STANDARDS PROJECT

Draft Standard for Information Technology — Portable Operating System Interface (POSIX) Part 2: Shell and Utilities — Amendment

Sponsor

Portable Application Standards Committee
of the
IEEE Computer Society

Work Item Number: JTC 1 22.41

Abstract: P1003.2b is part of the POSIX series of standards for applications and user interfaces to open systems. It consists of modifications and clarifications to ISO/IEC 9945-2: 1993 (IEEE Std 1003.2-1992), including support for symbolic links, a new archive/interchange format, and other modifications and clarifications prompted by ISO/IEC balloting.

Keywords: API, application portability, data processing, open systems, operating system, portable application, POSIX, shell and utilities, user portability

P1003.2b / D12
June 1999

Editor's Notes

This section will not appear in the final document. It is used for editorial comments concerning this draft.

This is the second recirculation balloting draft of P1003.2b. Please see the balloting instructions in Annex I. See the Change History later in these notes for a summary of the nontrivial changes from the last working group meeting. This draft uses small numbers in the right margin in lieu of change bars. Diff marks "C" denote changes from Draft 11 to Draft 12. Diff marks "B" denote changes from Draft 10 to Draft 11. Editorial changes such as typos, grammatical errors or changes, changes in cross references, and removal of editorial notes are not diff-marked. Please note that it is not always feasible to get the diff marks exactly right; they will sometimes start or end a line too soon.

This draft attempts to fully document the authorization sources of all changes being made to IEEE Std 1003.2-1992. Thus, all interpretation requests and international balloting comments resulting in changes are cited explicitly. However, there is a large collection of changes related to the addition of symbolic link support that are not specifically cited; it was felt that these changes are so obvious in identification that no specific citations were required. See the Introduction (page v) for a list of authorized changes.

This draft modifies IEEE Std 1003.2-1992, which is technically identical to ISO/IEC 9945-2: 1993. (However, note that there are very minor editorial and line number differences between these two documents.) You can purchase the standard by contacting:

IEEE Publications
P.O. Box 1331
445 Hoes Lane
Piscataway, NJ 08855-1331

1 (800) 678-IEEE
+1 (732) 562-3800 (outside US)

Since portions of this standard are meant to be modifications of the base POSIX.2 standard, the draft headings have been set up to match the affected clauses and still go into the table of contents. Therefore, there are gaps in the clause numbers of some sections.

POSIX.2b Change History

This section is provided to track major changes between drafts.

Draft 12	[June 1999] Second IEEE recirculation draft.
	— Changes incorporated from Draft 11 ballot resolution, including substantial rework for `ex`, `more`, `vi`, and `cd`.
Draft 11	[March 1995] First IEEE recirculation draft.

- Major changes to the charmap format to accommodate ISO/IEC 10646 and to move character width information from **LC_CTYPE**. (The latter change also affected REs, `localedef` and `tr`.)
- Major revisions to `ex`, `more`, and `vi`.
- A number of `pax` changes to address Canadian concerns about the effects of invalid pathnames in `cpio` and `tar` archives, and other balloting resolution issues.
- Miscellaneous utility changes to address balloting comments and interpretation requests.

Draft 10 [June 1994] First IEEE balloting draft. This draft includes the working group input from the April 1994 meeting.

- The subclauses on BREs and EREs Matching Multiple Characters (2.8.3.3 and 2.8.4.3) were updated.
- The synopses of utilities dealing with the **[** `−h` **|** `−R` **]** and **[** `−H` **|** `−L` **]** options were cleaned up.
- Th effect of SIGQUIT on `ed` was specified.
- The `pax` list-mode format in 4.48.3.1 was changed significantly, based on a proposal from David Korn.
- A number of terminology changes were made in `sed`.
- The `xargs` `−E` option was changed.
- Escaping in `csplit` REs was specified.
- A security hole in `ex` (and `vi`) initialization was plugged. The meaning of `\l` et al was clarified. The indentation behavior using *eof* was clarified. The `beautify` option was deleted.
- References to {POSIX2_C_BIND} were deleted from `c89`.

Draft 9 [February 1994] This draft includes the working group input from the January 1994 meeting.

- The reorganization of standards with the APIs transferring to P1003.1a caused changes primarily in Sections 1 and 2, and the deletion of Section 7 and Annex B.
- The new `pax` format was changed significantly, based on a proposal from Hal Jespersen.
- The symbolic link interfaces were changed significantly, based on a proposal from Keith Bostic.
- The `file` command added support for the traditional *magic* file. Thanks to Keith for this big addition.
- Miscellaneous minor changes to `dd`, `ed`, `ex`, `sed`, `tr`, and `write`.

Draft 8 [December 1993] This draft includes the working group input from the October 1993 meeting.

— Miscellaneous minor changes to `ed`, `ex`, `find`, `patch`, `test`, `uudecode`, `uuencode`, `vi`, `xargs`, and the rationale for *system*().

Draft 7 [October 1993] This draft includes the working group input from the April and July 1993 meetings.

— A number of the Annex H changes were addressed.

Draft 6 [March 1993] This draft includes the working group input from the January 1993 meeting.

— Mods to the `date` and `pax` commands.

— Minor mods to LC_CTYPE (2.5), `tr`

Draft 5 [December 1992] This draft includes the working group input from the October 1992 meeting.

— Modifications based on Japanese proposals for state-dependent encoding, character width definitions, and era date/time formats.

— Minor mods to `iconv`, `pax`, and `sed`.

Draft 4 [August 1992] This draft includes the working group input from the April and July 1992 meetings.

— Integration of the WG15 requirements (POSIX.2/D12 Annex H) for enhancements. Although many of these are currently placeholders for promised proposals from Japan and Denmark, there are substantive additions as follows:

— Locale definition (2.5) has a new LC_CTYPE `charclass` keyword.

— The `date` utility has added field widths.

— The `pax` format has been updated, based on work by Mark Brown and David Rowley, to include support for the 10646 UTF canonical form.

— The `uuencode` utility has added an option for the Internet Base64 format.

— The `uudecode` utility has added a `−o` option to override the output pathname.

— A new `iconv` utility has been added to convert codesets.

Draft 3 [February 1992] Miscellaneous minor changes to the `pax` format, provided by Mark Brown. Symbolic link material added, based on initial proposals from Dawn Burnett, as modified by the working group.

Draft 2	[December 1991] Miscellaneous minor changes to the pax format, provided by Mark Brown. Limited online access provided as part of an IEEE Computer Society experiment.
Draft 1	[September 1991] Conversion of pax formatting from P1003.1b Draft 5 and cpio and pax from IEEE Std 1003.1-1990.

Contents

FIGURES

TABLES

Introduction

[This introduction is not a normative part of P1003.2b, Draft Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities — Amendment, but is included for information only.]

This amendment to ISO/IEC 9945-2: 1993 (IEEE Std 1003.2-1992) was developed to address issues associated with the harmonization of the IEEE standard and the ISO/IEC International Standard. When the Draft International Standard was approved, ISO/IEC JTC 1/SC22/WG15 listed specific areas in which enhancements should be evaluated. Furthermore, it was realized that such a large standard would encounter various problems (interpretations, clarifications, elimination of ambiguities, conflicts with test suites, etc.) as it was implemented. Therefore, this amendment work was authorized with the following goals.[1)]

(1) Resolve international comments on ISO/IEC 9945-2: 1993. (See Annex H of that International Standard for a specific list of these areas.)

(2) Resolve issues resulting from requests for interpretation of IEEE Std 1003.2-1992.

(3) Improve the clarity, accuracy, and precision of the language in IEEE Std 1003.2-1992, correcting deficiencies found in implementing systems, test suites, or applications based on the documents.

(4) Resolve issues identified by IEEE working groups producing functional standards (profiles) that desire finer granularity in groupings of optional utilities and features.

(5) Incorporate interfaces associated with new facilities being produced by the P1003.1a project, such as symbolic links.

(6) Assume responsibility for definition of file interchange and archiving formats from P1003.1. This would involve movement of the current section 10 in IEEE Std 1003.1-1990 and the proposed new format from P1003.1a to the clause in P1003.2 that describes the pax utility.

1) These goals are paraphrased from the IEEE P1003.2b Project Authorization Request (PAR).

Related Standards Activities

Activities to extend this standard to address additional requirements are in progress, and similar efforts can be anticipated in the future.

The following areas are under active consideration at this time, or are expected to become active in the near future:[2)]

(1) Shell and Utility facilities 1

(2) Verification testing methods

(3) Realtime facilities 1

(4) Network interface facilities

(5) System Administration

(6) Profiles describing application- or user-specific combinations of Open Systems standards for: supercomputing, multiprocessor, and batch extensions; transaction processing; realtime systems; and multiuser systems based on historical models

(7) Services for reliable, available and serviceable systems 1

Extensions are approved as “amendments” or “revisions” to this document, following the IEEE and ISO/IEC Procedures.

Approved amendments are published separately until the full document is reprinted and such amendments are incorporated in their proper positions.

If you have interest in participating in the Portable Application Standards Committe (PASC) working groups addressing these issues, please send your name, address, and phone number to the Secretary, IEEE Standards Board, Institute of Electrical and Electronics Engineers, Inc., P.O. Box 1331, 445 Hoes Lane, Piscataway, NJ 08855-1331, and ask to have this forwarded to the chairperson of the appropriate PASC working group. If you have interest in participating in this work at the international level, contact your ISO/IEC national body.

2) A *Standards Status Report* that lists all current IEEE Computer Society standards projects is available from the IEEE Computer Society, 1730 Massachusetts Avenue NW, Washington, DC 20036-1903; Telephone: +1 202 371-0101; FAX: +1 202 728-9614. Working drafts of POSIX standards under development are also available from this office.

This amendment to IEEE Std 1003.2-1992 was prepared by the Shell and Utilities Working Group, sponsored by the Portable Application Standards Committee of the IEEE Computer Society. At the time this standard was approved, the membership of the Shell and Utilities Working Group was as follows:

Editor's Note: The full membership list will be provided in a future draft.

Draft Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities — Amendment

Section 1: Revisions to General

1.1 Scope

⇒ **1.1 Scope.** *Update references to POSIX.1-1990 to be the version amended by P1003.1a.*

Rationale: The P1003.1 and P1003.2 working groups have agreed that the P1003.1a and P1003.2b drafts will be submitted to the IEEE Standards Board for approval at the same time.

⇒ **1.1 Scope.** *At the beginning of the eleventh paragraph, delete the following:*

> Portions of this standard comprise optional language bindings to system service interfaces. (See, for example, the C-Language Bindings Option in Annex B.)

Rationale: The P1003.1 and P1003.2 working groups have agreed that all C-language APIs will be transferred into the P1003.1a amendment.

⇒ **1.1 Scope.** *Delete the twelfth paragraph, which reads:*

For language interfaces, or functions, this standard has been defined exclusively at the source-code level. The objective is that a conforming portable application source program can be translated to execute on a conforming implementation. This standard assumes that the source program may need to be retranslated to produce target code for a new environment prior to execution in that environment.

1.2 Normative References

⇒ **1.2 Normative References.** *Update the reference to POSIX.1 {8} to represent the version including the IEEE Std 1003.1a-199x and IEEE Std 1003.1b-1993 amendments (and 1003.1c if it is approved in time).*

⇒ **1.2 Normative References.** *Add the following entry to the Normative References clause:*

{10} ISO/IEC 10646-1: 1993, *Information technology—Universal Multiple-Octet Coded Character Set (UCS)—Part 1: Architecture and Basic Multilingual Plane.*

1.3 Conformance

⇒ **1.3 Conformance.** *Delete all references to the C-Language Bindings Option and the* {POSIX2_C_BIND} *symbol from 1.3 and all of its subclauses.*

⇒ **1.3.1.1 Requirements.** *Change item (3) to:*

(3) The system may provide additional or enhanced utilities or facilities not required by this standard. Nonstandard extensions should be identified as such in the system documentation. Nonstandard extensions, when used, may change the behavior of utilities or facilities defined by this standard. In such cases, the conformance document of the implementation (see 2.2.1.3) shall define an execution environment (i.e., shall provide general operating instructions) in which an application can be run with the behavior specified by this standard. In no case shall such an environment require modification of a Strictly Conforming POSIX.2 Application.

Rationale: Since Annex B is gone, all references to “functions” have to be removed.

1.4 Test Methods

⇒ **1.4 Test Methods.** *Change the entire clause to:* B

The test methods for this standard are described in P2003.2 {Bxx}.

Editor’s Note: This will be updated to indicate a revised P2003.2 (if and when a PAR is authorized).

Section 2: Revisions to Terminology and General Requirements

Editor's Note: The following material on symbolic links is related to P1003.1a/D12; the definition is from that draft verbatim. All of the symbolic link material in this and later sections is contingent on P1003.1a being approved before P1003.2b. The P1003.1 and P1003.2 working groups have agreed that the P1003.1a and P1003.2b drafts will be submitted to the IEEE Standards Board for approval at the same time. When P1003.1a is approved, a number of the P1003.2 definitions copied from POSIX.1 {8} will be updated automatically. See also the considerable rationale text about symbolic links being added to Annex E.

2.2.2 General Terms

⇒ **2.2.2.35 collation sequence.** *Change the second paragraph of this definition (the one beginning with* "The character order, . . . "*) to:*

The collation sequence is used for sorting and is determined from the collating weights assigned to each collating element. In the absence of weights, the collation sequence is also the *collating element order* (see 2.2.2.201).

Rationale: This change is the result of interpretation requests PASC 1003.2-92 #27 and #40 submitted for IEEE Std 1003.2-1992.

⇒ **2.2.2.87 hard link.** *Replace the definition with the following:*

The relationship between two directory entries that represent the same file; the result of an execution of the `ln` utility (without the `−s` option) or the POSIX.1 {8} *link*() function.

⇒ **2.2.2.165 source code.** *Change the second and third paragraphs to:*

When dealing with an ISO/IEC conforming programming language, source code is input to a compiler conforming to that ISO/IEC standard.

Rationale: Since Annex B is gone, all references to C-Language Binding Option have to be removed.

⇒ **2.2.2 General Terms.** *Modify the contents of subclause 2.2.2, General Terms, to add the following definitions in the correct sorted order [disregarding the subclause numbers shown here].*

2.2.2.201 collating element order: The relative order of collating elements as determined by the setting of the LC_COLLATE category in the current locale.

The collating element order is used in range expressions in REs (see 2.8) and is determined by the order in which collating elements are specified between `order_start` and `order_end` keywords in the LC_COLLATE category.

2.2.2.202 symbolic link: A type of file that contains a string whose length is less than or equal to {SYMLINK_MAX}.

The string in the file is interposed into a pathname being resolved, when the file is encountered during pathname resolution, to create a new pathname. [P1003.1a/D12]

2.2.3 Abbreviations

⇒ **2.2.3 Abbreviations.** *Modify the contents of subclause 2.2.3, Abbreviations, to add the following definition in the correct sorted order [disregarding the subclause numbers shown here].*

2.2.3.201 UTF8: The File-System Safe Universal Translation Format defined in Annex N of ISO/IEC 10646 {10}, as amended by ISO/IEC JTC 1/SC2/WG2 N993.

2.3 Built-In Utilities

⇒ **2.3 Built-In Utilities.** *In Table 2-3, add the* `pwd` *utility in the proper sorted order.*

Rationale: Changes to the `pwd` utility in this draft require it to affect the environment variable **PWD**, so it must become a shell built-in. C C

⇒ **2.3 Built-In Utilities.** *Delete the final paragraph in this subclause (the one beginning* "Since exec-able versions . . . "*).* B B

Rationale: As part of a general cleanup to remove references to the now-deleted Chapter 7, this paragraph was removed because it is little more than rationale and duplicates material in the previous paragraph. B B B

2.4 Character Set

⇒ **2.4 Character Set.** *Replace the paragraph and following dashed list that begins* "The current version of this standard does not address fully"*, with:*

State-dependent character encodings are described in 2.4.2.

⇒ **2.4.1 Character Set Description File.** *Change the second paragraph (the one beginning* "Each character set . . . "*) to:* B B

Each character set description file, except those that use ISO/IEC 10646 {10} position values as the encoding values, shall define characteristics for the coded character set and the encoding for the characters specified in Table 2-4, and may define encoding for additional characters supported by the implementation. Other information about the coded character set may also be in the file. Coded character set character values shall be defined using symbolic character names followed by character encoding values. B B B B B B B

⇒ **2.4.1 Character Set Description File.** *Change the two consecutive paragraph that begin* "The encoding part . . . " *and* "Decimal constants . . . " *to:* B B

The encoding part shall be expressed as one (for single-byte character values) or more concatenated decimal, octal, or hexadecimal constants in the following formats: B B B

`"%cd%2d"`, *<escape_char>*, *<decimal byte value>* B

`"%cx%2x"`, *<escape_char>*, *<hexadecimal byte value>* B

`"%c%2o"`, *<escape_char>*, *<octal byte value>* B

C

Decimal constants shall be represented by two or three decimal digits, preceded by the escape character and the lowercase letter `d`; for example, `\d05`, `\d97`, or `\d143`. Hexadecimal constants shall be represented by two hexadecimal digits, preceded by the escape character and the lowercase letter `x`; for example, `\x05`, `\x61`, or `\x8f`. Octal constants shall be represented by two or B B B B B three octal digits, preceded by the escape character; for example, `\05`, `\141`, or `\217`. In a portable charmap file, each constant shall represent an 8 b byte. Implementations supporting other byte sizes may allow constants to represent values larger than those that can be represented in 8 b bytes, and to allow additional digits in constants. When constants are concatenated for multibyte character values, they shall be of the same type and interpreted in byte order from first to last with the least significant byte of the multibyte character specified by the last constant. The manner in which these constants are represented in the character stored in the system is implementation defined. Omitting bytes from a multibyte character definition produces undefined results. B

⇒ **2.4.1 Character Set Description File.** *Add a new paragraph preceding the one that consists of* "The comment is optional." B

In lines defining ranges of symbolic names that also use ISO/IEC 10646 {10} position constant values, the conversion to the target codeset encoding value shall be performed before assignment of encoding values to symbolic names. B

Editor's Note: The following rationale will be added to E.2.4.1, but is kept here for this draft: B

(Rationale text deleted in Draft 12.) B

⇒ **2.4.1 Character Set Description File.** *Delete the final paragraph in the subclause, which was:*

For interpretation of the dollar sign and the number sign, see 2.2.2.45 and 2.2.2.110.

Rationale: This change satisfies the following corrigendum request from ISO/IEC 9945-2: 1993 Annex H.2:

(2) The final paragraph of 2.4.1 implies that there are special interpretations of the dollar sign and number sign characters described in 2.2.2, but no text appears in 2.2.2.45 or 2.2.2.110 to explain these interpretations.

⇒ **2.4.1 Character Set Description File.** *Add the following text to the end of the subclause:* B

The following declarations can follow the character set mapping definitions (after the `END CHARMAP` statement). Each shall consist of the keyword shown in the following list, starting in column 1, followed by the value(s) to be associated to the keyword, as defined below. B

`WIDTH`	An unsigned positive integer value defining the column width (see 2.2.2.36) for the coded character set specified in Table 2-4 and Table 2-5. Coded character set character values shall be defined using symbolic character names followed by column width values. B

Defining a character with more than one WIDTH produces undefined results. The END WIDTH keyword shall be used to terminate the WIDTH definitions.

WIDTH_DEFAULT An unsigned positive integer value defining the default column width for any printable character not listed by one of the WIDTH keywords. If no WIDTH_DEFAULT keyword is included in the charmap, the default character width shall be 1.

Example:

After the END CHARMAP statement, a syntax for a width definition would be:

```
WIDTH
<NUL>...<IS1>              -1
<A>                        1
<B>                        1
<C>...<Z>                  1
...
<foo1>...<foon>            2
...
END WIDTH
```

The code point values represented by the symbols <A> and <B> are assigned a width of 1. Also, the code point values <C> to <Z> inclusive (<C>, <D>, <E>, <F>, <G>, <H>, <I>, and so on) are assigned a width of 1.

In this example, <A>...<Z> would have required fewer lines, but the alternative was given to demonstrate flexibility.

The keyword WIDTH_DEFAULT can be added as appropriate. All nonprintable characters shall have a width of –1.

Rationale: This change satisfies the following requirement from ISO/IEC 9945-2: 1993 Annex H.1:

(9) The definition of column position (see 2.2.2.36) relies on the implementation's knowledge of the integral width of the characters. The charmap (2.4) or LC_CTYPE (2.5.2.1) locale definitions should be enhanced to allow application specification of these widths.

The character "width" information was first considered for inclusion under LC_CTYPE but was moved because it is more closely associated with the information in the charmap than information in the locale source (cultural conventions information). Concerns were raised that formalizing this type of information is moving the locale source definition from the codeset independent entity that it was designed to be to a repository of codeset specific information. A similar issue occured with the <code_set_name>, <mb_cur_max>, and <mb_cur_min> information, which was resolved to reside in the charmap definition.

The width definition was added to the POSIX.2b standard with the intent that the functions *wcswidth*() and/or *wcwidth*() [currently specified in the X/Open

Common Application Environment Specification (X/Open CAE), System Interfaces and Headers, Version 2] be the mechanism to retrieve the character width information. B

⇒ **2.4 Character Set.** *Add the following new subclause:*

2.4.2 State-Dependent Character Encodings

This subclause addresses the use of state-dependent character encodings (i.e., those in which the encoding of a character is dependent on one or more shift codes that may precede it).

A single-shift encoding (where each character not in the initial shift state is preceded by a shift code) can be defined in the charmap format if each shift-code/character sequence is considered a multibyte character, defined using the concatenated-constant format described in 2.4.1. If the implementation supports a character encoding of this type, all of the standard utilities shall support it.

A locking-shift encoding (where the state of the character is determined by a shift code that may affect more than the single character following it) could be defined with an extension to the charmap format described in 2.4.1. If the implementation supports a character encoding of this type, any of the standard utilities that describe character (versus byte) or text-file manipulation shall have the following characteristics: B

(1) The utility shall process the statefully encoded data as a concatenation of state-independent characters. The presence of redundant locking shifts shall not affect the comparison of two statefully encoded strings.

(2) A utility that divides, truncates, or extracts substrings from statefully encoded data contain locking shifts at the beginning or end of the resulting data, if appropriate, to retain correct state information. B

State-Dependent Character Encodings Rationale

A requirement was considered that would force utilities to eliminate any redundant locking shifts, but this was left as a quality of implementation issue.

Rationale: This change satisfies the following requirement from ISO/IEC 9945-2: 1993 Annex H.1:

(8) The support of state-dependent (shift encoding) character sets should be addressed fully. See descriptions of these in 2.4. If such character encodings are supported, it is expected that this will impact 2.4 (charmap), 2.5 (locale definition), 2.8 (regular expressions), and the `comm`, `cut`, `diff`, `grep`, `head`, `join`, `paste`, and `tail` utilities.

2.5 Locale

B

⇒ **2.5 Locale.** *Change the second paragraph (the one following the list of environment variable names) to:*

Conforming implementations shall implement the standard utilities so that their behavior is based on the current locale, as defined in the Environment Variables subclause for each utility.

Rationale: Since Annex B is gone, all references to it and to "functions" have to be removed.

⇒ **2.5.2 Locale Definition.** *In the numbered list, change the first sentence of item (2) to:*

(2) A character in the portable character set can be represented by the character itself, in which case the value of the character is implementation defined. (Implementations may allow other characters to be represented as themselves, but such locale definitions are not portable.)

Rationale: This change was made in response to a Japanese ballot comment to ISO/IEC 9945-2: 1993.

⇒ **2.5.2.1 LC_CTYPE.** *Add the following keyword items between the items labeled* `blank` *and* `toupper`*:*

`charclass` Define one or more locale-specific character class names as strings separated by semicolons. Each named character class can then be defined subsequently in the LC_CTYPE definition. B A character class name shall consist of at least one and no B more than {CHARCLASS_NAME_MAX} bytes of alphanumeric B characters from the portable filename character set. The first B character cannot be a digit. The name cannot match any of B the LC_CTYPE keywords defined in this standard. Future B revisions of this standard will not specify any LC_CTYPE key- B words containing uppercase letters. B

charclass-name
Define characters to be classified as belonging to the named locale-specific character class. In the POSIX Locale, the locale-specific named character classes need not exist.

If a class name is defined by a `charclass` keyword, but no characters are subsequently assigned to it, this is not an error; it shall represent a class without any characters belonging to it.

The *charclass-name* can be used in regular expression and shell pattern-matching bracket expressions, and by the `tr` utility. B B

Rationale: This addition was adopted from XPG4 {B49} to satisfy the following requirement from ISO/IEC 9945-2: 1993 Annex H.1:

(3) The LC_CTYPE (2.5.2.1) locale definition should be enhanced to allow user-specified additional character classes, similar in concept to the C Standard {7} Multibyte Support Extension (MSE) *is_wctype*() function. C C

The symbolic constant {CHARCLASS_NAME_MAX} was adopted from the XPG4 {B49}. Application portability is enhanced by the use of symbolic constants. C C

⇒ **2.5.2.1 LC_CTYPE.** *Add the following keyword items between the items labeled* `digit` *and* `space`*:*

`alnum`	Define characters to be classified as letters and numeric digits. Only the characters specified for the `alpha` and `digit` keywords shall be specified. Characters specified for the keywords `alpha` and `digit` are automatically included in this class.

Rationale: The `alnum` keyword was added to correct an oversight in POSIX.2-1992; it was clearly implied by the POSIX Locale table, but since it was mentioned only in a comment field, it was considered not normative.

B

⇒ **2.5.2.2.4 Collation Sequence.** *Remove the following sentence from the second paragraph:*

The NUL character shall compare lower than any other character.

Rationale: This change partially satisfies the following requirement from ISO/IEC 9945-2: 1993 Annex H.1:

(7) The specific encoding and collation requirements for the character NUL should be removed.

The specific encoding was retained because the C Standard {7} requires it.

⇒ **2.5.2.3 LC_MONETARY.** *In Table 2-9, add the following after the entry for* `int_frac_digits`*:*

```
frac_digits      -1
```

Rationale: This change satisfies the following corrigendum request from ISO/IEC 9945-2: 1993 Annex H.2:

(3) Table 2-9, listing the LC_MONETARY Category Definition in the POSIX Locale, omits the value to be assigned to `frac_digits`.

⇒ **2.5.2.5 LC_TIME.** *Add new keywords in between* `era_d_fmt` *and* `alt_-digits`*:* B

`era_d_t_fmt`	The format of the date and time in alternate era notation, corresponding to the `%Ec` field descriptor. B
`era_t_fmt`	The format of the time in alternate era notation, corresponding to the `%EX` field descriptor. B

Rationale: This change was to correct an oversight in ISO/IEC 9945-2: 1993, pointed out by Japan. It is identical to an extension in XPG4 {B49}.

⇒ **2.5.2.5 LC_TIME.** *In Table 2-11, change the lines defining* `t_fmt_ampm` *to:*

```
# Appropriate 12 h time representation (%r) "%I:%M:%S %p"
t_fmt_ampm "<percent-sign><I><colon><percent-sign><M><colon>\
<percent-sign><S><space><percent_sign><p>"
```

Rationale: This change satisfies the following corrigendum request from ISO/IEC 9945-2: 1993 Annex H.2:

(5) Table 2-11, listing the LC_TIME Category Definition in the POSIX Locale, contains the following entry:

```
# Appropriate 12 h time representation (%r) "%I:%M:%S %p"
t_fmt_ampm "<percent-sign><I><colon><percent-sign><M><colon>\
<percent-sign><S> <percent_sign><p>"
```

It in unclear whether there is a space between `<S>` and `<percent_-sign>` (which should have been represented as `<space>` to match the other entries) or whether this is a typographical error.

⇒ **2.5.3.1 Locale Lexical Conventions.** *Add the following token description:*

`CHARCLASS`	A string of alphanumeric characters from the portable character set, the first of which shall not be a digit, consisting of at least one and at most {CHARCLASS_NAME_MAX} bytes, and optionally surrounded by double-quotes. B

Rationale: See the 2.5.2.1 changes.

⇒ **2.5.3.2 Locale Grammar.** *Modify the* `ctype_keyword` *and* `charclass_keyword` *descriptions as follows:*

```
ctype_keyword          : charclass_keyword charclass_list EOL
                       | charwidth_keyword charclass_list EOL
                       | defwidth_keyword defwidth_value EOL
                       | charconv_keyword charconv_list EOL
                       | 'charclass' charclass_namelist EOL
                       ;
charclass_namelist     : charclass_namelist ';' CHARCLASS
                       | CHARCLASS
                       ;
charclass_keyword      : 'upper' | 'lower' | 'alpha' | 'digit'
                       | 'alnum' | 'xdigit' | 'space' | 'print'
                       | 'graph' | 'blank' | 'cntrl' | 'punct'
                       | CHARCLASS
                       ;
```
B

⇒ **2.5.3.2 Locale Grammar.** *In the* `time_keyword_opt` *description, add* `'era_d_t_fmt'` *and* `'era_t_fmt'` *as alternatives to the four existing entries.* B

Rationale: See the 2.5.2.1 changes.

2.6 Environment Variables B

⇒ **2.6 Environment Variables.** *In the fourth paragraph, change the sentence* "See 7.2 and 3.12 for methods of accessing these variables." *to:* B

See the *getenv*() function in POSIX.1 {8} and 3.12 for methods of accessing these variables. B

Rationale: This change is part of a general cleanup to remove references to the now-deleted Chapter 7. B

⇒ **2.6 Environment Variables.** *Add the following variable in proper sorted order:* B

PWD	This variable shall represent an absolute pathname of the current working directory. It shall not contain any filename components of dot or dot-dot. The value is set by the `cd` utility. B

2.8 Regular Expression Notation

⇒ **2.8.1 RE Introduction.** *Delete the final sentence in this subclause:* "Both BREs and EREs are supported by the RE Matching interface in 7.3."

Rationale: As part of a general cleanup to remove references to the now-deleted Chapter 7, this sentence was removed because it was little more than rationale.

⇒ **2.8.3.2 RE Bracket Expression.** *Change the first paragraph of item (7) to:*

(7) A *range expression* represents the set of collating elements that fall between two elements in the collating element order (see 2.2.2.201) of the current locale, inclusive. A range expression shall be expressed as the starting point and the ending point separated by a hyphen (–).

Rationale: This change is the result of interpretation request PASC 1003.2-92 #27 submitted for IEEE Std 1003.2-1992.

⇒ **2.8.3.3 BREs Matching Multiple Characters.** *In the numbered list, change item (3) to:*

(3) The *backreference* expression `\`*n* shall match the same (possibly empty) string of characters as was matched by a subexpression enclosed between `\(` and `\)` preceding the `\`*n*. The character *n* shall be a digit from `1` through `9`, specifying the *n*-th subexpression [the one that begins with the *n*-th `\(` and ends with the corresponding paired `\)`]. The expression is invalid if less than *n* subexpressions precede the `\`*n*. For example, the expression `^\(.*\)\1$` matches a line consisting of two adjacent appearances of the same string, and the expression `\(a\)*\1` fails to match `a`. When the referenced subexpression matched more than one string, the backreferenced expression shall refer to the last matched string. If the subexpression referenced by the backreference matches more than one string because of an asterisk (*) or an interval expression [see item (5)], the backreference shall match the last (rightmost) of these strings.

Rationale: The changes to 2.8.3.3 and 2.8.4.3 remove an unspecified or ambiguous behavior in POSIX.2, aligning it with the requirements specified for the *regcomp*() expression, and is the result of interpretation request PASC 1003.2-92 #43 submitted for IEEE Std 1003.2-1992.

⇒ **2.8.3.3 BREs Matching Multiple Characters.** *At the end of the subclause, add a new paragraph:*

A subexpression repeated by an asterisk (*) or an interval expression shall not match a null expression unless this is the only match for the repetition or it is necessary to satisfy the exact or minimum number of occurrences for the interval expression.

⇒ **2.8.4.3 EREs Matching Multiple Characters.** *At the end of the subclause, add a new paragraph:*

An ERE matching a single character repeated by an *, ?, or an interval expression shall not match a null expression unless this is the only match for the repetition or it is necessary to satisfy the exact or minimum number of occurrences for the interval expression. B

⇒ **2.8.5.2 RE and Bracket Expression Grammar.** *In the section of the grammar for the* `nondupl_RE` *nonterminal, remove the third line:*

```
                    | Back_open_paren Back_close_paren
```

Rationale: This change is the result of interpretation request PASC 1003.2-92 #43 submitted for IEEE Std 1003.2-1992. Although the grammar required support for null subexpressions, subclause 2.8.3.3 does not describe the meaning of, and historical practice did not support, this construct.

2.9.1.4 File Read, Write, and Creation

⇒ **2.9.1.4 File Read, Write, and Creation.** *In the first numbered list, change item (3) to:*

(3) If the file is a regular file, the permission bits are set to

S_IROTH | S_IWOTH | S_IRGRP | S_IWGRP | S_IRUSR | S_IWUSR

(see Section 5.6.1.2 of POSIX.1 {8}), except that the bits specified by the file mode creation mask of the process are cleared.

If the file is a directory, the permission bits are set to

S_IRWXU | S_IRWXG | S_IRWXO

(see Section 5.6.1.2 of POSIX.1 {8}), except that the bits specified by the file mode creation mask of the process are cleared.

Rationale: This change is required to match historical practice and is the result of interpretation request PASC 1003.2-92 #18 submitted for IEEE Std 1003.2-1992.

⇒ **2.9.1.4 File Read, Write, and Creation.** *In the first numbered list, change item (6) to:* B

(6) If the file is a symbolic link, the effect shall be undefined unless the {POSIX2_SYMLINKS} variable is in effect for the directory in which the symbolic link would be created. B

(7) Unless otherwise specified, the file created shall be a regular file. B

Editor's Note: The following rationale will be added to E.2.9.1.8, but is kept here for this draft:

Pathname Resolution Rationale. *(This subclause is not a part of P1003.2b)*

P1003.1a now includes symbolic links in pathname resolution and a number of concepts are automatically inherited in POSIX.2 by this inclusion. The large majority of standard utilities resolve pathnames and operate on files without special arrangements for symbolic links. Because of the global POSIX.1 {8} inheritance, this entails very few modifications to utility descriptions.

2.10 Utility Conventions B

2.10.2 Utility Syntax Guidelines B

⇒ **2.10.2 Utility Syntax Guidelines.** *Change the first paragraph to:* B

The following guidelines are established for the naming of utilities and for the specification of options, option-arguments, and operands. The *getopt*() function in POSIX.1 {8} assists utilities in handling options and operands that conform to these guidelines. B

Rationale: This change is part of a general cleanup to remove references to the now-deleted Chapter 7. All of the applicable functions are now in POSIX.1-199x, the version created by the currently balloting P1003.1a. B

2.13 Configuration Values

⇒ **2.13.1 Symbolic Limits.** *Change the second paragraph (the one beginning* “The values specified in Table 2-17 . . . ”*) to:*

The values specified in Table 2-17 represent the lowest values conforming implementations shall provide and, consequently, the largest values on which an application can rely without further enquiries, as described below. These values shall be accessible to applications via the `getconf` utility (see 4.26).

⇒ **2.13.1 Symbolic Limits.** *Change the fourth paragraph (the one beginning* “The functions in 7.8.2 . . . ”*) to:*

The `getconf` utility shall return the value of each symbol on each specific implementation. The value so retrieved shall be the largest, or most liberal, value that shall be available throughout the session lifetime, as determined at session creation.

⇒ **2.13.1 Symbolic Limits.** *Add a new symbol to Table 2-17:* B

Name	Description	Value
{POSIX2_CHARCLASS_NAME_MAX}	The maximum number of bytes in a character class name.	14

⇒ **2.13.1 Symbolic Limits.** *Add a new symbol to Table 2-18:* B

Name	Description	Minimum Value
{CHARCLASS_NAME_MAX}	The maximum number of bytes in a character class name.	{POSIX2_CHARCLASS_NAME_MAX}

⇒ **2.13.2 Symbolic Constants for Portability Specifications.** *Change the first paragraph to:*

Table 2-19 lists symbols that can be used by the application to determine which optional facilities are present on the implementation. The `getconf` utility can be used to retrieve the value of each symbol on each specific implementation.

⇒ **2.13.2 Symbolic Constants for Portability Specifications.** *Delete the* Table 2-19 *entry for* {POSIX2_C_BIND}.

Rationale: The preceding four changes are related to the removal of Annex B.

2.13.3 Pathname Variable Values

⇒ **2.13.3 Pathname Variable Values.** *Add a new subclause, 2.13.3, Pathname Variable Values, as follows:*

The values in Table 2-100 may be constants within an implementation or may vary from one pathname to another.

Table 2-100 – Pathname Variable Values

Name	Description
{POSIX2_SYMLINKS}	When referring to a directory, the system supports the creation of symbolic links within that directory; for nondirectory files, the meaning of {POSIX2_SYMLINKS} is undefined.

Symbolic Constants Rationale. *(This subclause is not a part of P1003.2b)*

The {POSIX2_SYMLINKS} variable indicates that the underlying operating system supports the creation of symbolic links in specific directories. Many of the POSIX.2 utilities that deal with symbolic links do not depend on this value. For example, a utility that follows symbolic links (or does not, as the case may be) will only be affected by a symbolic link if it encounters one. Presumably, a file system that does not support symbolic links will not contain any. This variable does affect such utilities as `ln –s` and `pax` that attempt to create symbolic links.

{POSIX2_SYMLINKS} was developed even though there is no comparable configuration value in P1003.1a. Since POSIX.2 does not depend on a fully conforming POSIX.1 {8} system underneath, the developers of the standard wished to allow systems in which this was an optional feature, perhaps on a file system basis.

Section 3: Revisions to Shell Command Language

⇒ **3.1 Shell Introduction.** *Change the first paragraph to:*

> The shell is a command language interpreter. This section describes the syntax of that command language as it is used by the `sh` utility and the POSIX.1 {8} *system*() and *popen*() functions.

Rationale: This and the following change are part of a general cleanup to remove references to the now-deleted Chapter 7. All of the applicable functions are now in POSIX.1-199x, the version created by the currently balloting P1003.1a.

⇒ **3.1 Shell Introduction.** *Change the first numbered item to:*

> (1) Reads its input from a file (see `sh` in 4.56), from the −`c` option, or from the POSIX.1 {8} *system*() or *popen*() functions. If the first line of a file of shell commands starts with the characters #!, the results are unspecified.

⇒ **3.2.3 Double Quotes.** *Change the description of backslash to:*

> \\ The backslash shall retain its special meaning as an escape character (see 3.2.1) only when followed by one of the following characters when considered special:
>
> `$` `‘` `"` `\` `<newline>`

Rationale: This change is the result of interpretation request PASC 1003.2-92 #102 submitted for IEEE Std 1003.2-1992.

⇒ **3.5.3 Environment Variables.** *Change the description of* **ENV** *to:*

> **ENV** This variable, when and only when an interactive shell is invoked, shall be subjected to parameter expansion (see 3.6.2) by the shell, and the resulting value shall be used as a pathname of a file containing shell commands to execute in the current environment. The file need not be executable. If the expanded value of **ENV** is not an absolute pathname, the results are unspecified. **ENV** shall be ignored if the real and effective user IDs or real and effective group IDs of the user are different.

⇒ **3.5.3 Environment Variables.** *Add a new variable in the proper sorted order:*

PWD	This variable shall be set by the shell to be an absolute pathname of the current working directory, containing no components of type symbolic link, no components that are dot, and no components that are dot-dot when the shell is initialized. If an application sets or unsets the value of **PWD**, the behaviors of the `cd` and `pwd` utilities are unspecified.

Editor's Note: The following rationale will be added to E.3.5.3, but is kept here with Environment Variables for this draft:

Environment Variables Rationale. *(This subclause is not a part of P1003.2b)*

A previous version of this standard did not specify whether **ENV** file processing was performed by noninteractive shells. Historical practice supports **ENV** processing only for interactive shells, and this is what the standard now requires.

⇒ **3.9.4.3 `case` Conditional Construct.** *In the first paragraph, replace the sentence* "The compound-list for each list of patterns shall be" *terminated with ;;." with:*

The *compound-list* for each list of patterns, with the possible exception of the last, shall be terminated with `;;`.

Rationale: This change is the result of interpretation request PASC 1003.2-92 #46 submitted for IEEE Std 1003.2-1992.

⇒ **3.9.4.3 `case` Conditional Construct.** *Replace the synopsis of the* `case` *statement with:*

```
case word in
     [[(]pattern[|pattern]... )   compound-list;;]...
     [[(]pattern[|pattern]... )   compound-list]
esac
```

⇒ **3.10.2 Shell Grammar Rules.** *Replace the rules for* `case_clause`, `case_list`, *and* `case_item` *with:*

```
case_clause  : Case WORD linebreak in linebreak case_list    Esac
             | Case WORD linebreak in linebreak case_list_ns Esac
             | Case WORD linebreak in linebreak              Esac
             ;
```

```
case_list_ns : case_list case_item_ns
             |           case_item_ns
             ;
case_list    : case_list case_item
             |           case_item
             ;
case_item_ns :     pattern ')' linebreak     linebreak
             |     pattern ')' compound_list linebreak
             | '(' pattern ')' linebreak     linebreak
             | '(' pattern ')' compound_list linebreak
             ;
case_item    :     pattern ')' linebreak     DSEMI linebreak
             |     pattern ')' compound_list DSEMI linebreak
             | '(' pattern ')' linebreak     DSEMI linebreak
             | '(' pattern ')' compound_list DSEMI linebreak
             ;
```

Rationale: This change is required to match historical practice and is the result of interpretation request PASC 1003.2-92 #46 submitted for IEEE Std 1003.2-1992. The `case` construct in 3.9.4.3 was incorrectly described in 1992 as requiring a minimum of two compound lists, when the grammar and historical practice allowed the `case_item` to be omitted. The grammar did not allow the historical practice of omitting the final `;;` (that was already described in 3.9.4.3).

3.13 Pattern Matching Notation

⇒ **3.13 Pattern Matching Notation.** *At the end of the first paragraph, change* "... the description of RE notation." *to:*

... the description of RE notation, modified to include backslash escape processing.

Rationale: This change, and the following in 3.13.1, are required to match historical practice and are the result of interpretation request PASC 1003.2-92 #21 submitted for IEEE Std 1003.2-1992.

⇒ **3.13.1 Patterns Matching a Single Character.** *At the end of the first paragraph, add:*

A `<backslash>` character shall escape the following character. The escaping `<backslash>` shall be discarded.

⇒ **3.14.11 `set` – Set/unset options and positional parameters.** *(This change should be read only in conjunction with the following change.) Change the Synopsis to:*

```
set [-abCefmnuvx] [-o option] ... [argument ...]
set [+abCefmnuvx] [+o option] ... [argument ...]
set –– [argument ...]
set -o
set +o
```

Obsolescent version:

```
set – [argument ...]
```

⇒ **3.14.11 `set` – Set/unset options and positional parameters.** *Add the following after the description of the* `–n` *option:*

`–o`	Write the current settings of the options to standard output in an unspecified format.
`+o`	Write the current option settings to standard output in a format that is suitable for reinput to the shell as commands that achieve the same options settings.

⇒ **3.13 `set` – Set/unset options and positional parameters.** *Change the description of the* `–x` *option to:*

`–x`	Write to standard error a trace for each command after the shell expands the command and before it executes it. It is unspecified whether the command that turns tracing off is traced.

Editor's Note: The following rationale will be added to E.3.14.11, but is kept here with `set` *for this draft:*

`set` Rationale. *(This subclause is not a part of P1003.2b)*

Historical implementations are inconsistent in the format used for `–o` option status reporting. The `+o` format without an option-argument was added to allow portable access to the options that can be saved and then later restored using, for instance, a dot script.

Historically, `sh` did trace the command `set +x`, but `ksh` did not.

Rationale: The preceding three changes are the result of interpretation requests PASC 1003.2-92 #79 and #99 submitted for IEEE Std 1003.2-1992. B

Section 4: Revisions to Execution Environment Utilities

4.1 awk – Pattern scanning and processing language

Rationale: The changes to awk are the result of interpretation requests PASC 1003.2-92 #91 and #107 submitted for IEEE Std 1003.2-1992.

⇒ **4.1.4 awk Operands.** *In the description of the assignment operand, change the fourth and fifth sentences (the ones beginning* "The variable shall be assigned . . . " *and* "If that value is considered a numeric string . . . "*) to*

The variable shall be assigned the value of that STRING token and, if appropriate, shall be considered a *numeric string* (see 4.1.7.2).

⇒ **4.1.5.1 awk Standard Input.** *Add to the end of the paragraph:*

If the awk program contains no actions and no patterns, but is otherwise a valid awk program, standard input and any *file* operands shall not be read and awk shall exit with a return status of zero.

⇒ **4.1.7.1 awk Overall Program Structure.** *Change the second paragraph (the one beginning* "A missing pattern . . . "*) to:*

A missing pattern shall match any record of input, and a missing action shall be equivalent to

```
{ print }
```

If the awk program contains no actions and no patterns, but is otherwise a valid awk program, standard input and any *file* operands shall not be read and awk shall exit with a return status of zero.

⇒ **4.1.7.1 awk Overall Program Structure.** *Change the last paragraph to:*

Execution of the awk program shall start by first executing the actions associated with all BEGIN patterns in the order they occur in the program. Then each *file* operand (or standard input if no files were specified) shall be processed in turn by reading data from the file until a record separator is seen (<newline> by default). Before the first reference to a field in the record is evaluated, the record shall be split into fields, according to the rules in 4.1.7.4, using the value of FS that was current at the time the record was read. Each pattern in the program then shall be evaluated in the order of occurrence, and the action associated with each pattern that matches the current record executed. The action for a matching pattern shall be executed before evaluating subsequent patterns. Finally, the actions associated with all `END` patterns shall be executed in the order they occur in the program.

⇒ **4.1.7.2 awk Expressions.** *Change the fourth paragraph (the one beginning* "A string value shall be converted to a numeric value . . . "*) and the following dashed list to the following text. In the paragraph following the list, change* "in the above steps" *to* "in the preceding description".

A string value shall be considered a *numeric string* if it comes from one of the following:

— Field variables

— Input from the `getline` function

— `FILENAME`

— `ARGV` array elements

— `ENVIRON` array elements

— Array elements created by the `split` function

— A command-line variable assignment

— Variable assignment from another *numeric string* variable

and after all the following conversions have been applied, the resulting string would lexically be recognized as a `NUMBER` token as described by the lexical conventions in 4.1.7.8:

— All leading and trailing blanks are discarded

— If the first non-<`blank`> character is + or −, it is discarded

— Changing each occurrence of the decimal point character from the current locale to a period

⇒ **4.1.7.2 awk Expressions.** *Change the final paragraph to:*

Comparisons (with the <, <=, !=, ==, >, and >= operators) shall be made numerically if both operands are numeric, if one is numeric and the other has a string value that is a *numeric string*, or if one is numeric and the other has the *uninitialized value*. Otherwise, operands shall be converted to strings as required, and a string comparison shall be made using the locale-specific collation sequence. The value of the comparison expression shall be 1 if the relation is true, or zero if the relation is false.

⇒ **4.1.7.3 awk Variable and Special Variables.** *Change the first paragraph (which currently contains four lines of text across a page break) to:*

Variables can be used in an awk program by referencing them. With the exception of function parameters (see 4.1.7.6.2.4), they are not explicitly declared. Function parameter names shall be local to the function; all other variable names shall be global. The same name shall not be used as both a function parameter name and as the name of a function or a special awk variable. The same name shall not be used both as a variable name with global scope and as the name of a function. The same name shall not be used within the same scope both as a scalar variable and as an array. Uninitialized variables, including scalar variables, array elements, and field variables shall have an *uninitialized value*. An *uninitialized value* shall have both a numeric value of zero and a string value of the empty string. Evaluation of variables with an *uninitialized value*, to either string or numeric, shall be determined by the context in which they are used.

⇒ **4.1.7.3 awk Variable and Special Variables.** *Change the second paragraph (the one beginning* "Field variables shall be designated . . . "*) to:*

Field variables shall be designated by a $ followed by a number or numeric expression. The effect of the field number *expression* evaluating to anything other than a nonnegative integer is unspecified; uninitialized variables or string values need not be converted to numeric values in this context. New field variables can be created by assigning a value to them. References to nonexistent fields (i.e., fields after $NF), shall evaluate to the *uninitialized value*. Such references shall not create new fields. However, assigning to a nonexistent field [e.g., `$(NF+2) = 5`] shall increase the value of NF; create any intervening fields with the *uninitialized value*; and cause the value of $0 to be recomputed, with the fields being separated by the value of OFS. Each field variable shall have a string value or an *uninitialized value* when created. Field variables shall have the *uninitialized value* when created from $0 using FS and the variable does not contain any characters. If appropriate, the field variable shall be considered a *numeric string* (see 4.1.7.2).

⇒ **4.1.7.3 awk Variable and Special Variables.** *In the first paragraph of the* ENVIRON *description, change the sentence* "If the value of an environment variable is considered a numeric string . . . " *to*

If appropriate, the environment variable shall be considered a *numeric string* (see 4.1.7.2).

⇒ **4.1.7.3 awk Variable and Special Variables.** *Change the description of* OFS *to:*

OFS The print statement output field separator; <space> by default.

⇒ **4.1.7.4 awk Regular Expressions.** *Change the final sentence in the first paragraph to:*

Using a slash character within an ERE token requires the escaping shown in Table 4-2.

⇒ **4.1.7.4 awk Regular Expressions.** *Add a new item (1) to the numbered list, changing the existing items to (2) and (3):*

(1) If FS is a null string, the behavior is unspecified.

⇒ **4.1.7.4 awk Regular Expressions.** *Change the first paragraph that follows the numbered list (which begins* "Except in the gsub, . . . "*) to:*

Except for the ~ and !~ operators, and in the gsub, match, split, and sub built-in functions, ERE matching shall be based on input records; i.e., record separator characters (the first character of the value of the variable RS, <newline> by default) cannot be embedded in the expression, and no expression shall match the record separator character. If the record separator is not <newline>, <newline> characters embedded in the expression can be matched. For the ~ and !~ operators, and in those four built-in functions, ERE matching shall be based on text strings; i.e., any character (including <newline> and the record separator) can be embedded in the pattern, and an appropriate pattern shall match any character. However, in all awk ERE matching, the use of one or more NUL characters in the pattern, input record, or text string produces undefined results.

⇒ **4.1.7.6.1 awk Output Statements.** *Change the first sentence of the second paragraph to:*

In all cases, the *expression* shall be evaluated to produce a string that is used as a pathname into which to write (for > or >>) or as a command to be executed (for |).

⇒ **4.1.7.6.2.1 `awk` Arithmetic Functions.** *Change the description of atan2 to:*

`atan2(`*y*`,`*x*`)`	Return arctangent of *y*/*x* in radians in the range −π to π.

⇒ **4.1.7.6.2.2 `awk` String Functions.** *Change the description of* `split` *to:*

`split(`*s*`,` *a*`[,` *fs*`])`	Split the string *s* into array elements *a*[1], *a*[2], ... , *a*[*n*], and return *n*. All elements of the array shall be deleted before the split is performed. The separation shall be done with the ERE *fs* or with the field separator `FS` if *fs* is not given. Each array element shall have a string value when created and, if appropriate, the array element shall be considered a *numeric string* (see 4.1.7.2). The effect of a null string as the value of *fs* is unspecified.

⇒ **4.1.7.6.2.2 `awk` String Functions.** *Change the description of* `sub` *to:*

`sub(`*ERE*`,` *repl*`[,` *in*`])`	Substitute the string *repl* in place of the first instance of the extended regular expression *ERE* in string *in* and return the number of substitutions. An ampersand (&) appearing in the string *repl* shall be replaced by the string from *in* that matches the ERE. An ampersand preceded with a backslash (\) shall be interpreted as the literal ampersand character. Any other occurrence of a backslash (e.g., preceding any other character) shall be treated as a literal backslash character. [Note that if *repl* is a string literal (the lexical token `STRING`, see 4.1.7.8), the handling of the ampersand character occurs after any lexical processing, including any lexical backslash escape sequence processing.] If *in* is specified and it is not an *lvalue* (see 4.1.7.2), the behavior is undefined. If *in* is omitted, `awk` shall use the current record (`$0`) in its place.

⇒ **4.1.7.6.2.2 awk String Functions.** *Change the description of* `substr` *to:*

`substr(`*s*`, `*m*`[, `*n*`])`
: Return the at most *n*-character substring of *s* that begins at position *m*, numbering from 1. If *n* is missing, or if *n* specifies more characters than are left in the string, the length of the substring shall be limited by the length of the string *s*.

⇒ **4.1.7.6.2.3 awk Input/Output and General Functions.** *In the description of expression* `| getline` **[***var***]**, *change* "file" *to* "stream".

⇒ **4.1.7.6.2.3 awk Input/Output and General Functions.** *Change the description of* `getline` *var to:*

`getline` *var*
: Set variable *var* to the next input record from the current input file and, if appropriate, *var* shall be considered a *numeric string* (see 4.1.7.2). This form of `getline` shall set the `FNR` and `NR` variables.

⇒ **4.1.7.6.2.3 awk Input/Output and General Functions.** *In the first paragraph of the description of expression* `| getline` **[***var***]**, *change the last sentence (the one beginning with* "If var is missing . . . "*) to:*

If *var* is missing, `$0` and `NF` shall be set; otherwise, *var* shall be set and, if appropriate, it shall be considered a *numeric string* (see 4.1.7.2).

⇒ **4.1.7.6.2.3 awk Input/Output and General Functions.** *In the description of* `getline` **[***var***]** `<` *expression, change* "full pathname" *to* "pathname".

⇒ **4.1.7.6.2.3 awk Input/Output and General Functions.** *In the first paragraph of the description of* `getline` **[***var***]** `<` *expression, change the last sentence (the one beginning with* "If var is missing . . . "*) to:*

If *var* is missing, `$0` and `NF` shall be set; otherwise, *var* shall be set and, if appropriate, it shall be considered a *numeric string* (see 4.1.7.2).

⇒ **4.1.7.6.2.4 awk User-Defined Functions.** *Change the first paragraph to:* B

The `awk` language provides for user-defined functions. Such functions can be defined as B

`function` *name*(**[***parameter*,...**]**) { *statements* } C

⇒ **4.1.7.6.2.4 awk User-Defined Functions.** *Change the third paragraph (the one beginning* "Function arguments ... "*) to:* B

Function parameters, if present, can be either scalars or arrays; the behavior is undefined if an array name is passed as a parameter that the function uses as a scalar, or if a scalar expression is passed as a parameter that the function uses as an array. Function parameters shall be passed by value if scalar and by reference if array name. B

⇒ **4.1.7.6.2.4 awk User-Defined Functions.** *In the fourth paragraph, change the third sentence (the one beginning* "If fewer arguments are supplied ... "*) to:* B

If fewer arguments are supplied in a function call than are in the function definition, the extra parameters that are used in the function body as scalars shall evaluate to the *uninitialized value* until they are otherwise initialized, and the extra parameters that are used in the function body as arrays shall be treated as uninitialized arrays where each element evaluates to the *uninitialized value* until otherwise initialized. B

⇒ **4.1.7.8 awk Lexical Conventions.** *In item (6), change the fifth sentence from* "An ERE constant shall be terminated by the first unescaped occurrence of the slash character after the one that begins the string constant." *to:* B

An ERE constant shall be terminated by the first unescaped occurrence of the slash character after the one that begins the ERE constant. B

Editor's Note: The following rationale will be added to E.4.1, but is kept here with `awk` *for this draft:* B

awk Rationale. *(This subclause is not a part of P1003.2b)* B

In `sub` and `gsub`, if *repl* is a string literal (the lexical token `STRING`, see 4.1.7.8), then two consecutive backslash characters should be used in the string to ensure a single backslash will precede the ampersand when the resultant string is passed to the function. [For example, to specify one literal ampersand in the replacement string, use `gsub(ERE, "\\&")`.] B

Historically the only special character in the *repl* argument of `sub` and `gsub` string functions was the ampersand (&) character and preceding it with the backslash character was used to turn off its special meaning. B

The description in the 1992 standard introduced behavior such that the backslash character was another special character and it was unspecified whether there B were any other special characters. This description introduced several portability problems, some of which are described below, and so it has been replaced with the more historical description. Some of the problems include:

— Historically, to create the replacement string, a script could use `gsub(ERE, "\\&")`, but with the 1992 wording, it was necessary to use `gsub(ERE, "\\\\&")`. Backslash characters are doubled here because all string literals are subject to lexical analysis, which would reduce each pair of backslash characters to a single backslash before being passed to `gsub`.

— Since it was unspecified what the special characters were, for portable scripts to guarantee that characters are printed literally, each character had to be preceded with a backslash. (For examples, a portable script had to use `gsub(ERE, "\\h\\i")` to produce a replacement string of `hi`.)

The description for comparisons in the 1992 version of 4.1.7.2 did not properly describe historical practice because of the way numeric strings are compared as numbers. The current rules cause the following code:

```
if (0 == "000")
        print "strange, but true"
else
        print "not true"
```

to do a numeric comparison, causing the `if` to succeed. It should be intuitively obvious that this is incorrect behavior, and indeed, no historical implementation of `awk` actually behaves this way.

To fix this problem, the definition of *numeric string* was enhanced to include only those values obtained from specific circumstances (mostly external sources) where it is not possible to determine unambiguously whether the value is intended to be a string or a numeric.

Variables that are assigned to a *numeric string* shall also be treated as a *numeric string*. (For example, the notion of a *numeric string* can be propagated across assignments.) In comparisons, all variables having the *uninitialized value* are to be treated as a numeric operand evaluating to the numeric value zero.

Uninitialized variables includes all types of variables including scalars, array elements, and fields. The definition of an *uninitialized value* in 4.1.7.3 is necessary to describe the value placed on uninitialized variables and on fields that are valid (e.g., `< $NF`) but have no characters in them and to describe how these variables are to be used in comparisons. A valid field, such as `$1`, that has no characters in it can be obtained by from an input line of “`\t\t`” when `FS="\t"`. Historically, the comparison (`$1 < 10`) was done numerically after evaluating `$1` to the value zero.

The phrase “... also shall have the numeric value of the *numeric string*” was removed from several sections of the 1992 version because they specify an unnecessary implementation detail. It is not necessary for this standard to specify that these objects be assigned two different values. It is only necessary to specify that these objects may evaluate to two different values depending on context.

The description of numeric string processing is based on the behavior of the *atof*() function in the C Standard {7}. While it is not a requirement for an implementation to use this function, many historical implementations of awk do. In the C Standard {7}, floating point constants use a period as a decimal point character for the language itself, independent of the current locale, but the *atof*() function and the associated *strtod*() function use the decimal point character of the current locale when converting strings to numeric values. Similarly in awk, floating point constants in an awk script use a period independent of the locale, but input strings use the decimal point character of the locale.

4.3 bc – Arbitrary-precision arithmetic language

⇒ **4.3.7.1 bc Operations.** *Change the paragraph with the numbered list (the one beginning* "For all values of obase . . . "*) to:*

For all values of obase specified by this standard, bc shall output numeric values by performing each of the following steps in order:

(1) If the value is less than zero, bc shall write a hyphen (–) character.

(2) Depending on the numeric value, bc shall write one of the following:

— If the absolute value of the numeric value is greater than or equal to one, bc shall write the integer portion of the value as a series of digits appropriate to obase (as described below), most signifigant digit first. It shall write the most significant nonzero digit next, followed by each successively less significant digit.

— If the absolute value of the numeric value is less than one but greater than zero and the scale of the numeric value is greater than zero, it is unspecified whether bc writes the character 0.

— If the numeric value is zero, bc shall write the character 0.

(3) If the scale of the value is greater than zero and the numeric value is not zero, bc shall write a period character, followed by a series of digits appropriate to obase (as described below) representing the most significant portion of the fractional part of the value. If s represents the scale of the value being written, the number of digits written shall be s if obase is 10, less than or equal to s if obase is greater than 10, or greater than or equal to s if obase is less than 10. For obase values other than 10, this should be the number of digits needed to represent a precision of 10^s.

⇒ **4.3.7.1 bc Operations.** *Change the paragraph describing the* `return` *statements (the fourth last paragraph in the subclause) to:* B

The `return` statements [`return` and `return`(*expression*)] shall cause termination of a function, popping of its auto variables, and specification of the result of the function. The first form shall be equivalent to `return(0)`. The value and scale of the result returned by the function shall be the value and scale of the expression returned. B

⇒ **4.3.7.1 bc Operations.** *Change the last paragraph in the subclause (the one beginning* "The scale of an invocation . . . "*) to:* B

The scale of the result returned by these functions shall be the value of the `scale` register at the time the function is invoked. The value of the `scale` register after these functions have completed their execution shall be the same value it had upon invocation. The behavior is undefined if any of these functions is invoked with an argument outside the domain of the mathematical function. B

Rationale: The preceding three changes are the result of interpretation request PASC 1003.2-92 #96 submitted for IEEE Std 1003.2-1992. B

⇒ **4.3.7.2 bc Grammar.** *Change the definition of* `argument_list` *to:* B

```
argument_list : expression                              B
              | argument_list ',' expression            B
              | LETTER '[' ']' ',' argument_list        C
              ;                                         B
```

Rationale: The preceding change is the result of interpretation request PASC 1003.2-92 #101 submitted for IEEE Std 1003.2-1992. B

Editor's Note: The following rationale will be added to E.4.3, but is kept here with `bc` *for this draft:* B

bc Rationale. *(This subclause is not a part of P1003.2b)* B

Historical implementations of `bc` did not allow array parameters to be passed as the last parameter to a function. New implementations are encouraged to remove this restriction even though it is not required by the grammar. B

4.5 cd – Change working directory

Editor's Note: Virtually all of this clause has been changed in Draft 11. To avoid clutter, it is not further diffmarked.

⇒ **4.5.1 cd Synopsis.** *Modify the Synopsis to be:*

```
cd  [–L] [–P] [directory]
```

⇒ **4.5.2 cd Description.** *Change the entire subclause to:*

The cd utility shall change the working directory of the current shell execution environment (see 3.12) by executing the following steps in sequence. (In the following steps, the symbol *curpath* represents an intermediate value used to simplify the description of the algorithm used by cd. There is no requirement that *curpath* be made visible to the application.)

(1) If no *directory* operand is given and the **HOME** environment variable is empty or undefined, the default behavior is implementation defined and no further steps shall be taken.

(2) If no *directory* operand is given and the **HOME** environment variable is set to a nonempty value, the cd utility shall behave as if the directory named in the **HOME** environment variable was specified as the *directory* operand.

(3) If the operand begins with a slash, *curpath* shall be set to the operand. If the first component is dot or dot-dot, *curpath* shall be set to the **PWD** environment variable with a slash character and the operand appended. Otherwise, *curpath* shall be set as affected by the **CDPATH** environment variable. The cd utility shall construct a directory name to store in *curpath* by appending a slash and the operand to each directory named in the **CDPATH** variable, in the order listed. The resulting value of *curpath* shall be the first of these strings that is a directory. If none of the resulting strings represented a directory, *curpath* shall be set to the equivalent of the **PWD** environment variable with a slash character and the operand appended.

(4) If *curpath* is being handled dot-dot physically, the cd utility shall perform actions equivalent to the POSIX.1 {8} *chdir*() function, called with *curpath* as the *path* argument. If these actions succeed, the **PWD** environment variable shall be set to an absolute pathname for the current working directory and shall not contain filename components that, in the context of pathname resolution, refer to a file of type symbolic link. If there is insufficient permission on the new directory, or on any parent of that directory, to determine the current working directory, it is unspecified to what the PWD**environment**variable shall be set. If the actions equivalent to *chdir*() fail for any reason, the cd utility shall display an appropriate error message and not alter the **PWD** environment variable. In either case, no further steps shall be taken.

(5) The *curpath* value shall then be converted to canonical form as follows, considering each component from beginning to end, in sequence:

(a) Dot components and any slashes that separate them from the next component shall be deleted.

(b) For each dot-dot component, if there is a preceding component and it is neither root nor dot-dot, the preceding component, all slashes separating the preceding component from dot-dot, dot-dot, and all slashes separating dot-dot from the following component shall be deleted.

(c) An implementation may further simplify *curpath* by removing any trailing slash characters that are not also leading slashes, replacing multiple non-leading consecutive slashes with a single slash, and replacing three or or more leading slashes with a single slash.

If as a result of this canonicalization the *curpath* variable is null, no further steps shall be taken.

(6) The cd utility shall then perform actions equivalent to the POSIX.1 {8} *chdir*() function called with *curpath* as the *path* argument. If these actions failed for any reason, the cd utility shall display an appropriate error message and no further steps shall be taken. The **PWD** environment variable shall be set to *curpath*.

⇒ 4.5.3 cd Options. *Change the entire subclause to:*

The cd utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

−L	Handle the operand dot-dot logically; see 4.5.2.
−P	Handle the operand dot-dot physically, resolving any filename components that refer to symbolic links.

If both −L and −P options are specified, the last of these options shall be used and all others ignored. If neither −L nor −P is specified, the operand shall be handled dot-dot logically; see 4.5.2.

⇒ 4.5.4 cd Operands. *Change the directory entry with:*

directory	An absolute or relative pathname of the directory that shall become the new working directory. The interpretation of a relative pathname by cd depends on the −L option and the **CDPATH** and **PWD** environment variables. If *directory* is −, the results are implementation defined. If *directory* is an empty string, the results are unspecified.

⇒ **4.5.5.3 cd Environment Variables.** *Change the CDPATH entry to:*

CDPATH	This variable shall consist of a colon-separated list of pathnames that refer to directories. The `cd` utility shall use this list in its attempt to change the directory, as described in 4.5.2. An empty string in place of a directory pathname represents the current directory. If **CDPATH** is not set, it shall be treated as if it were an empty string.

⇒ **4.5.5.3 cd Environment Variables.** *Add the following entry in the proper sorted order:*

PWD	This variable shall be set as specified in 4.5.2. If an application sets or unsets the value of **PWD**, the behavior of `cd` is unspecified. C C C

Editor's Note: The following rationale will be added to E.4.5, but is kept here with `cd` *for this draft:*

cd Rationale. *(This subclause is not a part of P1003.2b)*

Some historical shells, such as the KornShell, took special actions when the directory name contained a dot-dot component, selecting the logical parent of the directory, rather than the actual parent directory; i.e., it moved up one level toward the / in the pathname, remembering what the user typed, rather than performing the equivalent of the POSIX.1 {8} call

```
chdir("..");
```

In such a shell, the following commands would not necessarily produce equivalent output for all directories:

```
cd .. && ls              ls ..
```

This behavior is not permitted by default because it is not consistent with the definition of dot-dot in most historical practice; i.e., while this behavior has been optionally available in the KornShell, other shells have historically not supported this functionality.

The logical pathname is stored in the **PWD** environment variable when the `cd` utility completes and this value is used to construct the next directory name if `cd` is invoked with the –L option.

4.6 chgrp – Change file group ownership

⇒ **4.6.1 chgrp Synopsis.** *Modify the Synopsis to be:*

chgrp **[** –R **[** –H **|** –L **]] [**–h**]** *group file . . .*

⇒ **4.6.3 chgrp Options.** *Change the entire subclause to:*

The chgrp utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

–h	If the system supports group IDs for symbolic links, for each *file* operand that names a file of type symbolic link, chgrp shall attempt to set the group ID of the symbolic link instead of the file referenced by the symbolic link. If the system does not support group IDs for symbolic links, for each *file* operand that names a file of type symbolic link, chgrp shall do nothing more with the current file and shall go on to any remaining files.
–H	If the –R option is specified and a symbolic link referencing a file of type directory is specified on the command line, chgrp shall change the group of the directory referenced by the symbolic link and all files in the file hierarchy below it.
–L	If the –R option is specified and a symbolic link referencing a file of type directory is specified on the command line or encountered during the traversal of a file hierarchy, chgrp shall change the group of the directory referenced by the symbolic link and all files in the file hierarchy below it.
–R	Recursively change file group IDs. For each *file* operand that names a directory, chgrp shall change the group of the directory and all files in the file hierarchy below it. When a symbolic link is specified on the command line or encountered during the traversal of a file hierarchy, chgrp shall change the group ID of the symbolic link if the system supports this operation. Unless the –H or –L options are specified, the chgrp utility shall not follow the symbolic link to any other part of the file hierarchy.

Specifying more than one of the mutually exclusive options –H and –L shall not be considered an error. The last option specified shall determine the behavior of the utility.

4.7 chmod – Change file modes

⇒ **4.7.1 chmod Synopsis.** *Modify the Synopsis to be:*

```
chmod [ –R [ –H | –L ] ] [–h] mode file . . .
```

⇒ **4.7.3 chmod Options.** *Change the entire subclause to:*

The chmod utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

–h If the system supports permissions for symbolic links, for each *file* operand that names a file of type symbolic link, chmod shall attempt to set the permissions of the symbolic link instead of the file referenced by the symbolic link. If the system does not support permissions for symbolic links, for each *file* operand that names a file of type symbolic link, chmod shall do nothing more with the current file and shall go on to any remaining files.

–H If the –R option is specified and a symbolic link referencing a file of type directory is specified on the command line, chmod shall change the file mode bits of the directory referenced by the symbolic link and all files in the file hierarchy below it.

–L If the –R option is specified and a symbolic link referencing a file of type directory is specified on the command line or encountered during the traversal of a file hierarchy, chmod shall change the file mode bits of the directory referenced by the symbolic link and all files in the file hierarchy below it.

–R Recursively change file mode bits. For each *file* operand that names a directory, chmod shall change the file mode bits of the directory and all files in the file hierarchy below it. When a symbolic link is specified on the command line or encountered during the traversal of a file hierarchy, chmod shall change the file mode bits of the symbolic link if the system supports this operation. Unless the –H or –L options are specified, the chmod utility shall not follow the symbolic link to any other part of the file hierarchy.

Specifying more than one of the mutually exclusive options –H and –L shall not be considered an error. The last option specified shall determine the behavior of the utility.

4.8 `chown` – Change file ownership

⇒ **4.8.1 `chown` Synopsis.** *Modify the Synopsis to be:*

`chown` **[** `–R` **[** `–H` **|** `–L` **]] [**`–h`**]** *owner***[**`:`*group***]** *file* . . .

⇒ **4.8.3 `chown` Options.** *Change the entire subclause to:*

The `chown` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

`–h` If the system supports user IDs for symbolic links, for each *file* operand that names a file of type symbolic link, `chown` shall attempt to set the user ID of the symbolic link. If the system supports group IDs for symbolic links, and a group ID was specified, for each *file* operand that names a file of type symbolic link, `chown` shall attempt to set the group ID of the symbolic link. By default, `chown` shall not attempt to set the user ID or group ID of the file referenced by the symbolic link. If the system does not support user or group IDs for symbolic links, for each *file* operand that names a file of type symbolic link, `chown` shall do nothing more with the current file and shall go on to any remaining files.

`–H` If the `–R` option is specified and a symbolic link referencing a file of type directory is specified on the command line, `chown` shall change the user ID (and group ID, if specified) of the directory referenced by the symbolic link and all files in the file hierarchy below it.

`–L` If the `–R` option is specified and a symbolic link referencing a file of type directory is specified on the command line or encountered during the traversal of a file hierarchy, `chown` shall change the user ID (and group ID, if specified) of the directory referenced by the symbolic link and all files in the file hierarchy below it.

`–R` Recursively change file user and group IDs. For each *file* operand that names a directory, `chown` shall change the user ID (and group ID, if specified) of the directory and all files in the file hierarchy below it. When a symbolic link is specified on the command line or encountered during the traversal of a file hierarchy, `chown` shall change the user ID (and group ID, if specified) of the symbolic link if the system supports this operation. Unless the `–H` or `–L` options are specified, the `chown` utility shall not follow the symbolic link to any other part of the file hierarchy.

Specifying more than one of the mutually exclusive options −H and −L shall not be considered an error. The last option specified shall determine the behavior of the utility.

4.13 cp – Copy files

⇒ **4.13.1 cp Synopsis.** *Modify the Synopsis to be:*

```
cp [−fip] source_file target_file
cp [−fip] source_file ... target
cp −R [ −H | −L ] [−fip] source_file ... target
cp −r [ −H | −L ] [−fip] source_file ... target
```

⇒ **4.13.2 cp Description.** *Change the second sentence of the first paragraph to:*

The cp utility shall copy the contents of *source_file* (or, if *source_file* is a file of type symbolic link, the contents of the file referenced by *source_file*) to the destination path named by *target_file*.

⇒ **4.13.2 cp Description.** *Change the last sentence of the second paragraph to:*

The cp utility shall copy the contents of each *source_file* (or, if *source_file* is a file of type symbolic link, the contents of the file referenced by *source_file*) to the destination path named by the concatenation of *target*, a slash character, and the last component of *source_file*.

⇒ **4.13.2 cp Description.** *Change the seventh paragraph to:*

In the following description, the term *dest_file* refers to the file named by the destination path. The term *source_file* refers to the file that is being copied, whether specified as an operand or a file in a file hierarchy rooted in a *source_file* operand. If *source_file* is a file of type symbolic link:

— If neither the −R nor −r options were specified, cp shall take actions based on the type and contents of the file referenced by the link, and not by the link itself.

— If the −R option was specified:

  — If neither the −H nor −L options were specified, cp shall take actions based on the file being of type symbolic link.

  — If the −H option was specified, cp shall take actions based on the type and contents of the file referenced by any symbolic link specified as a *source_file* operand.

— If the –L option was specified, cp shall take actions based on the type and contents of the file referenced by any symbolic link specified as a *source_file* operand or any symbolic links encountered during traversal of a file hierarchy.

— If the –r option was specified, the behavior is implementation defined.

⇒ **4.13.2 cp Description.** *In item (4b), add a subitem [3] at the end:*

[3] If *source_file* is a file of type symbolic link, the pathname contained in *dest_file* shall be the same as the pathname contained in *source_file*.

If this fails for any reason, cp shall write a diagnostic message to standard error, do nothing more with *source_file*, and go on to any remaining files.

⇒ **4.13.3 cp Options.** *Add the following options in the proper sorted order:*

–H	Take actions based on the type and contents of the file referenced by any symbolic link specified as a *source_file* operand.
–L	Take actions based on the type and contents of the file referenced by any symbolic link specified as a *source_file* operand or any symbolic links encountered during traversal of a file hierarchy.

⇒ **4.13.3 cp Options.** *Add the following paragraph to the end of the subclause:*

Specifying more than one of the mutually exclusive options –H and –L shall not be considered an error. The last option specified shall determine the behavior of the utility.

4.14 `cut` – Cut out selected fields of each line of a file

⇒ **4.14.3 `cut` Options.** *Change the last sentence of the second paragraph from* "The elements in list can be . . . in any order." *to:*

The elements in list can be repeated, can overlap, and can be specified in any order, but the bytes, characters, or fields selected shall be written in the order of the input data. If an element appears in the selection list more than once, it shall be written exactly once.

Rationale: This change is in response to P1003.2-N149. It represents historical practice on all known systems. The original standard was ambiguous on the nature of the output. Add the following example to E.4.14:

The *list* option-arguments are historically used to select the portions of the line to be written, but do not affect the order of the data. For example,

```
echo abcdefghi | cut -c6,2,4-7,1
```

yields `abdefg`.

A proposal to enhance `cut` with the following option:

`–o`	Preserve the selected field order. When this option is specified, each byte, character, or field (or ranges of such) shall be written in the order specified by the *list* option-argument, even if this requires multiple outputs of the same bytes, characters, or fields.

was rejected because this type of enhancement is outside the scope of the P1003.2b amendment.

4.15 date – Write the date and time

⇒ **4.15.4.2 date Modified Field Descriptors.** *Add the following list item following* %Ex*:*

%EX	Alternate time representation of the locale.

Rationale: This change was to correct an oversight in ISO/IEC 9945-2: 1993, pointed out by Japan. It is identical to an extension in XPG4 {B49}.

4.16 dd – Convert and copy a file

⇒ **4.16.2 dd Description.** *Change processing order step (4) to:*

(4) If the swab conversion is specified, each pair of input data bytes shall be swapped. If there are an odd number of bytes in the input block, the last byte in the input record shall not be swapped.

Rationale: This change is required to match historical practice and is the result of interpretation requests PASC 1003.2-92 #03 and PASC 1003.2-92 #04 submitted for IEEE Std 1003.2-1992.

⇒ **4.16.5.4 dd Asynchronous Events.** *Change the entire subclause to:*

For SIGINT, the dd utility shall interrupt its current processing, write status information to standard error, and exit as though terminated by SIGINT. It shall take the standard action for all other signals; see 2.11.5.4.

Rationale: This change is required to match historical practice and is the result of interpretation request PASC 1003.2-92 #06 submitted for IEEE Std 1003.2-1992.

4.17 **diff** – Compare two files B

Editor's Note: This clause is new in Draft 11. To avoid clutter, it is not further diffmarked. B

⇒ **4.17.3 diff Options.** *Change the description of* `–b` *to:*

`–b`	Cause any amount of white space at the end of a line to be treated as a single `<newline>` (i.e., the white-space characters preceding the `<newline>` are ignored) and other strings of white-space characters, not including `<newline>`s, to compare equally. The `–b` option shall not affect the comparison of files of type symbolic link.

⇒ **4.17.3 diff Options.** *Change the description of* `–r` *to:*

`–r`	Apply `diff` recursively to files and directories of the same name when *file1* and *file2* are both directories. If a symbolic link is encountered during the traversal of the file hierarchy, the `diff` utility shall take actions based on the file being of type symbolic link, rather than based on the type of the file referenced by the symbolic link.

⇒ **4.17.4 diff Operands.** *Change the second paragraph (the one beginning* "If both file1 and file2 . . . *") to:*

If *file1* or *file2* is a symbolic link, the `diff` utility shall take actions based on the type and contents of the file referenced by the symbolic link; e.g., if *file1* is a symbolic link that references a file of type directory, `diff` shall behave as if it were a file of type directory.

If both *file1* and *file2* are directories, `diff` shall not compare block special files, character special files, or FIFO special files to any files and shall not compare files of different types. The system documentation shall specify the behavior of `diff` on implementation-specific file types not specified by POSIX.1 {8} when found in directories. Further details are as specified in 4.17.6.1.1.

⇒ **4.17.6.1.1 diff Directory Comparison Format.** *Change the fifth paragraph from:* For each file common to the two directories, if the files are to be compared and are identical, no output shall be written. If the two files differ, the following format: shall be written:

`"diff %s %s %s\n"`, *<diff_options>*, *<filename1>*, *<filename2>*

where *<diff_options>* are the options as specified on the command line. Depending on these options, one of the following output formats shall be used to write the differences.

to:

For each file common to the two directories, if the files are symbolic links and their contents differ, the following format shall be written in the POSIX Locale:

> `"Symbolic links: %s -> %s and %s -> %s\n"`, <*filename1*>, <*filename1 contents*>, <*filename2*>, <*filename2 contents*>

Otherwise, for each file common to the two directories, if the files are to be compared and are identical, no output shall be written. If the two files differ, the following format shall be written:

> `"diff %s %s %s\n"`, <*diff_options*>, <*filename1*>, <*filename2*>

where <*diff_options*> are the options as specified on the command line. Depending on these options, one of the following output formats shall be used to write the differences.

⇒ **4.17.6.1.4 `diff` –`c` or –`C` Output Format.** *(This change should be read only in conjunction with the following change.) Delete the phrase:*

and a string of 15 asterisks:

> `"***************\n"`

⇒ **4.17.6.1.4 `diff` –`c` or –`C` Output Format.** *Change the line* "First, the range of lines in file1 shall be written in the" *following format:" to:*

First, a line shall be written in the following format:

> `"***************\n"`

Next, the range of lines in *file1* shall be written in the following format:

Rationale: The two preceding changes are the result of interpretation request PASC 1003.2-92 #71 submitted for IEEE Std 1003.2-1992.

4.20 ed – Edit text

Editor's Note: All instances or RE have been changed to BRE without specific diff marks. This was an editorial error and was not intended to deviate from the 1992 text.

⇒ **4.20.5.4 ed Asynchronous Events.** *Add a new list item at the end of the list:*

SIGQUIT The ed utility shall ignore this event.

Rationale: This change is to align with historical practice and is the result of interpretation request PASC 1003.2-92 #7 submitted for IEEE Std 1003.2-1992.

⇒ **4.20.7.2 ed Addressing.** *Change the entire subclause to:*

Addressing in ed relates to the current line. Generally, the current line is the last line affected by a command. The current line number is the address of the current line. If the edit buffer is not empty, the initial value for the current line shall be the last line in the edit buffer; otherwise, zero.

Addresses shall be constructed as follows:

(1) The period character (.) shall address the current line.

(2) The dollar-sign character ($) shall address the last line of the edit buffer.

(3) The positive decimal number *n* shall address the *n*-th line of the edit buffer.

(4) The apostrophe-*x* character pair (′*x*) shall address the line marked with the mark name character *x*, which shall be a lowercase letter from the portable character set. It shall be an error if the character has not been set to mark a line, or if the line that was marked is not currently present in the edit buffer, or the mark has not been set. Lines can be marked with the k command.

(5) A BRE (see 2.8.3) enclosed by slash characters (/) shall address the first line found by searching forwards from the line following the current line toward the end of the edit buffer and stopping at the first line containing a string matching the BRE. The BRE consisting of a null BRE delimited by a pair of slash characters shall address the next line containing the last BRE encountered. In addition, the second slash can be omitted at the end of a command line. Within the BRE, a backslash-slash pair (\/) shall represent a literal slash instead of the BRE delimiter.

(6) A BRE enclosed by question-mark characters (?) shall address the first line found by searching backwards from the line preceeding the current line toward the beginning of the edit buffer and stopping at the first line containing a string matching the BRE. The BRE consisting of a null BRE delimited by a pair of question-mark characters (??) shall address the previous line containing the last BRE encountered. In addition, the second question-mark can be omitted at the end of a command line.

Within the BRE, a backslash-question-mark pair (\?) shall represent a literal question mark instead of the BRE delimiter.

(7) A plus-sign (+) or hyphen character (-) followed by a decimal number shall address the current line plus or minus the number. A plus-sign or hyphen character not followed by a decimal number shall address the current line plus or minus 1.

Addresses can be followed by zero or more address offsets, optionally `<blank>` separated. Address offsets are constructed as follows:

— A plus-sign or hyphen character followed by a decimal number shall add or subtract, respectively, the indicated number of lines to or from the address. A plus-sign or hyphen character not followed by a decimal number shall add or subtract 1 to or from the address.

— A decimal number shall add the indicated number of lines to the address.

It shall not be an error for an intermediate address value to be less than zero or greater than the last line in the edit buffer. It shall be an error for the final address value to be less than zero or greater than the last line in the edit buffer.

Commands accepts zero, one, or two addresses. If more than the required number of addresses are provided to a command that requires zero addresses, it shall be an error. Otherwise, if more than the required number of addresses are provided to a command, the addresses specified first shall be evaluated and then discarded until the maximum number of valid addresses remain, for the specified command.

Addresses shall be separated from each other by a comma (`,`) or semicolon character (`;`). In the case of a semicolon separator, the current line (`.`) shall be set to the first address, and only then will the second address be calculated. This feature can be used to determine the starting line for forwards and backwards searches [see rules (5) and (6)].

Addresses can be omitted on either side of the comma or semicolon separator, in which case the resulting address pairs shall be as follows:

Specified	**Resulting**
`,`	`1 , $`
`,` *addr*	`1 ,` *addr*
addr `,`	*addr* `,` *addr*
`;`	`. ; $`
`;` *addr*	`. ;` *addr*
addr `;`	*addr* `;` *addr*

Any `<blank>` characters included between addresses, address separators, or address offsets shall be ignored.

Rationale: This change is is the result of interpretation request PASC 1003.2-92 #XX submitted for IEEE Std 1003.2-1992. B

⇒ **4.20.7.3 ed Commands.** *Replace the sixth paragraph (the one beginning* "If an end-of-file is detected . . . "*) with:*

If a terminal disconnect is detected: B

— If the buffer is not empty and has changed since the last write, the `ed` utility shall attempt to write a copy of the buffer to a file. First, the file named `ed.hup` in the current directory shall be used; if that fails, the file named `ed.hup` in the directory named by the **HOME** environment variable shall be used. B

— The `ed` utility shall not write the file to the currently remembered pathname or return to command mode, and shall terminate with a nonzero exit status. B

If an end-of-file is detected on standard input: B

— If the `ed` utility is in input mode, `ed` shall terminate input mode and return to command mode. It is unspecified if any partially entered lines, (i.e., input text without a terminating `<newline>` character) are discarded from the input text. B

— If the `ed` utility is in command mode, it shall act as if a `q` command had been entered. B

Rationale: This change is required to match historical practice and is the result of interpretation request PASC 1003.2-92 #36 submitted for IEEE Std 1003.2-1992.

⇒ **4.20.7.3.2 ed Change Command.** *Add a new sentence at the end of the paragraph:* C

Address 0 shall be valid for this command; it shall be interpreted as if address 1 were specified. C

⇒ **4.20.7.3.7 ed Global Command.** *Change the second sentence (the one beginning with* "Then, for every such line, . . . "*) to:* B

Then, going sequentially from the beginning of the file to the end of the file, the given *command list* shall be executed for each marked line, with the current line number set to the address of that line. Any line modified by the command list shall be unmarked. B

⇒ **4.20.7.3.8 ed Interactive Global Command.** B

Rationale: The preceding two changes are the result of interpretation request PASC 1003.2-92 #119 submitted for IEEE Std 1003.2-1992. B

⇒ **4.20.7.3.11 ed Insert Command.** *Change the final sentence of the paragraph to:* C

Address 0 shall be valid for this command; it shall be interpreted as if address 1 were specified. C

⇒ **4.20.7.3.14 ed List Command.** *Replace the second sentence with:*

The characters listed in Table 2-16 (see 2.12), except for `\n`, shall be written as the corresponding escape sequences.

Rationale: The exception for `\n` was added to avoid breaking historical practice and is the result of interpretation request PASC 1003.2-92 #32 submitted for IEEE Std 1003.2-1992.

⇒ **4.20.7.3.14 ed List Command.** *In the second paragraph, change the sentence* "The end of each line shall be marked with a $." *to:* C

The end of each line shall be marked with a `$`, and `$` characters within the text shall be written with a preceding backslash. C

Editor's Note: The following rationale will be added to E.4.20, but is kept here with ed *for this draft:* B

ed Rationale. *(This subclause is not a part of P1003.2b)* B

It is difficult under some modes of some versions of historical operating system terminal drivers to distinguish between an end-of-file condition and terminal disconnect. POSIX.2 does not require implementations to distinguish between the two situations, which permits historical implementations of the ed utility on historical platforms to conform. Implementations are encouraged to distinguish between the two, if possible, and take appropriate action on terminal disconnect. B

Historically, ed accepted a zero address for the `a` and `r` commands in order to insert text at the start of the edit buffer. When the buffer was empty the command "`.=`" returned zero. This standard requires conformance to historical practice. B

For consistency with the `a` and `r` commands and better user functionality, the `i` and `c` commands must also accept an address of 0, in which case `0i` is treated as `1i` and likewise for the `c` command. B

All of the following are valid addresses: B

+++	Three lines after the current line

B

/pattern/–	One line before the next occurrence of pattern
`–2`	Two lines before the current line
`3 ---- 2`	Line one (note the intermediate negative address)
`1 2 3`	Line six

Any number of addresses can be provided to commands taking addresses; e.g., `1,2,3,4,5p` prints lines 4 and 5, because two is the greatest valid number of addresses accepted by the `print` command. This, in combination with the semicolon delimiter, permits users to create commands based on ordered patterns in the file. For example, the command `3;/foo/;+2p` will display the first line after line 3 that contains the pattern `foo`, plus the next two lines. Note that the address "`3;`" still must be evaluated before being discarded, because the search origin for the `/foo/` command depends on this.

Historically, `ed` disallowed address chains, as discussed above, consisting solely of comma or semicolon separators; e.g., "`,,,`" or "`;;;`" were considered an error. For consistency of address specification, this restriction is removed. The following table list some of the address forms now possible:

Address	**Addr1**	**Addr2**	**Status**	**Comment**
`7,`	7	7	historical	
`7,5,`	5	5	historical	
`7,5,9`	5	9	historical	
`7,9`	7	9	historical	
`7,+`	7	8	historical	
`,`	1	$	historical	
`,7`	1	7	extension	
`,,`	$	$	extension	
`,;`	$	$	extension	
`7;`	7	7	historical	
`7;5;`	5	5	historical	
`7;5;9`	5	9	historical	
`7;5,9`	5	9	historical	
`7;$;4`	$	4	historical	valid, but erroneous
`7;9`	7	9	historical	
`7;+`	7	8	historical	
`;`	.	$	historical	
`;7`	.	7	extension	
`;;`	$	$	extension	
`;,`	$	$	extension	

Historically, values could be added to addresses by including them after one or more `<blank>` characters; e.g., "`3 - 5p`" wrote the seventh line of the file, and "`/foo/ 5`" was the same as `/foo/+5`. However, only absolute values could be added; e.g., "`5 /foo/`" was an error. This standard requires conformance to historical practice.

Historically, ed accepted the ˆ character as an address, in which case it was identical to the hyphen character. This standard does not require or prohibit this behavior. B

4.22 expr – Evaluate arguments as an expression B

⇒ **4.22.6.1 expr Standard Output.** *Change the contents of this subclause to:* B

The expr utility shall write the evaluation of the expression to standard output followed by a <newline> character. B

Rationale: This change is the result of interpretation request PASC 1003.2-92 #104 submitted for IEEE Std 1003.2-1992. B

⇒ **4.22.7 expr Extended Description.** *Change the first row in Table 4-5 to:* B

Expression	Description
expr1 | *expr2*	Returns the evaluation of *expr1* if it is neither null nor zero; otherwise, the evaluation of *expr2* if it is not null; otherwise, zero.

B

Rationale: This change is the result of interpretation request PASC 1003.2-92 #104 submitted for IEEE Std 1003.2-1992. B

4.24 find – Find files

⇒ **4.24.1 find Synopsis.** *Change the Synopsis to:*

find **[** –H **|** –L **]** *path* . . . **[***operand_expression* . . . **]**

⇒ **4.24.2 find Description.** *Add at the end of the second paragraph:*

The find utility shall detect infinite loops; i.e., entering a previously visited directory that is an ancestor of the last file encountered. When it detects an infinite loop, find shall write a diagnostic message to standard error and shall either recover its position in the hierarchy or terminate. B

⇒ **4.24.3 `find` Options.** *Change the entire subclause to:*

The `find` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

`−H` Cause the file information and file type evaluated for each symbolic link encountered on the command line to be those of the file referenced by the link, and not the link itself. If the referenced file does not exist, the file information and type shall be for the link itself. File information for all symbolic links not on the command line shall be that of the link itself.

`−L` Cause the file information and file type evaluated for each symbolic link to be those of the file referenced by the link, and not the link itself. If the referenced file does not exist, the file information and type shall be for the link itself.

Specifying more than one of the mutually exclusive options `−H` and `−L` shall not be considered an error. The last option specified shall determine the behavior of the utility. C

Editor's Note: The following rationale will be added to E.4.24, but is kept here with `find` *for this draft:*

`find` Rationale. *(This subclause is not a part of P1003.2b)*

Historically, the `−L` option was implemented using the primary `−follow`. The `−H` and `−L` options were added for two reasons. First, they offer a finer granularity of control and consistency with other programs that walk file hierarchies. Second, the `−follow` primary always evaluated to true. As they were historically really global variables that took effect before the traversal began, some valid expressions had unexpected results. An example is the expression `−print −o −follow`. Because `−print` always evaluates to true, the standard order of evaluation implies that `−follow` would never be evaluated. This was never the case.

B

⇒ **4.24.4 `find` Operands.** *Replace the* `−atime`, `−ctime`, *and* `−mtime` *descriptions with:*

`−atime` *n* The primary shall evaluate as true if the file access time subtracted from the initialization time, divided by 86 400 (with any remainder discarded), is *n*. B

`−ctime` *n* The primary shall evaluate as true if the time of last change of file status information subtracted from the initialization time, divided by 86 400 (with any remainder discarded), is *n*. B

`−mtime` *n*	The primary shall evaluate as true if the file modification time subtracted from the initialization time, divided by 86 400 (with any remainder discarded), is *n*.

Rationale: This change is required to match historical practice and is the result of interpretation request PASC 1003.2-92 #58 submitted for IEEE Std 1003.2-1992.

⇒ **4.24.4 `find` Operands.** *Add the following primary in the proper sorted order:*

`−follow`	The primary always shall evaluate as true. If it occurs anywhere in *operand_expression*, it shall cause `find` to evaluate the file information and file type for all symbolic links (whether named on the command line or encountered in a file hierarchy) to be those of the file referenced by the link, and not the link itself. If the referenced file does not exist, the file information and type shall be for the link itself. By default, `find` shall not follow symbolic links. If any `−follow` primary is specified, it shall apply to the entire expression even if the `−follow` primary would not normally be evaluated.

⇒ **4.24.4 `find` Operands.** *In the* `−type` *c description, add the character* `l` *(ell) to represent a symbolic link.*

4.26 `getconf` – Get configuration values

⇒ **4.26.4 `getconf` Operands.** *Change the first paragraph of the system_var operand to:*

system_var	A name of a configuration variable or minimum value available from the *confstr*() or *sysconf*() functions in POSIX.1 {8}.

Rationale: The `getconf` changes are part of a general cleanup to remove references to the now-deleted Chapter 7. All of the applicable functions are now in POSIX.1-199x, the version created by the currently balloting P1003.1a.

⇒ **4.26.6.1 `getconf` Standard Output.** *In the first paragraph, change the phrase* "the function in 7.8.1" *to:*

the POSIX.1 {8} *confstr*() function

4.33 `ln` – Link files

⇒ **4.33.1 `ln` Synopsis.** *Modify the Synopsis to be:*

`ln` **[**`−fs`**]** *source_file target_file*

`ln` **[**`−fs`**]** *source_file* . . . *target_dir*

⇒ **4.33.2 `ln` Description.** *Change the first two paragraphs to:*

In the first synopsis form, the `ln` utility shall create a new directory entry (link), or if the `−s` option is specified, a symbolic link, for the file specified by the *source_file* operand at the *destination* path specified by the *target_file* operand. This first synopsis form shall be assumed when the final operand does not name an existing directory; if more than two operands are specified and the final operand is not an existing directory, an error shall result.

In the second synopsis form, the `ln` utility shall create a new directory entry, or if the `−s` option is specified, a symbolic link, for each file specified by a *source_file* operand at a *destination* path in the existing directory named by *target_dir*.

Editor's Note: The third paragraph of POSIX.2-1992 (If the last operand specifies an existing file of a type not specified by POSIX.1 {8}, the behavior is implementation defined.) is referring to the version of POSIX.1 {8} at the time the dot2b amendment is approved, not the 1990 version. Since dot2b and dot1a are proceeding in sync, this will be P1003.1a, which includes symlinks.

⇒ **4.33.2 `ln` Description.** *In the fourth paragraph, change* "The corresponding destination path . . . " *to:*

The corresponding *destination* path . . .

⇒ **4.33.2 `ln` Description.** *Change item (2) to:*

(2) If the `−s` option is specified, `ln` shall create a symbolic link named by the *destination* path and containing as its pathname *source_file*. The `ln` utility shall do nothing more with *source_file* and shall go on to any remaining files.

(3) If *source_file* is a symbolic link, actions shall be performed equivalent to the POSIX.1 {8} *link*() function using the object that *source_file* references

as the *path1* argument and the *destination* path as the *path2* argument. The `ln` utility shall do nothing more with *source_file* and shall go on to any remaining files.

(4) Actions shall be performed equivalent to the POSIX.1 {8} *link*() function using *source_file* as the *path1* argument and the *destination* path as the *path2* argument.

⇒ **4.33.3 `ln` Options.** *Add the following option in the proper sorted order:*

`−s`	Create symbolic links instead of hard links.

⇒ **4.33.4 `ln` Operands.** *Replace the description of source_file with:*

source_file	A pathname of a file to be linked. If the `−s` option is specified, no restrictions on the type of file or on its existence shall be made. If the `−s` option is not specified, whether a directory can be linked is implementation defined.

4.35 `localedef` – Define locale environment

Editor's Note: This clause is new in Draft 11. To avoid clutter, it is not further diffmarked.

⇒ **4.35.1 `localedef` Synopsis.** *Modify the Synopsis to be:*

`localedef` **[**`−c`**] [**`−f` *charmap***] [**`−i` *sourcefile***] [**`−u` *code_set_name***]** *name*

⇒ **4.35.3 `localedef` Options.** *Add the following option in the proper sorted order:*

`−u` *code_set_name*
: Specify the name of a code set used as the target mapping of character symbols and collating element symbols whose encoding values are defined in terms of ISO/IEC 10646 {10} position constant values.

⇒ **4.35.7 `localedef` Extended Description.** *Change this subclause from* "None." *to:*

When the –`u` option is used, the *code_set_name* option-argument shall be interpreted as an implementation-defined name of a code set to which the ISO/IEC 10646 {10} position constant values shall be converted via an implementation-defined method. Both ISO/IEC 10646 {10} position constant values and other formats (decimal, hexadecimal, or octal) shall be valid as encoding values within the charmap file. The code set represented by the implementation-defined name can be any codeset that is supported by the implementation.

When conflicts occur between the charmap specification of `<code_set_name>`, `<mb_cur_max>`, or `<mb_cur_min>` and the implementation-defined interpretation of these respective items for the codeset represented by the –`u` option-argument *code_set_name*, the result is unspecified.

When conflicts occur between the charmap encoding values specified for symbolic names of characters of the portable character set (Table 2-4) and the implementation-defined assignment of character encoding values, the result is unspecified.

If a nonprintable character in the charmap has a width specified that is not –1, `localedef` shall generate a warning. C

⇒ **4.35.9 `localedef` Consequences of Errors.** *Add a final list entry to the dashed list of conditions for warning messages:* C

— If a nonprintable character has a width specified other than –1. C

4.39 ls – List directory contents

⇒ **4.39.1 ls Synopsis.** *Modify the Synopsis to be:*

ls **[**−CFRacdilqrtu1**] [** −H **|** −L **] [** *file* ... **]**

⇒ **4.39.2 ls Description.** *Replace the entire subclause with:*

For each operand that names a file of a type other than directory or symbolic link to a directory, ls shall write the name of the file as well as any requested, associated information. For each operand that names a file of type directory, ls shall write the names of files contained within the directory as well as any requested, associated information. If one of the −d, −F, or −l options are specified, and one of the −H or −L options are not specified, for each operand that names a file of type symbolic link to a directory, ls shall write the name of the file as well as any requested, associated information. If none of the −d, −F, or −l options are specified, or the −H or −L options are specified, for each operand that names a file of type symbolic link to a directory, ls shall write the names of files contained within the directory as well as any requested, associated information.

If no operands are specified, ls shall write the contents of the current directory. If more than one operand is specified, ls shall write nondirectory operands first; it shall sort directory and nondirectory operands separately according to the collating sequence in the current locale.

The ls utility shall detect infinite loops; i.e., entering a previously visited directory that is an ancestor of the last file encountered. When it detects an infinite loop, ls shall write a diagnostic message to standard error and shall either recover its position in the hierarchy or terminate.

⇒ **4.39.3 ls Options.** *Replace the descriptions of the* −d, −F, *and* −l *options with the following:*

−d	Do not follow symbolic links named as operands unless the −H or −L options are specified. Do not treat directories differently than other types of files. The use of −d with −R produces unspecified results.
−F	Do not follow symbolic links named as operands unless the −H or −L options are specified. Write a slash (/) immediately after each pathname that is a directory, an asterisk (*) after each that is executable, a vertical bar (|) after each that is a FIFO, and an at-sign (@) after each that is a symbolic link.

`–l` (The letter ell.) Do not follow symbolic links named as operands unless the `–H` or `–L` options are specified. Write out in long format (see 4.39.6.1). When `–l` (ell) is specified, `–1` (one) shall be assumed. B

⇒ **4.39.3 `ls` Options.** *Add the following options in the proper sorted order:*

`–H` If a symbolic link referencing a file of type directory is specified on the command line, `ls` shall evaluate the file information and file type to be those of the file referenced by the link, and not the link itself; however, `ls` shall write the name of the link itself and not the file referenced by the link.

`–L` Evaluate the file information and file type for all symbolic links (whether named on the command line or encountered in a file hierarchy) to be those of the file referenced by the link, and not the link itself; however, `ls` shall write the name of the link itself and not the file referenced by the link. When `–L` is used with `–l`, write the contents of symbolic links in the long format (see 4.39.6.1).

⇒ **4.39.3 `ls` Options.** *Change the final paragraph in this subclause to:* B

Specifying more than one of the options in the following mutually exclusive pairs shall not be considered an error: `–C` and `–l` (ell), `–C` and `–1` (one), `–H` and `–L`, `–c` and `–u`. The last option specified in each pair shall determine the output format. B

⇒ **4.39.6.1 `ls` Standard Output.** *Replace the six-line description of* `–l` *(beginning with* "If the –l option is specified, . . . "*) with:*

If the `–l` option is specified without `–L`, the following information shall be written:

> `"%s %u %s %s %u %s %s\n"`, <*file mode*>, <*number of links*>, <*owner name*>, <*group name*>, <*number of bytes in the file*>, <*date and time*>, <*pathname*>

If the file is a symbolic link, this information shall be about the link itself and the <*pathname*> field shall be of the form:

> `"%s -> %s"`, <*pathname of link*>, <*contents of link*>

If both `–l` and `–L` are specified, the following information shall be written:

> `"%s %u %s %s %u %s %s\n"`, <*file mode*>, <*number of links*>, <*owner name*>, <*group name*>, <*number of bytes in the file*>, <*date and time*>, <*pathname of link*>

where all fields except <*pathname of link*> shall be for the file resolved from the symbolic link.

In both of the preceding `-l` forms, if <*owner name*> or <*group name*> cannot be determined, they shall be replaced with their associated numeric values using the format `"%u"`.

⇒ **4.39.6.1 `ls` Standard Output.** *Add the following to the list of <entry type> characters:*

`l` (ell)	Symbolic link

⇒ **4.39.8 `ls` Exit Status.** *Change the zero exit status from* "All files were written successfully." *to:*

0	Successful completion.

Rationale: This change is in response to confusion about whether `ls` was supposed to write to the files about which it was reporting. It is the result of interpretation request PASC 1003.2-92 #39 submitted for IEEE Std 1003.2-1992.

4.40 `mailx` – Process Messages

Rationale: The majority of changes to the `mailx` utility arise from interpretation requests submitted for IEEE Std 1003.2-1992. In particular, the changes here address interpretation requests PASC 1003.2-92 #10, 11, 103, 106, 108, 114, 115, 122 and 129. Where a change is particularly relevant to an interpretation request, it is highlighted by additional in-line rationale. Where there is no additional rationale given, the change has been caused by problems highlighted by the resolution of these interpretations. C

⇒ **4.40.5.3 `mailx` Environment Variables.** *In the description of the* **LISTER** *variable, delete the sentence* "The default value shall be unset."

Rationale: This change satisfies the following corrigendum request from ISO/IEC 9945-2:1993 Annex H.2:

(6) In the 4.40.5.3 description of the `mailx` **LISTER** variable, the sentence "The default value shall be unset" may be redundant.

⇒ **4.40.7 `mailx` Extended Description.** *Change the second paragraph (the one beginning with* "When mailx is invoked . . . "*) to:*

When `mailx` is invoked using one of the Receive Mode synopsis forms, it shall write a page of header-summary lines (if `–N` was not specified and there are messages, see below), followed by a prompt indicating `mailx` can accept regular commands (see 4.40.7.2); this is termed *command mode*. The page of header-summary lines shall contain the first new message if there are new messages, or the first unread message if there are unread messages, or the first message. When `mailx` is invoked using the Send Mode synopsis and standard input is a terminal, if no subject is specified on the command line and the `asksub` variable is set, a prompt for the subject shall be written. At this point `mailx` is in *input mode*. This input mode is also entered when using one of the Receive Mode synopsis forms and a reply or new message is composed using the `reply`, `Reply`, or `mail` commands and standard input is a terminal. When the message is typed and the end of message is encountered, the message shall be passed to the mail delivery software. Commands can be entered by beginning a line with the escape character [by default, tilde (**~**)] followed by a single command letter and optional arguments. See 4.40.7.3 for a summary of these commands. It is unspecified what effect these commands will have if standard input is not a terminal when a message is entered using either the Send Mode synopsis, or the Read Mode commands `reply`, `Reply`, or `mail`.

Rationale: The preceding change is the result of interpretation request PASC 1003.2-92 #103, submitted for IEEE Std 1003.2-1992.

⇒ **4.40.7 `mailx` Extended Description.** *Change the fifth paragraph (the one beginning* "If no command is specified . . . "*) to:*

If no *command* is specified in command mode, `next` shall be assumed. In input mode, commands shall be recognized by the escape character, and lines not treated as commands shall be taken as input for the message.

Rationale: The preceding change is the result of interpretation requests PASC 1003.2-92 #103 and 115, submitted for IEEE Std 1003.2-1992.

⇒ **4.40.7 `mailx` Extended Description.** *In the seventh paragraph (the one beginning* "All messages have a state . . . "*), change the sentence* "All messages are in one of the following states:" *to:*

When `mailx` is invoked using one of the Receive Mode synopsis forms, the current message shall be the first new message, if there is a new message, or the first unread message if there is an unread message, or the first message if there are any messages, or unspecified if there are no messages in the mailbox. Each command that takes an optional list of messages (*msglist*) or an optional single message (*message*) on which to operate shall leave the current message set to the hignest-numbered message of the messages specified, unless the command deletes messages, in which case the current message shall be set to the first undeleted message (i.e., a message not in the deleted state) after the hignest-numbered message deleted by the command, if one exists, or the first undeleted message before the hignest-numbered message deleted by the command, if one exists, or to an unspecified value if there are no remaining undeleted messages. All messages are in one of the following states:

⇒ **4.40.7 `mailx` Extended Description.** *Change the description of the* deleted *state to:*

deleted The message has been processed by one of the following commands: `delete`, `dp`, `dt`. Messages in state *deleted* when `mailx` quits shall be deleted. Deleted messages shall be ignored until `mailx` quits or changes mailboxes or they are specified to the `undelete` command; e.g., the message specification /*string* shall only search the subject lines of messages that have not yet been deleted, unless the command operating on the list of messages is `undelete`. No deleted message or deleted message header shall be displayed by any `mailx` command other than `undelete`.

⇒ **4.40.7 `mailx` Extended Description.** *Add a description of the* saved *state:*

saved The message has been processed by one of the following commands: `save` or `write`. If the current mailbox is the system mailbox, and the internal variable `keepsave` is set, messages in the state *saved* shall be saved to the file designated by the **MBOX** variable (see 4.40.5.3). If the current mailbox is the system mailbox, messages in the state *saved* shall be deleted from the current mailbox, when the `quit` or `file` command is used to exit the current mailbox.

⇒ **4.40.7.1 `mailx` Internal Variables.** *Change the description of the* `keepsave` *variable to:*

`keepsave` Keep the messages that have been saved from the system mailbox into other files in the file designated by the variable **MBOX**, instead of deleting them. The default shall be `nokeepsave`.

⇒ **4.40.7.2.5 `mailx` Delete messages.** *Change the paragraph following the Synopsis to:* B

Mark messages for deletion from the mailbox. The deletions shall not occur until `mailx` quits (see 4.40.7.2.24) or changes mailboxes (see 4.40.7.2.10). If `autoprint` is set and there are messages remaining after the `delete` command, the current message shall be written as described for the `print` command (see 4.40.7.2.23); otherwise, the `mailx` prompt shall be written. B

Rationale: The preceding change is the result of interpretation requests PASC 1003.2-92 #129, submitted for IEEE Std 1003.2-1992. C

⇒ **4.40.7.2.7 `mailx` Delete messages and display.** *Change the paragraph following the Synopsis to:* B

Delete the specified messages as described for the `delete` command, except that the `autoprint` variable shall have no effect, and the current message shall be written only if it was set to a message after the last message deleted by the command. Otherwise, an informational message to the effect that there are no further messages in the mailbox shall be written, followed by the `mailx` prompt. B

Rationale: The preceding change is the result of interpretation requests PASC 1003.2-92 #129, submitted for IEEE Std 1003.2-1992. C

⇒ **4.40.7.2.8 `mailx` Edit messages.** *Change the paragraph following the Synopsis to:* B

Edit the given messages. Each message shall be placed in a temporary file, and the utility named by the **EDITOR** variable (see 4.40.5.3) shall be invoked to edit each file in sequence. The default editor is unspecified. C C B

Rationale: The preceding change is the result of interpretation requests PASC 1003.2-92 #108 submitted for IEEE Std 1003.2-1992. B

⇒ **4.40.7.2.11 `mailx` Display list of folders.** *Change the sentence following the synopsis to:*

Write the names of the files in the directory set by the `folder` variable (see 4.40.7.1). The command specified by the **LISTER** environment variable shall be used (see 4.40.5.3).

Rationale: This change satisfies the following corrigendum request from ISO/IEC 9945-2: 1993 Annex H.2:

(7) In 4.40.7.2.11, the `mailx folders` command does not indicate how the value of the **LISTER** variable affects this command.

⇒ **4.40.7.2.13 `mailx` Display header summary.** *Change the entire subclause to:*

Synopsis: `h[eaders]` **[***message***]**

Write the page of headers that includes the message specified. If the *message* argument is not specified, the current message shall not change. However, if the *message* argument is specified, the current message shall become the message that appears at the top of the page of headers that includes the message specified. The `screen` variable sets the number of headers per page. See also the `z` command.

⇒ **4.40.7.2.20 `mailx` Process next specified message.** *Change the sentence following the synopsis to:*

If the current message has not been written (e.g., by the `print` command) since `mailx` started or since any other message was the current message, behave as if the `print` command was entered. Otherwise, if there is a undeleted message after the current message, make it the current message and behave as if the `print` command was entered. Otherwise, an informational message to the effect that there are no further messages in the mailbox shall be written, followed by the `mailx` prompt.

⇒ **4.40.7.2.28 `mailx` Save messages.** *Change the final sentence,* "The message shall be deleted from the mailbox . . . " *to:*

The message shall be put in the state *saved*, and shall behave as specified in the description of the *saved* state when the current mailbox is exited by the `quit` or `file` command (see 4.40.7).

⇒ **4.40.7.2.36 `mailx` Undelete messages.** *Change all of the subclause following the synopsis to:*

Change the state of the specified messages from *deleted* to *read*. If `autoprint` is set, the last message of those restored shall be written. If *msglist* is not specified, the message shall be selected as follows:

— If there are any deleted messages that follow the current message, the first of these shall be chosen.

— Otherwise, the last deleted message that also precedes the current message shall be chosen.

⇒ **4.40.7.2.38 `mailx` Edit message with full-screen editor.** *Change the paragraph following the synopsis to:*

Edit the given messages with a screen editor. Each message shall be placed in a temporary file, and the utility named by the **VISUAL** variable (see 4.40.5.3) shall be invoked to edit each file in sequence. The default editor shall be `vi`. C

Rationale: The preceding change is the result of interpretation requests PASC 1003.2-92 #115 submitted for IEEE Std 1003.2-1992.

Editor's Note: The following rationale will be added to E.4.40, but is kept here with `mailx` *for this draft:*

`mailx` Rationale. *(This subclause is not a part of P1003.2b)*

The intent of the wording for the `next` command is that if any command has already displayed the current message it should display a following message, but otherwise, it should display the current message. Consider the command sequence:

```
next 3
delete 3
next
```

where the `autoprint` option was not set. The normative text specifies that the second `next` command should display a message following the third message, because even though the current message has not been displayed since it was set by the `delete` command, it has been displayed since the current message was anything other than message number 3. This does not always match historical practice in some implementations, where the command `file` *address* followed by `next` (or the default command) would skip the message for which the user had searched.

4.41 mkdir – Make directories

⇒ **4.41.2 mkdir Description.** *Change item (2) to:*

(2) The value of the bitwise inclusive OR of S_IRWXU, S_IRWXG, and S_IRWXO is used as the *mode* argument. (If the –m option is specified, the value of the *mkdir*() *mode* argument is unspecified, but the directory shall at no time have permissions less restrictive than the –m *mode* option-argument.)

⇒ **4.41.3 mkdir Options.** *Change the description of* –m *to:*

–m *mode*	The file permission bits of the directory shall be set to the specified *mode* value. The *mode* option-argument shall be the same as the *mode* operand defined for the chmod utility (see 4.7). In the *symbolic_mode* strings, the *op* characters + and - shall be interpreted relative to an assumed initial mode of a=rwx; + shall add permissions to the default mode, - shall delete permissions from the default mode.

Rationale: The preceding two changes are the result of interpretation request PASC 1003.2-92 #67 submitted for IEEE Std 1003.2-1992. Identical changes were made for mkdir and mkfifo.

4.42 mkfifo – Make Make FIFO special files

⇒ **4.42.2 mkfifo Description.** *Change item (2) to:*

(2) The value of the bitwise inclusive OR of S_IRWXU, S_IRWXG, and S_IRWXO is used as the *mode* argument. (If the –m option is specified, the value of the *mkfifo*() *mode* argument is unspecified, but the FIFO shall at no time have permissions less restrictive than the –m *mode* option-argument.)

⇒ **4.42.3 mkfifo Options.** *Change the description of* –m *to:*

–m *mode*	The file permission bits of the FIFO shall be set to the specified *mode* value. The *mode* option-argument shall be the same as the *mode* operand defined for the chmod utility (see 4.7). In the *symbolic_mode* strings, the *op* characters + and - shall be interpreted relative to an assumed initial mode of a=rwx; + shall add permissions to the default mode, - shall delete permissions from the default mode.

Rationale: The preceding two changes are the result of interpretation request PASC 1003.2-92 #67 submitted for IEEE Std 1003.2-1992. Identical changes were made for `mkdir` and `mkfifo`. C C C

4.43 `mv` – Move files

⇒ **4.43.2 `mv` Description.** *Replace the first two paragraphs of the Description with:*

In the first synopsis form, the `mv` utility shall move the file named by the *source_file* operand to the *destination* specified by the *target_file*. This first synopsis form is assumed when the final operand does not name an existing directory and is not a symbolic link referring to an existing directory.

In the second synopsis form, `mv` shall move each file named by a *source_file* operand to a *destination* file in the existing directory named by the *target_dir* operand, or referenced if *target_dir* is a symbolic link referring to an existing directory. The *destination* path for each *source_file* shall be the concatenation of the target directory, a single slash character, and the last pathname component of the *source_file*. This second form is assumed when the final operand names an existing directory.

⇒ **4.43.2 `mv` Description.** *Replace the first sentence of item (5) with:*

The file hierarchy rooted in *source_file* shall be duplicated as a file hierarchy rooted in the destination path. If *source_file* or any of the files below it in the hierarchy are symbolic links, the links themselves shall be duplicated, including their contents, rather than any files to which they refer. B

Editor's Note: The following rationale will be added to E.4.43, but is kept here with `mv` *for this draft:*

`mv` Rationale. *(This subclause is not a part of P1003.2b)*

When `mv` is dealing with a single file system and *source_file* is a symbolic link, the link itself is moved as a consequence of the dependence on the POSIX.1 {8} *rename*() functionality, per the Description. Across file systems, this has to be made explicit.

4.45 `od` – Dump files in various formats

⇒ **4.45.4 `od` Operands.** *Change the description of the* file *operand to:* B

file A pathname of a file to be read. If no *file* operands are specified, the standard input shall be used. If there are more than two operands, none of the `–A`, `–j`, `–N`, or `–t` options is specified, and either of the following are true: B

— the first character of the last operand is a plus sign (+), or B

— the first character of the second operand is numeric B

then the results are unspecified. B

⇒ **4.45.7 `od` Extended Description.** *Replace the second sentence with:*

If no output type is specified, the default output shall be as if `–t oS` had been specified.

Rationale: The changes to `od` are required to match historical practice and are the result of interpretation requests PASC 1003.2-92 #47 and #95 submitted for IEEE Std 1003.2-1992. B

⇒ **4.45.7 `od` Extended Description.** *Change the first dashed list item to:*

— The default number of bytes transformed by output type specifiers `d`, `o`, `u`, and `x` corresponds to the various C-language types, as follows: C

  — If the `c89` compiler is present on the system, these specifiers shall correspond to the sizes used by default in that compiler. C

  — Otherwise, these sizes may vary among systems that conform to this standard. For the type specifier characters `d`, `o`, `u`, and `x` the default number of bytes shall correspond to the size of the basic integral data type of the underlying implementation. For these specifier characters, systems that conform to this standard shall support values of the optional number of bytes to be converted corresponding to the number of bytes in the C-language types *char*, *short*, *int*, and *long*. These numbers can also be specified by an application as the characters `C`, `S`, `I`, and `L`, respectively. The implementation shall also support the values 1, 2, and 4, even if it provides no C-Language types of those sizes. C

  The byte order used when interpreting numeric values is implementation defined, but shall correspond to the order in which a constant of the corresponding type is stored in memory on the system. C

Editor's Note: The following rationale will be added to E.4.45, but is kept here with `od` *for this draft:*

od Rationale. *(This subclause is not a part of P1003.2b)*

The original standard specified `-t o2` as the default when no output type was given. This was changed to `-t oS` (the length of a *short*) to accommodate a supercomputer implementation that historically used 64 b as its default (and that defined *short*s as 64 b). This change should not affect portable applications. The requirement to support lengths of 1, 2, and 4 was added at the same time to address an historical implementation that had no two-byte data types in its C compiler.

4.48 pax – Portable archive interchange

Editor's Note: This note is a road map to the many changes in `pax` proposed by this draft. In Draft 11, the volume of changes became such that I chose to integrate the changes in with the original `pax` text from the 1992 standard. All of the `pax` rationale is now merged into E.4.48. In the merged normative and rationale text, only the changes from Draft 11 onwards are diff-marked. As is standard with recirculation ballots, only diff-marked text is subject to objections. B

(1) Support has been added for symbolic links in the options and interchange formats.

(2) A new format has been devised, based on extensions to `ustar`. This new format should satisfy the following requirement from ISO/IEC 9945-2: 1993 Annex H.1: (13) The `pax` utility should provide a new file interchange format, in addition to `cpio` and `ustar`, that allows extended characters in file, user, and group names. Rules should be given for the cases where an archived name cannot be represented by the local character set in the file system.

(3) The descriptions of the `ustar` and `cpio` formats have been moved from Sections 10.1.1 and 10.1.2 of POSIX.1 {8}, but have been cleaned up in three areas:

(a) Rather than referring to a generic "reading or writing utility," they refer directly to `pax`.

(b) Some instances in POSIX.1 where "byte" had not been expressed correctly as "octet" have been converted.

(c) The C-language header file orientation has been converted to a more tabular approach.

This converted text is intended to have no normative difference from that in POSIX.1 {8}.

(4) References to the "extended" `tar` and `cpio` formats derived from POSIX.1 {8} have been changed to remove the "extended" adjective because this could cause confusion with the extended `tar` header added in this revision. (All references to `tar` are actually to `ustar`).

(5) In Draft 11, the `−o invalid=` option was added to address Canadian National Body concerns about overwriting existing files, expressed originally during international balloting on the `tar` and `cpio` formats in POSIX.1 {8}. Also, various numeric fields were added to the extended header record to allow for the cases where the original `ustar` format was too small; this was prompted by communications from a group designing support for very large files.

4.48.1 Synopsis

`pax` **[**`−cdnv`**] [** `−H` **|** `−L` **] [**`−f` *archive***] [**`−s` *replstr***]** . . . **[***pattern* . . . **]**

`pax −r` **[**`−cdiknuv`**] [** `−H` **|** `−L` **] [**`−f` *archive***] [**`−o` *options***]** . . . **[**`−p` *string***]** . . .
[`−s` *replstr***]** . . . **[***pattern* . . . **]**

`pax −w` **[**`−dituvX`**] [** `−H` **|** `−L` **] [**`−b` *blocksize***] [[**`−a`**] [**`−f` *archive***]]**
[`−o` *options***]** . . . **[**`−s` *replstr***]** . . . **[**`−x` *format***] [***file* . . . **]**

`pax −r −w` **[**`−dikltuvX`**] [** `−H` **|** `−L` **] [**`−p` *string***]** . . . **[**`−s` *replstr***]** . . . **[***file* . . . **]**
directory

4.48.2 Description

The `pax` utility shall read, write, and write lists of the members of archive files and copy directory hierarchies. A variety of archive formats shall be supported; see 4.48.7.

The action to be taken depends on the presence of the `−r` and `−w` options. The four combinations of `−r` and `−w` are referred to as the four modes of operation: *list*, *read*, *write*, and *copy* modes, corresponding respectively to the four forms shown in 4.48.1.

list In list mode (when neither the `−r` option nor the `−w` option is specified), `pax` shall write the names of the members of the archive file read from the standard input, with pathnames matching the specified patterns, to standard output. If a named file is of type directory, the file hierarchy rooted at that file shall be listed as well.

read In read mode (when `−r` is specified, but `−w` is not), `pax` shall extract the members of the archive file read from the standard input, with pathnames matching the specified patterns. If an extracted file is of type directory, the file hierarchy rooted at that file shall be extracted as well. The extracted files shall be created relative to the current file hierarchy.

The ownership, access and modification times, and file mode of the restored files are discussed under the `−p` option.

write In write mode (when −w is specified, but −r is not), pax shall write the contents of the file operands to the standard output in an archive format. If no *file* operands are specified, a list of files to copy, one per line, shall be read from the standard input. A file of type directory shall include all of the files in the file hierarchy rooted at the file.

copy In copy mode (when both −r and −w are specified), pax shall copy the *file* operands to the destination directory.

If no *file* operands are specified, a list of files to copy, one per line, shall be read from the standard input. A file of type directory shall include all of the files in the file hierarchy rooted at the file.

The effect of the copy shall be as if the copied files were written to an archive file and then subsequently extracted, except that there may be hard links between the original and the copied files. If the destination directory is a subdirectory of one of the files to be copied, the results are unspecified. If the destination directory is a file of a type not defined by POSIX.1 {8}, the results are implementation defined; otherwise, it shall be an error for the file named by the *directory* operand not to exist, not be writable by the user, or not be a file of type directory.

In read or copy modes, if intermediate directories are necessary to extract an archive member, pax shall perform actions equivalent to the POSIX.1 {8} *mkdir*() function, called with the following arguments:

— The intermediate directory used as the *path* argument.

— The value of the bitwise inclusive OR of S_IRWXU, S_IRWXG, and S_IRWXO as the *mode* argument. C

If any specified *pattern* or *file* operands are not matched by at least one file or archive member, pax shall write a diagnostic message to standard error for each one that did not match and exit with a nonzero exit status.

The archive formats described in 4.48.7 shall be automatically detected on input. B
The default output archive format shall be implementation defined.

A single archive can span multiple files. The pax utility shall determine, in an implementation-defined manner, what file to read or write as the next file.

If the selected archive format supports the specification of linked files, it shall be an error if these files cannot be linked when the archive is extracted. For archive formats that do not store file contents with each name that causes a hard link, if the file that contains the data is not extracted during this pax session, either the data shall be restored from the original file, or a diagnostic message shall be displayed with the name of a file that can be used to extract the data. B

In traversing directories, pax shall detect infinite loops; i.e., entering a previously visited directory that is an ancestor of the last file visited. When it detects an infinite loop, pax shall write a diagnostic message to standard error and shall terminate. B

4.48.3 Options

The `pax` utility shall conform to the utility argument syntax guidelines described in 2.10.2, except that the order of presentation of the `−o`, `−p`, and `−s` options is significant.

The following options shall be supported by the implementation:

`−r` Read an archive file from standard input.

`−w` Write files to the standard output in the specified archive format.

`−a` Append files to the end of the archive. It is implementation defined which devices on the system support appending. Additional file formats unspecified by this standard may impose restrictions on appending.

`−b` *blocksize*
Block the output at a positive decimal integer number of bytes per write to the archive file. Devices and archive formats may impose restrictions on blocking. Blocking shall be automatically determined on input. Conforming POSIX.2 applications shall not specify a *blocksize* value larger than 32 256 B. Default blocking when creating archives depends on the archive format. (See the `−x` option below.)

`−c` Match all archive members except those specified by the *pattern* operands.

`−d` Cause files of type directory being copied or archived or archive members of type directory being extracted or listed to match only the file or archive member itself and not the file hierarchy rooted at the file.

`−f` *archive* Specify the pathname of the input or output archive, overriding the default standard input (in list or read modes) or standard output (write mode).

`−H` If a symbolic link referencing a file of type directory is specified on the command line, `pax` shall archive the file hierarchy rooted in the file referenced by the link, using the name of the link as the root of the file hierarchy. The default behavior shall be to archive the symbolic link itself.

`−i` Interactively rename files or archive members. For each archive member matching a *pattern* operand or file matching a *file* operand, a prompt shall be written to the file `/dev/tty`. The prompt shall contain the name of the file or archive member, but the format is otherwise unspecified. A line shall then be read from `/dev/tty`. If this line is blank, the file or archive member shall be skipped. If this line consists of a single period, the file or archive member shall be processed with no modification to its name. Otherwise, its name shall be replaced with the contents of the line. The pax utility shall immediately exit with a nonzero exit status if end-of-file is encountered when reading a response or if /dev/tty cannot be opened for reading and writing.

The results of extracting a hard link to a file that has been renamed during extraction are unspecified. B

−k Prevent the overwriting of existing files.

−l (The letter ell.) In copy mode, hard links shall be made between the source and destination file hierarchies whenever possible. B

−L If a symbolic link referencing a file of type directory is specified on the command line or encountered during the traversal of a file hierarchy, pax shall archive the file hierarchy rooted in the file referenced by the link, using the name of the link as the root of the file hierarchy. The default behavior shall be to archive the symbolic link itself.

−n Select the first archive member that matches each *pattern* operand. No more than one archive member shall be matched for each pattern (although members of type directory shall still match the file hierarchy rooted at that file).

−o *options* Provide information to the implementation to modify the algorithm for extracting or writing files. The value of *options* shall consist of one or more comma-separated keywords of the form:

keyword[[:]=*value*][,*keyword*[[:]=*value*], ...]

Some keywords apply only to certain file formats, as indicated with each description. Use of keywords that are inapplicable to the file format being processed produces undefined results. B

Keywords in the options argument shall be a string that would be a valid portable filename as described in portable filename character set (see 2.2.2.131). C

NOTE: Keywords are not expected to be filenames, merely to follow the same character composition rules as portable filenames. C

Keywords can be preceded with white space. The *value* field shall consist of zero or more characters; within *value*, the application shall precede any literal comma with a backslash, which shall be ignored, but preserves the comma as part of *value*. A comma as the final character, or a comma followed solely by white space as the final characters, in *options* shall be ignored. Multiple −o options can be specified; if keywords given to these multiple −o options conflict, the keywords and values appearing later in command-line sequence shall take precedence and the earlier shall be silently ignored. The following keyword values of *options* shall be supported for the file formats as indicated:

`delete=`*pattern*

(Applicable only to the −x `pax` format.) When used in write or copy mode, `pax` shall omit from extended header records that it produces any keywords matching the string *pattern*. When used in read or list mode, `pax` shall ignore any keywords matching the string *pattern* in the extended header records. In both cases, matching shall be performed using the pattern matching notation described in 3.13.1 and 3.13.2. For example,

```
−o delete=security.∗
```

would suppress security-related information. See 4.48.7.1.2 for extended header record keyword usage.

`exthdr.name=`*string*

(Applicable only to the −x `pax` format.) This keyword allows user control over the name that is written into the `ustar` header blocks for the extended header records produced under the circumstances described in 4.48.7.1.1. The name shall be the contents of *string*, after the following character substitutions have been made: B

***string* Includes**	**Replaced By**
`%d`	The directory name of the file, equivalent to the result of the `dirname` utility on the translated pathname.
`%f`	The filename of the file, equivalent to the result of the `basename` utility on the translated pathname.
`%%`	A % character.

Any other % characters in *string* produce undefined results.

If no −o `exthdr.name=`*string* is specified, `pax` shall use the following default value:

```
%d/PaxHeaders/%f
```

`globexthdr.name=`*string*

(Applicable only to the −x `pax` format.) When used in write or copy mode with the appropriate options, `pax` creates global extended header records with `ustar` header blocks that will be treated as regular files by previous versions of `pax`. This keyword allows user control over the name that is written into the `ustar` header blocks for global extended header records. The name shall be the contents of *string*, after the following character substitutions have been made:

string Includes	Replaced By
`%n`	An integer that represents the sequence number of the global extended header record in the archive, starting at 1.
`%%`	A % character.

Any other % characters in *string* produce undefined results.

If no `−o globexthdr.name=`*string* is specified, `pax` shall use the following default value:

```
$TMPDIR/GlobalHead.%n
```

where `$TMPDIR` represents the value of the **TMPDIR** environment variable. If **TMPDIR** is not set, `pax` shall use `/tmp`.

`invalid=`*action*

(Applicable only to the `−x pax` format.) This keyword allows user control over the action `pax` takes upon encountering values in an extended header record that, in read or copy mode, are invalid in the destination hierarchy or, in list mode, cannot be written in the codeset and current locale of the implementation. The following are invalid values that shall be recognized by `pax`:

— In read or copy mode, a file name or link name that contains character encodings invalid in the destination hierarchy. (For example, the name may contain embedded NULs.)

— In read or copy mode, a file name or link name that is longer than the maximum allowed in the destination hierarchy (for either a pathname component or the entire pathname).

— In list mode, any character string value (file name, link name, user name, etc.) that cannot be written in the codeset and current locale of the implementation.

The following mutually exclusive values of the *action* argument are supported:

`bypass`

In read or copy mode, `pax` shall bypass the file, causing no change to the destination hierarchy. In list mode, `pax` shall write all requested valid values for the file, but its method for writing invalid values is unspecified.

`rename`
: In read or copy mode, `pax` shall act as if the `−i` option were in effect for each file with invalid file name or link name values, allowing the user to provide a replacement name interactively. In list mode, `pax` shall behave identically to the `bypass` action.

`UTF8`
: When used in read, copy, or list mode and a file name, link name, owner name, or any other field in an extended header record cannot be translated from the `pax` UTF8 codeset format to the codeset and current locale of the implementation, `pax` shall use the actual UTF8 encoding for the name.

`write`
: In read or copy mode, `pax` shall write the file, translating or truncating the name, regardless of whether this may overwrite an existing file with a valid name. In list mode, `pax` shall behave identically to the `bypass` action.

If no `−o invalid=` option is specified, `pax` shall act as if `−o invalid=bypass` were specified. Any overwriting of existing files that may be allowed by the `−o invalid=` actions shall be subject to permission (`−p`) and modification time (`−u`) restrictions, and shall be suppressed if the `−k` option is also specified.

`linkdata`
: (Applicable only to the `−x pax` format.) In write mode, `pax` shall write the contents of a file to the archive even when that file is merely a hard link to a file whose contents have already been written to the archive.

`listopt=`*format*
: This keyword specifies the output format of the table of contents produced when the `−v` option is specified in list mode. See 4.48.3.1. To avoid ambiguity, the `listopt=`*format* shall be the only or final *keyword=value* pair in a `−o` option-argument; all characters in the remainder of the option-argument shall be considered part of the *format* string. When multiple `−o listopt=`*format* options are specified, the *format* strings shall be considered a single, concatenated string, evaluated in command-line order.

`times`
: (Applicable only to the `−x pax` format.) When used in

write or copy mode, `pax` shall include `atime`, `ctime`, and `mtime` extended header records for each file. See 4.48.7.1.4.

In addition to these keywords, if the `−x pax` format is specified, any of the keywords and values defined in 4.48.7.1.2, including implementation extensions, can be used in `−o` option-arguments, in either of two modes:

keyword=*value*
: When used in write or copy mode, these keyword/value pairs shall be included at the beginning of the archive as *typeflag* `g` global extended header records. When used in read or list mode, these keyword/value pairs shall act as if they had been at the beginning of the archive as *typeflag* `g` global extended header records.

keyword:=*value*
: When used in write or copy mode, these keyword/value pairs shall be included as records at the beginning of a *typeflag* `x` extended header for each file. (This is equivalent to the equal-sign form except that it creates no *typeflag* `g` global extended header records.) When used in read or list mode, these keyword/value pairs shall act as if they were included as records at the end of each extended header; thus, they shall override any global or file-specific extended header record keywords of the same names. For example, in the command

```
pax -r -o "
gname:=mygroup,
" <archive
```

the group name will be forced to a new value for all files read from the archive.

The precedences of `−o` keywords over various fields in the archive are described in 4.48.7.1.3.

`−p` *string*
: Specify one or more file characteristic options (privileges). The *string* option-argument shall be a string specifying file characteristics to be retained or discarded on extraction. The string shall consist of the specification characters `a`, `e`, `m`, `o`, and `p`, and/or other implementation-defined characters. Multiple characteristics can be concatenated within the same string, and multiple `−p` options can be specified. The meanings of the specification characters are as follows:

`a`
: Do not preserve file access times.

`e`
: Preserve the user ID, group ID, file mode bits (see 2.2.2.71), access time, modification time, and any other implementation-defined file characteristics.

- `m` Do not preserve file modification times.
- `o` Preserve the user ID and group ID.
- `p` Preserve the file mode bits. Other, implementation-defined file-mode attributes may be preserved.

In the preceding list, "preserve" indicates that an attribute stored in the archive shall be given to the extracted file, subject to the permissions of the invoking process. The access and modification times of the file shall be preserved unless otherwise specified with the −`p` option or not stored in the archive. All attributes that are not preserved shall be determined as part of the normal file creation action (see 2.9.1.4). B

If neither the `e` nor the `o` specification character is specified, or the user ID and group ID are not preserved for any reason, `pax` shall not set the S_ISUID and S_ISGID bits of the file mode.

If the preservation of any of these items fails for any reason, `pax` shall write a diagnostic message to standard error. Failure to preserve these items shall affect the final exit status, but shall not cause the extracted file to be deleted.

If file-characteristic letters in any of the *string* option-arguments are duplicated or conflict with each other, the one(s) given last shall take precedence. For example, if −`p eme` is specified, file modification times shall be preserved.

−`s` *replstr* Modify file or archive member names named by *pattern* or *file* operands according to the substitution expression *replstr*, using the syntax of the `ed` utility (see 4.20). The concepts of "address" and "line" are meaningless in the context of the `pax` utility and shall not be supplied. The format shall be

−`s` /*old*/*new*/[`gp`]

where (as in `ed`) *old* is a BRE and *new* can contain an ampersand, *n* (where *n* is a digit) backreferences, or subexpression matching. The *old* string also shall be permitted to contain `<newline>` characters.

Any nonnull character can be used as a delimiter (`/` shown here). Multiple −`s` expressions can be specified; the expressions shall be applied in the order specified, terminating with the first successful substitution. The optional trailing `g` shall be as defined in the `ed` utility. The optional trailing `p` shall cause successful substitutions to be written to standard error. File or archive member names that are replaced with the empty string shall be ignored when reading and writing archives. B

`–t` Cause the access times of the archived files to be the same as they were before being read by `pax`.

`–u` Ignore files that are older (having a less recent file modification time) than a pre-existing file or archive member with the same name. In read mode, an archive member with the same name as a file in the file system shall be extracted if the archive member is newer than the file. In write mode, an archive file member with the same name as a file in the file system shall be superseded if the file is newer than the archive member. If `–a` is also specified, this is accomplished by appending to the archive; otherwise, it is unspecified if this is accomplished by actual replacement in the archive or by appending to the archive. In copy mode, the file in the destination hierarchy shall be replaced by the file in the source hierarchy or by a link to the file in the source hierarchy if the file in the source hierarchy is newer. B B

`–v` In list mode, produce a verbose table of contents (see 4.48.6.1). Otherwise, write archive member pathnames to standard error (see 4.48.6.2).

`–x` *format* Specify the output archive format. The `pax` utility shall support the following formats: B B

`cpio` The `cpio` interchange format specified in 4.48.7.3. The default *blocksize* for this format for character special archive files shall be 5120 B. Implementations shall support all *blocksize* values less than or equal to 32 256 B that are multiples of 512 B.

`pax` The interchange format specified in 4.48.7.1, based on an extension to the `ustar` format. The default *blocksize* for this format for character special archive files shall be 10 240 B. Implementations shall support all *blocksize* values less than or equal to 32 256 B that are multiples of 512 B.

`ustar` The `ustar` interchange format specified in 4.48.7.2. The default *blocksize* for this format for character special archive files shall be 10 240 B. Implementations shall support all *blocksize* values less than or equal to 32 256 B that are multiples of 512 B.

Implementation-defined formats shall specify a default block size as well as any other block sizes supported for character special archive files.

Any attempt to append to an archive file in a format different from the existing archive format shall cause `pax` to exit immediately with a nonzero exit status.

In copy mode, if no `–x` format is specified, `pax` shall behave as if `–x pax` were specified. B B

−X When traversing the file hierarchy specified by a pathname, pax shall not descend into directories that have a different device ID [*st_dev*, see POSIX.1 {8} *stat*()].

The options that operate on the names of files or archive members (−c, −i, −n, −s, −u, and −v) shall interact as follows. In read mode, the archive members shall be "selected" based on the user-specified *pattern* operands as modified by the −c, −n, and −u options. Then, any −s and −i options shall modify, in that order, the names of the selected files. The −v option shall write names resulting from these modifications.

In write mode, the files shall be selected based on the user-specified pathnames as modified by the −n and −u options. Then, any −s and −i options shall, in that order, modify the names of these selected files. The −v option shall write names resulting from these modifications.

If both the −u and −n options are specified, pax shall not consider a file selected unless it is newer than the file to which it is compared.

4.48.3.1 List-Mode Format Specifications

In list mode with the −o listopt=*format* option, the *format* argument shall be applied for each selected file. The pax utility shall append a <newline> character to the listopt output for each selected file. B

The *format* argument shall be used as the *format* string described in 2.12, with the exceptions (1) through (5) defined in 4.50.7, plus the following exceptions: B

(6) The sequence (*keyword*) can occur before a format conversion specifier. The conversion argument is defined by the value of *keyword*. The implementation shall support the following keywords:

— Any of the Field Name entries in Table 4-100 and Table 4-102. The implementation may support the cpio keywords without the leading c_ in addition to the form required by Table 4-102.

— Any keyword defined for the the extended header in 4.48.7.1.2.

— Any keyword provided as an implementation-defined extension within the extended header defined in 4.48.7.1.2.

For example, the sequence %(charset)s is the string value of the name of the character set in the extended header.

The result of the keyword conversion argument shall be the value from the applicable header field or extended header, without any trailing NULs.

All keyword values used as conversion arguments shall be translated from the UTF8 encoding to the character set appropriate for the local file system, user database, etc., as applicable.

(7) An additional conversion character, T, shall be used to specify time formats. The T conversion character can be preceded by the sequence

(*keyword*=*subformat*), where *subformat* is a date format as defined by 4.15.4.1. The default *keyword* shall be `mtime` and the default *subformat* shall be: `%b %e %H:%M %Y`. B

(8) An additional conversion character, `M`, shall be used to specify the file mode string as defined in 4.39.6.1. If (*keyword*) is omitted, the `mode` keyword shall be used. For example, `%.1M` writes the single character corresponding to the <*entry type*> field of the `ls –l` command. B B B

(9) An additional conversion character, `D`, shall be used to specify the device for block or special files, if applicable, in an implementation-defined format. If not applicable, and (*keyword*) is specified, then this conversion shall be equivalent to `%(`*keyword*`)u`. If not applicable, and (*keyword*) is omitted, then this conversion shall be equivalent to `<space>`.

(10) An additional conversion character, `F`, shall be used to specify a pathname. The `F` conversion character can be preceded by a sequence of comma separated keywords:

(*keyword*[,*keyword*] . . .)

The values for all the keywords that are non-null shall be concatenated together, each separated by a `/`. The default shall be `(path)` if the keyword *path* is defined; otherwise, the default shall be `(prefix,name)`.

(11) An additional conversion character, `L`, shall be used to specify a symbolic link expansion. If the current file is a symbolic link, then `%L` shall expand to:

`"%s -> %s"`, <*value of keyword*>, <*contents of link*>

Otherwise, the `%L` conversion character shall be the equivalent of `%F`. B

4.48.4 Operands

The following operands shall be supported by the implementation:

directory	The destination directory pathname for copy mode.
file	A pathname of a file to be copied or archived.
pattern	A pattern matching one or more pathnames of archive members. A pattern shall be given in the name-generating notation of the pattern matching notation in 3.13, including the filename expansion rules in 3.13.3. The default, if no *pattern* is specified, is to select all members in the archive.

4.48.5 External Influences

4.48.5.1 Standard Input

In write mode, the standard input shall be used only if no *file* operands are specified. It shall be a text file containing a list of pathnames, one per line, without leading or trailing `<blank>`s.

In list and read modes, if `−f` is not specified, the standard input shall be an archive file. (See 4.48.5.2.)

Otherwise, the standard input shall not be used.

4.48.5.2 Input Files

The input file named by the *archive* option-argument, or standard input when the archive is read from there, shall be a file formatted according to one of the specifications in 4.48.7 or some other implementation-defined format. B

The file `/dev/tty` shall be used to write prompts and read responses.

4.48.5.3 Environment Variables

The following environment variables shall affect the execution of `pax`:

LANG	This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.
LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.
LC_COLLATE	This variable shall determine the locale for the behavior of ranges, equivalence classes, and multicharacter collating elements used in the pattern matching expressions for the *pattern* operand, the BRE for the `−s` option, and the ERE defined for the `yesexpr` locale keyword in the LC_MESSAGES category.
LC_CTYPE	This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files) and the behavior of character classes within REs and pattern matching.
LC_MESSAGES	This variable shall determine the processing of affirmative responses and the language in which messages should be written.

LC_TIME	This variable shall determine the format and contents of date and time strings when the `−v` option is specified.
TMPDIR	This variable shall be interpreted as a pathname that provides part of the default global extended header record file name, as described for the `−o globexthdr=` keyword in 4.48.3.

4.48.5.4 Asynchronous Events

Default.

4.48.6 External Effects

4.48.6.1 Standard Output

In write mode, if `−f` is not specified, the standard output shall be the archive formatted according to one of the specifications in 4.48.7 or some other implementation-defined format. (See `−x` *format* under 4.48.3.)

In list mode, when the `−o listopt=`*format* option has been specified, the selected archive members shall be written to standard output using the format described in 4.48.3.1. In list mode without the `−o listopt=`*format* option, the table of contents of the selected archive members shall be written to standard output using the following format:

`"%s\n"`, <*pathname*>

If the `−v` option is specified in list mode, the table of contents of the selected archive members shall be written to standard output using the following formats:

For pathnames representing hard links to previous members of the archive:

`"%s∆==∆%s\n"`, <*ls −l listing*>, <*linkname*>

For all other pathnames:

`"%s\n"`, <*ls −l listing*>

where <*ls −l listing*> shall be the format specified by the `ls` utility (see 4.39) with the `−l` option. When writing pathnames in this format, it is unspecified what is written for fields for which the underlying archive format does not have the correct information, although the correct number of `<blank>`-separated fields shall be written.

In list mode, standard output shall not be buffered more than one line at a time.

4.48.6.2 Standard Error

If `−v` is specified in read, write, or copy modes, `pax` shall write the pathnames it processes to standard error using the following format:

"%s\n", <*pathname*>

These pathnames shall be written as soon as processing is begun on the file or archive member and shall be flushed to standard error. The trailing <newline>, which shall not be buffered, shall be written when the file has been read or written.

If the −s option is specified, and the replacement string has a trailing p, substitutions shall be written to standard error in the following format:

"%s∆>>∆%s\n", <*original pathname*>, <*new pathname*>

In all operating modes of pax (see 4.48.2), optional messages of unspecified format concerning the input archive format and volume number, and the number of files, blocks, volumes, and media parts, as well as other diagnostic messages, may be written to standard error.

In all formats, for both standard output and standard error, it is unspecified how nonprintable characters in pathnames or linknames are written.

Editor's Note: The Draft 10 editing instructions mistakenly called for the following paragraph to replace all of 4.48.6.2. I believe the correct action is merely to add it to the end of the subclause, as I've done here. B

When pax is in read mode or list mode, using the −x pax archive format, and a file name, link name, owner name, or any other field in an extended header record cannot be translated from the pax UTF8 codeset format to the codeset and current locale of the implementation, pax shall write a diagnostic message to standard error, shall process the file as described for the −o invalid= option, and then shall process the next file in the archive. B

4.48.6.3 Output Files

In read mode, the extracted output files shall be of the archived file type. In copy mode, the copied output files shall be the type of the file being copied. In either mode, existing files in the destination hierarchy shall be overwritten only when all permission (−p), modification time (−u), and invalid-value (−o invalid=) tests allow it. B

In write mode, the output file named by the −f option argument shall be a file formatted according to one of the specifications in 4.48.7 or some other implementation-defined format.

4.48.7 Extended Description

4.48.7.1 pax Interchange Format

A pax archive tape or file produced in the −x pax format shall contain a series of blocks. The physical layout of the archive shall be identical to the ustar format described in 4.48.7.2. Each file archived shall be represented by the following sequence:

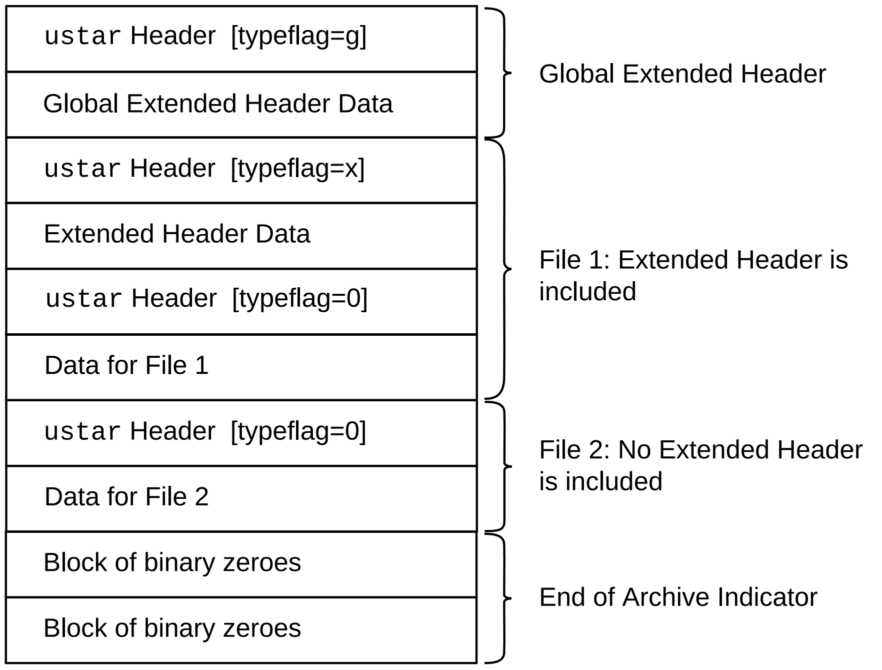
— An optional header block with extended header records. This header block is of the form described in 4.48.7.1.1, with a *typeflag* value of x or g. The extended header records, described in 4.48.7.1.2, are included as the data for this header block.

— A header block that describes the file. Any fields in the preceding optional extended header override the associated fields in this header block for this file.

— Zero or more blocks that contain the contents of the file. B

At the end of the archive file there shall be two 512 B blocks filled with binary zeroes, interpreted as an end-of-archive indicator.

A schematic of an example archive with global extended header records and two actual files is shown in Figure 4-1. In the example, the second file in the archive has no extended header preceding it, presumably because it has no need for extended attributes.



ustar Header [typeflag=g]
Global Extended Header Data
ustar Header [typeflag=x]
Extended Header Data
ustar Header [typeflag=0]
Data for File 1
ustar Header [typeflag=0]
Data for File 2
Block of binary zeroes
Block of binary zeroes
Global Extended Header
File 1: Extended Header is included
File 2: No Extended Header is included
End of Archive Indicator

Figure 4-1 – pax Format Archive Example

4.48.7.1.1 Header Block

The header block shall be identical to the ustar header block described in 4.48.7.2, except that two additional *typeflag* values are defined:

`'x'` Represents extended header records for the following file in the archive (which shall have its own `ustar` header block). The format of these extended header records shall be as described in 4.48.7.1.2.

`'g'` Represents global extended header records for the following files in the archive. The format of these extended header records shall be as described in 4.48.7.1.2. Each value shall affect all subsequent files that do not override that value in their own extended header record and until another global extended header record is reached that provides another value for the same field. The *typeflag* `g` global headers should not be used with interchange media that could suffer partial data loss in transporting the archive.

For both of these types, the *size* field shall be the size of the extended header records in octets. The other fields in the header block are not meaningful to this version of the `pax` utility. However, if this archive is read by a `pax` utility conforming to a previous version of this standard, the header block fields are used to create a regular file that contains the extended header records as data. Therefore, header block field values should be selected to provide reasonable file access to this regular file.

A further difference from the `ustar` header block is that data blocks for files of *typeflag* `1` (hard link) may be included, which means that the *size* field may be greater than zero. Archives created by `pax –o linkdata` shall include these data blocks with the hard links.

4.48.7.1.2 Extended Header

An extended header contains values that are inappropriate for the `ustar` header block because of limitations in that format: fields requiring a character encoding other than ISO/IEC 646 {1}; fields representing file attributes not described in the `ustar` header; fields whose format or length do not fit the requirements of the `ustar` header. The values in an extended header add attributes to the following file (or files—see the description of the *typeflag* `g` header block) or override values in the following header block(s), as indicated in the following list of keywords.

An extended header shall consist of one or more records, each constructed as follows:

`"%d %s=%s\n"`, <*length*>, <*keyword*>, <*value*>

The extended header records shall be encoded in ISO/IEC 10646 {10} Universal Translation Format 8 (UTF8). The <*length*>, `<blank>`s, equals sign, and `<newline>` shown shall be limited to the portable character set, as encoded in UTF8. The <*keyword*> and <*value*> fields can be any UTF8 characters.

The <*length*> field shall be the decimal length of the extended header record in octets, including the trailing `<newline>`.

The <*keyword*> field shall be one of the entries from the following list or a keyword provided as an implementation extension. Keywords consisting entirely of lowercase letters, digits, and periods are reserved for future standardization. A keyword shall not include an equals sign. [In the following list, the notations “file(s)” or “block(s)” are used to acknowledge that a keyword affects the following single file after a *typeflag* `x` extended header, but possibly multiple files after *typeflag* `g`. Any requirements in the list for `pax` to include a record when in write or copy mode shall apply only when such a record has not already been provided through the use of the `−o` option. When used in copy mode, `pax` shall behave as if an archive had been created with applicable extended header records and then extracted.]

`atime` The file access time for the following file(s), equivalent to the value of the *st_atime* member of the *stat* structure for a file, as described in POSIX.1 {8}. The access time shall be restored if the process has the appropriate privilege required to do so. The format of the <*value*> shall be as described in 4.48.7.1.4.

`charset` The name of the character set used to encode the data in the following file(s). The entries in the following table are defined to refer to known standards; additional names may be agreed on between the originator and recipient.

<*value*>	**Formal Standard**
`ISO-IR△646△1990`	ISO/IEC 646 IRV {1}
`ISO-IR△8859△1△1987`	ISO 8859-1 {5}
`ISO-IR△8859△2△1987`	ISO 8859-2 {6}
`ISO-IR△10646△1993`	ISO/IEC 10646 {10}
`ISO-IR△10646△1993△UTF8`	ISO/IEC 10646 {10}, UTF8 encoding
`BINARY`	None

The encoding is included in an extended header for information only; when `pax` is used as described in this standard, it shall not translate the file data into any other encoding. The BINARY entry indicates unencoded binary data.

When used in write or copy mode, it is implementation defined whether `pax` includes a `charset` extended header record for a file.

`comment` A series of characters used as a comment. All characters in the <*value*> field shall be ignored by `pax`.

`ctime` The file creation time for the following file(s), equivalent to the value of the *st_ctime* member of the *stat* structure for a file, as described in POSIX.1 {8}. The creation time shall be restored if the process has the appropriate privilege required to do so. The format of the <*value*> shall be as described in 4.48.7.1.4.

`gid` The group ID of the group that owns the file, expressed as a decimal number using digits from ISO/IEC 646 {1}. This record shall override the *gid* field in the following header block(s). When used in write or copy mode, `pax` shall include a `gid` extended header record for each file whose group ID is greater than 99 999 999.

`gname` The group of the file(s), formatted as a group name in the group database. This record shall override the *gid* and *gname* fields in the following header block(s), and any `gid` extended header record. When used in read, copy, or list mode, `pax` shall translate the name from the UTF8 encoding in the header record to the character set appropriate for the group database on the receiving system. If any of the UTF8 characters cannot be translated, and if the `−o invalid=UTF8` option is not specified, the results are implementation defined. When used in write or copy mode, `pax` shall include a `gname` extended header record for each file whose group name cannot be represented entirely with the letters and digits of the portable character set.

`linkpath` The pathname of a link being created to another file, of any type, previously archived. This record shall override the *linkname* field in the following `ustar` header block(s).

The following `ustar` header block shall determine the type of link created. If *typeflag* of the following header block is `1`, it shall be a hard link. If *typeflag* is `2`, it shall be a symbolic link and the `linkpath` value shall be the contents of the symbolic link.

The `pax` utility shall translate the name of the link (contents of the symbolic link) from the UTF8 encoding to the character set appropriate for the local file system.

When used in write or copy mode, `pax` shall include a `linkpath` extended header record for each link whose pathname cannot be represented entirely with the members of the portable character set other than NUL.

`mtime` The file modification time of the following file(s), equivalent to the value of the *st_mtime* member of the *stat* structure for a file, as described in POSIX.1 {8}. This record shall override the *mtime* field in the following header block(s). The modification time shall be restored if the process has the appropriate privilege required to do so. The format of the <*value*> shall be as described in 4.48.7.1.4.

`path` The pathname of the following file(s). This record shall override the *name* and *prefix* fields in the following header block(s). The `pax` utility shall translate the pathname of the file from the UTF8 encoding to the character set appropriate for the local file system.

When used in write or copy mode, `pax` shall include a `path` extended header record for each file whose pathname cannot be represented entirely with the members of the portable character set other than NUL.

`realtime.`*any*

The keywords prefixed by "`realtime.`" are reserved for future POSIX realtime standardization.

`security.`*any*
: The keywords prefixed by "`security.`" are reserved for future POSIX security standardization.

`size`
: The size the file in octets, expressed as a decimal number using digits from ISO/IEC 646 {1}. This record shall override the *size* field in the following header block(s). When used in write or copy mode, `pax` shall include a `size` extended header record for each file with a *size* value greater than 999 999 999 999.

`uid`
: The user ID of the file owner, expressed as a decimal number using digits from ISO/IEC 646 {1}. This record shall override the *uid* field in the following header block(s). When used in write or copy mode, `pax` shall include a `uid` extended header record for each file whose owner ID is greater than 99 999 999.

`uname`
: The owner of the following file(s), formatted as a user name in the user database. This record shall override the *uid* and *uname* fields in the following header block(s), and any `uid` extended header record. When used in read, copy, or list mode, `pax` shall translate the name from the UTF8 encoding in the header record to the character set appropriate for the user database on the receiving system. If any of the UTF8 characters cannot be translated, and if the `–o invalid=UTF8` option is not specified, the results are implementation defined. When used in write or copy mode, `pax` shall include a `uname` extended header record for each file whose user name cannot be represented entirely with the letters and digits of the portable character set.

If the <*value*> field is zero length, it shall delete any header block field, previously entered extended header value, or global extended header value of the same name.

If a keyword in an extended header record (or in a `–o` option-argument) overrides or deletes a corresponding field in the `ustar` header block, `pax` shall ignore the contents of that header block field.

Unlike the `ustar` header block fields, NULs shall not delimit <*value*>s; all characters within the <*value*> field shall be considered data for the field. None of the length limitations of the `ustar` header block fields in Table 4-100 shall apply to the extended header records.

4.48.7.1.3 Extended Header Keyword Precedence

This subclause describes the precedence in which the various header records and fields and command-line options are selected to apply to a file in the archive. When `pax` is used in read or list modes, it shall determine a file attribute in the following sequence:

(1) If `–o delete=`*keyword-prefix* is used, the affected attributes shall be determined from step (7), if applicable, or ignored otherwise.

(2) If −o *keyword*:= is used, the affected attributes shall be ignored.

(3) If −o *keyword*:=*value* is used, the affected attribute shall be assigned the *value*.

(4) If there is a *typeflag* x extended header record, the affected attribute shall be assigned the <*value*>. When extended header records conflict, the last one given in the header shall take precedence.

(5) If −o *keyword*=*value* is used, the affected attribute shall be assigned the *value*.

(6) If there is a *typeflag* g global extended header record, the affected attribute shall be assigned the <*value*>. When global extended header records conflict, the last one given in the global header shall take precedence.

(7) Otherwise, the attribute shall be determined from the ustar header block.

4.48.7.1.4 Extended Header File Times

The pax utility shall write atime and ctime records for each file in write or copy modes only if the −o times option is specified; pax shall write a mtime record for each file in write or copy modes if the file system of the underlying implementation supports time granularities smaller than that required by the ustar header block described in 4.48.7.2. All of these time records shall be formatted as a decimal representation of the time in seconds since the Epoch. If a period (.) decimal point character is present, the digits to the right of the point shall represent the units of a subsecond timing granularity, where the first digit is tenths of a second and each subsequent digit is a tenth of the previous digit. Implementations may ignore any portion of the subsecond digits for which they do not support the necessary timing granularity; they shall not perform any rounding operation.

4.48.7.2 ustar Interchange Format

A ustar archive tape or file shall contain a series of blocks. Each block shall be a fixed-size block of 512 octets (see below). Although this format may be thought of as being stored on 9-track industry-standard 12,7 mm (0,5 in) magnetic tape, other types of transportable media are not excluded. Each file archived shall be represented by a header block that describes the file, followed by zero or more blocks that give the contents of the file. At the end of the archive file there shall be two 512 B blocks filled with binary zeroes, interpreted as an end-of-archive indicator.

The blocks may be grouped for physical I/O operations, as described under the −b *blocksize* and −x ustar options. Each group of blocks may be written with a single operation equivalent to the *write*() function in POSIX.1 {8}. On magnetic tape, the result of this write shall be a single tape record. The last group of blocks always shall be at the full size, so blocks after the two zero blocks may contain undefined data.

The header block shall be structured as shown in Table 4-100. All lengths and offsets are in decimal.

Table 4-100 – ustar Header Block

Field Name	Offset (in octets)	Length (in octets)
name	0	100
mode	100	8
uid	108	8
gid	116	8
size	124	12
mtime	136	12
chksum	148	8
typeflag	156	1
linkname	157	100
magic	257	6
version	263	2
uname	265	32
gname	297	32
devmajor	329	8
devminor	337	8
prefix	345	155

All characters in the header block shall be represented in the coded character set of ISO/IEC 646 {1}. For maximum portability between implementations, names should be selected from characters represented by the portable filename character set as octets with the most significant bit zero. If an implementation supports the use of characters outside of slash and the portable filename character set in names for files, users, and groups, one or more implementation-defined encodings of these characters shall be provided for interchange purposes.

Each field within the header block shall be contiguous; that is, there shall be no padding used. Each character on the archive medium shall be stored contiguously.

The fields *magic*, *uname*, and *gname* shall be character strings each terminated by a NUL character. The fields *name*, *linkname*, and *prefix* shall be NUL-terminated character strings except when all characters in the array contain non-NUL characters including the last character. The *version* field shall be two octets containing the characters "00" (zero-zero). The *typeflag* shall contain a single character. All other fields shall be leading zero-filled octal numbers using digits from ISO/IEC 646 {1} IRV. Each numeric field shall be terminated by one or more `<space>` or NUL characters.

The *name* and the *prefix* fields shall produce the pathname of the file. A new pathname shall be formed, if *prefix* is not an empty string (its first character is not NUL), by concatenating *prefix* (up to the first NUL character), a slash character, and *name*; otherwise, *name* shall be used alone. In either case, *name* shall be terminated at the first NUL character. If *prefix* begins with a NUL character, it shall be ignored. In this manner, pathnames of at most 256 characters can be supported. If a pathname does not fit in the space provided, the pax utility shall notify the user of the error, and shall not attempt to store any part of the file—header or data—on the medium.

The *linkname* field, described below, shall not use the *prefix* to produce a pathname. As such, a *linkname* is limited to 100 characters. If the name does not fit in the space provided, the pax utility shall notify the user of the error, and shall not attempt to store the link on the medium.

The *mode* field provides 12 b encoded in ISO/IEC 646 {1} octal digit representation. The encoded bits shall represent the bitwise inclusive OR of the values in Table 4-101, expressed in terms of their equivalent POSIX.1 {8} bits.

Table 4-101 – ustar *mode* Field

Bit Value	POSIX.1 {8} Bit	Description
04 000	S_ISUID	Set user ID on execution
02 000	S_ISGID	Set group ID on execution
01 000	*<reserved>*	Reserved for future standardization
00 400	S_IRUSR	Read permission for file owner class
00 200	S_IWUSR	Write permission for file owner class
00 100	S_IXUSR	Execute/search permission for file owner class
00 040	S_IRGRP	Read permission for file group class
00 020	S_IWGRP	Write permission for file group class
00 010	S_IXGRP	Execute/search permission for file group class
00 004	S_IROTH	Read permission for file other class
00 002	S_IWOTH	Write permission for file other class
00 001	S_IXOTH	Execute/search permission for file other class

When appropriate privilege is required to set one of these mode bits, and the user restoring the files from the archive does not have the appropriate privilege, the mode bits for which the user does not have appropriate privilege shall be ignored. Some of the mode bits in the archive format are not mentioned elsewhere in this standard or POSIX.1 {8}. If the implementation does not support those bits, they may be ignored.

The *uid* and *gid* fields shall be the user and group ID of the owner and group of the file, respectively.

The *size* field shall be the size of the file in octets. If the *typeflag* field is set to specify a file to be of type 1 (hard link) or 2 (symbolic link), the *size* field shall be specified as zero. If the *typeflag* field is set to specify a file of type 5 (directory), the *size* field shall be interpreted as described under the definition of that record type. No data blocks shall be stored for types 1, 2, or 5. If the *typeflag* field is set to 3 (character special file), 4 (block special file), or 6 (FIFO), the meaning of the *size* field is unspecified by this standard, and no data blocks shall be stored on the medium. Additionally, for 6, the *size* field shall be ignored when reading. If the *typeflag* field is set to any other value, the number of blocks written following the header shall be (*size*+511)/512, ignoring any fraction in the result of the division.

The *mtime* field shall be the modification time of the file at the time it was archived. It is the ISO/IEC 646 {1} representation of the octal value of the modification time obtained from the equivalent of the POSIX.1 {8} *stat*() function.

The *chksum* field shall be the ISO/IEC 646 {1} IRV representation of the octal value of the simple sum of all octets in the header block. Each octet in the header shall be treated as an unsigned value. These values shall be added to an unsigned integer, initialized to zero, the precision of which shall be not less than 17 b. When calculating the checksum, the *chksum* field shall be treated as if it were all `<space>`s.

The *typeflag* field shall specify the type of file archived. If a particular implementation does not recognize the type, or the user does not have appropriate privilege to create that type, the file shall be extracted as if it were a regular file if the file type is defined to have a meaning for the size field that could cause data blocks to be written on the medium (see the previous description for *size*). If conversion to a regular file occurs, the `pax` utility shall produce an error indicating that the conversion took place. All of the *typeflag* fields shall be coded in ISO/IEC 646 {1} IRV:

`'0'` Represents a regular file. For backward compatibility, a *typeflag* value of binary zero (`'\0'`) should be recognized as meaning a regular file when extracting files from the archive. Archives written with this version of the archive file format shall create regular files with a *typeflag* value of ISO/IEC 646 {1} IRV `'0'`.

`'1'` Represents a file linked to another file, of any type, previously archived. Such files shall be identified by each file having the same device and file serial number. The linked-to name shall be specified in the *linkname* field with a NUL-character terminator if it is less than 100 octets in length.

`'2'` Represents a symbolic link. The contents of the symbolic link shall be stored in the *linkname* field.

`'3'`,`'4'` Represent character special files and block special files respectively. In this case the *devmajor* and *devminor* fields shall contain information defining the device, the format of which is unspecified by this standard. Implementations may map the device specifications to their own local specification or may ignore the entry.

`'5'` Specifies a directory or subdirectory. On systems where disk allocation is performed on a directory basis, the *size* field shall contain the maximum number of octets (which may be rounded to the nearest disk block allocation unit) that the directory may hold. A *size* field of zero shall indicate no such limiting. Systems that do not support limiting in this manner should ignore the *size* field.

`'6'` Specifies a FIFO special file. Note that the archiving of a FIFO file archives the existence of this file and not its contents.

`'7'`	Reserved to represent a file to which an implementation has associated some high performance attribute. Implementations without such extensions should treat this file as a regular file (type `'0'`).
`'A'-'Z'`	The letters `A` through `Z` are reserved for custom implementations. All other values are reserved for specification in future revisions of this standard.

The *magic* field is the specification that this archive was output in this archive format. If this field contains `"ustar"` (the five ISO/IEC 646 {1} IRV characters shown followed by NUL), the *uname* and *gname* fields shall contain the ISO/IEC 646 {1} IRV representation of the owner and group of the file respectively (truncated to fit, if necessary). When the file is restored by a privileged, protection-preserving version of the utility, the password and group files shall be scanned for these names. If found, the user and group IDs contained within these files shall be used rather than the values contained within the *uid* and *gid* fields.

4.48.7.3 cpio Interchange Format

The octet-oriented `cpio` archive format shall be a series of entries, each comprising a header that describes the file, the name of the file, and then the contents of the file.

An archive may be recorded as a series of fixed-size blocks of octets. This blocking shall be used only to make physical I/O more efficient. The last group of blocks always shall be at the full size.

For the octet-oriented `cpio` archive format, the individual entry information shall be in the order indicated and described by Table 4-102.

4.48.7.3.1 cpio Header

For each file in the archive, a header as defined previously shall be written. The information in the header fields shall be written as streams of ISO/IEC 646 {1} characters interpreted as octal numbers. The octal numbers shall be extended to the necessary length by appending ISO/IEC 646 {1} IRV zeros at the most-significant-digit end of the number; the result is written to the stream of octets most-significant-digit first. The fields shall be interpreted as follows:

c_magic	Identifies the archive as being a transportable archive by containing the identifying value `"070707"`.
c_dev	
c_ino	Contains values that uniquely identify the file within the archive (i.e., no files shall contain the same pair of *c_dev* and *c_ino* values unless they are links to the same file). The values shall be determined in an unspecified manner.

Table 4-102 – Octet-Oriented cpio Archive Entry

Header		
Field Name	**Length (in octets)**	**Interpreted as**
c_magic	6	Octal number
c_dev	6	Octal number
c_ino	6	Octal number
c_mode	6	Octal number
c_uid	6	Octal number
c_gid	6	Octal number
c_nlink	6	Octal number
c_rdev	6	Octal number
c_mtime	11	Octal number
c_namesize	6	Octal number
c_filesize	11	Octal number
File Name		
Field Name	**Length**	**Interpreted as**
c_name	*c_namesize*	Pathname string
File Data		
Field Name	**Length**	**Interpreted as**
c_filedata	*c_filesize*	Data

c_mode The encoded bits shall represent the bitwise inclusive OR of the values in Table 4-103, expressed in terms of their equivalent POSIX.1 {8} bits, added to one of the values in Table 4-104. Directories, FIFOs, and regular files shall be supported on a system conforming to this standard; additional values defined previously are reserved for compatibility with existing systems. Additional file types may be supported; however, such files should not be written on archives intended for transport to portable systems.

c_uid Contains the user ID of the owner.

c_gid Contains the group ID of the group.

c_nlink Contains the number of links referencing the file at the time the archive was created.

c_rdev Contains implementation-defined information for character or block special files.

c_mtime Contains the latest time of modification of the file at the time the archive was created.

c_namesize Contains the length of the pathname, including the terminating NUL character.

Table 4-103 – cpio *c_mode* File Modes

Bit Value	POSIX.1 {8} Bit	Description
04 000	S_ISUID	Set user ID on execution
02 000	S_ISGID	Set group ID on execution
01 000	*<reserved>*	Reserved for future standardization
00 400	S_IRUSR	Read permission for file owner class
00 200	S_IWUSR	Write permission for file owner class
00 100	S_IXUSR	Execute/search permission for file owner class
00 040	S_IRGRP	Read permission for file group class
00 020	S_IWGRP	Write permission for file group class
00 010	S_IXGRP	Execute/search permission for file group class
00 004	S_IROTH	Read permission for file other class
00 002	S_IWOTH	Write permission for file other class
00 001	S_IXOTH	Execute/search permission for file other class

Table 4-104 – cpio *c_mode* File Types

Bit Value	Description
040 000	Directory
010 000	FIFO
0100 000	Regular file
060 000	Block special file
020 000	Character special file
0110 000	Reserved for future standardization
0120 000	Symbolic link
0140 000	Reserved for future standardization

c_filesize Contains the length of the file in octets. This shall be the length of the data section following the header structure.

4.48.7.3.2 cpio File Name

The *c_name* field shall contain the pathname of the file. The length of this field in octets shall be the value of *c_namesize*.

All characters shall be represented in ISO/IEC 646 {1} IRV. For maximum portability between implementations, names should be selected from characters represented by the portable filename character set as octets with the most significant bit zero. If an implementation supports the use of characters outside of slash and the portable filename character set in names for files, users, and groups, one or more implementation-defined encodings of these characters shall be provided for interchange purposes. B

4.48.7.3.3 cpio File Data

Following *c_name*, there shall be *c_filesize* octets of data. Interpretation of such data shall occur in a manner dependent on the file. If *c_filesize* is zero, no data shall be contained in *c_filedata*. Only regular files have data to be restored. B

4.48.7.3.4 cpio Special Entries

FIFO special files, directories, and the trailer shall be recorded with *c_filesize* equal to zero. For other special files, *c_filesize* is unspecified by this standard. The header for the next file entry in the archive shall be written directly after the last octet of the file entry preceding it. A header denoting the file name “TRAILER!!!” shall indicate the end of the archive; the contents of octets in the last block of the archive following such a header are undefined.

4.48.8 Exit Status

The pax utility shall exit with one of the following values:

0	All files were processed successfully.
>0	An error occurred.

4.48.9 Consequences of Errors

If pax cannot create a file or a link when reading an archive; cannot find a file when writing an archive; or cannot preserve the user ID, group ID, or file mode when the –p option is specified; a diagnostic message shall be written to standard error and a nonzero exit status shall be returned, but processing shall continue. In the case where pax cannot create a link to a file, pax shall not, by default, create a second copy of the file.

If the extraction of a file from an archive is prematurely terminated by a signal or error, pax may have only partially extracted the file or (if the –n option was not specified) may have extracted a file of the same name as that specified by the user, but that is not the file the user wanted. Additionally, the file modes of extracted directories may have additional bits from the S_IRWXU mask set as well as incorrect modification and access times.

4.51 pwd – Return working directory name

⇒ **4.51.1 pwd Synopsis.** *Change the Synopsis to:*

```
pwd  [ –L | –P ]
```

⇒ **4.51.2 pwd Description.** *Change this subclause to:*

The pwd utility shall write to standard output an absolute pathname of the current working directory, which does not contain the filenames dot or dot-dot.

⇒ **4.51.3 pwd Options.** *Change the entire subclause to:*

The pwd utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

–L	If the **PWD** environment variable contains an absolute pathname of the current directory that does not contain the filenames dot or dot-dot, pwd shall write this pathname to standard output. Otherwise, the –L option shall behave as the –P option.
–P	The absolute pathname written shall not contain filenames that, in the context of the pathname, refer to files of type symbolic link.

If both –L and –P are specified, the last one shall apply. If neither –L nor –P is specified, the pwd utility shall behave as if –L had been specified.

⇒ **4.51.5.3 pwd Environment Variables.** *Add the following variable in the correct sorted order:*

PWD	If the –P option is in effect, this variable shall be set to an absolute pathname of the current working directory that does not contain any components that specify symbolic links, does not contain any components that are dot, and does not contain any components that are dot-dot. If an application sets or unsets the value of **PWD**, the behavior of pwd is unspecified.

4.53 `rm` – Remove directory entries

⇒ **4.53.2 `rm` Description.** *Replace item (2c) with:*

For each entry contained in *file*, other than dot or dot-dot, the four steps listed here [(1)-(4)] shall be taken with the entry as if it were a *file* operand. The `rm` utility shall not traverse directories by following symbolic links into other parts of the hierarchy, but shall remove the links themselves.

⇒ **4.53.8 `rm` Exit Status.** *Change the description of the 0 value to:* B

0 All of the named directory entries for which `rm` performed actions equivalent to the POSIX.1 {8} *rmdir*() or *unlink*() functions were removed. B

Rationale: This change is the result of interpretation request PASC 1003.2-92 #75 submitted for IEEE Std 1003.2-1992. B

Editor's Note: The following rationale will be added to E.4.53, but is kept here with `rm` *for this draft:*

`rm` Rationale. *(This subclause is not a part of P1003.2b)*

The `rm` utility removes symbolic links themselves, not the files they refer to, as a consequence of the dependence on the POSIX.1 {8} *unlink*() functionality, per the Description. When removing hierarchies with `−r` or `−R`, the prohibition on following symbolic links has to be made explicit.

4.55 sed – Stream editor

Rationale: The changes to sed are to align with historical practice and are the result of interpretation requests PASC 1003.2-92 #34 and #35 submitted for IEEE Std 1003.2-1992.

⇒ **4.55.5.2 sed Input Files.** *Replace this subclause with the following:*

The input files shall be text files. The *script_file*s named by the −f option shall consist of editing commands.

⇒ **4.55.7 sed Extended Description.** *Replace the entire Extended Description with the following.*

Editor's Note: There were numerous terminology changes in this clause, which would have resulted in many dozens of individual change descriptions. I chose to reprint the entire clause with the changes embedded. Lines changed from POSIX.2-1992 are diffmarked for Draft 10 only; these are the lines subject to P1003.2b balloting. The diffmarks were removed in Draft 11.

4.55.7 Extended Description

The *script* shall consist of editing commands of the following form:

[*address***[**,*address***]]***function*

where *function* represents a single-character command verb from the list in 4.55.7.3, followed by any applicable arguments.

Zero or more <blank>s shall be accepted before the first address and before *function*. Any number of semicolons shall be accepted before the first *address*.

In default operation, sed cyclically shall copy a line of input, less its terminating <newline>, into a *pattern space* (unless there is something left after a D command), apply in sequence all commands whose addresses select that pattern space, and at the end of the script copy the pattern space to standard output (except when −n is specified) and delete the pattern space. Whenever the pattern space is written to standard output or a named file, sed shall immediately follow it with a <newline>.

Some of the editing commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval. The *pattern* and *hold spaces* shall each be able to hold at least 8192 B.

4.55.7.1 sed Addresses

An address is either a decimal number that counts input lines cumulatively across files, a $ character that addresses the last line of input, or a context address (which consists of a BRE, as described in 4.55.7.2, preceded and followed by a delimiter, usually a slash).

An editing command with no addresses shall select every pattern space.

An editing command with one address shall select each pattern space that matches the address.

An editing command with two addresses shall select the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line shall be selected.) Starting at the first line following the selected range, `sed` shall look again for the first address. Thereafter, the process shall be repeated. Omitting either or both of the *address* components in the **[***address***[**,*address***]]** form produces undefined results.

4.55.7.2 **sed** REs

The `sed` utility shall support the BREs described in 2.8.3, with the following additions:

(1) In a context address, the construction \cBREc, where *c* is any character other than backslash or `<newline>`, shall be identical to /*BRE*/. If the character designated by *c* appears following a backslash, then it shall be considered to be that literal character, which shall not terminate the BRE. For example, in the context address `\xabc\xdefx`, the second `x` stands for itself, so that the BRE is `abcxdef`.

(2) The escape sequence `\n` shall match a `<newline>` embedded in the pattern space. A literal `<newline>` character shall not be used in the BRE of a context address or in the substitute function.

(3) If an RE is empty (i.e., no pattern is specified) `sed` shall behave as if the last RE used in the last command applied (either as an address or as part of a substitute command) was specified.

4.55.7.3 **sed** Editing Commands

In the following list of editing commands, the maximum number of permissible addresses for each function is indicated by **[***0addr***]**, **[***1addr***]**, or **[***2addr***]**, representing zero, one, or two addresses.

The argument *text* shall consist of one or more lines. Each embedded `<newline>` in the text shall be preceded by a backslash. Other backslashes in text shall be removed, and the following character shall be treated literally.

The `r` and `w` command verbs, and the `w` flag to the `s` command, take an optional *rfile* (or *wfile*) parameter, separated from the command verb letter or flag by one or more `<blank>`s; implementations may allow zero separation as an extension.

The argument *rfile* or the argument *wfile* shall terminate the editing command. Each *wfile* shall be created before processing begins. Implementations shall support at least nine *wfile* arguments in the script; the actual number (≥9) that shall be supported by the implementation is unspecified. The use of the *wfile* parameter shall cause that file to be initially created, if it does not exist, or shall replace the contents of an existing file.

The `b`, `r`, `s`, `t`, `w`, `y`, and `:` command verbs shall accept additional arguments. The following synopses indicate which arguments shall be separated from the command verbs by a single `<space>`.

The `a` and `r` commands schedule text for later output. The text specified for the `a` command, and the contents of the file specified for the `r` command, shall be written to standard output just before the next attempt to fetch a line of input when executing the `N` or `n` commands, or when reaching the end of the script. If written when reaching the end of the script, and the `−n` option was not specified, the text shall be written after copying the pattern space to standard output. The contents of the file specified for the `r` command shall be as of the time the output is written, not the time the `r` command is applied. The text shall be output in the order in which the `a` and `r` commands were applied to the input.

Command verbs other than `{`, `a`, `b`, `c`, `i`, `r`, `t`, `w`, `:`, and `#` can be followed by a semicolon, optional `<blank>`s, and another command verb. However, when the `s` command verb is used with the `w` flag, following it with another command in this manner produces undefined results.

A function can be preceded by one or more `!` characters, in which case the function shall be applied if the addresses do not select the pattern space. Zero or more `<blank>`s shall be accepted before the first `!` character. It is unspecified if `<blank>` characters can follow a `!` character, and conforming applications shall not follow a `!` character with `<blank>`s.

[*2addr***]** `{` *function*
function
. . .
`}`

Execute a list of `sed` functions only when the pattern space is selected. The list of `sed` functions shall be surrounded by braces and separated by `<newline>`s, as follows. The braces can be preceded or followed by `<blank>`s. The *function*s can be preceded by `<blank>`s, but shall not be followed by `<blank>`s. The `<right-brace>` shall be preceded by a `<newline>` and can be preceded or followed by `<blank>`s.

[*1addr***]**`a\`
text

Write *text* to standard output as described previously.

[*2addr***]**`b` **[***label***]**

Branch to the `:` function bearing the *label*. If *label* is not specified, branch to the end of the script. The implementation shall support *labels* recognized as unique up to at least 8 characters; the actual length (≥8) that shall be supported by the implementation is unspecified. It is unspecified whether exceeding a label length causes an error or a silent truncation.

[*2addr***]**`c\`
text Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output and start the next cycle. B

[*2addr***]**`d` Delete the pattern space and start the next cycle.

[*2addr***]**`D` Delete the initial segment of the pattern space through the first `<newline>` and start the next cycle.

[*2addr***]**`g` Replace the contents of the pattern space by the contents of the hold space.

[*2addr***]**`G` Append to the pattern space a `<newline>` followed by the contents of the hold space.

[*2addr***]**`h` Replace the contents of the hold space with the contents of the pattern space.

[*2addr***]**`H` Append to the hold space a `<newline>` followed by the contents of the pattern space.

[*1addr***]**`i\`
text Write *text* to standard output.

[*2addr***]**`l` (The letter ell.) Write the pattern space to standard output in a visually unambiguous form. The characters listed in Table 2-16 (see 2.12), except for `\n`, shall be written as the corresponding escape sequence. Nonprintable characters not in Table 2-16 shall be written as one three-digit octal number (with a preceding backslash) for each byte in the character (most significant byte first). If the size of a byte on the system is greater than 9 b, the format used for nonprintable characters is implementation defined.

Long lines shall be folded, with the point of folding indicated by writing `<backslash><newline>`; the length at which folding occurs is unspecified, but should be appropriate for the output device. The end of each line shall be marked with a `$`.

[*2addr***]**`n` Write the pattern space to standard output if the default output has not been suppressed, and replace the pattern space with the next line of input.

If no next line of input is available, the `n` command verb shall branch to the end of the script and quit without starting a new cycle. B

[*2addr***]**`N` Append the next line of input to the pattern space, using an embedded `<newline>` to separate the appended material from the original material. Note that the current line number changes.

If no next line of input is available, the `N` command verb shall branch to the end of the script and quit without starting a new cycle or copying the pattern space to standard output. B

[*2addr***]**`p` Write the pattern space to standard output.

[*2addr***]**`P` Write the pattern space, up to the first `<newline>`, to standard output.

[*1addr***]**`q` Branch to the end of the script and quit without starting a new cycle.

[*1addr***]**`r` *rfile*

Copy the contents of *rfile* to standard output as described previously. If *rfile* does not exist or cannot be read, it shall be treated as if it were an empty file, causing no error condition. B

[*2addr***]**`s/`*BRE*`/`*replacement*`/`*flags*

Substitute the *replacement* string for instances of the *BRE* in the pattern space. Any character other than backslash or `<newline>` can be used instead of a slash to delimit the BRE and the replacement. Within the BRE and the replacement, the BRE delimiter itself can be used as a literal character if it is preceded by a backslash.

An ampersand (&) appearing in the *replacement* shall be replaced by the string matching the BRE. The special meaning of & in this context can be suppressed by preceding it by backslash. The characters \n, where *n* is a digit, shall be replaced by the text matched by the corresponding backreference expression (see 2.8.3.3). For each backslash (\\) encountered in scanning *replacement* from beginning to end, the backslash shall be discarded and the following character shall lose its special meaning (if any). It is unspecified what special meaning is given to any character other than &, \\, or digits.

A line can be split by substituting a `<newline>` character into it. The application shall escape the `<newline>` in the *replacement* by preceding it by backslash. A substitution shall be considered to have been performed even if the replacement string is identical to the string that it replaces. Any backslash used to alter the default meaning of a subsequent character shall be discarded from the BRE or the replacement before evaluating the BRE or using the replacement. B

The value of *flags* shall be zero or more of

n Substitute for the *n*th occurrence only of the *BRE* found within the pattern space.

`g` Globally substitute for all nonoverlapping instances of the *BRE* rather than just the first one. If both `g` and *n* are specified, the results are unspecified.

`p` Write the pattern space to standard output if a replacement was made.

`w` *wfile* Write. Append the pattern space to *wfile* if a replacement was made. A conforming application shall precede the *wfile* argument with one or more `<blank>`s. If the `w` flag is not the last flag value given in a concatenation of multiple flag values, the results are undefined.

[*2addr***]**`t` **[***label***]**
Test. Branch to the `:` command verb bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a `t`. If *label* is not specified, branch to the end of the script.

[*2addr***]**`w` *wfile*
Append [write] the pattern space to *wfile*.

[*2addr***]**`x` Exchange the contents of the pattern and hold spaces.

[*2addr***]**`y/`*string1*`/`*string2*`/`
Replace all occurrences of characters in *string1* with the corresponding characters in *string2*. If a backslash followed by an `n` appear in *string1* or *string2*, the two characters shall be handled as a single `<newline>` character. If the number of characters in *string1* and *string2* are not equal, or if any of the characters in *string1* appear more than once, the results are undefined. Any character other than backslash or `<newline>` can be used instead of slash to delimit the strings. If the delimiter is not `n`, within *string1* and *string2*, the delimiter itself can be used as a literal character if it is preceded by a backslash. If a backslash character is immediately followed by a backslash character in *string1* or *string2*, the two backslash characters shall be counted as a single literal backslash character. The meaning of a backslash followed by any character that is not `n`, a backslash, or the delimiter character is undefined. B B B B B B

[*0addr***]**`:`*label*
Do nothing. This command bears a *label* to which the `b` and `t` commands branch.

[*1addr***]=** Write the following to standard output:

`"%d\n"`, <*current line number*>

[*0addr***]** Ignore this empty command.

[*0addr***]#** Ignore the # and the remainder of the line (treat them as a comment), with the single exception that if the first two characters in the script are `#n`, the default output shall be suppressed; this shall be the equivalent of specifying `–n` on the command line.

Editor's Note: The following rationale will be added to E.4.51, but is kept here with `sed` *for this draft:*

`sed` Rationale. *(This subclause is not a part of P1003.2b)*

B

The `b`, `t`, and `:` commands are documented to ignore leading white space, but no mention is made of trailing white space. Historical implementations of `sed` assigned different locations to the labels `'x'` and `'x '`. This is not useful, and leads to subtle programming errors, but it is historical practice, and changing it could theoretically break working scripts. Implementors are encouraged to provide warning messages about labels that are never used or jumps to labels that do not exist.

Editor's Note: The terminology changes in the normative text will carry over into the rationale as well. They are summarized here using POSIX.2-1992 line numbers within E.4.55:

Line 8018: change "commands" to "editing commands."

Line 8021: change "command" to "function."

Line 8029: change "command lines" to "editing commands."

Line 8035: change "command line" to "editing command."

Line 8038: change "command" to "command verb."

Line 8050: change "command" to "function."

Line 8067: change "commands" to "command verbs."

Line 8078: change "command" to "function."

Line 8081: change "command" to "editing command."

Lines 8083–8084: change "commands" to "editing commands."

Editor's Note: Replace the rationale paragraph (E.4.55, POSIX.2-1992 lines 8083-86) with: B

Historically, the `sed` `!` and `}` editing commands did not permit multiple commands on a single line using a semicolon as a command delimiter. Implementations are permitted, but not required, to support this extension. B

4.56 sh – Shell, the standard command language interpreter

⇒ **4.56.4 sh Operands.** *Change the command_string description to:*

command_string
A string that shall be interpreted by the shell as one or more commands, as if the string were the argument to the POSIX.1 {8} *system*() function. If the *command_string* operand is an empty string, sh shall exit with a zero exit status.

Rationale: This change is part of a general cleanup to remove references to the now-deleted Chapter 7. All of the applicable functions are now in POSIX.1-199x, the version created by the currently balloting P1003.1a.

⇒ **4.56.5.3 sh Environment Variables.** *Change the description of* **ENV** *to:*

ENV This variable, when and only when an interactive shell is invoked, shall be subjected to parameter expansion (see 3.6.2) by the shell, and the resulting value shall be used as a pathname of a file containing shell commands to execute in the current environment. The file need not be executable. If the expanded value of **ENV** is not an absolute pathname, the results are unspecified. **ENV** shall be ignored if the real and effective user IDs or real and effective group IDs of the user are different.

Rationale: The preceding change is the result of interpretation request PASC 1003.2-92 #110 submitted for IEEE Std 1003.2-1992.

⇒ **4.56.5.3 sh Environment Variables.** *Add the following variable in proper sorted order:*

PWD This variable shall represent an absolute pathname of the current working directory. Assignments to this variable may be ignored unless the value is an absolute pathname of the current working directory and there are no filename components of dot or dot-dot.

4.62 `test` – Evaluate expression

⇒ **4.62.4 `test` Operands.** *Replace the* `–r`, `–w`, *and* `–x` *descriptions with the following:*

`–r` *file*	True if *file* exists and is readable. True shall indicate that permission to read from *file* will be granted, as defined in 2.2.2.66.
`–w` *file*	True if *file* exists and is writable. True shall indicate that permission to write to *file* will be granted, as defined in 2.2.2.66.
`–x` *file*	True if *file* exists and is executable. True shall indicate that permission to execute *file* will be granted, as defined in 2.2.2.66. If *file* is a directory, true shall indicate that permission to search *file* will be granted.

Rationale: This change is a clarification and is the result of interpretation request PASC 1003.2-92 #23 submitted for IEEE Std 1003.2-1992.

⇒ **4.62.4 `test` Operands.** *Add the following primary in the proper sorted order:*

`–h` *file*	True if *file* exists and is a symbolic link.

⇒ **4.62.4 `test` Operands.** *Add the following at the end of the primaries list (before the paragraph that begins* "A primary can be preceded by . . . "

With the exception of the `–h` *file* primary, if a *file* argument is a symbolic link, `test` shall evaluate the expression by resolving the symbolic link and using the file referenced by the link.

4.64 tr – Translate characters

Rationale: The following changes related to –C are the result of interpretation requests PASC 1003.2-92 #24 and #25 submitted for IEEE Std 1003.2-1992.

⇒ **4.64.1 tr Synopsis.** *Change the Synopsis to:*

```
tr [–c|–C][–s] string1 string2
tr –s [–c|–C] string1
tr –d [–c|–C] string1
tr –ds [–c|–C] string1 string2
```

⇒ **4.64.3 tr Options.** *Change the description of* –c *to:*

–c Complement the range of values specified by *string1*. See 4.64.7.

–C Complement the set of characters specified by *string1*. See 4.64.7.

⇒ **4.64.7 tr Extended Description.** *Change the description of \octal to:*

\octal Represents octal sequences that can be used to represent specific coded values. An octal sequence shall consist of a backslash followed by the longest sequence of one, two, or three octal-digit characters (01234567). The sequence shall cause the value whose encoding is represented by the one, two, or three digit octal integer to be placed into the array. If the size of a byte on the system is greater than 9 b, the valid escape sequence used to represent a byte is implementation defined.

⇒ **4.64.7 tr Extended Description.** *Change the description of \c–c to:*

c–c Represents the range of characters between the range end-points (as long as neither endpoint is an octal sequence of the form \octal), inclusive, as defined by the current setting of the LC_COLLATE locale category. The starting endpoint shall precede the second endpoint in the current collation order. The characters in the range shall be placed in the array in ascending collation sequence. C C C

If either or both of the range endpoints are octal sequences of the form \octal, this shall represent the range of specific coded values between the two range endpoints, inclusive.

⇒ **4.64.7 `tr` Extended Description.** *In the dashed list following* "When the –d option is not specified", *change the second item to:*

- If the –C option is specified, the complement of the characters specified by *string1*—the set of all characters in the current character set, as defined by the current setting of LC_CTYPE, except for those actually specified in the *string1* operand—shall be placed in the array in ascending collation sequence, as defined by the current setting of LC_COLLATE.
- If the –c option is specified, the complement of the values specified by *string1* shall be placed in the array in ascending order by binary value.

⇒ **4.64.7 `tr` Extended Description.** *In the dashed list following* "When the –d option is specified", *change the second item to:*

- When the –C option is specified with –d, all characters except those specified by *string1* shall be deleted. The contents of *string2* shall be ignored, unless the –s option is also specified.
- When the –c option is specified with –d, all values except those specified by *string1* shall be deleted. The contents of *string2* shall be ignored, unless the –s option is also specified.

Editor's Note: The following rationale will be added to E.4.64, but is kept here with `tr` *for this draft:*

`tr` Rationale. *(This subclause is not a part of P1003.2b)*

A prior version of this standard had a –c option that behaved similarly to the –C option, but did not supply functionality equivalent to the –c option specified in this standard. This meant that historical practice of being able to specify `tr –d \200-\377` (which would delete all bytes with the top bit set) would have no effect because, in the C locale, bytes with the values octal 200 to octal 377 are not characters.

The earlier standard also said that octal sequences referred to collating elements and could be placed adjacent to each other to specify multibyte characters. However, it was noted that this caused ambiguities because `tr` would not be able to tell whether adjacent octal sequences were intending to specify multibyte characters or multiple single byte characters. This standard specifies that octal sequences always refer to single byte binary values.

B

4.72 xargs – Construct argument list(s) and invoke utility

⇒ **4.72.1 xargs Synopsis.** *Change the synopsis to:*

```
xargs [–t] [–E eofstr] [–n number [–x] ] [–s size] [utility [argument
        ... ]]
```

Rationale: This change is required to match historical practice and is the result of interpretation request PASC 1003.2-92 #53 submitted for IEEE Std 1003.2-1992. See the added rationale in E.4.72 and the following three changes.

⇒ **4.72.2 xargs Description.** *Replace the last sentence of the first paragraph of the Description (the one beginning with* "This sequence shall ... "*) with:*

This sequence shall be repeated until one of the following occurs:

— An end-of-file condition is detected on standard input

— The logical end-of-file string (see the –E *eofstr* option) is found on standard input after double-quote processing, apostrophe processing, and backslash escape processing (see next paragraph)

— An invocation of a constructed command line returns an exit status of 255

Rationale: See 4.72.1 change.

⇒ **4.72.2 xargs Description.** *In the second paragraph, replace the second-to-last sentence (*"The utility shall be executed one or more times until the end-of-file is reached."*) with:*

The *utility* shall be executed one or more times until the end-of-file is reached or the logical end-of-file string is found.

Rationale: See 4.72.1 change.

⇒ **4.72.3 xargs Options.** *Add the following option:*

–E *eofstr*	Use *eofstr* as the logical end-of-file string. If –E is not specified, it is unspecified whether the logical end-of-file string is the underscore character (_) or the end-of-file string capability is disabled. When *eofstr* is the null string, the logical end-of-file string capability shall be disabled and underscore characters shall be taken literally.

Rationale: See 4.72.1 change.

Editor's Note: The following rationale will be added to E.4.72, but is kept here with `xargs` *for this draft:*

`xargs` Rationale. *(This subclause is not a part of P1003.2b)*

The −`e` option was omitted from IEEE Std 1003.2-1992 in the belief that the *eofstr* option-argument was recognized only when it was on a line by itself and before quote and escape processing were performed and that the logical end-of-file processing was only enabled if a −`e` option was specified. In that case, a simple `sed` script could be used to duplicate the −`e` functionality. Further investigation revealed that

— The logical end-of-file string was checked for *after* quote and escape processing, making a `sed` script that provided equivalent functionality much more difficult to write

— The default was to perform logical end-of-file processing with an underscore as the logical end-of-file string

To correct this misunderstanding, the −`E` *eofstr* option was adopted from XPG4 {B49} in the first revision of this standard. Users should note that the description of the −`E` option matches historical documentation of the −`e` option (which was not adopted because it did not support the utility syntax guidelines), by saying that if *eofstr* is the null string, logical end-of-file processing is disabled. Historical implementations of `xargs` actually did not disable logical end-of-file processing; they treated a null argument found in the input as a logical end-of-file string. (A null string argument could be generated using single or double quotes (`''` or `""`). Since this behavior was not documented historically, it is considered to be a bug.

Editor's Note: The rationale in E.4.72 will also be modified editorially to remove the now incorrect reference to −`e`*eofstr being replaced by a* `sed` *script (IEEE Std 1003.2-1992 page 970, lines 8986–87).*

⇒ **4.73 `iconv` — Convert file codesets.** *Add the following new clause:*

Rationale: This addition was adopted from XPG4 {B49} to satisfy the following requirement from ISO/IEC 9945-2: 1993 Annex H.1:

(10) A utility (or feature of another utility, such as `tr`) should be provided that converts between character sets encodings based on two charmap files.

4.73 iconv – Convert file codesets

4.73.1 Synopsis

`iconv` **[**`–cs`**] [**`–f` *fromcode***] [**`–t` *tocode***] [***file* . . . **]**

`iconv –l`

4.73.2 Description

The `iconv` utility shall convert the encoding of characters in *file* from one codeset to another and write the results to standard output.

When the options indicate that charmap files are used to specify the codesets (see 4.73.3), the codeset conversion shall be accomplished by performing a logical join on the symbolic character names in the two charmaps. The implementation need not support the use of charmap files for codeset conversion unless the {POSIX2_LOCALEDEF} symbol is defined on the system; see 2.13.2. B B B

4.73.3 Options

The `iconv` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

`–c` Omit any invalid characters from the output. When `–c` is not used, the results of encountering invalid characters in the input stream (either those that are not valid members of the *fromcode* or those that have no corresponding value in *tocode*) shall be specified in the system documentation. The presence or absence of `–c` shall not affect the exit status of `iconv`.

`–f` *fromcode*
Identify the codeset of the input file. If the option-argument contains a slash character, `iconv` shall attempt to use it as the pathname of a charmap file, as defined in 2.4.1. If the pathname does not represent a valid, readable charmap file, the results are undefined. If the option-argument does not contain a slash, it shall be considered the name of one of the codeset descriptions provided by the system, in an unspecified format. The valid values of the option-argument without a slash are implementation defined. If this option is omitted, the codeset of the current locale shall be used.

`–l` Write all supported *fromcode* and *tocode* values to standard output in an unspecified format.

−s Suppress any messages written to standard error concerning invalid characters. When −s is not used, the results of encountering invalid characters in the input stream (either those that are not valid members of the *fromcode* or those that have no corresponding value in *tocode*) shall be specified in the system documentation. The presence or absence of −s shall not affect the exit status of `iconv`.

−t *tocode* Identify the codeset of the output file. The semantics are equivalent to the −f *fromcode* option.

If either −f or −t represents a charmap file, but the other does not (or is omitted), or both −f and −t are omitted, the results are undefined.

4.73.4 Operands

The following operands shall be supported by the implementation:

file A pathname of an input file. If no *file* operands are specified, or if a *file* operand is –, the standard input shall be used.

4.73.5 External Influences

4.73.5.1 Standard Input

The standard input shall be used only if no *file* operands are specified, or if a *file* operand is –. See Input Files.

4.73.5.2 Input Files

The input files shall be text files.

4.73.5.3 Environment Variables

The following environment variables shall affect the execution of `iconv`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files). During translation of the file, this variable

	shall be superseded by the use of the *fromcode* and *tocode* option-arguments.
LC_MESSAGES	This variable shall determine the language in which messages should be written.

4.73.5.4 Asynchronous Events

Default.

4.73.6 External Effects

4.73.6.1 Standard Output

When the −l option is used, the standard output shall contain all supported *fromcode* and *tocode* values, written in an unspecified format.

When the −l option is not used, the standard output shall contain the sequence of characters read from the input file(s), translated to the specified codeset. Nothing else shall be written to the standard output.

4.73.6.2 Standard Error

Used only for diagnostic messages.

4.73.6.3 Output Files

None.

4.73.7 Extended Description

None.

4.73.8 Exit Status

The iconv utility shall exit with one of the following values:

0	All input files were output successfully.
>0	An error occurred.

4.73.9 Consequences of Errors

Default.

4.73.10 Rationale. *(This subclause is not a part of P1003.2b)*

Usage, Examples

The `iconv` utility can be used portably only when the user provides two charmap files as option-arguments. This is because a single charmap provided by the user cannot reliably be joined with the names in a system-provided character set description. The valid values for *fromcode* and *tocode* are implementation defined and do not have to have any relation to the charmap mechanisms. As an aid to interactive users, the `−l` option was adopted from the Plan 9 operating system. It writes information concerning these implementation-defined values. The format is unspecified because there are many possible useful formats that could be chosen, such as a matrix of valid combinations of *fromcode* and *tocode*. The `−l` option is not intended for shell script usage; portable applications will have to use charmaps.

The user must ensure that both charmap files use the same symbolic names for characters the two codesets have in common.

History of Decisions Made

The `iconv` utility is based on one of the same name in XPG4 {B49}. Because of requirements from WG15, the ability to use charmap files for the conversion was added.

Section 5: Revisions to User Portability Utilities Option

5.2 at – Execute commands at a later time

⇒ **5.2.1 at Synopsis.** *Change the first synopsis line to:*

`at` **[**`–m`**] [**`–f` *file*]**[**`–q` *queuename*] `–t` *time_arg*

⇒ **5.2.3 at Options.** *Change the description of the* `–t` *time option to:*

`–t` *time_arg*
: Submit the job to be run at the time specified by the *time_arg* option-argument, which shall have the format as specified by the `touch –t` *time* argument (see 4.63).

Rationale: The two preceding changes satisfy the following corrigendum request from ISO/IEC 9945-2: 1993 Annex H.2:

(8) In 5.2, the `at` utility description is confusing because the same symbol *time* is used for two different values: the `–t` *time* option-argument and one of the *timespec* fields.

⇒ **5.2.3 at Options.** *Add the following sentence to the end of the* `–q` *description:*

If `–q b` is specified along with either of the `–t` *time_arg* or *timespec* arguments, the results are unspecified.

Rationale: This change satisfies the following requirement from ISO/IEC 9945-2: 1993 Annex H.1:

(21) The effects of the combined use of the `at –q b` option and the *timespec* operand should be specified.

⇒ **5.2.6.2 `at` Standard Error.** *Change the beginning of the first sentence from* "The following shall be written to standard error . . . " *to:*

In the POSIX Locale, the following shall be written to standard error . . .

Rationale: This change satisfies the following corrigendum request from ISO/IEC 9945-2: 1993 Annex H.2:

(9) In 5.2.6.2, the `at` message

`"job %s at %s\n"`, *at_job_id*, <*date*>

is in English, but there is no indication of whether it is dependent on the POSIX Locale.

5.3 `batch` – Execute commands at a later time

⇒ **5.3 `batch` <title>.** *Change the clause title to be:*

5.3 `batch` – Schedule commands to be executed in a batch queue

⇒ **5.3.2 `batch` Description.** *Change the first sentence from* "The batch utility shall read commands to be executed at a later time" *to:*

The `batch` utility shall read commands from standard input and schedule them for execution in a batch queue.

Rationale: The preceding two changes satisfy the following requirement from ISO/IEC 9945-2: 1993 Annex H.1:

(22) The title and description of the `batch` utility should be re-examined for their appropriateness and accuracy. Specific reference to "execution in a batch queue" should be included.

5.6 `csplit` – Split files based on context

⇒ **5.6.4 `csplit` Operands.** *Change the descriptions of the* rexp *operands as follows:*

*/rexp/***[***offset***]**
: A file shall be created using the content of the lines from the current line up to, but not including, the line that results from the evaluation of the BRE with *offset*, if any, applied. The BRE *rexp* shall follow the rules described in 2.8.3. The application shall use the sequence `\/` to specify a slash character within the *rexp*. The optional *offset* shall be a positive or negative integer value representing a number of lines. A positive integer value can be preceded by `+`. If the selection of lines from an offset expression of this type would create a file with zero lines, or one with greater than the number of lines left in the input file, the results are unspecified. After the section is created, the current line shall be set to the line that results from the evaluation of the BRE with any offset applied. If the current line is the first line in the file and an RE operation has not yet been performed, the pattern match of *rexp* shall be applied from the current line to the end of the file. Otherwise, the pattern match of *rexp* shall be applied from the line following the current line to the end of the file.

*%rexp%***[***offset***]**
: Equivalent to */rexp/***[***offset***]**, except that no file shall be created for the selected section of the input file. The application shall use the sequence `\%` to specify a percent-sign character within the *rexp*.

Rationale: These `csplit` changes are required to match historical practice and are the result of interpretation request PASC 1003.2-92 #59 submitted for IEEE Std 1003.2-1992.

5.7 `ctags` – Create a tags file

⇒ **5.7.2 `ctags` Description.** *Change the third sentence (*“A locator consists *. . . ”) to:*

A locator consists of a name, pathname, and either a search pattern or a line number that can be used in searching for the object definition.

⇒ **5.7.6.3 `ctags` Output Files.** *Change this subclause to:*

When the `−x` option is not specified, the format of the output file shall be

`"%s\t%s\t/%s/\n"`, <*identifier*>, <*filename*>, <*pattern*>

where <*pattern*> is a search pattern that could be used by an editor to find the defining instance of <*identifier*> in <*filename*> (where “defining instance” is indicated by the declarations listed in 5.7.7).

An optional circumflex (ˆ) can be added as a prefix to <*pattern*>, and an optional dollar sign can be appended to <*pattern*> to indicate that the pattern is anchored to the beginning (end) of a line of text (see 2.8.4.6). Any slash or backslash characters in <*pattern*> shall be preceded by a backslash character. The anchoring circumflex, dollar sign, and escaping backslash characters shall not be considered part of the search pattern. All other characters in the search pattern shall be considered literal characters.

An alternative format is

`"%s\t%s\t?%s?\n"`, <*identifier*>, <*filename*>, <*pattern*>

which is identical to the first format except that slashes in <*pattern*> shall not be preceded by escaping backslash characters, and question mark characters in <*pattern*> shall be preceded by backslash characters.

A second alternative format is

`"%s\t%s\t%d\n"`, <*identifier*>, <*filename*>, <*lineno*>

where <*lineno*> is a decimal line number that could be used by an editor to find <*identifier*> in <*filename*>.

Neither alternative format shall be produced by `ctags` when it is used as described by this standard, but the standard utilities that process tags files shall be able to process those formats as well as the first format.

In any of these formats, the file shall be sorted by identifier, based on the collation sequence in the POSIX Locale.

Rationale: The preceding changes are the result of interpretation request PASC 1003.2-92 #116 submitted for IEEE Std 1003.2-1992. Note related rationale changes in E.5.7.

5.9 du – Estimate file usage

⇒ **5.9.1 du Synopsis.** *Modify the Synopsis to be:*

```
du  [ −a | −s ] [−kx] [ −H | −L ] [file . . . ]
```

⇒ **5.9.2 du Description.** *Add a new sentence in the first paragraph, following the sentence beginning with* "The du utility, by default . . . "

By default, when a symbolic link is encountered on the command line or in the file hierarchy, du shall count the size of the symbolic link (rather than the file referenced by the link), and shall not follow the link to another portion of the file hierarchy.

⇒ **5.9.3 du Options.** *Add the following options in the proper sorted order:*

−H	If a symbolic link is specified on the command line, du shall count the size of the symbolic link and the size of the file or file hierarchy referenced by the link.
−L	If a symbolic link is specified on the command line or encountered during the traversal of a file hierarchy, du shall count the size of the symbolic link and the size of the file or file hierarchy referenced by the link.

⇒ **5.9.3 du Options.** *Add the following paragraph to the end of the subclause:*

Specifying more than one of the mutually exclusive options −H and −L shall not be considered an error. The last option specified shall determine the behavior of the utility.

⇒ **5.10 `ex` – Text editor.** *Replace the entire* `ex` *clause with the following.*

Editor's Note: All of this clause has been changed in Draft 11 from the POSIX.2-1992 version. To avoid clutter, it is not further diffmarked. The rationale in Annex E is also completely replaced.

Rationale: The changes to the `ex` and `vi` clauses are the result of interpretation requests PASC 1003.2-92 #31, 38, 49, 50, 51, 52, 55, 56, 57, 61, 62, 63, 64, 65, and 78, submitted for IEEE Std 1003.2-1992.

5.10 `ex` – Text editor

5.10.1 Synopsis

`ex` **[**`−rR`**] [**`−s` **|** `−v` **] [**`−c` *command***] [**`−t` *tagstring***] [**`−w` *size***] [***file* ... **]**

Obsolescent Version:

`ex` **[**`−rR`**] [** `−` **|** `−v` **] [***+command***] [**`−t` *tagstring***] [**`−w` *size***] [***file* ... **]**

5.10.2 Description

The `ex` utility is a line-oriented text editor. There are two other modes of the editor—open and visual—in which screen-oriented editing is available. This is described more fully by the `ex open` and `visual` commands and in 5.35. The user can switch back and forth between `ex` and the screen-oriented editor modes. All three modes can be further partitioned into two other modes: command mode and text input mode. In command mode, the user is entering commands for the editor to execute; in text input mode, the user is entering text into the edit buffer.

This clause uses the term "edit buffer" to describe the current working text. No specific implementation is implied by this term. All editing changes are performed on the edit buffer, and no changes to it shall affect any file until an editor command writes a file.

Certain terminals do not have all the capabilities necessary to support the complete `ex` definition, such as the full-screen editing commands (open and visual). When these commands cannot be supported on such terminals, this condition shall neither produce an error message such as "not an editor command" nor report a syntax error. The implementation may either accept the commands and produce results on the screen that are the result of an unsuccessful attempt to meet the requirements of this standard or report an error describing the terminal-related deficiency.

5.10.3 Options

The `ex` utility shall conform to the utility argument syntax guidelines described in 2.10.2, except for the obsolescent +*command* and – "options," and that the order of presentation of the +*command* and –`c` options is significant. The following options shall be supported by the implementation:

–`c` *command*
+*command* (Obsolescent.)
: Specify an initial command to be executed in the first edit buffer loaded from an existing file (see 5.10.7.1). Implementations may support more than a single +*command* or –`c` option. In such implementations, the specified commands shall be executed in the order specified on the command line.

–`r`
: Recover the named files (see 5.10.7.1). Recovery information for a file shall be saved during an editor or system crash (e.g., when the editor is terminated by a signal which the editor can catch), or after the use of an `ex` `preserve` command.

A "crash" in this context is an unexpected failure of the system or utility that requires restarting the failed system or utility. A system crash implies that any utilities running at the time also crash. In the case of an editor or system crash, the number of changes to the edit buffer (since the most recent `preserve` command) that will be recovered is unspecified.

If no *file* operands are given and the –`t` option is not specified, all other options, the **EXINIT** variable, and any `.exrc` files shall be ignored; a list of all recoverable files available to the invoking user shall be written, and the editor shall exit normally without further action.

–`R`
: Set the `readonly` edit option.

–`s`
– (Obsolescent.)
: Prepare `ex` for batch use by taking the following actions:

— Suppress writing prompts and informational (but not diagnostic) messages.

— Ignore the value of **TERM** and any implementation default terminal type and assume the terminal is a type incapable of supporting open or visual modes; see 5.10.7.5.19, 5.10.7.5.37, and the description of `vi` in 5.35.

— Suppress the use of the **EXINIT** environment variable (see 5.10.5.3) and the reading of any `.exrc` file (see 5.10.7.1).

— Suppress autoindentation, ignoring the value of the `autoindent` edit option.

`−t` *tagstring*
: Edit the file containing the specified tagstring; see 5.10.7.1 and `ctags` in 5.7. The tags feature represented by `−t` *tagstring* and the `tag` command (see 5.10.7.5.32) is optional. It shall be provided on any system that also provides a conforming implementation of `ctags`; otherwise, the use of `−t` produces undefined results. On any system, it shall be an error to specify more than a single `−t` option. C C

`−v`
: Begin in visual mode (see 5.35).

`−w` *size*
: Set the value of the `window` edit option to *size*.

5.10.4 Operands

The following operand shall be supported by the implementation:

file
: A pathname of a file to be edited.

5.10.5 External Influences

5.10.5.1 Standard Input

The standard input consists of a series of commands and input text, as described in 5.10.7. The implementation may limit each line of standard input to a length of {LINE_MAX}. C C

If the standard input is not a terminal device, it shall be as if the `−s` option had been specified.

If a read from the standard input returns an error, or if the editor detects an end-of-file condition from the standard input, it shall be equivalent to a SIGHUP asynchronous event.

5.10.5.2 Input Files

Input files shall be text files or files that would be text files except for an incomplete last line that is not longer than {LINE_MAX} − 1 B in length and contains no NUL characters. By default, any incomplete last line shall be treated as if it had a trailing `<newline>` character. Other forms of files may optionally be edited by implementations. The `.exrc` files (see 5.10.7.1) and `source` (see 5.10.7.5.30) files shall be text files consisting of `ex` commands.

By default, the editor shall read lines from the files to be edited without interpreting any of those lines as any form of editor command.

5.10.5.3 Environment Variables

The following environment variables shall affect the execution of `ex`:

COLUMNS	This variable shall override the system-selected horizontal screen size. See 2.6 for valid values and results when it is unset or null.
EXINIT	This variable shall be interpreted to contain `ex` commands, executed during startup. See 5.10.7.1 for more details.
HOME	This variable shall be interpreted as a pathname of a directory that shall be searched for an editor startup file named `.exrc`; see 5.10.7.1 for details.
LANG	This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.
LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.
LC_COLLATE	This variable shall determine the locale for character collation information in REs.
LC_CTYPE	This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files), the behavior of character classes within REs, the classification of characters as upper- or lowercase letters, the case conversion of letters, and the detection of word boundaries.
LC_MESSAGES	This variable shall determine the processing of affirmative responses and the language in which messages should be displayed or written.
LINES	This variable shall override the system-selected vertical screen size, and shall set the value of the `window` edit option. See 2.6 for valid values and results when it is unset or null. c c c
PATH	This variable shall determine the search path for the shell command specified in the `ex` editor commands `!`, `shell`, `read`, `write`, and the open and visual mode command `!`; see the description of command search and execution in 3.9.1.1.

SHELL	This variable shall be used as the default value of the `shell` edit option. See 5.10.7.8.18.
TERM	This variable shall be interpreted as the name of the terminal type. If this variable is unset or null, an unspecified default terminal type shall be used.

5.10.5.4 Asynchronous Events

The following symbol is used in this and following subclauses to specify command and asynchronous event actions:

complete write
A complete write is a write of the entire contents of the edit buffer to a file of a type other than a terminal device, or, the saving of the edit buffer caused by the user executing the `ex preserve` command. Writing the contents of the edit buffer to a temporary file that will be removed when the editor exits shall not be considered a complete write. C

The following actions shall be taken upon receipt of signals:

SIGINT If the standard input is not a terminal device, `ex` shall not write the file or return to command or text input mode, and shall exit with a nonzero exit status. C

Otherwise, if executing an open or visual text input mode command, `ex` in receipt of SIGINT shall behave identically to its receipt of the `<ESC>` character. C

Otherwise: C

(1) If executing an `ex` text input mode command, all input lines that have been completely entered shall be resolved into the edit buffer, and any partially entered line shall be discarded. C

(2) If there is a currently executing command, it shall be aborted and a message displayed. Unless otherwise specified by the `ex` or `vi` command descriptions, it is unspecified if any lines modified by the executing command appear modified, or as they were before being modified by the executing command, in the buffer. C

If the currently executing command was a motion command, its associated command shall be discarded. C

(3) If in open or visual command mode, the terminal shall be alerted. C

(4) The editor shall then return to command mode. C

SIGCONT The screen shall be refreshed if in open or visual mode.

SIGHUP	
SIGTERM	If the edit buffer has been modified since the last complete write, ex shall attempt to save the edit buffer so that it can be recovered later using the −r option or the ex recover command. The editor shall not write the file or return to command or text input mode, and shall terminate with a nonzero exit status.

The action taken for all other signals is unspecified.

5.10.6 External Effects

5.10.6.1 Standard Output

The standard output shall be used only for writing prompts to the user, for informational messages, and for writing lines from the edit buffer.

5.10.6.2 Standard Error

Used only for diagnostic messages.

5.10.6.3 Output Files

The output from ex shall be text files.

5.10.7 Extended Description

Only the ex mode of the editor is described in this subclause. See 5.35 for additional editing capabilities available in ex.

When an error occurs, ex shall write a message. If the terminal supports a standout mode (such as inverse video), the message shall be written in standout mode. If the terminal does not support a standout mode, and the edit option errorbells is set, an alert action shall precede the error message.

By default, ex shall start in command mode, which shall be indicated by a “:” prompt (see 5.10.7.8.12). Text input mode can be entered by the append, insert, or change commands; it can be exited (and command mode re-entered) by typing a period (.) alone at the beginning of a line.

5.10.7.1 `ex` and `vi` Initialization

The following symbols are used in this and following clauses to specify locations in the edit buffer.

alternate and current pathnames
: Two pathnames, named *current* and *alternate*, are maintained by the editor. Any `ex` commands that take file names as arguments shall set them as follows:

  (1) If a file argument is specified to the `ex edit`, `ex`, or `recover` commands, or if an `ex tag` command replaces the contents of the edit buffer.

    (a) If the command replaces the contents of the edit buffer, the current pathname shall be set to the *file* argument or the file indicated by the tag, and the alternate pathname shall be set to the previous value of the current pathname.

    (b) Otherwise, the alternate pathname shall be set to the *file* argument.

  (2) If a *file* argument is specified to the `ex next` command:

    (a) If the command replaces the contents of the edit buffer, the current pathname shall be set to the first *file* argument, and the alternate pathname shall be set to the previous value of the current pathname.

  (3) If a *file* argument is specified to the `ex file` command, the current pathname shall be set to the *file* argument, and the alternate pathname shall be set to the previous value of the current pathname.

  (4) If a *file* argument is specified to the `ex read` and `write` commands (i.e., when reading or writing a file, and not to the program named by the `shell` edit option), or a *file* argument is specified to the `ex xit` command:

    (a) If the current pathname has no value, the current pathname shall be set to the *file* argument.

    (b) Otherwise, the alternate pathname shall be set to the *file* argument.

  If the alternate pathname is set to the previous value of the current pathname when the current pathname had no previous value, then the alternate pathname shall have no value as a result.

current line
: The line of the edit buffer referenced by the cursor. Each command description specifies the current line after the command has been executed, as the *Current line* value. When the edit buffer contains no lines, the current line shall be zero; see 5.10.7.2.

current column

The current screen column occupied by the cursor. (The columns shall be numbered beginning at 1.) Each command description specifies the current column after the command has been executed, as the *Current column* value. This column is an "ideal" column that is remembered over the lifetime of the editor. The actual screen column upon which the cursor rests may be different from the current column; see the cursor positioning discussion in `vi` (5.35.7.2).

set to nonblank

A description for a current column value, meaning that the current column shall be set to the last screen column on which is displayed any part of the first nonblank character of the line. If the line has no nonblank characters, the current column shall be set to the last screen column on which is displayed any part of the last character in the line. If the line is empty, the current column shall be set to column position 1.

The length of lines in the edit buffer may be limited to {LINE_MAX} bytes. In open and visual mode, the length of lines in the edit buffer may be limited to the number of characters that will fit in the display. If either limit is exceeded during editing, an error message shall be written. If either limit is exceeded by a line read in from a file, an error message shall be written and the edit session may be terminated.

If the editor stops running due to any reason other than a user command, and the edit buffer has been modified since the last complete write, it shall be equivalent to a SIGHUP asynchronous event. If the system crashes, it shall be equivalent to a SIGHUP asynchronous event.

During initialization (before the first file is copied into the edit buffer or any user commands from the terminal are processed)

(1) If the environment variable **EXINIT** is set, the editor shall execute the `ex` commands contained in that variable.

(2) If the **EXINIT** variable is not set, and all of the following are true:

   (a) The **HOME** environment variable is not null and not empty.

   (b) The file `.exrc` in the directory referred to by the **HOME** environment variable

      [1] exists

      [2] is owned by the same user ID as the real user ID of the process or the process has appropriate privileges

      [3] is not writeable by anyone other than the owner

   the editor shall execute the `ex` commands contained in that file.

(3) If and only if all of the following are true:

(a) The current directory is not referred to by the **HOME** environment variable.

(b) A command in the **EXINIT** environment variable or a command in the `.exrc` file in the directory referred to by the **HOME** environment variable sets the editor option `exrc`.

(c) The `.exrc` file in the current directory

[1] exists

[2] is owned by the same user ID as the real user ID of the process, or by one of a set of implementation defined user IDs

[3] is not writeable by anyone other than the owner

the editor shall attempt to execute the `ex` commands contained in that file.

Lines in any `.exrc` file that contain no characters or only `<blank>` characters shall be ignored. If any `.exrc` file exists, but is not read for ownership or permission reasons, it shall be an error.

After the **EXINIT** variable and any `.exrc` files are processed, the first file specified by the user shall be edited, as follows:

(1) If the user specified the `−t` option, the effect shall be as if the `ex` tag command was entered with the specified argument, with the exception that if tag processing does not result in a file to edit, the effect shall be as described in step (3) below.

(2) Otherwise, if the user specified any command-line *file* arguments, the effect shall be as if the `ex edit` command was entered with the first of those arguments as its *file* argument.

(3) Otherwise, the effect shall be as if the `ex edit` command was entered with a nonexistent file name as its file argument. It is unspecified if this action shall set the current pathname. In an implementation where this action does not set the current pathname, any editor command using the current pathname shall fail until an editor command sets the current pathname.

If the `−r` option was specified, the first time a file in the initial argument list or a file specified by the `−t` option is edited, if recovery information has previously been saved about it, that information shall be recovered and the editor shall behave as if the contents of the edit buffer have already been modified. If there are multiple instances of the file to be recovered, the one most recently saved shall be recovered, and an informational message that there are previous versions of the file that can be recovered shall be written. If no recovery information about a file is available, an informational message to this effect shall be written, and the edit shall proceed as usual.

If the −c option was specified, the first time a file that already exists (including a file that might not exist but for which recovery information is available, when the −r option is specified) replaces or initializes the contents of the edit buffer, the current line shall be set to the last line of the edit buffer, the current column shall be set to nonblank, and the ex commands specified with the −c option shall be executed. In this case, the current line and current column shall not be set as described for the command associated with the replacement or initialization of the edit buffer contents. However, if the −t option or a tag command is associated with this action, the −c option commands shall be executed and then the movement to the tag shall be performed.

The current argument list shall initially be set to the file names specified by the user on the command line. If no file names are specified by the user, the current argument list shall be empty. If the −t option was specified, it is unspecified if any file name resulting from tag processing shall be prepended to the current argument list. In the case where the file name is added as a prefix to the current argument list, the current argument list reference shall be set to that file name. In the case where the file name is not added as a prefix to the current argument list, the current argument list reference shall logically be located before the first of the file names specified on the command line (e.g., a subsequent ex next command shall edit the first file name from the command line). If the −t option was not specified, the current argument list reference shall be to the first of the file names on the command line.

5.10.7.2 Addressing

Addressing in ex relates to the current line and the current column; the address of a line is its 1-based line number, the address of a column is its 1-based count from the beginning of the line. Generally, the current line is the last line affected by a command. The current line number is the address of the current line. In each command description, the effect of the command on the current line number and the current column is described.

Addresses are constructed as follows:

(1) The character . (period) shall address the current line.

(2) The character $ shall address the last line of the edit buffer.

(3) The positive decimal number *n* shall address the *n*-th line of the edit buffer.

(4) '*x* shall address the line marked with the mark name character *x*, which shall be a lowercase letter from the portable character set or one of the characters ‘ or '. It shall be an error if the line that was marked is not currently present in the edit buffer or the mark has not been set. Lines can be marked with the ex mark or k commands, or the vi m command.

(5) An RE enclosed by slashes (/) shall address the first line found by searching forwards from the line following the current line toward the end of the edit buffer and stopping at the first line containing a string matching the RE. [As stated in 5.10.7.6, an address consisting of a null RE delimited by slashes (//) shall address the next line containing the last RE encountered.] In addition, the second slash can be omitted at the end of a command line. If the `wrapscan` edit option is set, the search shall wrap around to the beginning of the edit buffer and continue up to and including the current line, so that the entire edit buffer is searched. Within the RE, the sequence \/ shall represent a literal slash instead of the RE delimiter.

(6) An RE enclosed in question marks (?) shall address the first line found by searching backwards from the line preceding the current line toward the beginning of the edit buffer and stopping at the first line containing a string matching the RE. The second question mark can be omitted at the end of a command line. If the `wrapscan` edit option is set, the search shall wrap around from the beginning of the edit buffer to the end of the edit buffer and continue up to and including the current line, so that the entire edit buffer is searched. Within the RE, the sequence \? shall represent a literal question mark instead of the RE delimiter.

(7) A + or - immediately followed by a decimal number shall address the current line plus or minus the number. A + or - not followed by a decimal number shall address the current line plus or minus 1.

Addresses can be followed by zero or more address offsets, optionally `<blank>` separated. Address offsets are constructed as follows:

(1) A + or - immediately followed by a decimal number shall add (subtract) the indicated number of lines to (from) the address. A + or - not followed by a decimal number shall add (subtract) 1 to (from) the address.

(2) A decimal number shall add the indicated number of lines to the address.

It shall not be an error for an intermediate address value to be less than zero or greater than the last line in the edit buffer. It shall be an error for the final address value to be less than zero or greater than the last line in the edit buffer.

Commands take zero, one, or two addresses; see the descriptions of *1addr* and *2addr* in 5.10.7.5. If more than the required number of addresses are provided to a command that requires zero addresses, it shall be an error. Otherwise, if more than the required number of addresses are provided to a command, the addresses specified first shall be evaluated and then discarded until the maximum number of valid addresses remain.

Addresses shall be separated from each other by a comma (,) or a semicolon (;). If no address is specified before or after a comma or semicolon separator, it shall be as if the address of the current line was specified before or after the separator. In the case of a semicolon separator, the current line (.) shall be set to the first address, and only then will the next address be calculated. This feature can be used to determine the starting line for forwards and backwards searches [see rules (5) and (6)].

A percent sign (%) shall be equivalent to entering the two addresses `1,$`.

Any delimiting `<blank>` characters between addresses, address separators, or address offsets shall be discarded.

5.10.7.3 ex Command-Line Parsing

The following symbol is used in this and following subclauses to describe parsing behavior:

escape
: If a character is referred to as "backslash escaped" or "`<control-V>` escaped," it shall mean that the character acquired or lost a special meaning by virtue of being preceded, respectively, by a backslash or `<control-V>` character. Unless otherwise specified, the escaping character shall be discarded at that time and shall not be further considered for any purpose.

Command-line parsing shall be done in the following steps. For each step, characters already evaluated shall be ignored; i.e., the phrase "leading character" refers to the next character that has not yet been evaluated.

(1) Leading colon characters shall be skipped.

(2) Leading `<blank>` characters shall be skipped.

(3) If the leading character is a double-quote character, the characters up to and including the next non-backslash-escaped `<newline>` character shall be discarded, and any subsequent characters shall be parsed as a separate command.

(4) Leading characters that can be interpreted as addresses shall be evaluated; see 5.10.7.2.

(5) Leading `<blank>` characters shall be skipped.

(6) If the next character is a vertical-line character or a `<newline>` character:

  (a) If the next character is a `<newline>` character:

    [1] If ex is in open or visual mode, the current line shall be set to the last address specified, if any.

    [2] Otherwise, if the last command was terminated by a vertical-line character, no action shall be taken; e.g., the command "`||<newline>`" shall execute two implied commands, not three.

    [3] Otherwise, step (6b) shall apply.

  (b) Otherwise, the implied command shall be the `print` command. The last #, `p`, and `l` flags specified to any ex command shall be remembered and shall apply to this implied command. Executing the `ex` `number`, `print`, or `list` command shall set the remembered flags to #, nothing, and `l`, respectively, plus any other flags specified for that execution of the `number`, `print`, or `list` command.

If `ex` is not currently performing a `global` or `v` command, and no address or count is specified, the current line shall be incremented by 1 before the command is executed. If incrementing the current line would result in an address past the last line in the edit buffer, the command shall fail, and the increment shall not happen.

(c) The `<newline>` or vertical-line character shall be discarded and any subsequent characters shall be parsed as a separate command.

(7) The command name shall be comprised of the next character, (if the character is not alphabetic) or the next character and and any subsequent alphabetic characters (if the character is alphabetic), with the following exceptions:

(a) Commands that consist of any prefix of the characters in the command name `delete`, followed immediately by any of the characters `l`, `p`, `+`, `-`, or # shall be interpreted as a `delete` command, followed by a `<blank>` character, followed by the characters that were not part of the prefix of the `delete` command. The maximum number of characters shall be matched to the command name `delete`; e.g., “`del`” shall not be treated as “`de`” followed by the flag `l`.

(b) Commands that consist of the character `k`, followed by a character that can be used as the name of a mark, shall be equivalent to the `mark` command followed by a `<blank>` character, followed by the character that followed the `k`.

(c) Commands that consist of the character `s`, followed by character(s) that could be interpreted as valid options to the `s` command, shall be the equivalent of the `s` command, without any pattern or replacement values, followed by a `<blank>` character, followed by the characters after the `s`.

(8) The command name shall be matched against the possible command names, and a command name that contains a prefix matching the characters specified by the user shall be the executed command. In the case of commands where the characters specified by the user could be ambiguous, the executed command shall be as follows:

`a`	append	`n`	next	`t`	t
`c`	change	`p`	print	`u`	undo
`ch`	change	`pr`	print	`un`	undo
`e`	edit	`r`	read	`v`	v

`m`	move	`re`	read	`w`	write
`ma`	mark	`s`	s		

Implementation extensions with names causing similar ambiguities shall not be checked for a match until all possible matches for commands specified by this standard have been checked.

(9) If the command is a `!` command, or if the command is a `read` command followed by zero or more `<blank>` characters and a `!`, or if the command is a `write` command followed by one or more `<blank>` characters and a `!`, the rest of the command shall include all characters up to a non-backslash-escaped `<newline>`. The `<newline>` shall be discarded and any subsequent characters shall be parsed as a separate `ex` command.

(10) Otherwise, if the command is an `edit`, `ex` or `next` command, or a `visual` command while in open or visual mode, the next part of the command shall be parsed as follows:

(a) Any `!` character immediately following the command shall be skipped and be part of the command.

(b) Any leading `<blank>` characters shall be skipped and be part of the command.

(c) If the next character is a `+`, characters up to the first non-backslash-escaped `<newline>` or non-backslash-escaped `<blank>` shall be skipped and be part of the command.

(d) The rest of the command shall be determined by the steps specified in paragraph 12.

(11) Otherwise, if the command is a `global`, `open`, `s`, or `v` command, the next part of the command shall be parsed as follows:

(a) Any leading `<blank>` characters shall be skipped and be part of the command.

(b) If the next character is not an alphanumeric, double-quote, `<newline>`, backslash, or vertical-line character:

[1] The next character shall be used as a command delimiter.

[2] If the command is a `global`, `open`, or `v` command, characters up to the first non-backslash-escaped `<newline>` character, or first non-backslash-escaped delimiter character, shall be skipped and be part of the command.

[3] If the command is an `s` command, characters up to the first non-backslash-escaped `<newline>` character, or second non-backslash-escaped delimiter character, shall be skipped and be part of the command.

(c) If the command is a `global` or `v` command, characters up to the first non-backslash-escaped `<newline>` character shall be skipped and be part of the command.

(d) Otherwise, the rest of the command shall be determined by the steps specified in paragraph 12.

(12) Otherwise:

(a) If the command was a map, unmap, abbreviate, or unabbreviate command, characters up to the first non-<control-V>-escaped <newline>, vertical-line, or double-quote character shall be skipped and be part of the command.

(b) Otherwise, characters up to the first non-backslash-escaped <newline>, vertical-line, or double-quote character shall be skipped and be part of the command.

(c) If the command was an append, change, or insert command, and the step (12b) ended at a vertical-line character, any subsequent characters, up to the next non-backslash-escaped <newline> character shall be used as input text to the command.

(d) If the command was ended by a double-quote character, all subsequent characters, up to the next non-backslash-escaped <newline> character shall be discarded.

(e) The terminating <newline> or vertical-line character shall be discarded and any subsequent characters shall be parsed as a separate ex command.

Command arguments shall be parsed as described by the synopsis and description of each individual ex command. This parsing shall not be <blank>-sensitive, except for the ! argument, which must follow the command name without intervening <blank> characters, and where it would otherwise be ambiguous. For example, *count* and *flag* arguments need not be <blank>-separated because "d22p" is not ambiguous, but *file* arguments to the ex next command must be separated by one or more <blank> characters. Any <blank> character in command arguments for the abbreviate, unabbreviate, map, and unmap commands can be <control-V>-escaped, in which case the <blank> character shall not be used as an argument delimiter. Any <blank> character in the command argument for any other command can be backslash-escaped, in which case that <blank> character shall not be used as an argument delimiter.

Within command arguments for the abbreviate, unabbreviate, map, and unmap commands, any character can be <control-V>-escaped. All such escaped characters shall be treated literally and shall have no special meaning. Within command arguments for all other ex commands that are not REs or replacement strings, any character that would otherwise have a special meaning can be backslash escaped. Escaped characters shall be treated literally, without special meaning as shell expansion characters or !, %, and # expansion characters. See 5.10.7.6 and 5.10.7.7 for descriptions of command arguments that are REs or replacement strings.

Non-backslash-escaped % characters appearing in *file* arguments to any ex command shall be replaced by the current pathname; unescaped # characters shall be replaced by the alternate pathname. It shall be an error if % or # characters appear unescaped in an argument and their corresponding values are not set.

Non-backslash-escaped ! characters in the arguments to either the `ex` ! command or the open and visual mode ! command, or in the arguments to the `ex` `read` command, where the first non-`<blank>` character after the command name is a ! character, or in the arguments to the `ex` `write` command where the command name is followed by one or more `<blank>` characters and the first non-`<blank>` character after the command name is a ! character, shall be replaced with the arguments to the last of those three commands as they appeared after all unescaped %, #, and ! characters were replaced. It shall be an error if ! characters appear unescaped in one of these commands and there has been no previous execution of one of these commands.

If an error occurs during the parsing or execution of an `ex` command:

— An informational message to this effect shall be written. Execution of the `ex` command shall stop, and the cursor (e.g., the current line and column) shall not be further modified.

— If the `ex` command resulted from a map expansion, all characters from that map expansion shall be discarded, except as otherwise specified by the `map` command (see 5.10.7.5.14).

— Otherwise, if the `ex` command resulted from the processing of an **EXINIT** environment variable, a `.exrc` file, a `:source` command, a `–c` option, or a *+command* specified to an `ex` `edit`, `ex`, `next`, or `visual` command, no further commands from the source of the commands shall be executed.

— Otherwise, if the `ex` command resulted from the execution of a buffer or a `global` or `v` command, no further commands caused by the execution of the buffer or the `global` or `v` command shall be executed.

— Otherwise, if the `ex` command was not terminated by a `<newline>` character, all characters up to and including the next non-backslash-escaped `<newline>` shall be discarded.

5.10.7.4 `ex` Input Editing

The following symbols are used in this and following clauses to specify command actions.

word In the POSIX Locale, a word consists of a maximal sequence of letters, digits, and underscores, delimited at both ends by characters other than letters, digits, or underscores, or by the beginning or end of a line or the edit buffer.

When accepting input characters from the user, in either `ex` command mode or `ex` text input mode, `ex` shall enable canonical mode input processing, as defined in POSIX.1 {8}.

If in `ex` text input mode:

(1) If the `number` edit option is set, `ex` shall prompt for input using the line number that would be assigned to the line if it is entered, in the format specified for the `ex` `number` command.

(2) If the `autoindent` edit option is set, `ex` shall prompt for input using autoindent characters, as described by the `autoindent` edit option. Autoindent characters shall follow the line number, if any.

If in `ex` command mode:

(1) If the `prompt` edit option is set, input shall be prompted for using a single `:` character; otherwise, there shall be no prompt.

The input characters in the following subclauses shall have the following effects on the input line.

5.10.7.4.1 *eof*

Synopsis: *eof*

See the description of the `stty` *eof* character in 4.59.

If in `ex` command mode:

If the *eof* character is the first character entered on the line, the line shall be evaluated as if it contained two characters: a `<control-D>` and a `<newline>` character.

Otherwise, the *eof* character shall have no special meaning.

If in `ex` text input mode:

If the cursor follows an `autoindent` character, the `autoindent` characters in the line shall be modified so that a part of the next text input character will be displayed on the first column in the line after the previous `shiftwidth` edit option column boundary, and the user shall be prompted again for input for the same line.

Otherwise, if the cursor follows a `0`, which follows an `autoindent` character, and the `0` was the previous text input character, the `0` and all `autoindent` characters in the line shall be discarded, and the user shall be prompted again for input for the same line.

Otherwise, if the cursor follows a ^, which follows an `autoindent` character, and the ^ was the previous text input character, the ^ and all `autoindent` characters in the line shall be discarded, and the user shall be prompted again for input for the same line. In addition, the `autoindent` level for the next input line shall be derived from the same line from which the `autoindent` level for the current input line was derived.

Otherwise, if there are no `autoindent` or text input characters in the line, the *eof* character shall be discarded.

Otherwise, the *eof* character shall have no special meaning.

5.10.7.4.2 `<newline>`

Synopsis: `<newline>`
Synopsis: `<control-J>`

If in `ex` command mode:

Cause the command line to be parsed; `<control-J>` shall be mapped to the `<newline>` character for this purpose.

If in `ex` text input mode:

Terminate the current line. If there are no characters other than autoindent characters on the line, all characters on the line shall be discarded. Prompt for text input on a new line after the current line. If the `autoindent` edit option is set, an appropriate number of autoindent characters shall be added as a prefix to the line as described by the `ex` `autoindent` edit option.

5.10.7.4.3 `<backslash>`

Synopsis: `<backslash>`

Allow the entry of a subsequent `<newline>` or `<control-J>` as a literal character, removing any special meaning that it may have to the editor during text input mode. The backslash character shall be retained and evaluated when the command line is parsed, or retained and included when the input text becomes part of the edit buffer.

5.10.7.4.4 `<control-V>`

Synopsis: `<control-V>`

Allow the entry of any subsequent character as a literal character, removing any special meaning that it may have to the editor during text input mode. The `<control-V>` character shall be discarded before the command line is parsed or the input text becomes part of the edit buffer.

If the “literal next” functionality is performed by the underlying system, it is implementation defined if a character other than `<control-V>` performs this function.

5.10.7.4.5 `<control-W>`

Synopsis: `<control-W>`

Discard the `<control-W>`, and the word previous to it in the input line, including any `<blank>` characters following the word and preceding the `<control-W>`.

If the “word erase” functionality is performed by the underlying system, it is implementation-defined if a character other than `<control-W>` performs this function.

5.10.7.5 `ex` Command Descriptions

The following symbols are used in this subclause to represent command modifiers. Some of these modifiers can be omitted, in which case the specified defaults shall be used.

1addr A single address, given in any of the forms described in 5.10.7.2; the default shall be the current line (`.`), unless otherwise specified.

If the line address is zero, it shall be an error, unless otherwise specified in the following command descriptions.

If the edit buffer is empty, and the address is specified with a command other than =, `append`, `insert`, `open`, `put`, `read`, or `visual`, or the address is not zero, it shall be an error. C

2addr Two addresses specifying an inclusive range of lines. If no addresses are specified, the default for *2addr* shall be the current line only (`.,.`), unless otherwise specified in the following command descriptions. If one address is specified, *2addr* shall specify that line only, unless otherwise specified in the following command descriptions.

It shall be an error if the first address is greater than the second address.

If the edit buffer is empty, and the two addresses are specified with a command other than the `!`, `write`, `wq`, or `xit` commands, or either address is not zero, it shall be an error.

count A positive decimal number. If *count* is specified, it shall be equivalent to specifying an additional address to the command, unless otherwise specified by the following command descriptions. The additional address shall be equal to the last address specified to the command (either explicitly or by default) plus *count* – 1.

If this would result in an address greater than the last line of the edit buffer, it shall be corrected to equal the last line of the edit buffer.

flags One or more of the characters +, –, #, `p`, or `l` (ell). The *flag* characters can be `<blank>`-separated, and in any order or combination.

The characters #, `p`, and `l` shall cause line(s) to be written in the format specified by the `print` command with the specified flags. C C

The line(s) to be written are as follows:

(1) All edit buffer lines written during the execution of the `ex` &, `~`, `list`, `number`, `open`, `print`, `s`, `visual`, and `z` commands shall be written as specified by any flags.

(2) After the completion of an `ex` command with a flag as an argument, the current line shall be written as specified by the flag(s), unless the current line was the last line written by the command.

The characters + and − cause the value of the current line after the execution of the `ex` command to be adjusted by the offset address as described in section 5.10.7.2. This adjustment shall occur before the current line is written as described in (2) above.

The default for *flags* shall be none.

buffer One of a number of named areas for holding text. The named buffers are specified by the alphanumeric characters of the POSIX Locale. There shall also be one "unnamed" buffer. When no buffer is specified for editor commands that use a buffer, the unnamed buffer shall be used. Commands that store text into buffers shall store the text as it was before the command took effect, and shall store text occurring earlier in the file before text occurring later in the file, regardless of how the text region was specified. Commands that store text into buffers shall store the text into the unnamed buffer as well as any specified buffer.

In `ex` commands, buffer names are specified as the name by itself. In open or visual mode commands the name is preceded by a double quote (`"`) character.

If the specified buffer name is an uppercase character, and the buffer contents are to be modified, the buffer shall be appended to rather than being overwritten. If the buffer is not being modified, specifying the buffer name in lowercase and uppercase shall have identical results.

There shall also be buffers named by the numbers 1 through 9. In open and visual mode, if a region of text including characters from more than a single line is being modified by the `vi` `c` or `d` commands, the motion character associated with the `c` or `d` commands specifies that the buffer text shall be in line mode, or the commands `%`, `'`, `/`, `?`, `(`, `)`, `N`, `n`, `{`, or `}` are used to define a region of text for the `c` or `d` commands, the contents of buffers 1 through 8 shall be moved into the buffer named by the next numerically greater value, the contents of buffer 9 shall be discarded, and the region of text shall be copied into buffer 1. This shall be in addition to copying the text into a user-specified buffer or unnamed buffer, or both. Numeric buffers can be specified as a source buffer for open and visual mode commands; however, specifying a numeric buffer as the write target of an open or visual mode command shall have unspecified results.

The text of each buffer shall have the characteristic of being in either line or character mode. Appending text to a nonempty buffer shall set the mode to match the characteristic of the text being appended. Appending text to a buffer shall cause the creation of at least one additional line in the buffer. All text stored into buffers by `ex` commands shall be in line mode. The `ex` commands that use buffers as the source of text specify individually how buffers of different modes are handled. Each open or visual mode command that uses buffers for any purpose specifies individually the mode of the text stored into the buffer and how buffers of different modes are handled.

file Command text used to derive a pathname. The default shall be the current pathname, as defined previously, in which case, if no current pathname has yet been established it shall be an error, except where specifically noted in the individual command descriptions that follow. If the command text contains any of the characters `~`, `{`, `[`, `*`, `?`, `$`, `‘`, `’`, `"`, and `\`, it shall be subjected to the process of “shell expansions,” as described below; if more than a single pathname results and the command expects only one, it shall be an error.

The process of shell expansions in the editor shall be done as follows. The `ex` utility shall pass two arguments to the program named by the `shell` edit option; the first shall be `−c`, and the second shall be the string “`echo` ” and the command text as a single argument. The standard output and standard error of that command shall replace the command text.

! A character that can be appended to the command name to modify its operation, as detailed in the individual command descriptions. With the exception of the `ex` `read`, `write`, and `!` commands, the `!` character shall only act as a modifier if there are no `<blank>` characters between it and the command name.

remembered search direction
The `vi` commands `N` and `n` begin searching in a forwards or backwards direction in the edit buffer based on a remembered search direction, which is initially unset, and is set by the `ex` `global`, `v`, `s`, and `tag` commands, and the `vi` `/` and `?` commands.

5.10.7.5.1 abbreviate

Synopsis: `ab`**[**`breviate`**] [***lhs rhs***]**

If *lhs* and *rhs* are not specified, write the current list of abbreviations and do nothing more.

Implementations may restrict the set of characters accepted in *lhs* or *rhs*, except that printable characters and `<blank>`s shall not be restricted. Additional restrictions shall be implementation defined.

In both *lhs* and *rhs*, any character may be escaped with a `<control-V>`, in which case the character shall not be used to delimit *lhs* from *rhs*, and the escaping `<control-V>` shall be discarded.

In open and visual text input mode, if a nonword or `<ESC>` character that is not escaped by a `<control-V>` character is entered after a word character, a check shall be made for a set of characters matching *lhs*, in the text input entered during this command. If it is found, the effect shall be as if *rhs* was entered instead of *lhs*.

The set of characters that are checked is defined as follows:

(1) If there are no characters inserted before the word and nonword or <ESC> characters that triggered the check, the set of characters shall consist of the word character.

(2) If the character inserted before the word and nonword or <ESC> characters that triggered the check is a word character, the set of characters shall consist of the characters inserted immediately before the triggering character(s) that are word characters, plus the triggering word character.

(3) If the character inserted before the word and nonword or <ESC> characters that triggered the check is not a word character, the set of characters shall consist of the characters that were inserted before the triggering character(s) that are neither <blank>s nor word characters, plus the triggering word character.

It is unspecified if the *lhs* argument entered for the ex abbreviate and unabbreviate commands is replaced in this fashion. Regardless of whether or not the replacement occurs, the effect of the command shall be as if the replacement had not occurred.

Current line: Unchanged.

Current column: Unchanged.

5.10.7.5.2 append

Synopsis: **[***1addr***]** a**[**ppend**][**!**]**

Enter ex text input mode; the input text shall be placed after the specified line. If the line is zero, the text shall be placed at the beginning of the edit buffer.

This command shall be affected by the number and autoindent edit options; following the command name with ! shall cause the autoindent edit option setting to be toggled for the duration of this command only.

Current line: Set to the last input line; if no lines were input, set to the specified line, or to the first line of the edit buffer if a line of zero was specified, or zero if the edit buffer is empty.

Current column: Set to nonblank.

5.10.7.5.3 args

Synopsis: ar**[**gs**]**

Write the current argument list, with the current argument-list entry, if any, between [and] characters.

Current line: Unchanged.

Current column: Unchanged.

5.10.7.5.4 change

Synopsis: **[***2addr***]** `c`**[**`hange`**][**`!`**]** **[***count***]**

Enter `ex` text input mode; the input text shall replace the specified lines. The specified lines shall be copied into the unnamed buffer, which shall become a line mode buffer.

This command shall be affected by the `number` and `autoindent` edit options; following the command name with `!` shall cause the `autoindent` edit option setting to be toggled for the duration of this command only.

Current line: Set to the last input line; if no lines were input, set to the line before the first address, or to the first line of the edit buffer if there are no lines preceding the first address, or to zero if the edit buffer is empty.

Current column: Set to nonblank.

5.10.7.5.5 chdir

Synopsis: `chd`**[**`ir`**][**`!`**]** **[***file***]**
Synopsis: `cd`**[**`!`**]** **[***file***]**

Change the current working directory to *file*.

If no *file* argument is specified, and the **HOME** environment variable is set to a nonnull and nonempty value, *file* shall default to the value named in the **HOME** environment variable. If the **HOME** environment variable is empty or is undefined, the default value of *file* is implementation defined.

If no `!` is appended to the command name, and the edit buffer has been modified since the last complete write, and the current pathname does not begin with a `/`, it shall be an error.

Current line: Unchanged.

Current column: Unchanged.

5.10.7.5.6 copy

Synopsis: **[***2addr***]** `co`**[**`py`**]** *1addr* **[***flags***]**
Synopsis: **[***2addr***]** `t` *1addr* **[***flags***]**

Copy the specified lines after the specified destination line; line zero specifies that the lines shall be placed at the beginning of the edit buffer.

Current line: Set to the last line copied.

Current column: Set to nonblank.

5.10.7.5.7 delete

Synopsis: **[***2addr***]** `d`**[**`elete`**]** **[***buffer***]** **[***count***]** **[***flags***]**

Delete the specified lines into a buffer (defaulting to the unnamed buffer), which shall become a line-mode buffer.

Flags can immediately follow the command name; see 5.10.7.3.

Current line: Set to the line following the deleted lines, or to the last line in the edit buffer if that line is past the end of the edit buffer, or to zero if the edit buffer is empty.

Current column: Set to nonblank.

5.10.7.5.8 edit

Synopsis: `e`**[**`dit`**][**`!`**]** **[**+*command***]** **[***file***]**
Synopsis: `ex`**[**`!`**]** **[**+*command***]** **[***file***]**

If no ! is appended to the command name, and the edit buffer has been modified since the last complete write, it shall be an error.

If *file* is specified, replace the current contents of the edit buffer with the current contents of file, and set the current pathname to *file*. If *file* is not specified, replace the current contents of the edit buffer with the current contents of the file named by the current pathname. If for any reason the current contents of the file cannot be accessed, the edit buffer shall be empty.

The +*command* option shall be `<blank>`-delimited; `<blank>` characters within +*command* can be escaped by preceding them with a backslash character. The +*command* shall be interpreted as an `ex` command immediately after the contents of the edit buffer have been replaced and the current line and column have been set. C

If the edit buffer is empty:

Current line:
Set to 0.

Current column:
Set to 1.

Otherwise, if executed while in `ex` command mode or if the +*command* argument is specified:

Current line:
Set to the last line of the edit buffer.

Current column:
Set to nonblank.

Otherwise, if *file* is omitted or results in the current pathname:

Current line:
Set to the first line of the edit buffer.

Current column:
Set to nonblank.

Otherwise, if *file* is the same as the last file edited, the line and column shall be set as follows; if the file was previously edited, the line and column may be set as follows:

Current line:
Set to the last value held when that file was last edited. If this value is not a valid line in the new edit buffer, set to the first line of the edit buffer.

Current column:
If the current line was set to the last value held when the file was last edited, set to the last value held when the file was last edited. Otherwise, or if the last value is not a valid column in the new edit buffer, set to non-blank.

Otherwise:

Current line:
Set to the first line of the edit buffer.

Current column:
Set to nonblank.

5.10.7.5.9 `file`

Synopsis: `f`**[**`ile`**]** **[***file***]**

If a *file* argument is specified, the alternate pathname shall be set to the current pathname, and the current pathname shall be set to *file*.

Write an informational message. If the file has a current pathname, it shall be included in this message; otherwise, the message shall indicate that there is no current pathname. If the edit buffer contains lines, the current line number and the number of lines in the edit buffer shall be included in this message; otherwise, the message shall indicate that the edit buffer is empty. If the edit buffer has been modified since the last complete write, this fact shall be included in this message. If the `readonly` edit option is set, this fact shall be included in this message. The message may contain other unspecified information.

Current line: Unchanged.

Current column: Unchanged.

5.10.7.5.10 `global`

Synopsis: **[***2addr***]** `g`**[**`lobal`**][**`!`**]** `/`**[[***pattern***]**`/` **[***commands***]]** C

Synopsis: **[***2addr***]** `v /`**[[***pattern***]**`/` **[***commands***]]** C

The optional ! character after the `global` command shall be the same as executing the `v` command.

If *pattern* is empty (e.g., `//`) or not specified, the last RE used in the editor command shall be used as the pattern. The pattern can be delimited by slashes (shown in the Synopsis line), as well as any nonalphanumeric or non-`<blank>` character other than backslash, vertical line, double quote, or `<newline>`. C

If no lines are specified, the lines shall default to the entire file.

The `global` and `v` commands are logically two-pass operations. First, mark the lines within the specified lines that match (`global`) or do not match (`v` or `global!`) the specified pattern. Second, execute the `ex` command(s) given by *commands*, with the current line (`.`) set to each marked line. If an error occurs during this process, or the contents of the edit buffer are replaced (e.g., by the `ex :edit` command) an error message shall be written and no more commands resulting from the execution of this command shall be processed. C C

Multiple `ex` commands can be specified by entering multiple commands on a single line using a vertical line to delimit them, or one per line, by escaping each `<newline>` with a backslash.

If no commands are specified:

(1) If in `ex` command mode, it shall be as if the `print` command were specified.

(2) Otherwise, no command shall be executed.

For the `append`, `change`, and `insert` commands, the input text shall be included as part of the command, and the terminating period can be omitted if the command ends the list of commands. The `open` and `visual` commands can be specified as one of the commands, in which case each marked line shall cause the editor to enter open or visual mode. If open or visual mode is exited using the `vi Q` command, the current line shall be set to the next marked line, and open or visual mode reentered, until the list of marked lines is exhausted.

The `global`, `v`, and `undo` commands cannot be used in *commands*. Marked lines may be deleted by commands executed for lines occurring earlier in the file than the marked lines. In this case, no commands shall be executed for the deleted lines.

If the remembered search direction is not set, the `global` and `v` commands shall set it to forward. C C

The `autoprint` and `autoindent` edit options shall be inhibited for the duration of the `g` or `v` command. C

Current line:

If no commands executed, set to the last marked line. Otherwise, as specified for the executed `ex` commands.

Current column:

If no commands are executed, set to nonblank; otherwise, as specified for the individual `ex` commands.

5.10.7.5.11 **insert**

Synopsis: **[***1addr***]** `i`**[**`nsert`**][**`!`**]**

Enter `ex` text input mode; the input text shall be placed before the specified line. If the line is zero or 1, the text shall be placed at the beginning of the edit buffer.

This command shall be affected by the `number` and `autoindent` edit options; following the command name with `!` shall cause the `autoindent` edit option setting to be toggled for the duration of this command only.

Current line: Set to the last input line; if no lines were input, set to the line before the specified line, or to the first line of the edit buffer if there are no lines preceding the specified line, or zero if the edit buffer is empty.

Current column: Set to nonblank.

5.10.7.5.12 **join**

Synopsis: **[***2addr***]** `j`**[**`oin`**][**`!`**]** **[***count***]** **[***flags***]**

If *count* is specified:

If no address was specified, the `join` command shall behave as if *2addr* were the current line and the current line plus *count* (`.,.` + *count*).

If one address was specified, the `join` command shall behave as if *2addr* were the specified address and the specified address plus *count* (*addr*,*addr* + *count*).

If two addresses were specified, the `join` command shall behave as if an additional address, equal to the last address plus *count* − 1 (*addr1*,*addr2*,*addr2* + *count* − 1), was specified.

If this would result in a second address greater than the last line of the edit buffer, it shall be corrected to be equal to the last line of the edit buffer.

If no *count* is specified:

If no address was specified, the `join` command shall behave as if *2addr* were the current line and the next line (`.,.+1`).

If one address was specified, the `join` command shall behave as if *2addr* were the specified address and the next line (*addr*,*addr* + 1).

Join the text from the specified lines into a single line, which shall replace the specified lines.

If a ! character is appended to the command name, the join shall be without modification of any line, independent of the current locale.

Otherwise, in the POSIX Locale, set the current line to the first of the specified lines, and then, for each subsequent line, proceed as follows:

(1) Discard leading spaces from the line to be joined.

(2) If the line to be joined is now empty, delete it, and skip steps (3) through (5).

(3) If the current line ends in a `<blank>` character, or the first character of the line to be joined is a) character, join the lines without further modification.

(4) If the last character of the current line is a ., join the lines with two `<space>` characters between them.

(5) Otherwise, join the lines with a single `<space>` character between them.

Current line: Set to the first line specified.

Current column: Set to nonblank.

5.10.7.5.13 list

Synopsis: **[***2addr***]** `l`**[**`ist`**]** **[***count***]** **[***flags***]**

This command shall be equivalent to the `ex` command:

[*2addr***]** `p`**[**`rint`**]** **[***count***]** `l`**[***flags***]**

See 5.10.7.5.21.

5.10.7.5.14 map

Synopsis: `map`**[**`!`**]** **[***lhs rhs***]**

If *lhs* and *rhs* are not specified:

(1) If ! is specified, write the current list of text input mode maps.

(2) Otherwise, write the current list of command mode maps.

(3) Do nothing more.

Implementations may restrict the set of characters accepted in *lhs* or *rhs*, except that printable characters and `<blank>`s shall not be restricted. Additional restrictions shall be implementation defined.

In both *lhs* and *rhs*, any character can be escaped with a `<control-V>`, in which case the character shall not be used to delimit *lhs* from *rhs*, and the escaping `<control-V>` shall be discarded.

If the character ! is appended to the `map` command name, the mapping shall be effective during open or visual text input mode rather than open or visual command mode. This allows *lhs* to have two different map definitions at the same time: one for command mode and one for text input mode.

For command mode mappings:

When the *lhs* is entered as any part of a `vi` command in open or visual mode (but not as part of the arguments to the command), the action shall be as if the corresponding *rhs* had been entered.

If any character in the command, other than the first, is escaped using a `<control-V>` character, that character shall not be part of a match to an *lhs*.

C

It is unspecified if implementations shall support command maps where the *lhs* is more than a single character in length, where the first character of the *lhs* is printable.

If *lhs* contains more than one character and the first character is #, additional, unspecified character(s), representing the function key named by the characters in *lhs* following the #, may be mapped to *rhs*. It is unspecified how function keys are named or what function keys are supported.

For text input mode mappings:

When the *lhs* is entered as any part of text entered in open or visual text input modes, the action shall be as if the corresponding *rhs* had been entered.

If any character in the input text is escaped using a `<control-V>` character, that character shall not be part of a match to an *lhs*.

It is unspecified if the *lhs* argument entered for the `map` or `unmap` commands is replaced in this fashion. Regardless of whether or not the replacement occurs, the effect of the command shall be as if the replacement had not occurred. C

If only part of the *lhs* is entered, it is unspecified how long the editor will wait for additional, possibly matching characters before treating the already entered characters as not matching the *lhs*.

The *rhs* characters shall themselves be subject to remapping, unless otherwise specified by the `remap` edit option, except that if the characters in *lhs* occur as prefix characters in *rhs*, those characters shall not be remapped.

On block-mode terminals, the mapping need not occur immediately (for example, it may occur after the terminal transmits a group of characters to the system), but it shall achieve the same results as if it occurred immediately.

Current line: Unchanged.

Current column: Unchanged.

5.10.7.5.15 **mark**

Synopsis: **[***1addr***]** `ma`**[**`rk`**]** *character*
Synopsis: **[***1addr***]** `k` *character*

Implementations shall support *character* values of a single lowercase letter of the POSIX Locale and the characters ‘ and ′; support of other characters is implementation defined.

If executing the `vi` `m` command, set the specified mark to the current line and 1-based numbered character referenced by the current column, if any; otherwise, column position 1.

Otherwise, set the specified mark to the specified line and 1-based numbered first non-`<blank>` character in the line, if any; otherwise, the last character in the line, if any; otherwise, column position 1.

The mark shall remain associated with the line until the mark is reset or the line is deleted. If a deleted line is restored by a subsequent `undo` command, any marks previously associated with the line, which have not been reset, shall be restored as well. Any use of a mark not associated with a current line in the edit buffer shall be an error.

The marks ‘ and ′ shall be set as described previously, immediately before the following events occur in the editor:

(1) The use of `$` as an `ex` address

(2) The use of a positive decimal number as an `ex` address

(3) The use of a search command as an `ex` address

(4) The use of a mark reference as an `ex` address

(5) The use of the following open and visual mode commands:

```
<control-]> % ( ) [ ] { }
```

(6) The use of the following open and visual mode commands:

```
′ G H L M z
```

if the current line will change as a result of the command

(7) The use of the open and visual mode commands:

```
/ ? N ‘ n
```

if the current line or column will change as a result of the command

(8) The use of the `ex` mode commands:

```
z undo global v
```

For rules (1), (2), (3), and (4), the ‘ and ′ marks shall not be set if the `ex` command is parsed as specified by rule (6a) in 5.10.7.3.

For rules (5), (6), and (7), the ‘ and ′ marks shall not be set if the commands are used as motion commands in open and visual mode.

For rules (1), (2), (3), (4), (5), (6), (7), and (8), the ‘ and ’ marks shall not be set if the command fails.

The ‘ and ’ marks shall be set as described previously, each time the contents of the edit buffer are replaced (including the editing of the initial buffer), if in open or visual mode, or if in `ex` mode and the edit buffer is not empty, before any commands or movements (including commands or movements specified by the `–c` or `–t` options or the +*command* argument) are executed on the edit buffer. If in open or visual mode, the marks shall be set as if executing the `vi m` command; otherwise, as if executing the `ex mark` command.

When changing from `ex` mode to open or visual mode, if the ‘ and ’ marks are not already set, the ‘ and ’ marks shall be set as described previously.

Current line: Unchanged.

Current column: Unchanged.

5.10.7.5.16 move

Synopsis: **[***2addr***]** `m`**[**`ove`**]** *1addr* **[***flags***]**

Move the specified lines after the specified destination line. A destination of line zero specifies that the lines shall be placed at the beginning of the edit buffer. It shall be an error if the destination line is within the range of lines to be moved.

Current line: Set to the last of the moved lines.

Current column: Set to nonblank.

5.10.7.5.17 next

Synopsis: `n`**[**`ext`**][**`!`**]** **[**+*command***]** **[***file* ...**]**

If no ! is appended to the command name, and the edit buffer has been modified since the last complete write, it shall be an error, unless the file is successfully written as specified by the `autowrite` option.

If one or more files is specified:

(1) Set the argument list to the specified file names.

(2) Set the current argument list reference to be the first entry in the argument list.

(3) Set the current pathname to the first file name specified.

Otherwise:

(1) It shall be an error if there are no more file names in the argument list after the file name currently referenced.

(2) Set the current pathname and the current argument list reference to the file name after the file name currently referenced in the argument list.

Replace the contents of the edit buffer with the contents of the file named by the current pathname. If for any reason the contents of the file cannot be accessed, the edit buffer shall be empty.

This command shall be affected by the `autowrite` and `writeany` edit options.

The +*command* option shall be `<blank>`-delimited; `<blank>` characters can be escaped by preceding them with a backslash character. The +*command* shall be interpreted as an `ex` command immediately after the contents of the edit buffer have been replaced and the current line and column have been set.

Current line: Set as described for the `edit` command.

Current column: Set as described for the `edit` command.

5.10.7.5.18 number

Synopsis: **[***2addr***]** `nu`**[**`mber`**]** **[***count***]** **[***flags***]**
Synopsis: **[***2addr***]** # **[***count***]** **[***flags***]**

These commands shall be equivalent to the `ex` command:

[*2addr***]** `p`**[**`rint`**]** **[***count***]** #**[***flags***]**

See 5.10.7.5.21.

5.10.7.5.19 open

Synopsis: **[***1addr***]** `o`**[**`pen`**]** **[**/*pattern***[**/**]]** **[***flags***]**

This command need not be supported on block-mode terminals or terminals with insufficient capabilities. If standard input, standard output, or standard error are not terminal devices, the results are unspecified.

Enter open mode.

The trailing delimiter can be omitted from pattern at the end of the command line. If pattern is empty (e.g., `//`) or not specified, the last RE used in the editor shall be used as the pattern. The pattern can be delimited by slashes (shown in the Synopsis line), as well as any alphanumeric, or non-`<blank>` character other than backslash, vertical line, double quote, or `<newline>`.

If a match is found for the optional RE in the line, the cursor shall be placed at the start of the matching pattern. If the pattern is not found, it shall be an error.

Current line: Set to the specified line.

Current column: Set to nonblank.

5.10.7.5.20 preserve

Synopsis: `pre`**[**`serve`**]**

Save the edit buffer in a form that can later be recovered by using the `−r` option or by using the `ex recover` command. After the file has been preserved, a mail message shall be sent to the user. This message shall be readable by invoking the `mailx` utility (see 4.40). The message shall contain the name of the file, the time of preservation, and an `ex` command that could be used to recover the file.

Additional, unspecified, information may be included in the mail message.

Current line: Unchanged.

Current column: Unchanged.

5.10.7.5.21 `print`

Synopsis: **[***2addr***]** `p`**[**`rint`**]** **[***count***]** **[***flags***]**

Write the addressed lines. The behavior is unspecified if the number of columns on the display is less than the number of columns required to write any single character in the line(s) being written.

Nonprintable characters, except for `<tab>`, shall be written as implementation-defined multicharacter sequences.

If the # flag is specified or the `number` edit option is set, each line shall be preceded by its line number in the following format:

`"%6d△△"`, <*line number*>

If the `l` flag is specified or the `list` edit option is set:

(1) The characters listed in Table 2-16 (see 2.12) shall be written as the corresponding escape sequence.

(2) Nonprintable characters not in Table 2-16 shall be written as one three-digit octal number (with a preceding `<backslash>`) for each byte in the character (most significant byte first). If the size of a byte on the system is greater than 9 b, the format used for nonprintable characters is implementation defined.

(3) The end of each line shall be marked with a `$`, and literal `$` characters within the line shall be written with a preceding backslash.

Long lines shall be folded. The length at which folding occurs is unspecified, but folding should be as appropriate for the output terminal, considering the number of columns of the terminal.

If a line is folded, and the `l` flag is specified or the `list` edit option is set:

(1) The point of folding shall be indicated by writing `<backslash>` `<newline>`.

(2) A multicolumn character at the folding position shall be neither separated nor discarded.

If a line is folded, and the `l` flag is not specified and the `list` edit option is not set, it is unspecified if a multicolumn character at the folding position is separated; it shall not be discarded.

Current line: Set to the last line written.

Current column: Unchanged if the current line is unchanged; otherwise, set to nonblank.

5.10.7.5.22 put

Synopsis: **[***1addr***]** pu**[**t**]** **[***buffer***]**

Append text from the specified buffer (by default, the unnamed buffer) to the specified line; line zero specifies that the text shall be placed at the beginning of the edit buffer. Each portion of a line in the buffer shall become a new line in the edit buffer, regardless of the mode of the buffer.

Current line: Set to the last line entered into the edit buffer.

Current column: Set to nonblank.

5.10.7.5.23 quit

Synopsis: q**[**uit**][**!**]**

If no ! is appended to the command name

(1) If the edit buffer has been modified since the last complete write, it shall be an error.

(2) If there are file names in the argument list after the file name currently referenced, and the last command was not a quit, wq, xit, or ZZ (see 5.35.7.2.85) command, it shall be an error.

Otherwise, terminate the editing session.

5.10.7.5.24 read

Synopsis: **[***1addr***]** r**[**ead**][**!**]** **[***file***]**

If ! is not the first non-<blank> character to follow the command name, a copy of the specified file shall be appended into the edit buffer after the specified line; line zero specifies that the copy shall be placed at the beginning of the edit buffer. The number of lines and bytes read shall be written. If no *file* is named, the current pathname shall be the default. If there is no current pathname, then *file* shall become the current pathname. If there is no current pathname or *file* operand, it shall be an error. Specifying a *file* that is not of type regular shall have unspecified results.

Otherwise, if *file* is preceded by !, the rest of the line after the ! shall have %, #, and ! characters expanded as described in 5.10.7.3.

The ex utility shall then pass two arguments to the program named by the shell edit option; the first shall be "–c" and the second shall be the expanded argu- C
ments to the read command as a single argument. The standard input of the pro- C
gram shall be set to the standard input of the ex program when it was invoked.
The standard error and standard output of the program shall be appended into C
the edit buffer after the specified line.

Each line in the copied file or program output (as delimited by <newline> characters or the end of the file or output if it is not immediately preceded by a <newline> character), shall be a separate line in the edit buffer. Any occurrences of <carriage-return> and <newline> character pairs in the output shall be treated as single `<newline>` characters.

The special meaning of the `!` following the `read` command can be overridden by escaping it with a backslash character.

Current line:

If no lines are added to the edit buffer, unchanged.

Otherwise, if in open or visual mode, set to the first line entered into the edit buffer.

Otherwise, set to the last line entered into the edit buffer.

Current column:

Set to nonblank.

5.10.7.5.25 recover

Synopsis: `rec`**[**`over`**][**`!`**] [***file***]**

If no `!` is appended to the command name, and the edit buffer has been modified since the last complete write, it shall be an error.

If no *file* operand is specified, then the current pathname shall be used. If there is no current pathname or *file* operand, it shall be an error. C C

If no recovery information has previously been saved about *file*, the `recover` command shall behave identically to the `edit` command, and an informational message to this effect shall be written.

Otherwise, set the current pathname to *file*, and replace the current contents of the edit buffer with the recovered contents of *file*. If there are multiple instances of the file to be recovered, the one most recently saved shall be recovered, and an informational message that there are previous versions of the file that can be recovered shall be written. The editor shall behave as if the contents of the edit buffer have already been modified.

Current line: Set as described for the `edit` command.

Current column: Set as described for the `edit` command.

5.10.7.5.26 rewind

Synopsis: `rew`**[**`ind`**][**`!`**]**

If no `!` is appended to the command name, and the edit buffer has been modified since the last complete write, it shall be an error, unless the file is successfully written as specified by the `autowrite` option.

If the argument list is empty, it shall be an error.

The current argument list reference and the current pathname shall be set to the first file name in the argument list.

Replace the contents of the edit buffer with the contents of the file named by the current pathname. If for any reason the contents of the file cannot be accessed, the edit buffer shall be empty.

This command shall be affected by the `autowrite` and `writeany` edit options.

Current line: Set as described for the `edit` command.

Current column: Set as described for the `edit` command.

5.10.7.5.27 s

Synopsis: **[***2addr***]** `s` **[**/**[***pattern***][**/**[***repl***][**/**]]] [***options***] [***count***] [***flags***]]**
Synopsis: **[***2addr***]** & **[***options***] [***count***] [***flags***]**
Synopsis: **[***2addr***]** **~** **[***options***] [***count***] [***flags***]**

Replace the first instance of *pattern* with the string *repl* on each specified line. (See 5.10.7.6 and 5.10.7.7.) Any nonalphabetic, nonblank delimiter other than \, |, double quote or `<newline>` can be used instead of `/`. Backslash characters can be used to escape delimiters, backslash characters, and other special characters.

The trailing delimiter can be omitted from *pattern* or from *repl* at the end of the command line. If both *pattern* and *repl* are not specified or are empty (e.g., `//`), the last `s` command shall be repeated. If only *pattern* is not specified or is empty, the last RE used in the editor shall be used as the pattern. If only *repl* is not specified or is empty, the pattern shall be replaced by nothing. If the entire replacement pattern is %, the last replacement pattern to an `s` command shall be used.

Entering a `<carriage-return>` in *repl* (which requires an escaping backslash in `ex` mode and an escaping `<control-V>` in open or `vi` mode) shall split the line at that point, creating a new line in the edit buffer. The `<carriage-return>` shall be discarded.

If *options* includes the letter `g` (`global`), all nonoverlapping instances of the *pattern* in the line shall be replaced.

If *options* includes the letter `c` (confirm), then before each substitution the line shall be written; the written line shall reflect all previous substitutions. On the following line, `<space>` characters shall be written beneath the characters from the line that are before the *pattern* to be replaced, and ^ characters written beneath the characters included in the *pattern* to be replaced. The `ex` utility shall then wait for a response from the user. An affirmative response shall cause the substitution to be done, while any other input shall not make the substitution. An affirmative response shall consist of a line with the affirmative response (as defined by the current locale) at the beginning of the line. This line shall be subject to editing in the same way as the `ex` command line.

If interrupted (see 5.10.5.4), any modifications confirmed by the user shall be preserved in the edit buffer after the interrupt.

If the remembered search direction is not set, the `s` command shall set it to forward (see 5.35.7.2.63 and 5.35.7.2.64).

In the second synopsis, the & command shall repeat the previous substitution, as if the & command were replaced by `s/`*pattern*`/`*repl*`/`, where *pattern* and *repl* are as specified in the previous `s`, &, or **~** command.

In the third synopsis, the **~** command shall repeat the previous substitution, as if the **~** were replaced by `s/`*pattern*`/`*repl*`/`, where *pattern* shall be the last RE specified to the editor, and *repl* shall be from the previous substitution (including & and **~**) command.

These commands shall be affected by the **LC_MESSAGES** environment variable.

Current line: Set to the last line in which a substitution occurred, or, unchanged if no substitution occurred.

Current column: Set to nonblank.

5.10.7.5.28 set

Synopsis: `se`**[**`t`**]** **[***option***[**`=`**[***value***]]** `. . .` **] [[**`no`**]***option . . .* **] [***option*`?` `. . .` **] [**`all`**]**

When no arguments are specified, write the value of the `term` edit option and those options whose values have been changed from the default settings; when the argument `all` is specified, write all of the option values.

Giving an option name followed by the character `?` shall cause the current value of that option to be written. The `?` can be separated from the option name by zero or more `<blank>`s. The `?` shall be necessary only for Boolean valued options. Boolean options can be given values by the form `set` *option* to turn them on or `set no`*option* to turn them off; string and numeric options can be assigned by the form `set` *option*=*value*. Any `<blank>`s in strings can be included as is by preceding each `<blank>` with an escaping backslash. More than one option can be set or listed by a single set command by specifying multiple arguments, each separated from the next by one or more `<blank>`s.

See 5.10.7.8 for details about specific options.

Current line: Unchanged.

Current column: Unchanged.

5.10.7.5.29 shell

Synopsis: `sh`**[**`ell`**]**

Invoke the program named by the `shell` edit option with the single argument `–i`. Editing shall be resumed when the program exits.

Current line: Unchanged.

Current column: Unchanged.

5.10.7.5.30 source

Synopsis: `so[urce]` *file*

Read and execute `ex` commands from *file*. Lines in the file that contain no characters or only `<blank>` characters shall be ignored.

Current line: As specified for the individual `ex` commands.

Current column: As specified for the individual `ex` commands.

5.10.7.5.31 suspend

Synopsis: `su[spend][!]`
Synopsis: `st[op][!]`

Allow control to return to the invoking process; `ex` shall suspend itself as if it had received the SIGTSTP signal. If the system does not support job control as described in POSIX.1 {8}, it shall be an error. If job control is not enabled for any reason, the results of the command are unspecified.

These commands shall be affected by the `autowrite` and `writeany` edit options.

The current *susp* character (see `stty` in 4.59) shall have the same effect as the `suspend` command.

Current line: Unchanged.

Current column: Unchanged.

5.10.7.5.32 tag

Synopsis: `ta[g][!]` *tagstring*

The results are unspecified if the format of a tags file is not as specified by the `ctags` utility (5.7) description.

The `tag` command shall search for *tagstring* in the tag file(s) referred to by the `tag` edit option, in the order they are specified, until a reference to *tagstring* is found. Files shall be searched from beginning to end. If no reference is found, it shall be an error and an error message to this effect shall be written. If no reference is found and a file referred to by the `tag` edit option does not exist, is not readable, or has an unspecified problem, an error message shall be written. This error message shall only be displayed the first time a tag is not found and a file in the `tag` edit option has a problem.

Otherwise, if the tags file contained a pattern, the pattern shall be treated as an RE used in the editor; e.g., for the purposes of the `s` command.

If the *tagstring* is in a file with a different name than the current pathname, set the current pathname to the name of that file, and replace the contents of the edit buffer with the contents of that file. In this case, if no ! is appended to the command name, and the edit buffer has been modified since the last complete write, it shall be an error, unless the file is successfully written as specified by the `autowrite` option.

This command shall be affected by the `autowrite`, `tag`, `taglength`, and `writeany` edit options.

Current line:

If the tags file contained a line number, set to that line number. If the line number is larger than the last line in the edit buffer, an error message shall be written and the current line shall be set as specified for the `edit` command.

If the tags file contained a pattern, set to the first occurrence of the pattern in the file. If no matching pattern is found, an error message shall be written and the current line shall be set as specified for the `edit` command.

Current column:

If the tags file contained a line-number reference and that line-number was not larger than the last line in the edit buffer, or if the tags file contained a pattern and that pattern was found, set to nonblank.

Otherwise, set as specified for the `edit` command.

5.10.7.5.33 unabbreviate

Synopsis: `una`**[**`bbreviate`**]** *lhs*

If *lhs* is not an entry in the current list of abbreviations (see 5.10.7.5.1), it shall be an error. Otherwise, delete *lhs* from the list of abbreviations.

Current line: Unchanged.

Current column: Unchanged.

5.10.7.5.34 undo

Synopsis: `u`**[**`ndo`**]**

Reverse the changes made by the last command that modified the contents of the edit buffer, including `undo`. For this purpose, the `global`, `v`, `open`, and `visual` commands, and commands resulting from buffer executions and mapped character expansions, are considered single commands.

If no action that can be undone preceded the `undo` command, it shall be an error.

If the `undo` command restores lines that were marked, the mark shall also be restored unless it was reset subsequent to the deletion of the lines.

Current line:

(1) If lines are added or changed in the file, set to the first line added or changed.

(2) Set to the line before the first line deleted, if it exists.

(3) Set to 1 if the edit buffer is not empty.

(4) Set to zero.

Current column: Set to nonblank.

5.10.7.5.35 unmap

Synopsis: `unm[ap][!]` *lhs*

If ! is appended to the command name, and if *lhs* is not an entry in the list of text input mode map definitions, it shall be an error. Otherwise, delete *lhs* from the list of text input mode map definitions.

If no ! is appended to the command name, and if *lhs* is not an entry in the list of command mode map definitions, it shall be an error. Otherwise, delete *lhs* from the list of command mode map definitions.

Current line: Unchanged.

Current column: Unchanged.

5.10.7.5.36 version

Synopsis: `ve[rsion]`

Write a message containing version information for the editor. The format of the message is unspecified.

Current line: Unchanged.

Current column: Unchanged.

5.10.7.5.37 visual

Synopsis: [*1addr*] `vi[sual]` [*type*] [*count*] [*flags*]

If `ex` is currently in open or visual mode, the Synopsis and behavior of the `visual` command shall be the same as the `edit` command, as specified by 5.10.7.5.8.

Otherwise, this command need not be supported on block-mode terminals or terminals with insufficient capabilities. If standard input, standard output, or standard error are not terminal devices, the results are unspecified. C

If *count* is specified, the value of the `window` edit option shall be set to *count* (as described in 5.10.7.8.29). If the ˆ type character was also specified, the `window` edit option shall be set before being used by the ˆ type character. C C

Enter visual mode. If *type* is not specified, it shall be as if a *type* of + was specified. The *type* shall cause the following effects:

+ Place the beginning of the specified line at the top of the display.

– Place the end of the specified line at the bottom of the display.

. Place the beginning of the specified line in the middle of the display.

ˆ If the specified line is less than or equal to the value of the `window` edit option, set the line to 1; otherwise, decrement the line by the value of the `window` edit option minus 1. Place the beginning of this line as close to the bottom of the displayed lines as possible, while still displaying the value of the `window` edit option number of lines.

Current line: Set to the specified line.

Current column: Set to nonblank.

5.10.7.5.38 write

Synopsis: **[***2addr***]** `w`**[**`rite`**][**`!`**]** **[**`>>`**]** **[***file***]**
Synopsis: **[***2addr***]** `w`**[**`rite`**]** **[**`!`**]** **[***file***]**
Synopsis: **[***2addr***]** `wq`**[**`!`**]** **[**`>>`**]** **[***file***]**

If no lines are specified, the lines shall default to the entire file.

The command `wq` shall be equivalent to a `write` command followed by a `quit` command; `wq!` shall be equivalent to `write!` followed by `quit`. In both cases, if the `write` fails, the `quit` shall not be attempted.

If the command name is not followed by one or more `<blank>`s, or *file* is not preceded by a ! character, the write shall be to a file.

(1) If the >> argument is specified, and the file already exists, the lines shall be appended to the file instead of replacing its contents. If the >> argument is specified, and the file does not already exist, it is unspecified if the write shall proceed as if the >> argument had not been specified or if the write shall fail.

(2) If the `readonly` edit option is set (see 5.10.7.8.13), the write shall fail.

(3) If *file* is specified, and is not the current pathname, and the file exists, the write shall fail.

(4) If *file* is not specified, the current pathname shall be used. If there is no current pathname, the `write` command shall fail.

(5) If the current pathname is used, and the current pathname has been changed by the `file` or `read` commands, and the file exists, the write shall fail. If the write is successful, subsequent writes shall not fail for this reason (unless the current pathname is changed again).

(6) If the whole edit buffer is not being written, and the file to be written exists, the write shall fail.

For rules (1), (2), (4), and (5), the write can be forced by appending the character ! to the command name.

For rules (2), (4), and (5), the write can be forced by setting the `writeany` edit option.

Additional, implementation-defined tests may cause the write to fail.

If the edit buffer is empty, a file without any contents shall be written.

An informational message shall be written noting the number of lines and bytes written.

Otherwise, if the command is followed by one or more `<blank>`s, and *file* is preceded by `!`, the rest of the line after the `!` shall have `%`, `#`, and `!` characters expanded as described in 5.10.7.3.

The `ex` utility shall then pass two arguments to the program named by the `shell` edit option; the first shall be "`−c`" and the second shall be the expanded arguments to the `write` command as a single argument. The specified lines shall be written to the standard input of the command. The standard error and standard output of the program. if any, shall be written as described for the `print` command. If the last character in that output is not a `<newline>` character, a `<newline>` shall be written at the end of the output.

The special meaning of the `!` following the `write` command can be overridden by escaping it with a backslash character.

Current line: Unchanged.

Current column: Unchanged.

5.10.7.5.39 xit

Synopsis: **[***2addr***]** `x`**[**`it`**][**`!`**]** **[***file***]**

If the edit buffer has not been modified since the last complete write, `xit` shall be equivalent to the `quit` command, or if a `!` is appended to the command name, to `quit!`.

Otherwise, `xit` shall be equivalent to the `wq` command, or if a `!` is appended to the command name, to `wq!`.

Current line: Unchanged.

Current column: Unchanged.

5.10.7.5.40 yank

Synopsis: **[***2addr***]** `ya`**[**`nk`**]** **[***buffer***]** **[***count***]**

Copy the specified lines to the specified buffer (by default, the unnamed buffer), which shall become a line-mode buffer.

Current line: Unchanged.

Current column: Unchanged.

5.10.7.5.41 z

Synopsis: **[***1addr***]** `z`**[**`!`**][***type* . . . **]** **[***count***]** **[***flags***]**

If no line is specified, the current line shall be the default; if *type* is omitted as well, the current line value shall first be incremented by 1. If incrementing the current line would cause it to be greater than the last line in the edit buffer, it shall be an error.

If there are `<blank>` characters between the *type* argument and the preceding `z` command name or optional `!` character, it shall be an error.

If *count* is specified, the value of the `window` edit option shall be set to *count* (as described in 5.10.7.8.29), If *count* is omitted, it shall default to 2 times the value of the `scroll` edit option, or if `!` was specified, the number of lines in the display minus 1.

If *type* is omitted, then *count* lines starting with the specified line shall be written. Otherwise, *count* lines starting with the line specified by the *type* argument shall be written.

The *type* argument shall change the line(s) to be written. The possible values of *type* are as follows:

– The specified line shall be decremented by the following value:

(((number of "–" characters) × *count*) – 1)

If the calculation would result in a number less than 1, it shall be an error. Write lines from the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit buffer has been written.

+ The specified line shall be incremented by the following value:

(((number of "+" characters) – 1) × *count*) + 1

If the calculation would result in a number greater than the last line in the edit buffer, it shall be an error. Write lines from the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit buffer has been written.

=

. If more than a single `.` or = is specified, it shall be an error. The following steps shall be taken:

(1) If *count* is zero, nothing shall be written.

(2) Write as many of the *N* lines before the current line in the edit buffer as exist. If *count* or `!` was specified, *N* shall be

(*count* – 1) / 2

Otherwise, *N* shall be

(*count* – 3) / 2

If *N* is a number less than 3, no lines shall be written.

(3) If = was specified as the *type* character, write a line consisting of the smaller of: the number of columns in the display divided by two, or 40 "–" characters.

(4) Write the current line.

(5) Repeat step 3.

(6) Write as many of the *N* lines after the current line in the edit buffer as exist. *N* shall be defined as in step (2). If *N* is a number less than 3, no lines shall be written.

lines after the current line in the edit buffer as exist. If *count* is less than 3, no lines shall be written.

ˆ The specified line shall be decremented by the following value:

$$(((\text{number of "}\hat{}\text{" characters}) + 1) \times \textit{count}) - 1$$

If the calculation would result in a number less than 1, it shall be an error. Write lines from the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit buffer has been written.

Current line: Set to the last line written, unless the type is =, in which case, set to the specified line.

Current column: Set to nonblank.

5.10.7.5.42 !

Synopsis: **[***2addr***]** ! *command*

The contents of the line after the ! shall have %, #, and ! characters expanded as described in 5.10.7.3. If the expansion causes the text of the line to change, it shall be redisplayed, preceded by a single ! character.

The `ex` utility shall execute the program named by the `shell` edit option. It shall pass two arguments to the program; the first shall be "–`c`", and the second shall be the expanded arguments to the ! command as a single argument.

If no lines are specified, the standard input, standard output, and standard error of the program shall be set to the standard input, standard output, and standard error of the `ex` program when it was invoked. In addition, a warning message shall be written if the edit buffer has been modified since the last complete write, and the `warn` edit option is set.

If lines are specified, they shall be passed to the program as standard input, and the standard output and standard error of the program shall replace those lines in the edit buffer. Each line in the program output (as delimited by `<newline>` characters or the end of the output if it is not immediately preceded by a `<newline>` character), shall be a separate line in the edit buffer. Any occurrences of `<carriage-return>` and `<newline>` character pairs in the output shall be treated as single `<newline>` characters. The specified lines shall be copied into the unnamed buffer before they are replaced, and the unnamed buffer shall become a line-mode buffer.

If in `ex` mode, a single ! character shall be written when the program completes. C

This command shall be affected by the `shell` and `warn` edit options. If no lines are specified, this command shall be affected by the `autowrite` and `writeany` edit options. If lines are specified, this command shall be affected by the `auto-print` edit option. C C

Current line:

(1) If no lines are specified, unchanged.

(2) Otherwise, set to the last line read in, if any lines are read in.

(3) Otherwise, set to the line before the first line of the lines specified, if that line exists.

(4) Otherwise, set to the first line of the edit buffer if the edit buffer is not empty.

(5) Otherwise, set to zero.

Current column:

If no lines are specified, unchanged.

Otherwise, set to nonblank.

5.10.7.5.43 <

Synopsis: **[***2addr***]** <**[**<. . . **] [***count***] [***flags***]**

Shift the specified lines toward the start of the line; the number of column positions to be shifted shall be the number of command characters times the value of the `shiftwidth` edit option. Only leading `<blank>`s shall be deleted or changed into other `<blank>` characters in shifting; other characters shall not be affected.

Lines to be shifted shall be copied into the unnamed buffer, which shall become a line-mode buffer.

This command shall be affected by the `autoprint` edit option. C

Current line: Set to the last line in the lines specified.

Current column: Set to nonblank.

5.10.7.5.44 >

Synopsis: **[***2addr***]** >**[**>. . . **] [***count***] [***flags***]**

Shift the specified lines away from the start of the line; the number of column positions to be shifted shall be the number of command characters times the value of the `shiftwidth` edit option. The shift shall be accomplished by adding `<blank>`s as a prefix to the line or changing leading `<blank>` characters into other `<blank>` characters. Empty lines shall not be changed.

Lines to be shifted shall be copied into the unnamed buffer, which shall become a line-mode buffer.

This command shall be affected by the `autoprint` edit option. C

Current line: Set to the last line in the lines specified.

Current column: Set to nonblank.

5.10.7.5.45 **`<control-D>`**

Synopsis: `<control-D>`

Write the next *n* lines, where *n* is the minimum of the values of the `scroll` edit option and the number of lines after the current line in the edit buffer. If the current line is the last line of the edit buffer it shall be an error.

Current line: Set to the last line written.

Current column: Set to nonblank.

5.10.7.5.46 =

Synopsis: **[***1addr***]** = **[***flags***]**

If line is not specified, it shall default to the last line in the edit buffer. Write the line number of the specified line.

Current line: Unchanged.

Current column: Unchanged.

5.10.7.5.47 @

Synopsis: **[***2addr***]** @ **[***buffer***]**
Synopsis: **[***2addr***]** * **[***buffer***]**

If no *buffer* is specified or is specified as @ or *, the last *buffer* executed shall be used. If no previous *buffer* has been executed, it shall be an error.

For each line specified by the addresses, set the current line (.) to the specified line, and execute the contents of the named *buffer* (as they were at the time the @ command was executed) as `ex` commands. For each line of a line-mode buffer, and all but the last line of a character-mode buffer, the `ex` command parser shall behave as if the line was terminated by a `<newline>` character.

If an error occurs during this process, or a line specified by the addresses does not exist when the current line would be set to it, or more than a single line was specified by the addresses, and the contents of the edit buffer are replaced (e.g., by the `ex :edit` command) an error message shall be written, and no more commands resulting from the execution of this command shall be processed. C C

Current line: As specified for the individual `ex` commands.

Current column: As specified for the individual `ex` commands.

5.10.7.6 REs

The ex utility shall support REs that are a superset of the BREs described in 2.8.3. A null RE (// or ??) shall be equivalent to the last RE encountered.

REs can be used in addresses to specify lines and, in some commands (for example, the s command), to specify portions of a line to be substituted.

The following constructs can be used to enhance the BREs:

\<	Match the beginning of a word. (See the definition of word at the beginning of 5.10.7.4.)
\>	Match the end of a word.
~	Match the replacement part of the last s command. The tilde (~) character can be escaped with a backslash in an RE to become a normal character with no special meaning. The backslash shall be discarded.

When the magic edit option is not set, the only characters with special meanings shall be ^ at the beginning of a pattern, $ at the end of a pattern, and backslash. The characters ., *, [, and ~ shall be treated as ordinary characters unless preceded by a backslash; when preceded by a backslash they shall regain their special meaning, or in the case of backslash, be handled as a single backslash. Backslashes used to escape other characters shall be discarded.

5.10.7.7 Replacement Strings

The character & (\& if the magic edit option is not set) in the replacement string shall stand for the text matched by the pattern to be replaced. The character ~ (\~ if the magic edit option is not set) shall be replaced by the replacement part of the previous s command. The sequence *n*, where *n* is an integer, shall be replaced by the text matched by the pattern enclosed in the *n*th set of parentheses \(and \).

The strings \l, \u, \L, and \U can be used to modify the case of elements in the replacement string. The string \l (\u) shall cause the character that follows to be converted to lowercase (uppercase). The string \L (\U) shall cause all characters subsequent to it to be converted to lowercase (uppercase) until the string \e or \E, or the end of the replacement string, is encountered.

Otherwise, any character following a backslash shall be treated as that literal character, and the escaping backslash shall be discarded.

5.10.7.8 Edit Options

The ex utility has a number of options that modify its behavior. These options have default settings, which can be changed using the set command.

Options are Boolean unless otherwise specified.

5.10.7.8.1 autoindent, ai

[Default: *unset*]

If autoindent is set, each line in text input mode shall be indented (first using as many <tab>s as possible, as determined by the tabstop edit option, and then using <space>s) to align with another line, as follows:

(1) If in open or visual mode and the text input is part of a line-oriented command (see 5.35.7), align to the first column.

(2) Otherwise, if in open or visual mode, indentation for each line shall be set as follows:

  (a) If a line was previously inserted as part of this command, it shall be set to the indentation of the last inserted line by default, or as otherwise specified for the <control-D> character in 5.35.7.3.2.

  (b) Otherwise, it shall be set to the indentation of the previous current line, if any; otherwise, to the first column.

(3) For the ex a, i, and c commands, indentation for each line shall be set as follows:

  (a) If a line was previously inserted as part of this command, it shall be set to the indentation of the last inserted line by default, or as otherwise specified for the *eof* character in 5.10.7.4.1.

  (b) Otherwise, if the command is the ex a command, it shall be set to the line appended after, if any; otherwise to the first column.

  (c) Otherwise, if the command is the ex i command, it shall be set to the line inserted before, if any; otherwise to the first column.

  (d) Otherwise, if the command is the ex c command, it shall be set to the indentation of the line replaced.

5.10.7.8.2 autoprint, ap

[Default: *set*]

If autoprint is set, the current line shall be written after each ex command that modifies the contents of the current edit buffer, and after each tag command for which the tag search pattern was found or tag line number was valid, unless:

(1) The command was executed while in open or visual mode.

(2) The command was executed as part of a global or v command or @ buffer execution.

(3) The command was the form of the read command that reads a file into the edit buffer.

(4) The command was the append, change, or insert command.

(5)

(6) The command was not terminated by a `<newline>` character.

(7) The current line shall be written by a flag specified to the command; e.g., "`delete #`" shall write the current line as specified for the flag modifier to the `delete` command, and not as specified by the `autoprint` edit option.

5.10.7.8.3 autowrite, aw

[Default: *unset*]

If `autowrite` is set, and the edit buffer has been modified since it was last completely written to any file, the contents of the edit buffer shall be written as if the `ex write` command had been specified without arguments, before each command affected by the `autowrite` edit option is executed. Appending the character `!` to the command name of any of the `ex` commands except `!` shall prevent the write. If the write fails, it shall be an error and the command shall not be executed.

5.10.7.8.4 errorbells, eb

[Default: *unset*]

If the editor is in `ex` mode, and the terminal does not support a standout mode (such as inverse video), and `errorbells` is set, error messages shall be preceded by alerting the terminal.

5.10.7.8.5 exrc, ex

[Default: *unset*]

If `exrc` is set, `ex` shall access any `.exrc` file in the current directory, as described in 5.10.7.1. If `exrc` is not set, `ex` shall ignore any `.exrc` file in the current directory during initialization, unless the current directory is named by the **HOME** environment variable.

5.10.7.8.6 ignorecase, ic

[Default: *unset*]

If `ignorecase` is set, characters that have uppercase and lowercase representations shall have those representations considered as equivalent for use in REs.

The `ignorecase` edit option shall affect all remembered REs; e.g., unsetting the `ignorecase` edit option shall cause a subsequent `vi n` command to search for the last BRE in a case-sensitive fashion.

5.10.7.8.7 list

[Default: *unset*]

If `list` is set, edit buffer lines written while in `ex` command mode shall be written as specified for the `print` command with the `l` flag specified.

In open or visual mode, each edit buffer line shall be displayed as specified for the `ex print` command with the `l` flag specified. In open or visual text input mode, when the cursor does not rest on any character in the line, it shall rest on the `$` marking the end of the line. C

5.10.7.8.8 magic

[Default: *set*]

If `magic` is set, modify the interpretation of characters in REs and substitution replacement strings as described in 5.10.7.6 and 5.10.7.7.

5.10.7.8.9 mesg

[Default: *set*]

If `mesg` is set, the permission for others to use the `write` or `talk` commands to write to the terminal shall be set while in open or visual mode. The shell-level command `mesg n` (see 5.17) shall take precedence over any setting of the `mesg` edit option; i.e., if `mesg y` was issued before the editor started (or in a shell escape, such as `:!mesg y`), the `mesg edit` option in the editor shall suppress incoming messages, but the `mesg edit` option shall not enable incoming messages if `mesg n` was issued.

5.10.7.8.10 number, nu

[Default: *unset*]

If `number` is set, edit buffer lines written while in `ex` command mode shall be written with line numbers, in the format specified by the `print` command with the # flag specified. In `ex` text input mode, each line shall be preceded by the line number it will have in the file. C

In open or visual mode, each edit buffer line shall be displayed with a preceding line number, in the format specified by the `ex print` command with the # flag specified. This line number shall not be considered part of the line for the purposes of evaluating the current column; i.e., column position 1 shall be the first column position after the format specified by the `print` command. C

5.10.7.8.11 paragraphs, para

[Default in the POSIX Locale: `IPLPPPQPP LIpplpipbp`]

The `paragraphs` edit option shall define additional paragraph boundaries for open and visual mode commands. The `paragraphs` edit option can be set to a character string consisting of zero or more character pairs; it shall be an error to set it to an odd number of characters.

5.10.7.8.12 prompt

[Default: *set*]

If prompt is set, ex command mode input shall be prompted for with a colon (:) character; when unset, no prompt shall be written.

5.10.7.8.13 readonly, ro

[Default: *see text*]

If the readonly edit option is set, read-only mode shall be enabled (see 5.10.7.5.38). The readonly edit option shall be initialized to set if either of the following conditions are true: C

— The command-line option −R was specified. C

— Performing actions equivalent to the POSIX.1 {8} *access*() function, called with the following arguments indicates that the file lacks write permission: C

(1) The current pathname is used as the *path* argument. C

(2) The constant W_OK is used as the *amode* argument. C

The readonly edit option may be initialized to set for other, implementation-defined reasons. The readonly edit option shall not be initialized to unset based on any special privileges of the user or process. C

The readonly edit option shall be reinitialized each time that the contents of the edit buffer are replaced (e.g., by an edit or next command) unless the user has explicitly set it, in which case it shall remain set until the user explicitly unsets it. Once unset, it shall again be reinitialized each time that the contents of the edit buffer are replaced. C

5.10.7.8.14 remap

[Default: *set*]

If remap is set, map translation shall allow for maps defined in terms of other maps; translation shall continue until a final product is obtained. If unset, only a one-step translation shall be done.

5.10.7.8.15 report

[Default: 5]

The value of the report edit option specifies what number of lines being added, copied, deleted or modified in the edit buffer will cause an informational message to be written to the user. The following conditions shall cause an informational message. The message shall contain the number of lines added, copied, deleted, or modified, but is otherwise unspecified. C

— An ex or vi editor command, other than open, undo, or visual, that modifies at least the value of the report edit option number of lines, and which is not part of an ex global or v command, or ex or vi buffer C execution, shall cause an informational message to be written.

— An ex yank or vi y or Y command, that copies at least the value of the report edit option plus 1 number of lines, and which is not part of an ex global or v command, or ex or vi buffer execution, shall cause an informational message to be written.

— An ex global, v, open, undo, or visual command or ex or vi buffer execution, that adds or deletes a total of at least the value of the report edit option number of lines, and which is not part of an ex global or v command, or ex or vi buffer execution, shall cause an informational message to be written. (For example, if 3 lines were added and 8 lines deleted during an ex visual command, 5 would be the number compared against the report edit option after the command completed.

5.10.7.8.16 scroll, scr

[Default: (number of lines in the display – 1) / 2]

The value of the scroll edit option shall affect the number of lines scrolled by the ex <control-D> and z commands. For the vi <control-D> and <control-U> commands, it shall be the initial number of lines to scroll when no previous <control-D> or <control-U> command has been executed.

5.10.7.8.17 sections, sect

[Default in the POSIX Locale: NHSHH HUnhsh]

The sections edit option shall define additional section boundaries for open and visual mode commands. The sections edit option can be set to a character string consisting of zero or more character pairs; it shall be an error to set it to an odd number of characters.

5.10.7.8.18 shell, sh

[Default: from the environment variable **SHELL**]

The value of this edit option shall be a string. The default shall be taken from the **SHELL** environment variable. If the **SHELL** environment variable is null or empty, the sh (see 4.56) utility shall be the default.

5.10.7.8.19 shiftwidth, sw

[Default: 8]

The value of this edit option shall give the width in columns of an indentation level used during autoindentation and by the ex and vi < and > commands.

5.10.7.8.20 showmatch, sm

[Default: *unset*]

The functionality described for the showmatch edit option need not be supported on block-mode terminals or terminals with insufficient capabilities.

If the showmatch option is set, in open and visual text input modes, when a) or } is typed, if the matching (or { is currently visible on the display, the matching (or { shall be flagged by moving the cursor to its location for an unspecified amount of time.

5.10.7.8.21 showmode, smd

[Default: *unset*]

If showmode is set in open or visual mode, the current mode of the editor shall be displayed on the last line of the display. Command mode and text input mode shall be differentiated; other unspecified modes and implementation-defined information may be displayed.

5.10.7.8.22 slowopen

[Default: *unset*]

If slowopen is set during open and visual text input modes, the editor shall not update portions of the display other than those screen columns that display the characters entered by the user (see 5.35.7.3).

5.10.7.8.23 tabstop, ts

[Default: 8]

The value of this edit option shall specify the column boundary used by a <tab> character in the display (see 5.10.7.8.2 and 5.35.7.2).

5.10.7.8.24 taglength, tl

[Default: zero]

The value of this edit option shall specify the maximum number of characters that are considered significant in the user-specified tag name and in the tag name from the tags file. If the value is zero, all characters in both tag names shall be significant.

5.10.7.8.25 tag, tags

[Default: *unspecified*]

The value of this edit option shall be a string of <blank>-delimited pathnames of files used by the tag command. The default value is unspecified.

5.10.7.8.26 term

[Default: from the environment variable **TERM**]

The value of this edit option shall be a string. The default shall be taken from the **TERM** environment variable. If the **TERM** environment variable is empty or null, the default is unspecified. The editor shall use the value of this edit option to determine the type of the display device.

The results are unspecified if the user changes the value of the `term` edit option after editor initialization.

5.10.7.8.27 terse

[Default: *unset*]

If `terse` is set, error messages may be less verbose. However, except for this caveat, error messages are unspecified.

5.10.7.8.28 warn

[Default: *set*]

If `warn` is set, and the contents of the edit buffer have been modified since they were last completely written, the editor shall write a warning message before certain ! commands (see 5.10.7.5.42).

5.10.7.8.29 window, wi

[Default: *see text*]

A value used in open and visual mode, by the `<control-B>` and `<control-F>` commands, and, in visual mode, to specify the number of lines displayed when the screen is repainted.

If the –`w` command-line option is not specified, the default value shall be set to the value of the **LINES** environment variable. If the **LINES** environment variable is empty or null, the default shall be the number of lines in the display minus 1.

Setting the `window` edit option to zero or to a value greater than the number of lines in the display minus 1 (either explicitly or based on the –`w` option or the **LINES** environment variable) shall cause the `window` edit option to be set to the number of lines in the display minus 1.

The baud rate of the terminal line may change the default in an implementation-defined manner.

5.10.7.8.30 wrapmargin, wm

[Default: zero]

If the value of this edit option is zero, it shall have no effect.

If not in the POSIX Locale, the effect of this edit option is implementation-defined.

Otherwise, it shall specify a number of columns from the ending margin of the terminal.

During open and visual text input modes, for each character for which any part of the character is displayed in a column that is less than `wrapmargin` columns from the ending margin of the screen, the editor shall behave as follows:

(1) If the character triggering this event is a `<blank>`, it, and all immediately preceding `<blank>` characters on the current line entered during the execution of the current text input command shall be discarded, and the editor shall behave as if the user had entered a single `<newline>` character instead. In addition, if the next user-entered character is a `<space>`, it shall be discarded as well.

(2) Otherwise, if there are one or more `<blank>` characters on the current line immediately preceding the last group of inserted non-`<blank>` characters which was entered during the execution of the current text input command, the `<blank>` characters shall be replaced as if the user had entered a single `<newline>` character instead. C

If the `autoindent` edit option is set, and the events described in (1) or (2) are performed, any `<blank>` characters at or after the cursor in the current line shall be discarded.

The ending margin shall be determined by the system or overridden by the user, as described for **COLUMNS** in 5.10.5.3 and 2.6.

5.10.7.8.31 `wrapscan, ws`

[Default: *set*]

If `wrapscan` is set, searches (the `ex` `/` and `?` addresses, or open and visual mode `/`, `?`, `N`, and `n` commands) shall wrap around the beginning or end of the edit buffer; when unset, searches shall stop at the beginning or end of the edit buffer. C

5.10.7.8.32 `writeany, wa`

[Default: *unset*]

If `writeany` is set, some of the checks performed when executing the `ex` `write` commands shall be inhibited, as described in 5.10.7.5.38.

5.10.8 Exit Status

The `ex` utility shall exit with one of the following values:

0	Successful completion.
>0	An error occurred.

5.10.9 Consequences of Errors

When any error is encountered and the standard input is not a terminal device file, ex shall not write the file or return to command or text input mode, and shall terminate with a nonzero exit status. C

Otherwise, when an unrecoverable error is encountered it shall be equivalent to a SIGHUP asynchronous event. C

Otherwise, when an error is encountered, the editor shall behave as specified in 5.10.7.3. C

5.11 expand – Convert tabs to spaces

⇒ **5.11.5.3 expand Environment Variables.** *In the description of* **LC_CTYPE**, *change the phrase* "... width in column positions each character would occupy on a constant-width-font output device" *to:*

> ... width in column positions each character would occupy on an output device.

Rationale: This change partially satisfies the following corrigendum request from ISO/IEC 9945-2: 1993 Annex H.2:

(15) In 5.11.5.3 and 5.32.5.3, in the last sentence of the LC_CTYPE paragraph for expand and unexpand, the phrase "on a constant-width-font output device" may be redundant because of definitions elsewhere in the standard.

5.14 `file` – Determine file type

Rationale: The changes in this clause, except for those related to symbolic links, satisfy the following requirement from ISO/IEC 9945-2: 1993 Annex H.1:

(12) The `file` utility should allow user-specified algorithms for file type recognition, similar to those used in the historical `/etc/magic` file.

⇒ **5.14.1 `file` Synopsis.** *Modify the Synopsis to be:*

`file` **[**`−dhi`**] [**`−M` *file***] [**`−m` *file***]** *file* . . .

⇒ **5.14.2 `file` Description.** *Add a new paragraph at the end of the subclause:*

If *file* is a symbolic link, by default the link shall be resolved and `file` shall test the type of file referenced by the symbolic link.

⇒ **5.14.3 `file` Options.** *Replace the entire Options subclause with:*

The `file` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

`−d`	Apply any default system tests to the file.
`−h`	When a symbolic link is encountered, identify the file as a symbolic link. If `−h` is not specified and *file* is a symbolic link that refers to a nonexistent file, `file` shall identify the file as a symbolic link, as if `−h` had been specified.
`−i`	If a file is a regular file, do not attempt to classify the type of the file further, but identify the file as specified in 5.14.6.1, using a <*type*> string that contains the string `regular file`.
`−M` *file*	Specify the name of a file containing tests that shall be applied to a file in order to classify it (see 5.14.7). No default system tests shall be applied.
`−m` *file*	Specify the name of a file containing tests that shall be applied to a file in order to classify it (see 5.14.7).

If multiple instances of the `−m`, `−d`, or `−M` options are specified, the concatenation of the tests specified, in the order specified, shall be the set of tests that are applied. If a `−M` option is specified, no tests other than those specified using the `−d`, `−M`, and `−m` options shall be applied to the file. If neither the `−d` nor `−M` options are specified, any default system tests shall be applied after any tests specified using the `−m` option.

⇒ **5.14.6.1 `file` Standard Output.** *Insert a new paragraph between the second and third (the one beginning* "If the file named . . . "*) paragraphs:*

If *file* is identified as a symbolic link (see `−h`), the following alternative output format shall be used:

`"%s: %s %s\n"`, <*file*>, <*type*>, <*contents of link*>

⇒ **5.14.6.1 `file` Standard Output.** *Change the third paragraph (the one beginning with* "If the file named . . . "*) to:*

If the file named by the *file* operand does not exist or cannot be read, the string `cannot open` shall be included as part of the <*type*> field, but this shall not be considered an error that affects the exit status. If the type of the file named by the *file* operand cannot be determined, the string `data` shall be included as part of the <*type*> field, but this shall not be considered an error that affects the exit status. C

B

⇒ **5.14.6.1 `file` Standard Output.** *Add the following entry to the table named* `file` *Output Strings:*

If *file* is a	<*type*> shall contain the string
symbolic link	`symbolic link to`

⇒ **5.14.7 `file` Extended Description.** *Change the entire subclause to:*

A file specified as an option-argument to the `−m` or `−M` options shall contain one test per line, which shall be applied to the file. If the test succeeds, the *message* field of the line shall be printed and no further tests shall be applied, with the exception that tests on immediately following lines beginning with a single > character shall be applied.

Each line shall be composed of the following four `<blank>`-separated fields:

offset An unsigned number (optionally preceded by a single > character) specifying the offset, in bytes, of the value in the file that is to be compared against the value field of the line. If the file is shorter than the specified offset, the test shall fail.

If the offset begins with the character >, the test contained in the line shall not be applied to the file unless the test on the last line for which the offset did not begin with a > was successful. By default, the offset shall be interpreted as an unsigned decimal number. With a leading `0x` or `0X`, the offset shall be interpreted as a hexadecimal number; otherwise, with a leading `0`, the offset shall be interpreted as an octal number.

type The type of the value in the file to be tested. The type shall consist of the type specification characters `c`, `d`, `f`, `s`, and `u`, specifying character, signed decimal, floating point, string, and unsigned decimal, respectively.

The type string shall be interpreted as the bytes from the file starting at the specified offset and including the same number of bytes specified by the *value* field. If insufficient bytes remain in the file past the offset to match the value field, the test shall fail.

The type specification characters `d`, `f`, and `u` can be followed by an optional unsigned decimal integer that specifies the number of bytes represented by the type. The type specification character `f` can be followed by an optional `F`, `D`, or `L`, indicating that the value is of type *float*, *double*, or *long double*, respectively. The type specification characters `d` and `u` can be followed by an optional `C`, `S`, `I`, or `L`, indicating that the value is of type *char*, *short*, *int*, or *long*, respectively.

The default number of bytes represented by the type specifiers `d`, `f`, and `u` shall correspond to their respective C-language types as follows. If the system claims conformance to the C-Language Development Utilities Option, those specifiers shall correspond to the default sizes used in the `c89` utility. Otherwise, the default sizes shall be implementation defined.

For the type specifier characters `d` and `u`, the default number of bytes shall correspond to the size of the basic integral data type of the implementation. For these specifier characters, the implementation shall support values of the optional number of bytes to be converted corresponding to the number of bytes in the C-language types *char*, *short*, *int*, or *long*. These numbers can also be specified by an application as the characters `C`, `S`, `I`, and `L`, respectively. The byte order used when interpreting numeric values is implementation defined, but shall correspond to the order in which a constant of the corresponding type is stored in memory on the system.

For the type specifier `f`, the default number of bytes shall correspond to the number of bytes in the basic double precision floating-point data type of the underlying implementation. The implementation shall support values of the optional number of bytes to be converted corresponding to the number of bytes in the C-language types *float*, *double*, and *long double*. These numbers can also be specified by an application as the characters `F`, `D`, and `L`, respectively.

All type specifiers, except for `s`, can be followed by a mask specifier of the form &*number*. The mask value shall be ANDed with the value before the comparison with the value from the file is made. By default, the mask shall be interpreted as an unsigned decimal number. With a leading `0x` or `0X`, the mask shall be interpreted as a unsigned hexadecimal number; otherwise, with a leading `0`, the mask shall be interpreted as an unsigned octal number.

The strings `byte`, `short`, `long`, and `string` shall also be supported as type fields, being interpreted as `dC`, `dS`, `dL`, and `s`, respectively.

value The value to be compared with the value from the file.

Any value that contains a character that is not a digit, other than a leading sign (+ or −) or a leading `0x` or `0X`, shall be interpreted as a string. The test shall succeed only when a string value exactly matches the bytes from the file.

If the value is a string, it can contain the following sequences:

\character
The backslash-escape sequences in Table 2-16 (see 2.12). The results of using any other character, other than an octal digit, following the backslash are unspecified.

\octal
Octal sequences that can be used to represent characters with specific coded values. An octal sequence shall consist of a backslash followed by the longest sequence of one, two, or three octal-digit characters (01234567). If the size of a byte on the system is greater than 9 b, the valid escape sequence used to represent a byte is implementation defined.

By default, any value that is not a string shall be interpreted as a signed decimal number. Any such value, with a leading `0x` or `0X`, shall be interpreted as an unsigned hexadecimal number; otherwise, with a leading zero, the value shall be interpreted as an unsigned octal number.

If the value is not a string, it can be preceded by a character indicating the comparison to be performed. Permissible characters and the comparisons they specify are as follows:

= The test shall succeed if the value from the file equals the value field.

< The test shall succeed if the value from the file is less than the value field.

> The test shall succeed if the value from the file is greater than the value field.

& The test shall succeed if all of the bits in the value field are set in the value from the file.

ˆ The test shall succeed if at least one of the bits in the value field is not set in the value from the file.

x The test shall succeed if there is any value in the file.

message
The message to be printed if the test succeeds. The message shall be interpreted using the notation for the `printf` formatting specification; see 4.50.7. If the *value* field was a string, the the value from the file shall be the argument for the `printf` formatting specification; otherwise, the value from the file shall be the argument. B B

Editor's Note: The rationale in E.5.14 (IEEE Std 1003.2-1992 pages 987-88, lines 9703-49) will be replaced by the following:

`file` Rationale. *(This subclause is not a part of P1003.2b)*

Historical systems have used a "magic file" named `/etc/magic` to help identify file types. Because it is generally useful for users and scripts to be able to identify special file types, the `−m` flag and a portable format for user-created magic files has been specified. No requirement is made that an implementation of `file` use this method of identifying files, only that users be permitted to add their own classifying tests.

In addition, three options have been added to historical practice. The `−d` flag has been added to permit users to cause their tests to follow any default system tests. The `−i` flag has been added to permit users to test portably for regular files in shell scripts. The `−M` flag has been added to permit users to ignore any default system tests.

The historical `−c` option was omitted as not particularly useful to users or portable shell scripts. In addition, a reasonable implementation of the `file` utility would report any errors found each time the magic file is read.

The historical format of the magic file was the same as that specified by the rationale in the previous version of this standard for the *offset*, *value*, and *message* fields; however, it used less precise *type* fields than the format specified by the current normative text. The new type field values are a superset of the historical ones.

The following is an example magic file:

```
0  short      070707     cpio archive
0  short      0143561    byte-swapped cpio archive
0  string     070707     ASCII cpio archive
0  long       0177555    very old archive
0  short      0177545    old archive
0  short      017437     old packed data
0  string     \037\036   packed data
0  string     \377\037   compacted data
0  string     \037\235   compressed data
>2 byte&0x80  >0         block compressed
>2 byte&0x1f  x          %d bits
0  string     \032\001   Compiled Terminfo Entry
0  short      0433       Curses screen image
```

```
    0  short      0434         Curses screen image
    0  string     <ar>         System V Release 1 archive
    0  string     !<arch>\n__.SYMDEF   archive random library
    0  string     !<arch>      archive
    0  string     ARF_BEGARF   PHIGS clear text archive
    0  long       0x137A2950   scalable OpenFont binary
    0  long       0x137A2951   encrypted scalable OpenFont binary
```

⇒ **5.18 `more` – Display files on a page-by-page basis.** *Replace the entire* `more` *clause with the following.*

Editor's Note: All of this clause has been changed in Draft 11 from the POSIX.2-1992 version. The rationale in Annex E is also completely replaced. Only the portions changed from Draft 10 and the 1992 standard are diffmarked.

Rationale: The changes to this clause are the result of interpretation requests PASC 1003.2-92 #37 and 109, submitted for IEEE Std 1003.2-1992, and the following requirement from ISO/IEC 9945-2: 1993 Annex H.1:

(27) The `more` utility should be able to handle underlined and emboldened displays of characters that are wider than a single column position.

5.18 `more` – Display files on a page-by-page basis

5.18.1 Synopsis

`more` **[**`–ceisu`**] [**`–n` *number***] [**`–t` *tagstring***] [**`–p` *command***] [***file* . . . **]**

Obsolescent Version:

`more` **[**`–ceisu`**] [**`–n` *number***] [***+command***] [**`–t` *tagstring***] [***file* . . . **]**

5.18.2 Description

The `more` utility shall read files and either write them to the terminal on a page-by-page basis or filter them to standard output. If standard output is not a terminal device, all input files shall be copied to standard output in their entirety, without modification, except as specified for the `–s` option. If standard output is a terminal device, the files shall be written a number of lines (one "screenful") at a time under the control of user commands; see 5.18.7.

Certain block-mode terminals do not have all the capabilities necessary to support the complete `more` definition; they are incapable of accepting commands that are not terminated with a `<newline>`. Implementations that support such terminals shall provide an operating mode to `more` in which all commands can be terminated with a `<newline>` on those terminals. This mode shall

— Be documented in the system documentation

— At invocation, inform the user of the terminal deficiency that requires the `<newline>` usage and provide instructions on how this warning can be suppressed in future invocations

— Not be required for implementations supporting only fully capable terminals

— Not affect commands already requiring `<newline>`s

— Not affect users on the capable terminals from using `more` as described in this standard

5.18.3 Options

The `more` utility shall conform to the utility argument syntax guidelines described in 2.10.2, except that +*command* of the obsolescent version uses a non-standard syntax, and that the order of presentation of the `−p` and `−t` options is significant.

The following options shall be supported by the implementation:

`−c` If a screen is to be written that has no lines in common with the current screen, or `more` is writing its first screen, do not scroll the screen, but instead redraw each line of the screen in turn, from the top of the screen to the bottom. In addition, if `more` is writing its first screen, clear the screen. This option may be silently ignored on devices with insufficient terminal capabilities.

`−e` By default, `more` shall exit immediately after writing the last line of the last file in the argument list. If the `−e` option is specified:

(1) If there is only a single file in the argument list and that file was completely displayed on a single screen, `more` shall exit immediately after writing the last line of that file.

(2) Otherwise, `more` shall exit only after reaching end-of-file on the last file in the argument list twice without an intervening operation; see 5.18.7.

`−i` Perform pattern matching in searches without regard to case. See 2.8.2.

`−n` *number* Specify the number of lines per screenful. The *number* argument is a positive decimal integer. The `−n` option shall override any values obtained from any other source.

`−p` *command*
+*command* (Obsolescent.)
Each time a screen from a new file is displayed or redisplayed (including as a result of `more` commands; e.g., `:p`), execute the `more` command(s) in the *command* arguments in the order specified, as if entered by the user after the first screen has been

displayed. No intermediate results shall be displayed (i.e., if the command is a movement to a screen different than the normal first screen, only the screen resulting from the command shall be displayed.) If any of the commands fail for any reason, an informational message to this effect shall be written, and no further commands specified using the `−p` or +*command* options shall be executed for this file.

`−s` Behave as if consecutive empty lines were a single empty line.

`−t` *tagstring*

Write the screenful of the file containing the tag named by the *tagstring* argument. See the `ctags` utility in 5.7. The tags feature represented by `−t` *tagstring* and the `:t` command (see 5.18.7.23) is optional. It shall be provided on any system that also provides a conforming implementation of `ctags`; otherwise, the use of `−t` produces undefined results.

The file name resulting from the `−t` option shall be logically added as a prefix to the list of command-line files, as if specified by the user. If the tag named by the *tagstring* argument is not found, it shall be an error, and `more` shall take no further action.

If the tag specifies a line number, the first line of the display shall contain the beginning of that line. If the tag specifies a pattern, the first line of the display shall contain the beginning of the matching text from the first line of the file that contains that pattern. If the line does not exist in the file or matching text is not found, an informational message to this effect shall be displayed, and `more` shall display the default screen as if `−t` had not been specified.

If both the `−t` *tagstring* and `−p` *command* (or the obsolescent +*command*) options are given, the `−t` *tagstring* shall be processed first; i.e., the file and starting line for the display shall be as specified by `−t`, and then the `−p` or +*command* `more` commands shall be executed. If the line (matching text) specified by the `−t` command does not exist (is not found), no `−p` or +*command* `more` commands shall be executed for this file at any time.

`−u` Treat `<backspace>` as a printable control character, displayed as an implementation-defined character sequence (see 5.18.7), suppressing backspacing and the special handling that produces underlined or standout-mode text on some terminal types. Also, do not ignore a `<carriage-return>` character at the end of a line.

5.18.4 Operands

The following operand shall be supported by the implementation:

file	A pathname of an input file. If no *file* operands are specified, the standard input shall be used. If a *file* is –, the standard input shall be read at that point in the sequence.

5.18.5 External Influences

5.18.5.1 Standard Input

The standard input shall be used only if no *file* operands are specified, or if a *file* operand is –.

5.18.5.2 Input Files

The input files being examined shall be files of any type. If standard output is a terminal, standard error shall be used to read commands from the user. If standard output is a terminal, standard error is not readable, and command input is needed, `more` may attempt to obtain user commands from the controlling terminal (e.g., `/dev/tty`); otherwise, `more` shall terminate with an error indicating that it was unable to read user commands. If standard output is not a terminal, no error shall result if standard error cannot be opened for reading.

5.18.5.3 Environment Variables

The following environment variables shall affect the execution of `more`:

COLUMNS	This variable shall override the system-selected horizontal screen size. See 2.6 for valid values and results when it is unset or null.
EDITOR	This variable shall be used by the `v` command to select an editor; see 5.18.7.
LANG	This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.
LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.
LC_COLLATE	This variable shall determine the locale for character collation information in BREs.

LC_CTYPE	This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files), and the behavior of character classes within BREs.
LC_MESSAGES	This variable shall determine the language in which messages should be written.
LINES	This variable shall override the system-selected vertical screen size, used as the number of lines in a screenful. See 2.6 for valid values and results when it is unset or null. The `-n` option shall take precedence over the **LINES** variable for determining the number of lines in a screenful.
MORE	This variable shall be interpreted as a string containing options described in 5.18.3, preceded with hyphens and `<blank>`-separated as on the command line. Any command-line options shall be processed after those in the **MORE** variable, as if the command line were `more $MORE` *options operands* The **MORE** variable shall take precedence over the **TERM** and **LINES** variables for determining the number of lines in a screenful.
TERM	This variable shall be interpreted as the name of the terminal type. If this variable is unset or null, an unspecified default terminal type shall be used.

5.18.5.4 Asynchronous Events

Default.

5.18.6 External Effects

5.18.6.1 Standard Output

The standard output shall be used to write the contents of the input files.

5.18.6.2 Standard Error

Used for diagnostic messages and user commands (see 5.18.5.2) and, if standard output is a terminal device, to write a prompting string. The prompting string shall shall appear on the screen line below the last line of the file displayed in the current screenful. The prompt shall contain the name of the file currently being examined and shall contain an end-of-file indication and the name of the next file, if any, when prompting at the end-of-file. If an error or informational message is displayed, it is unspecified if it is contained in the prompt. If it is not contained in the prompt, it shall be displayed and then the user shall be prompted for a continuation character, at which point another message or the user prompt may be displayed. The prompt is otherwise unspecified. It is unspecified if informational messages are written for other user commands.

5.18.6.3 Output Files

None.

5.18.7 Extended Description

The following subclause describes the behavior of `more` when the standard output is a terminal device. If the standard output is not a terminal device, no options other than `−s` shall have any effect, and all input files shall be copied to standard output otherwise unmodified, at which time `more` shall exit without further action.

The number of lines available per "screen" shall be determined by the `−n` option, if present, or by examining values in the environment (see 5.18.5.3). If neither method yields a number, an unspecified number of lines shall be used.

The maximum number of lines written shall be one less than this number because the screen line after the last line written shall be used to write a user prompt and user input. If the number of lines in the screen is less than two, the results are undefined. It is unspecified if user input is permitted to be longer than the remainder of a single line where the prompt has been written.

The number of columns available per line shall be determined by examining values in the environment (see 5.18.5.3), with a default value as described in 2.6. Lines that are longer than the display shall be folded; the length at which folding occurs is unspecified, but should be appropriate for the output device. Folding may occur between glyphs of single characters that take up multiple display columns.

When standard output is a terminal and `−u` is not specified, `more` shall treat `<backspace>`s and `<carriage-return>`s specially:

— A character, followed first by a sequence of *n* `<backspace>`s (where *n* is the same as the number of column positions that the character occupies), then by *n* underscores (_), shall cause that character to be written as underlined text, if the terminal type supports that. The *n* underscores, followed first by *n* `<backspace>`s, then any character with *n* column positions, also shall cause that character to be written as underlined text, if the terminal type supports that.

— A sequence of *n* `<backspace>`s (where *n* is the same as the number of column positions that the previous character occupies) that appears between two identical printable characters shall cause the first of those two characters to be written as emboldened text (i.e., visually brighter, standout mode, or inverse-video mode), if the terminal type supports that, and the second to be discarded. Immediately subsequent occurrences of

`<backspace>`s/character pairs for that same character also shall be discarded. (For example, the sequence `a\ba\ba\ba` is interpreted as a single emboldened `a`.)

— The `more` utility shall logically discard all other `<backspace>` characters from the line as well as the character which precedes them, if any.

— A `<carriage-return>` at the end of a line shall be ignored, rather than being written as a nonprintable character, as described in the next paragraph.

It is implementation defined how other nonprintable characters are written. Implementations should use the same format that they use for the `ex` `print` command; see 5.10.7.5.21. It is unspecified if a multicolumn character shall be separated if it crosses a logical line boundary; it shall not be discarded. The behavior is unspecified if the number of columns on the display is less than the number of columns any single character in the line being displayed would occupy.

When each new file is displayed (or redisplayed), `more` shall write the first screen of the file. Once the initial screen has been written, `more` shall prompt for a user command. If the execution of the user command results in a screen that has lines in common with the current screen, and the device has sufficient terminal capabilities, `more` shall scroll the screen; otherwise, it is unspecified if the screen is scrolled or redrawn.

For all files but the last (including standard input if no file was specified, and for the last file as well, if the `–e` option was not specified), when `more` has written the last line in the file, `more` shall prompt for a user command. This prompt shall contain the name of the next file as well as an indication that `more` has reached end-of-file. If the user command is `f`, `<control-F>`, `<space>`, `j`, `<newline>`, `d`, `<control-D>`, or `s`, `more` shall display the next file. Otherwise, if displaying the last file, `more` shall exit. Otherwise, `more` shall execute the user command specified.

Several of the commands described in this clause display a previous screen from the input stream. In the case that text is being taken from a nonrewindable stream, such as a pipe, it is implementation defined how much backwards motion is supported. If a command cannot be executed because of a limitation on backwards motion, an error message to this effect shall be displayed, the current screen shall not change, and the user shall be prompted for another command.

If a command cannot be performed because there are insufficient lines to display, `more` shall alert the terminal. If a command cannot be performed because there are insufficient lines to display or a `/` command fails: if the input is the standard input, the last screen in the file may be displayed; otherwise, the current file and screen shall not change, and the user shall be prompted for another command.

The interactive commands in the following subclauses shall be supported. Some commands can be preceded by a decimal integer, called *count* in the following descriptions. If not specified with the command, *count* shall default to 1.

In the following descriptions, *pattern* is a BRE, as described in 2.8.3. The term "examine" is historical usage meaning "open the file for viewing"; for example, `more foo` would be expressed as "examining" file `foo`. In the following descriptions, unless otherwise specified, *line* is a logical line in the `more` display, not a line from the file being examined.

In the following descriptions, the "current position" refers to two things:

— The position of the current line on the screen

— The line number (in the file) of the current line on the screen

Usually, the line on the screen corresponding to the current position is the third line on the screen. If this is not possible (there are fewer than three lines to display, or this is the first page of the file, or it is the last page of the file), then the current position is either the first or last line on the screen as described later.

5.18.7.1 Help

Synopsis: `h`

Write a summary of these commands and other implementation-defined commands. The behavior shall be as if the `more` utility were executed with the `−e` option on a file that contained the summary information. The user shall be prompted as described in 5.18.7 when end-of-file is reached. If the user command is one of those specified to continue to the next file, `more` shall return to the file and screen state from which the `h` command was executed.

5.18.7.2 Scroll forwards one screenful

Synopsis: **[***count***]**`f`
Synopsis: **[***count***]**`<control-F>`

Scroll forwards *count* lines, with a default of one screenful. If *count* is more than the screen size, only the final screenful shall be written.

5.18.7.3 Scroll backwards one screenful

Synopsis: **[***count***]**`b`
Synopsis: **[***count***]**`<control-B>`

Scroll backwards *count* lines, with a default of one screenful. If *count* is more than the screen size, only the final screenful shall be written.

5.18.7.4 Scroll forwards one line

Synopsis: **[***count***]**`<space>`
Synopsis: **[***count***]**`j`
Synopsis: **[***count***]**`<newline>`

Scroll forwards *count* lines. The default *count* for `<space>` shall be one screenful; for `j` and `<newline>`, one line. The entire *count* lines shall be written, even if *count* is more than the screen size.

5.18.7.5 Scroll backwards one line

Synopsis: **[***count***]**`k`

Scroll backwards *count* lines. The entire *count* lines shall be written, even if *count* is more than the screen size.

5.18.7.6 Scroll forwards one-half screenful

Synopsis: **[***count***]**`d`
Synopsis: **[***count***]**`<control-D>`

Scroll forwards *count* lines, with a default of one half of the screen size. If *count* is specified, it shall become the new default for subsequent `d` `<control-D>`, `u`, and `<control-U>` commands. The entire *count* lines shall be written, even if *count* is more than the screen size.

5.18.7.7 Skip forwards one line

Synopsis: **[***count***]**`s`

Display the screenful beginning with the line *count* lines after the last line on the current screen. If *count* would cause the current position to be such that less than one screenful would be written, the last screenful in the file shall be written.

5.18.7.8 Scroll backwards one-half screenful

Synopsis: **[***count***]**`u`
Synopsis: **[***count***]**`<control-U>`

Scroll backwards *count* lines, with a default of one half of the screen size. If *count* is specified, it shall become the new default for subsequent `d` `<control-D>`, `u`, and `<control-U>` commands. The entire *count* lines shall be written, even if *count* is more than the screen size.

5.18.7.9 Go to beginning of file

Synopsis: **[***count***]**`g`

Display the screenful beginning with the line *count*.

5.18.7.10 Go to end-of-file

Synopsis: **[***count***]**`G`

If *count* is specified, display the screenful beginning with the line *count*. Otherwise, display the last screenful of the file.

5.18.7.11 Refresh the screen

Synopsis: `r`
Synopsis: `<control-L>`

Refresh the screen.

5.18.7.12 Discard and refresh

Synopsis: `R`

Refresh the screen, discarding any buffered input. If the current file is nonseekable, buffered input shall not be discarded, and the `R` command is equivalent to the `r` command.

5.18.7.13 Mark position

Synopsis: `m`*letter*

Mark the current position with the letter named by *letter*, where *letter* represents the name of one of the lowercase letters of the portable character set. When a new file is examined, all marks may be lost.

5.18.7.14 Return to mark

Synopsis: `'`*letter*

Return to the position that was previously marked with the letter named by *letter*, making that line the current position.

5.18.7.15 Return to previous position

Synopsis: `''`

Return to the position from which the last large movement command was executed (where a "large movement" is defined as any movement of more than a screenful of lines). If no such movements have been made, return to the beginning of the file.

5.18.7.16 Search forwards for pattern

Synopsis: **[***count***]**`/`**[**`!`**]***pattern*`<newline>`

Display the screenful beginning with the *count*-th line containing the pattern. The search shall start after the first line currently displayed. The null BRE (`/<newline>`) shall repeat the search using the previous BRE, with a default *count*. If the character `!` is included, matching lines shall be those that do not contain the pattern. If no match is found for the pattern, a message to that effect shall be displayed.

5.18.7.17 Search backwards for pattern

Synopsis: [*count*]?[!]*pattern*`<newline>`

Display the screenful beginning with the *count*-th previous line containing the pattern. The search shall start on the last line before the first line currently displayed. The null BRE (`?<newline>`) shall repeat the search using the previous BRE, with a default *count*. If the character `!` is included, matching lines shall be those that do not contain the pattern. If no match is found for the pattern, a message to that effect shall be displayed.

5.18.7.18 Repeat search

Synopsis: [*count*]`n`

Repeat the previous search for *count*-th line containing the last *pattern* (or not containing the last *pattern*, if the previous search was `/!` or `?!`).

5.18.7.19 Repeat search in reverse

Synopsis: [*count*]`N`

Repeat the search in the opposite direction of the previous search for the *count*-th line containing the last *pattern* (or not containing the last *pattern*, if the previous search was `/!` or `?!`).

5.18.7.20 Examine new file

Synopsis: `:e` [*filename*]`<newline>`

Examine a new file. If the *filename* argument is not specified, the “current” file (see the `:n` and `:p` commands below) shall be re-examined. The *filename* shall be subjected to the process of shell word expansions (see 3.6); if more than a single pathname results, the effects are unspecified. If *filename* is a number sign (#), the previously examined file shall be re-examined. If *filename* is not accessible for any reason (including that it is a nonseekable file), an error message to this effect shall be displayed and the current file and screen shall not change.

5.18.7.21 Examine next file

Synopsis: [*count*]`:n`

Examine the next file. If a number *count* is specified, the *count*-th next file shall be examined. If *filename* refers to a nonseekable file, the results are unspecified.

5.18.7.22 Examine previous file

Synopsis: [*count*]`:p`

Examine the previous file. If a number *count* is specified, the *count*-th previous file shall be examined. If *filename* refers to a nonseekable file, the results are unspecified.

5.18.7.23 Go to tag

Synopsis: `:t` *tagstring*`<newline>`

If the file containing the tag named by the *tagstring* argument is not the current file, examine the file, as if the `:e` command was executed with that file as the argument. Otherwise, or in addition, display the screenful beginning with the tag, as described for the `−t` option (see 5.18.3). If the `ctags` utility is not supported by the system, the use of `:t` produces undefined results.

5.18.7.24 Invoke editor

Synopsis: `v`

Invoke an editor to edit the current file being examined. If standard input is being examined, the results are unspecified. The name of the editor shall be taken from the environment variable **EDITOR** or shall default to `vi`. If the last pathname component in **EDITOR** is either "`ex`" or "`vi`," the editor shall be invoked with a `−c` *linenumber* command-line argument, where *linenumber* is the line number of the physical line containing the logical line currently displayed as the first line of the screen. It is implementation defined whether line-setting options are passed to editors other than `vi` and `ex`.

When the editor exits, `more` shall resume with the same file and screen as when the editor was invoked.

5.18.7.25 Display position

Synopsis: `=`
Synopsis: `<control-G>`

Write a message for which the information references the first byte of the line after the last line of the file on the screen. This message shall include the name of the file currently being examined, its number relative to the total number of files there are to examine, the physical line number, the byte number and the total bytes in the file, and what percentage of the file precedes the current position. If `more` is reading from standard input, or the file is shorter than a single screen, the line number, the byte number, the total bytes, and the percentage need not be written.

5.18.7.26 Quit

Synopsis: `q`
Synopsis: `:q`
Synopsis: `ZZ`

Exit `more`.

5.18.8 Exit Status

The `more` utility shall exit with one of the following values:

0	Successful completion.
>0	An error occurred.

5.18.9 Consequences of Errors

If an error is encountered accessing a file when using the `:n` command, `more` shall attempt to examine the next file in the argument list, but the final exit status shall be affected. If an error is encountered accessing a file via the `:p` command, `more` shall attempt to examine the previous file in the argument list, but the final exit status shall be affected. If an error is encountered accessing a file via the `:e` command, `more` shall remain in the current file, and the final exit status shall not be affected.

5.22 `patch` – Apply changes to files

⇒ **5.22.3 `patch` Options.** *Change the* `–D` *description to:* B

`–D` *define* Mark changes with one of the following C preprocessor constructs: B

```
#ifdef define
...
#endif

#ifndef define
...
#endif
```

optionally combined with the C preprocessor construct `#else`. B

Rationale: This change is the result of interpretation request PASC 1003.2-92 #69 submitted for IEEE Std 1003.2-1992. B

⇒ **5.22.7.2 `patch` Filename Determination.** *Replace the entire subclause with:*

If no *file* operand is specified, `patch` shall perform the following steps to determine the filename to use:

(1) If the type of diff is context, the `patch` utility shall delete pathname components (as specified by the `–p` option) from the filename on the line beginning with ***, then test for the existence of this file relative to the current directory (or the directory specified with the `–d` option). If the file exists, the `patch` utility shall use this filename.

(2) If the type of diff is context, the `patch` utility shall delete the pathname components (as specified by the `−p` option) from the filename on the line beginning with `---`, then test for the existence of this file relative to the current directory (or the directory specified with the `−d` option). If the file exists, the `patch` utility shall use this filename.

(3) If the header information contains a line beginning with the string `Index:`, the `patch` utility shall delete pathname components (as specified by the `−p` option) from this line, then test for the existence of this file relative to the current directory (or the directory specified with the `−d` option). If the file exists, the `patch` utility shall use this filename.

(4) The `patch` utility shall write a prompt to standard output and request a filename interactively from the controlling terminal (e.g., `/dev/tty`).

Rationale: The change substituting `/dev/tty` for standard input corrects an error that deviated from historical practice and is the result of interpretation request PASC 1003.2-92 #19 submitted for IEEE Std 1003.2-1992.

The other wording changes are required to match historical practice and are the result of interpretation request PASC 1003.2-92 #15 submitted for IEEE Std 1003.2-1992.

5.24 renice – Set system scheduling priorities of running processes B

⇒ **5.24.1 renice Synopsis.** *Change the first Synopsis line (the non-Obsolescent one) to:* B

renice −n *increment* **[** −g **|** −p **|** −u **]** *ID . . .* B

⇒ **5.24.2 renice Description.** *Delete the second paragraph, which currently contains:* B

The system scheduling priority shall be bounded in an implementation-defined manner. If the requested *increment* (or *nice_value* in the obsolescent versions) would raise or lower the system scheduling priority of the executed utility beyond implementation-defined limits, then the limit whose value was exceeded shall be used. B

⇒ **5.24.3 renice Options.** *Change the full description of* −n *to:*

−n *increment*
: The −n option for the renice utility shall behave as described for the −n option for the nice utility (see 5.24).

⇒ **5.24.4 renice Operands.** *Change the description of the* nice_value *operand:*

nice_value
: (Obsolescent.) The value specified shall be taken as the actual system scheduling priority, rather than as an increment to the existing system scheduling priority. The system scheduling priority shall be bounded in an implementation-defined manner. If the requested *nice_value* would raise or lower the system scheduling priority of the executed utility beyond implementation-defined limits, then the limit whose value was exceeded shall be used. Specifying a scheduling priority higher than that of the existing process may require appropriate privileges.

Rationale: The preceding changes are the result of interpretation requests PASC 1003.2-92 #83 and #84 submitted for IEEE Std 1003.2-1992.

5.32 unexpand – Convert spaces to tabs

⇒ **5.32.5.3 unexpand Environment Variables.** *In the description of* **LC_CTYPE**, *change the phrase* "... width in column positions each character would occupy on a constant-width-font output device" *to:*

... width in column positions each character would occupy on an output device.

Rationale: This change partially satisfies the following corrigendum request from ISO/IEC 9945-2: 1993 Annex H.2:

(15) In 5.11.5.3 and 5.32.5.3, in the last sentence of the LC_CTYPE paragraph for expand and unexpand, the phrase "on a constant-width-font output device" may be redundant because of definitions elsewhere in the standard.

5.33 uudecode – Decode a binary file

Rationale: This change partially satisfies the following corrigendum request from ISO/IEC 9945-2: 1993 Annex H.2:

(14) The uuencode utility should support the BASE64-encoding specified in the MIME-RFC currently under consideration for Internet use. The uudecode utility should allow the user to override the output file name that is embedded in the file. Both utilities should be moved from Section 5 to Section 4.

⇒ **5.33.1 uudecode Synopsis.** *Change the Synopsis to be:*

uudecode **[**–o *outfile***]** **[***file***]**

⇒ **5.33.2 uudecode Description.** *Replace the first paragraph with:*

The uudecode utility shall read a file, or standard input if no file is specified, that includes data created by the uuencode utility (see 5.34). The uudecode utility shall scan the input file, searching for data compatible with one of the formats specified in 5.34.6.1 and attempt to create or overwrite the file described by the data (or overridden by the –o option). The pathname shall be contained in the data or specified by the –o option. The file access permission bits and contents for the file to be produced shall be contained in that data. The mode bits of the created file (other than standard output) shall be set from the file access permission bits contained in the data; i.e., other attributes of the mode, including the file mode creation mask (see umask in 4.67), shall not affect the file being produced.

⇒ **5.33.3 uudecode Options.** *Change this subclause to:*

The uudecode utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following option shall be supported by the implementation:

–o *outfile*	A pathname of a file that shall be used instead of any pathname contained in the input data. Specifying an *outfile* option-argument of /dev/stdout shall indicate standard output.

⇒ **5.33.6.1 uudecode Standard Output.** *Change this subclause to:*

If the file data header encoded by uuencode is – or the –o /dev/stdout option overrides the file data, the standard output shall be in the same format as the file originally encoded by uuencode. Otherwise, the standard output shall not be used.

Editor's Note: The following rationale will be added to E.5.33, but is kept here with uudecode *for this draft:*

uudecode Rationale. *(This subclause is not a part of P1003.2b)*

The –o option is not historical practice, but was added at the request of WG15 so that the user could override the target pathname without having to edit the input data itself.

In an early draft, the **[**–o *outfile***]** option-argument allowed the use of – to mean standard output. The symbol – has only been used previously in this standard as a standard input indicator. The developers of the standard did not wish to overload the meaning of – in this manner. The /dev/stdout concept exists on most modern systems. The /dev/stdout syntax does not refer to a new special file. It is just a magic cookie to specify standard output.

5.34 uuencode – Encode a binary file

Rationale: This change partially satisfies the following corrigendum request from ISO/IEC 9945-2: 1993 Annex H.2:

(14) The uuencode utility should support the BASE64-encoding specified in the MIME-RFC currently under consideration for Internet use. The uudecode utility should allow the user to override the output file name that is embedded in the file. Both utilities should be moved from Section 5 to Section 4.

⇒ **5.34.1 uuencode Synopsis.** *Change the Synopsis to be:*

uuencode **[**–m**]** **[***file***]** *decode_pathname*

⇒ **5.34.2 uuencode Description.** *Change the phrase* "the algorithm" *to* "one of the algorithms".

⇒ **5.34.3 uuencode Options.** *Change this subclause to:*

The uuencode utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following option shall be supported by the implementation:

−m Encode the output using the MIME Base64 algorithm described in 5.34.6.1.1. If −m is not specified, the historical algorithm described in 5.34.6.1.2 shall be used. C

⇒ **5.34.4 uuencode Operands.** *Change the description of decode_pathname to:*

decode_pathname
The pathname of the file into which the uudecode utility (see 5.33) shall place the decoded file. Specifying a *decode_pathname* operand of /dev/stdout shall indicate that uudecode is to use standard output. If there are characters in *decode_pathname* that are not in the portable filename character set (see Section 2.2.2.131 in POSIX.1 {8}), the results are unspecified.

⇒ **5.34.6.1 uuencode Standard Output.** *Replace this subclause with the following:*

5.34.6.1.1 uuencode Base64 Algorithm

The standard output shall be a text file (encoded in the character set of the current locale) that begins with the line:

"begin-base64∆%s∆%s\n", <*mode*>, *decode_pathname*

and ends with the line:

"====\n"

In both cases, the lines shall have no preceding or trailing <blank>s.

The encoding process represents 24 b groups of input bits as output strings of four encoded characters. Preceding from left to right, a 24 b input group shall be formed by concatenating three 8 b input groups. These 24 b then shall be treated as four concatenated 6 b groups, each of which shall be translated into a single digit in the base64 alphabet. When encoding a bit stream via the base64 encoding, the bit stream shall be presumed to be ordered with the most-significant-bit first. That is, the first bit in the stream shall be the high-order bit in the first byte, and the eighth bit shall be the low-order bit in the first byte, and so on.

Each 6 b group is used as an index into an array of 64 printable characters, as shown in Table 5-100.

Table 5-100 – uuencode Base64 Values

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

The character referenced by the index shall be placed in the output string.

The output stream (encoded bytes) shall be represented in lines of no more than 76 characters each. All line breaks or other characters not found in the table shall be ignored by decoding software (see uudecode in 5.33).

Special processing shall be performed if fewer than 24 b are available at the end of a message or encapsulated part of a message. A full encoding quantum shall be always completed at the end of a message. When fewer than 24 input bits are available in an input group, zero bits shall be added (on the right) to form an integral number of 6 b groups. Output character positions that are not required to represent actual input data shall be set to the character =. Since all base64 input is an integral number of octets, only the following cases can arise:

(1) The final quantum of encoding input is an integral multiple of 24 b; here, the final unit of encoded output shall be an integral multiple of 4 characters with no = padding.

(2) The final quantum of encoding input is exactly 8 b; here, the final unit of encoded output shall be two characters followed by two = padding characters.

(3) The final quantum of encoding input is exactly 16 b; here, the final unit of encoded output shall be three characters followed by one = padding character.

(4) The terminating ==== evaluates to nothing and denotes the end of the encoded data.

5.34.6.1.2 uuencode Historical Algorithm

<current contents of 5.34.6.1>

Editor's Note: The following rationale will be added to E.5.34, but is kept here with `uuencode` *for this draft:*

`uuencode` Rationale. *(This subclause is not a part of P1003.2b)*

A new algorithm was added at the request of the international community to parallel work in the Internet MIME RFC 2045 {B90}. As with the historical `uuencode` format, the Base64 Content-Transfer-Encoding is designed to represent arbitrary sequences of octets in a form that is not humanly readable. A 65-character subset of ISO/IEC 646 {1} is used, enabling 6 b to be represented per printable character. (The extra 65th character, =, is used to signify a special processing function.)

This subset has the important property that it is represented identically in all versions of ISO/IEC 646 {1}, including US ASCII, and all characters in the subset are also represented identically in all versions of EBCDIC. The historical `uuencode` algorithm does not share this property, which is the reason that a second algorithm was added to POSIX.2.

The string ==== was used for the termination instead of the `end` used in the original format because the latter is a string that could be valid encoded input.

In an early draft, the –`m` option was named –`b` (for Base64), but it was renamed to reflect its relationship to the Internet MIME RFC. A –`u` was also present to invoke the default algorithm, but since this was not historical practice, it was omitted as being unnecessary.

See the `uudecode` rationale for the derivation of the `/dev/stdout` symbol.

⇒ **5.35 `vi` – Screen-oriented (visual) display editor.** *Replace the entire* `vi` *clause with the following.*

Editor's Note: All of this clause has been changed in Draft 11 from the POSIX.2-1992 version. It is not further diffmarked. The rationale in Annex E is also completely replaced.

5.35 vi – Screen-oriented (visual) display editor

This utility shall be provided on systems that both support the User Portability Utilities Option and define the {POSIX2_CHAR_TERM} symbol. On other systems, it is optional.

5.35.1 Synopsis

`vi` **[**`–rR`**] [**`–c` *command***] [**`–t` *tagstring***] [**`–w` *size***] [***file* ...**]**

Obsolescent Version:

`vi` **[**`–rR`**] [***+command***] [**`–t` *tagstring***] [**`–w` *size***] [***file* ...**]**

5.35.2 Description

The `vi` (visual) utility is a screen-oriented text editor. Only the open and visual modes of the editor are described in this clause. See the line editor `ex` (5.10) for additional editing capabilities used in `vi`. The user can switch back and forth between `vi` and `ex`, and execute `ex` commands from within `vi`. C C

This clause uses the term "edit buffer" to describe the current working text. No specific implementation is implied by this term. All editing changes are performed on the edit buffer, and no changes to it shall affect any file until an editor command writes the file.

When using `vi`, the terminal screen acts as a "window" into the edit buffer. Changes made to the edit buffer shall be reflected in the screen display, and the position of the cursor on the screen shall indicate the current position within the edit buffer.

Certain terminals do not have all the capabilities necessary to support the complete `vi` definition. When these commands cannot be supported on such terminals, this condition shall neither produce an error message such as "not an editor command" nor report a syntax error. The implementation may either accept the commands and produce results on the screen that are the result of an unsuccessful attempt to meet the requirements of this standard or report an error describing the terminal-related deficiency.

5.35.3 Options

The `vi` utility shall conform to the utility argument syntax guidelines described in 2.10.2, except for the obsolescent *+command* "option."

The following options shall be supported by the implementation:

`–c` *command*
+command (Obsolescent.)
: See the `ex` command description of the `–c` and *+command* options (5.10.3).

`–r`
: See the `ex` command description of the `–r` option (5.10.3).

`–R`
: See the `ex` command description of the `–R` option (5.10.3).

`–t` *tagstring*
: See the `ex` command description of the `–t` option (5.10.3).

`–w` *size*
: See the `ex` command description of the `–w` option (5.10.3).

5.35.4 Operands

See the description of the operands supported by the `ex` command (5.10.4) for a description of the operands supported by the `vi` command.

5.35.5 External Influences

5.35.5.1 Standard Input

If standard input is not a terminal device, undefined results occur. The standard input consists of a series of commands and input text, as described in 5.35.7.

If a read from the standard input returns an error, or if the editor detects an end-of-file condition from the standard input, it shall be equivalent to a SIGHUP asynchronous event.

5.35.5.2 Input Files

See the description of the input files supported by the `ex` command (5.10.5.2) for a description of the input files supported by the `vi` command.

5.35.5.3 Environment Variables

See the description of the environment variables that affect the execution of the `ex` command (5.10.5.3) for a description of the environment variables that shall affect the `vi` command.

5.35.5.4 Asynchronous Events

See the description of the asynchronous events that affect the execution of the `ex` command (5.10.5.4) for a description of the asynchronous events that shall affect the `vi` command.

5.35.6 External Effects

5.35.6.1 Standard Output

If standard output is not a terminal device, undefined results occur.

Standard output may be used for writing prompts to the user, for informational and error messages, and for writing lines from the edit buffer.

5.35.6.2 Standard Error

If standard error is not a terminal device, undefined results occur.

Used only for diagnostic messages.

5.35.6.3 Output Files

See the description of the output files supported by the `ex` command (5.10.6.3) for a description of the output files supported by the `vi` command.

5.35.7 Extended Description

If the terminal does not have the capabilities necessary to support an unspecified portion of the `vi` definition, implementations shall start initially in `ex` mode (see 5.10) or open mode. Otherwise, after initialization, `vi` shall be in command mode; text input mode can be entered by one of several commands used to insert or change text. In text input mode, `<ESC>` can be used to return to command mode (see 5.35.7.3.9); other uses of `<ESC>` are described in 5.35.7.2.14.

5.35.7.1 `ex` and `vi` Initialization

See the description of `ex` and `vi` Initialization for the `ex` command (5.10.7.1) for a description of `ex` and `vi` Initialization for the `vi` utility.

5.35.7.2 `vi` Command Descriptions

The following symbols are used in this clause to represent arguments to commands.

buffer See the description of *buffer* in 5.10.7.5.

In open and visual mode, when a command synopsis shows both **[***buffer***]** and **[***count***]** preceding the command name, they can be specified in either order. C

count A positive integer used as an optional argument to most commands, normally to give a repeat count or as a size. This argument is optional and shall default to 1 unless otherwise specified.

The Synopsis lines for the `vi` commands `<control-G>`, `<control-L>`, `<control-R>`, `<control-]>`, `%`, `&`, `^`, `D`, `m`, `M`, `Q`, `u`, `U`, and `ZZ` do not have *count* as an optional argument. Regardless, it shall not be an error to specify a count to these commands, and any specified count shall be ignored.

motion An optional trailing argument used by the `!`, `<`, `>`, `c`, `d`, and `y` commands, which is used to indicate the region of text that shall be affected by the command. The motion can be either one of the command characters repeated or one of several other `vi` commands (listed in the following table). Each of the applicable commands specifies the region of text matched by repeating the command; each command that can be used as a motion command specifies the region of text it affects.

Commands that take *motion* arguments operate on either lines or characters, depending on the circumstances. When operating on lines, all lines that fall partially or wholly within the text region specified for the command shall be affected. When operating on characters, only the exact characters in the specified text region shall be affected. Each motion command specifies this individually.

When command that may be motion commands are not used as motion commands, they shall set the current position to the current line and column as specified.

The following shall be valid cursor motion commands:

`<control-H>`	`;`	`` ` ``*character*
`<newline>`	`?`	`b`
`<carriage-return>`	`B`	`e`
`<control-N>`	`E`	`f`
`<control-P>`	`F`	`h`
`<space>`	`G`	`j`
`$`	`H`	`k`
`%`	`L`	`l`
`'`*character*	`M`	`n`
`(`	`N`	`t`
`)`	`T`	`w`
`+`	`W`	`{`
`,`	`[[`	`|`
`-`	`]]`	`}`
`/`	`^`	`0`
`_`		

Any *count* that is specified to a command that has an associated motion command shall be applied to the motion command. If a *count* is applied to both the command and its associated motion command, the effect shall be multiplicative.

The following symbol is used in this clause to specify locations in the edit buffer:

current character
The character that is currently displayed by the cursor. C

The following symbols are used in this clause to specify command actions:

bigword
In the POSIX Locale, `vi` shall recognize four kinds of bigwords:

(1) A maximal sequence of nonblank characters preceded and followed by `<blank>` characters or the beginning or end of a line or the edit buffer

(2) One or more sequential empty or `<blank>`-filled lines

(3) The first character in the edit buffer

(4) The last character in the edit buffer

word
In the POSIX Locale, `vi` shall recognize five kinds of words:

(1) A maximal sequence of letters, digits and underscores, delimited at both ends by: characters other than letters, digits, or underscores; the beginning or end of a line; the beginning or end of the edit buffer

(2) A maximal sequence of characters other than letters, digits, underscores, or `<blank>`s, delimited at both ends by: a letter, digit, underscore, or `<blank>`s; the beginning or end of a line; the beginning or end of the edit buffer

(3) One or more sequential empty or `<blank>`-filled lines

(4) The first character in the edit buffer

(5) The last character in the edit buffer

section boundary

(1) A line whose first character is a `<form-feed>`

(2) A line whose first character is an open curly brace ({)

(3) A line whose first character is a period and whose second and third characters match a two-character pair in the `sections` edit option (see 5.10.7.8.17)

(4) A line whose first character is a period and whose only other character matches the first character of a two-character pair in the `sections` edit option, where the second character of the two-character pair is a `<space>`

(5) The first line of the edit buffer

(6) The last line of the edit buffer if the last line of the edit buffer is empty or if it is a `]]` or `}` command; otherwise, the last character of the last line of the edit buffer

paragraph boundary

(1) A section boundary

(2) A line whose first character is a period and whose second and third characters match a two-character pair in the `paragraphs` edit option (see 5.10.7.8.11)

(3) A line whose first character is a period and whose only other character matches the first character of a two-character pair in the `paragraphs` edit option, where the second character of the two-character pair is a `<space>`

(4) One or more sequential empty or `<blank>`-filled lines

remembered search direction

See the description of *remembered search direction* in 5.10.7.5.

sentence boundary

(1) A paragraph boundary

(2) The first nonblank character that occurs after a paragraph boundary

(3) The first nonblank character that occurs after a period (`.`), exclamation point (`!`) or question mark (`?`), followed by two `<space>`s or the end of a line; any number of closing parenthesis (`)`), closing brackets (`]`), double quote (`"`), or single quote (`'`) characters can appear between the punctuation mark and the two `<space>`s or end-of-line

Any lines displayed on the screen that logically represent lines after the last line in the edit buffer shall consist of a single tilde (`~`) character.

The last line of the screen shall be used to report errors or display informational messages. It shall also be used to display the input for "line-oriented commands" (`/`, `?`, `:`, and `!`). When a line-oriented command is executed, the editor shall enter text input mode on the last line on the screen, using the respective command characters as prompt characters. (In the case of the `!` command, the associated motion shall be entered by the user before the editor enters text input mode.) The line entered by the user shall be terminated by a `<newline>`, a non-`<control-V>`-escaped `<carriage-return>`, or unescaped `<ESC>`. It is unspecified if more characters than require a display width minus one column number of screen columns can be entered.

If any command is executed that overwrites a portion of the screen other than the last line of the screen, (e.g., the `ex suspend`, or `!` commands), other than the `ex shell` command, the user shall be prompted for a character before the screen is refreshed and the edit session continued.

Lines that are longer than the display shall be folded; the length at which folding occurs is unspecified, but should be appropriate for the output device. Folding may occur between glyphs of single characters that take up multiple display columns. By default, no `vi` command shall operate on screen lines; instead, all commands shall operate on physical lines, which may occupy one or more logical (i.e., display) lines. Unless otherwise specified, *line* refers to a physical line.

Tab characters shall take up the number of columns on the screen set by the `tabstop` edit option (see 5.10.7.8.23), unless there are less than that number of columns before the display margin that will cause the displayed line to be folded; in this case, they shall only take up the number of columns up to that boundary.

The cursor shall be placed on the current line and relative to the current column as specified by each command described in the following subclauses.

In open mode, if the current line is not already displayed, then it shall be displayed.

In visual mode, if the current line is not displayed, then the lines that are displayed shall be expanded, scrolled or redrawn to cause an unspecified portion of the current line to be displayed. If the screen is redrawn, no more than the number of logical lines specified by the value of th `window` edit option shall be displayed (unless the current line cannot be completely displayed in the number of logical lines specified by the `window` edit option) and the current line shall be positioned as close to the center of the displayed lines as possible (within the constraints imposed by the distance of the line from the beginning or end of the edit buffer). If the current line is before the first line in the display and the screen is scrolled, an unspecified portion of the current line shall be placed on the first line of the display. If the current line is after the last line in the display and the screen is scrolled, an unspecified portion of the current line shall be placed on the last line of the display.

In visual mode, if a line from the edit buffer (other than the current line) does not entirely fit into the lines at the bottom of the display that are available for its presentation, the editor may choose not to display any portion of the line. The lines of the display that do not contain text from the edit buffer for this reason shall each consist of a single @ character.

In visual mode, the editor may choose for unspecified reasons to not update lines in the display to correspond to the underlying edit buffer text. The lines of the display that do not correctly correspond to text from the edit buffer for this reason shall consist of a single @ character, and the `<control-R>` command shall cause the editor to update the screen to correctly represent the edit buffer.

Open and visual mode commands that set the current column set it to a column position in the display, and not a character position in the line. In this case, however, the column position in the display shall be calculated for a infinite width display; e.g., the column related to a character that is part of a line that has been folded onto additional screen lines will be offset from the screen column where the physical line begins, not from the beginning of a particular screen line.

The physical cursor column in the display is based on the value of the current column, as follows, with each rule applied in turn:

(1) If the current column is after the last screen column used by the displayed line, the physical cursor column shall be set to the last screen column occupied by the last character in the current line; otherwise, the physical cursor column shall be set to the current column.

(2) If the character of which some portion is displayed in the screen column specified by the physical cursor column requires more than a single

screen column:

(a) If in text input mode, the physical cursor column shall be adjusted to the first screen column in which any portion of that character is displayed.

(b) Otherwise, the physical cursor column shall be adjusted to the last screen column in which any portion of that character is displayed.

The current column shall not be changed by these adjustments to the physical cursor column.

If an error occurs during the parsing or execution of a `vi` command:

— The terminal shall be alerted. Execution of the `vi` command shall stop, and the cursor (e.g., the current line and column) shall not be further modified.

— Unless otherwise specified by the following command subclauses, it is unspecified if an informational message shall be displayed.

— Any partially entered `vi` command shall be discarded.

— If the `vi` command resulted from a map expansion, all characters from that map expansion shall be discarded, except as otherwise specified by the `map` command (see 5.10.7.5.14).

— If the `vi` command resulted from the execution of a buffer, no further commands caused by the execution of the buffer shall be executed.

The following subclauses describe command characters entered during command mode.

5.35.7.2.1 <control-B>

Synopsis: **[***count***]** `<control-B>`

If in open mode, the `<control-B>` command shall behave identically to the `z` command (see 5.35.7.2.84). Otherwise, if the current line is the first line of the edit buffer, it shall be an error.

If the `window` edit option is less than 3, display a screen where the last line of the display shall be some portion of:

(*current first line*) – 1

otherwise, display a screen where the first line of the display shall be some portion of:

(*current first line*) – *count* × ((`window` edit option) – 2)

If this calculation would result in a line that is before the first line of the edit buffer, the first line of the display shall display some portion of the first line of the edit buffer.

Current line: If no lines from the previous display remain on the screen, set to the last line of the display; otherwise, set to (*line – the number of new lines displayed on this screen*).

Current column: Set to nonblank.

5.35.7.2.2 `<control-D>`

Synopsis: **[***count***]** `<control-D>`

If the current line is the last line of the edit buffer, it shall be an error.

If no *count* is specified, *count* shall default to the *count* associated with the previous `<control-D>` or `<control-U>` command. If there was no previous `<control-D>` or `<control-U>` command, *count* shall default to the value of the `scroll` edit option.

If in open mode: Write lines starting with the line after the current line, until *count* lines or the last line of the file have been written.

Current line: If the current line + *count* is past the last line of the edit buffer, set to the last line of the edit buffer; otherwise, set to the current line + *count*.

Current column: Set to nonblank.

5.35.7.2.3 `<control-E>`

Synopsis: **[***count***]** `<control-E>`

Display the line *count* lines after the last line currently displayed.

If the last line of the edit buffer is displayed, it shall be an error. If there is no line *count* lines after the last line currently displayed, the last line of the display shall display some portion of the last line of the edit buffer.

Current line: Unchanged if the previous current character is displayed; otherwise, set to the first line displayed.

Current column: Unchanged.

5.35.7.2.4 `<control-F>`

Synopsis: **[***count***]** `<control-F>`

If in open mode, the `<control-F>` command shall behave identically to the `z` command (see 5.35.7.2.84).

Otherwise, if the current line is the last line of the edit buffer, it shall be an error.

If the `window` edit option is less than 3, display a screen where the first line of the display shall be some portion of:

(*current last line*) + 1

otherwise, display a screen where the first line of the display shall be some portion of:

(*current first line*) + *count* × ((`window` edit option) – 2)

If this calculation would result in a line that is after the last line of the edit buffer, the last line of the display shall display some portion of the last line of the edit buffer.

Current line: If no lines from the previous display remain on the screen, set to the first line of the display; otherwise, set to (line + the number of new lines displayed on this screen).

Current column: Set to nonblank.

5.35.7.2.5 `<control-G>`

Synopsis: `<control-G>`

This command shall be equivalent to the `ex file` command (see 5.10.7.5.9).

5.35.7.2.6 `<control-H>`

Synopsis: **[***count***]** `<control-H>`
Synopsis: **[***count***]** `h`
Synopsis: the current *erase* character (see `stty` in 4.59)

If there are no characters before the current character on the current line, it shall be an error. If there are less than *count* previous characters on the current line, *count* shall be adjusted to the number of previous characters on the line.

If used as a motion command:

(1) The text region shall be from the character before the starting cursor up to and including the *count*-th character before the starting cursor.

(2) Any text copied to a buffer shall be in character mode.

If not used as a motion command:

Current line: Unchanged.

Current column: Set to (*column* − the number of columns occupied by the *count* characters ending in the previous current column).

5.35.7.2.7 `<newline>`

Synopsis: **[***count***]** `<newline>`
Synopsis: **[***count***]** `<control-J>` C
Synopsis: **[***count***]** `<control-M>` C
Synopsis: **[***count***]** `<control-N>`
Synopsis: **[***count***]** `j`
Synopsis: **[***count***]** `<carriage-return>`
Synopsis: **[***count***]** `+`

If there are less than *count* lines after the current line in the edit buffer, it shall be an error.

If used as a motion command:

(1) The text region shall include the starting line and the next *count* − 1 lines.

(2) Any text copied to a buffer shall be in line mode.

If not used as a motion command:

Current line: Set to current line + *count*.

Current column: Set to nonblank for the `<carriage-return>`, `<control-M>`, and + commands; otherwise, unchanged. C C

5.35.7.2.8 `<control-L>`

Synopsis: `<control-L>`

If in open mode, clear the screen and redisplay the current line.

Otherwise, clear and redisplay the screen.

Current line: Unchanged.

Current column: Unchanged.

5.35.7.2.9 `<control-P>`

Synopsis: **[***count***]** `<control-P>`
Synopsis: **[***count***]** `k`
Synopsis: **[***count***]** `–`

If there are less than *count* lines before the current line in the edit buffer, it shall be an error.

If used as a motion command:

(1) The text region shall include the starting line and the previous *count* lines.

(2) Any text copied to a buffer shall be in line mode.

If not used as a motion command:

Current line: Set to current line – *count*.

Current column: Set to nonblank for the – command; otherwise, unchanged.

5.35.7.2.10 `<control-R>`

Synopsis: `<control-R>`

If any lines have been deleted from the logical screen in visual mode, and flagged as deleted on the terminal using the @ convention (see 5.35.7), they shall be redisplayed to match the contents of the edit buffer.

It is unspecified if lines flagged with @ because they do not fit on the terminal display shall be affected.

Current line: Unchanged.

Current column: Unchanged.

5.35.7.2.11 `<control-U>`

Synopsis: **[***count***]** `<control-U>`

If the current line is the first line of the edit buffer, it shall be an error.

If no *count* is specified, *count* shall default to the *count* associated with the previous `<control-D>` or `<control-U>` command. If there was no previous `<control-D>` or `<control-U>` command, *count* shall default to the value of the `scroll` edit option.

Current line: If *count* is greater than the current line, set to 1; otherwise, set to the current line – *count*.

Current column: Set to nonblank.

5.35.7.2.12 `<control-Y>`

Synopsis: **[***count***]** `<control-Y>`

C

Display the line *count* lines before the first line currently displayed.

If the current line is the first line of the edit buffer, it shall be an error. If this calculation would result in a line that is before the first line of the edit buffer, the first line of the display shall display some portion of the first line of the edit buffer.

Current line: Unchanged if the previous current character is displayed; otherwise, set to the first line displayed. C

Current column: Unchanged.

5.35.7.2.13 `<control-^>`

Synopsis: `<control-^>`

This command shall be equivalent to the `ex edit` command (see 5.10.7.5.8), with the alternate pathname as its argument.

5.35.7.2.14 `<ESC>`

Synopsis: `<ESC>`

If a partial `vi` command (as defined by at least one, non-*count* character) has been entered, discard the *count* and the command character(s). C

Otherwise, if no command characters have been entered, and the `<ESC>` was the result of a map expansion, the terminal shall be alerted and the `<ESC>` character shall be discarded, but it shall not be an error. C C C

Otherwise, it shall be an error. C

Current line: Unchanged.

Current column: Unchanged.

5.35.7.2.15 `<control-]>`

Synopsis: `<control-]>`

If the current character is not a word or `<blank>` character, it shall be an error.

This command shall be equivalent to the `ex tag` command (see 5.10.7.5.32), with the argument to that command defined as follows:

If the current character is a `<blank>`

(1) Skip all `<blank>` characters after the cursor up to the end of the line.

(2) If the end of the line is reached, it shall be an error.

Then, the argument to the `ex tag` command shall be the current character and all subsequent characters, up to the first nonword character or the end of the line.

5.35.7.2.16 `<space>`

Synopsis: **[***count***]** `<space>`
Synopsis: **[***count***]** `l` (ell)

If there are less than *count* characters after the cursor on the current line, *count* shall be adjusted to the number of characters after the cursor on the line.

If used as a motion command:

(1) If the current or *count*-th character after the cursor is the last character in the line, the text region shall be comprised of the current character up to and including the last character in the line. Otherwise, the text region shall be from the current character up to, but not including, the *count*-th character after the cursor.

(2) Any text copied to a buffer shall be in character mode.

If not used as a motion command:

If there are no characters after the current character on the current line, it shall be an error.

Current line: Unchanged.

Current column: Set to the last column that displays any portion of the *count*-th character after the current character.

5.35.7.2.17 `!`

Synopsis: **[***count***]** `!` *motion shell-command(s)* `<newline>`

If the *motion* command is the `!` command repeated:

(1) If the edit buffer is empty and no *count* was supplied, the command shall be the equivalent of the `ex :read !` command, with the text input, and no text shall be copied to any buffer.

(2) Otherwise:

(a) If there are less than *count* − 1 lines after the current line in the edit buffer, it shall be an error.

(b) The text region shall be from the current line up to and including the next *count* − 1 lines.

Otherwise, the text region shall be the lines in which any character of the text region specified by the motion command appear.

Any text copied to a buffer shall be in line mode.

This command shall be equivalent to the `ex` ! command (see 5.10.7.5.42) for the specified lines.

5.35.7.2.18 $

Synopsis: **[***count***]** $

It shall be an error if there are less than (*count* − 1) lines after the current line in the edit buffer.

If used as a motion command:

(1) If *count* is 1:

(a) It shall be an error if the line is empty.

(b) Otherwise, the text region shall consist of all characters from the starting cursor to the last character in the line, inclusive, and any text copied to a buffer shall be in character mode. c

(2) Otherwise, if the starting cursor position is at or before the first non-blank in the line, the text region shall consist of the current and the next *count* − 1 lines, and any text saved to a buffer shall be in line mode.

(3) Otherwise, the text region shall consist of all characters from the starting cursor to the last character in the line that is *count* − 1 lines forward from the current line, and any text copied to a buffer shall be in character mode.

If not used as a motion command:

Current line: Set to current line + *count* − 1.

Current column: The current column is set to the last screen column of the last character in the line, or column position 1 if the line is empty.

The current column shall be adjusted to be on the last screen column of the last character of the current line as subsequent commands change the current line, until a command changes the current column.

5.35.7.2.19 %

Synopsis: %

If the character at the current position is not a parenthesis, bracket, or curly brace, search forwards in the line to the first one of those characters. If no such character is found, it shall be an error.

The matching character shall be the parenthesis, bracket, or curly brace matching the parenthesis, bracket, or curly brace, respectively, that was at the current position or that was found on the current line.

Matching shall be determined as follows, for a open parenthesis:

(1) Set a counter to 1.

(2) Search forwards until a parenthesis is found or the end of the edit buffer is reached.

(3) If the end of the edit buffer is reached, it shall be an error.

(4) If a open parenthesis is found, increment the counter by 1.

(5) If a close parenthesis is found, decrement the counter by 1.

(6) If the counter is zero, the current character is the matching character.

Matching for a close parenthesis shall be equivalent, except that the search shall be backwards, from the starting character to the beginning of the buffer, a close parenthesis shall increment the counter by 1, and a open parenthesis shall decrement the counter by 1.

Matching for brackets and curly braces shall be equivalent, except that searching shall be done for open and close brackets or open and close curly braces.

It is implementation-defined if other characters are searched for and matched as well.

If used as a motion command:

(1) If the matching cursor was after the starting cursor in the edit buffer, and the starting cursor position was at or before the first nonblank in the starting line, and the matching cursor position was at or after the last nonblank in the matching line, the text region shall consist of the current line to the matching line, inclusive, and any text copied to a buffer shall be in line mode.

(2) If the matching cursor was before the starting cursor in the edit buffer, and the starting cursor position was at or after the last nonblank in the starting line, and the matching cursor position was at or before the first nonblank in the matching line, the text region shall consist of the current line to the matching line, inclusive, and any text copied to a buffer shall be in line mode.

(3) Otherwise, the text region shall consist of the starting character to the matching character, inclusive, and any text copied to a buffer shall be in character mode.

If not used as a motion command:

Current line: Set to the line where the matching character is located.

Current column: Set to the last column where any portion of the matching character is displayed.

5.35.7.2.20 &

Synopsis: &

This command shall be equivalent to the ex & command with the current line as C
its addresses, and without *options*, *count*, or *flags* (see 5.10.7.5.27). C

5.35.7.2.21 ′

Synopsis: ′ *character*

It shall be an error if the marked line is no longer in the edit buffer.

If used as a motion command:

(1) If the starting cursor is after the marked cursor, then the locations of the starting cursor and the marked cursor in the edit buffer shall be logically swapped.

(2) The text region shall consist of the starting line up to and including the marked line, and any text copied to a buffer shall be in line mode.

If not used as a motion command:

Current line: Set to the line referenced by the mark.

Current column: Set to nonblank.

5.35.7.2.22 ‘

Synopsis: ‘*character*

It shall be an error if the marked line is no longer in the edit buffer. If the marked line no longer contains a character in the saved numbered character position, it shall be as if the marked position is the first nonblank.

If used as a motion command:

(1) It shall be an error if the marked cursor references the same character in the edit buffer as the starting cursor.

(2) If the starting cursor is after the marked cursor, then the locations of the starting cursor and the marked cursor in the edit buffer shall be logically swapped.

(3) If the starting line is empty or the starting cursor is at or before the first nonblank character of the starting line, and the marked cursor line is empty or the marked cursor references the first character of the marked cursor line, the text region shall consist of all lines containing characters from the starting cursor to the line before the marked cursor line, inclusive, and any text copied to a buffer shall be in line mode.

(4) Otherwise, if the marked cursor line is empty or the marked cursor references a character at or before the first nonblank of the marked cursor line, the region of text shall be from the starting cursor to the last character of the line before the marked cursor line, inclusive, and any text copied to a buffer shall be in character mode.

(5) Otherwise, the region of text shall be from the starting cursor (inclusive), to the marked cursor (exclusive), and any text copied to a buffer shall be in character mode.

If not used as a motion command:

Current line: Set to the line referenced by the mark.

Current column: Set to the last column in which any portion of the character referenced by the mark is displayed.

5.35.7.2.23 [[

Synopsis: **[***count***]** [[

Move the cursor backward through the edit buffer to the first character of the previous section boundary, *count* times.

If used as a motion command:

(1) If the starting cursor was at the first character of the starting line or the starting line was empty, and the first character of the boundary was the first character of the boundary line, the text region shall consist of the current line up to and including the line where the *count*-th next boundary starts, and any text copied to a buffer shall be in line mode.

(2) If the boundary was the last line of the edit buffer or the last character of the last line of the edit buffer, the text region shall consist of the last character in the edit buffer up to and including the starting character, and any text saved to a buffer shall be in character mode.

(3) Otherwise, the text region shall consist of the starting character up to but not including the first character in the *count*-th next boundary, and any text copied to a buffer shall be in character mode.

If not used as a motion command:

Current line: Set to the line where the *count*-th next boundary in the edit buffer starts.

Current column: Set to the last column in which any portion of the first character of the *count*-th next boundary is displayed, or column position 1 if the line is empty.

5.35.7.2.24]]

Synopsis: **[***count***]**]]

Move the cursor forward through the edit buffer to the first character of the next section boundary, *count* times.

If used as a motion command:

(1) If the starting cursor was at the first character of the starting line or the starting line was empty, and the first character of the boundary was the first character of the boundary line, the text region shall consist of the current line up to and including the line where the *count*-th previous boundary starts, and any text copied to a buffer shall be in line mode.

(2) If the boundary was the first line of the edit buffer, the text region shall consist of the first character in the edit buffer up to but not including the starting character, and any text copied to a buffer shall be in character mode.

(3) Otherwise, the text region shall consist of the first character in the *count*-th previous section boundary up to but not including the starting character, and any text copied to a buffer shall be in character mode.

If not used as a motion command:

Current line: Set to the line where the *count*-th previous boundary in the edit buffer starts.

Current column: Set to the last column in which any portion of the first character of the *count*-th previous boundary is displayed, or column position 1 if the line is empty.

5.35.7.2.25 ˆ

Synopsis: ˆ

If used as a motion command:

(1) If the line has no nonblank characters, or if the cursor is at the first nonblank character of the line, it shall be an error.

(2) If the cursor is before the first nonblank character of the line, the text region shall be comprised of the current character, up to, but not including, the first nonblank character of the line.

(3) If the cursor is after the first nonblank character of the line, the text region shall be from the character before the starting cursor up to and including the first nonblank character of the line.

(4) Any text copied to a buffer shall be in character mode.

If not used as a motion command:

Current line: Unchanged.

Current column: Set to nonblank.

5.35.7.2.26 _

Synopsis: **[***count***]** _

If there are less than *count* − 1 lines after the current line in the edit buffer, it shall be an error.

If used as a motion command:

(1) If *count* is less than 2, the text region shall be the current line. C

(2) Otherwise, the text region shall include the starting line and the next *count* − 1 lines.

(3) Any text copied to a buffer shall be in line mode.

If not used as a motion command:

Current line: Set to current line + *count* − 1.

Current column: Set to nonblank.

5.35.7.2.27 (

Synopsis: **[***count***]** (

This command shall be equivalent to the [[command, with the exception that sentence boundaries shall be used instead of section boundaries.

5.35.7.2.28)

Synopsis: **[***count***]**)

This command shall be equivalent to the]] command, with the exception that sentence boundaries shall be used instead of section boundaries.

5.35.7.2.29 {

Synopsis: **[***count***]** {

This command shall be equivalent to the [[command, with the exception that paragraph boundaries shall be used instead of section boundaries.

5.35.7.2.30 }

Synopsis: **[***count***]** }

This command shall be equivalent to the]] command, with the exception that paragraph boundaries shall be used instead of section boundaries.

5.35.7.2.31 |

Synopsis: **[***count***]** |

For the purposes of this command, lines that are too long for the current display and that have been folded shall be treated as having a single, 1-based, number of columns.

If there are less than *count* columns in which characters from the current line are displayed on the screen, *count* shall be adjusted to be the last column in which any portion of the line is displayed on the screen.

If used as a motion character:

(1) If the line is empty, or the cursor character is the same as the character on the *count*-th column of the line, it shall be an error.

(2) If the cursor is before the *count*-th column of the line, the text region shall be comprised of the current character, up to but not including the character on the *count*-th column of the line.

(3) If the cursor is after the *count*-th column of the line, the text region shall be from the character before the starting cursor up to and including the character on the *count*-th column of the line.

(4) Any text copied to a buffer shall be in character mode.

If not used as a motion character:

Current line: Unchanged.

Current column: Set to the last column in which any portion of the character that is displayed in the *count* column of the line is displayed.

5.35.7.2.32 `,`

Synopsis: **[***count***]** `,`

If the last `F`, `f`, `T`, or `t` command was `F`, `f`, `T`, or `t`, this command shall be equivalent to an `f`, `F`, `t`, or `T` command, respectively, with the specified *count* and the same search character.

If there was no previous `F`, `f`, `T`, or `t` command, it shall be an error.

5.35.7.2.33 `.`

Synopsis: **[***count***]** `.`

Repeat the last `!`, `<`, `>`, `A`, `C`, `D`, `I`, `J`, `O`, `P`, `R`, `S`, `X`, `Y`, `a`, `c`, `d`, `i`, `o`, `p`, `r`, `s`, `x`, `y`, or `~` command. It shall be an error if none of these commands have been executed. Commands (other than commands that enter text input mode) executed as a result of map expansions, shall not change the value of the last repeatable command.

Repeated commands with associated motion commands shall repeat the motion command as well; however, any specified *count* shall replace the *count*(s) that were originally specified to the repeated command or its associated motion command.

If the motion component of the repeated command is `f`, `F`, `t`, or `T`, the repeated command shall not set the remembered search character for the `;` and `,` commands.

If the repeated command is `p` or `P`, and the buffer associated with that command was a numeric buffer named with a number less than 9, the buffer associated with the repeated command shall be set to be the buffer named by the name of the previous buffer logically incremented by 1.

If the repeated character is a text input command, the input text associated with that command is repeated literally:

— Input characters are neither macro or abbreviation expanded.

— Input characters are not interpreted in any special way with the exception that `<newline>` and `<carriage-return>` behave as described in 5.35.7.3.4, and `<control-T>` behaves as described in 5.35.7.3.5.

Current line: Set as described for the repeated command.

Current column: Set as described for the repeated command.

5.35.7.2.34 /

Synopsis: `/`

If the input line contains no characters, it shall be equivalent to a line containing only the last RE encountered. The enhanced REs supported by `vi` are described in `ex`; see 5.10.7.6.

Otherwise, the line shall be interpreted as one or more REs, optionally followed by an address offset or a `vi` `z` command.

If the RE is not the last RE on the line, or if a line offset or `z` command is specified, the RE shall be terminated by an unescaped `/` character, which shall not be used as part of the RE. If the RE is not the first RE on the line, it shall be preceded by zero or more `<blank>` characters, a semicolon, zero or more `<blank>` characters, and a leading `/` character, which shall not be interpreted as part of the RE. It shall be an error to precede any RE with any characters other than these.

Each search shall begin from the character after the first character of the last match (or, if it is the first search, after the cursor). If the `wrapscan` edit option is set, the search shall continue to the character before the starting cursor character; otherwise, to the end of the edit buffer. It shall be an error if any search fails to find a match, and an informational message to this effect shall be displayed.

An optional address offset (see 5.10.7.2) can be specified after the last RE by including a trailing `/` character after the RE and specifying the address offset. This offset will be from the line containing the match for the last RE specified. It shall be an error if the line offset would indicate a line address less than 1 or greater than the last line in the edit buffer. An address offset of zero shall be supported. It shall be an error to follow the address offset with any other characters than `<blank>`s.

If not used as a motion command, an optional `z` command (see 5.35.7.2.84) can be specified after the last RE by including a trailing `/` character after the RE, zero or more `<blank>` characters, a `z`, zero or more `<blank>` characters, an optional new `window` edit option value, zero or more `<blank>` characters, and a location character. The effect shall be as if the `z` command was executed after the `/` command(s). It shall be an error to follow the `z` command with any other characters than `<blank>`s.

The remembered search direction shall be set to forward.

If used as a motion command:

(1) It shall be an error if the last match references the same character in the edit buffer as the starting cursor.

(2) If any address offset is specified, the last match shall be adjusted by the specified offset as described previously.

(3) If the starting cursor is after the last match, then the locations of the starting cursor and the last match in the edit buffer shall be logically swapped.

(4) If any address offset is specified, the text region shall consist of all lines containing characters from the starting cursor to the last match line, inclusive, and any text copied to a buffer shall be in line mode.

(5) Otherwise, if the starting line is empty or the starting cursor is at or before the first nonblank character of the starting line, and the last match line is empty or the last match starts at the first character of the last match line, the text region shall consist of all lines containing characters from the starting cursor to the line before the last match line, inclusive, and any text copied to a buffer shall be in line mode.

(6) Otherwise, if the last match line is empty or the last match begins at a character at or before the first nonblank of the last match line, the region of text shall be from the current cursor to the last character of the line before the last match line, inclusive, and any text copied to a buffer shall be in character mode.

(7) Otherwise, the region of text shall be from the current cursor (inclusive), to the first character of the last match (exclusive), and any text copied to a buffer shall be be in character mode.

If not used as a motion command:

Current line: If a match is found, set to the last matched line plus the address offset, if any; otherwise, unchanged.

Current column: Set to the last column on which any portion of the first character in the last matched string is displayed, if a match is found, otherwise, unchanged.

5.35.7.2.35 0

Synopsis: `0` (zero)

The character `0` shall not be interpreted as a command if it is immediately preceded by a digit.

If used as a motion command:

(1) If the cursor character is the first character in the line, it shall be an error.

(2) The text region shall be from the character before the cursor character up to and including the first character in the line.

(3) Any text copied to a buffer shall be in character mode.

If not used as a motion command:

Current line: Unchanged.

Current column: The last column in which any portion of the first character in the line is displayed, or if the line is empty, unchanged.

5.35.7.2.36 **`:`**

Synopsis: `:`

Execute one or more `ex` command(s).

If any portion of the screen other than the last line of the screen was overwritten by any `ex` command (except `shell`), `vi` shall display a message indicating that it is waiting for an input from the user, and shall then read a character. This action may also be taken for other, unspecified reasons.

If the next character entered is a `:`, another `ex` command shall be accepted and executed. Any other character shall cause the screen to be refreshed and `vi` shall return to command mode.

Current line: As specified for the `ex` command(s).

Current column: As specified for the `ex` command(s).

5.35.7.2.37 **`;`**

Synopsis: **[***count***]** `;`

This command shall be equivalent to the last `F`, `f`, `T`, or `t` command, with the specified *count*, and with the same search character used for the last `F`, `f`, `T`, or `t` command.

If there was no previous `F`, `f`, `T`, or `t` command, it shall be an error.

5.35.7.2.38 **`<`**

Synopsis: **[***count***]** `<` *motion*

If the *motion* command is the < command repeated:

(1) If there are less than *count* − 1 lines after the current line in the edit buffer, it shall be an error.

(2) The text region shall be from the current line, up to and including the next *count* − 1 lines.

Shift any line in the text region specified by the *count* and motion command one `shiftwidth` (see 5.10.7.8.19) toward the start of the line, as described by the `ex` < command (see 5.10.7.5.42). The unshifted lines shall be copied to the unnamed buffer in line mode.

Current line: If the motion was from the current cursor position toward the end of the edit buffer, unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region specified by the motion command.

Current column: Set to nonblank.

5.35.7.2.39 >

Synopsis: **[***count***]** > *motion*

If the *motion* command is the > command repeated:

(1) If there are less than *count* − 1 lines after the current line in the edit buffer, it shall be an error.

(2) The text region shall be from the current line, up to and including the next *count* − 1 lines.

Shift any line with characters in the text region specified by the *count* and motion command one `shiftwidth` (see 5.10.7.8.19) away from the start of the line, as described by the `ex` > command (see 5.10.7.5.43). The unshifted lines shall be copied into the unnamed buffer in line mode.

Current line: If the motion was from the current cursor position toward the end of the edit buffer, unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region specified by the motion command.

Current column: Set to nonblank.

5.35.7.2.40 ?

Synopsis: ?

The ? command shall be equivalent to the / command (see 5.35.7.2.34) with the following exceptions:

(1) The input prompt shall be a ?.

(2) Each search shall begin from the character before the first character of the last match (or, if it is the first search, the character before the cursor character).

(3) The search direction shall be from the cursor toward the beginning of the edit buffer, and the `wrapscan` edit option shall affect whether the search wraps to the end of the edit buffer and continues.

C

(4) The remembered search direction shall be set to backward.

5.35.7.2.41 @

Synopsis: **[***count***]** *@buffer*

If the *buffer* is specified as @, the last buffer executed shall be used. If no previous buffer has been executed, it shall be an error.

Behave as if the contents of the named buffer were entered as standard input. After each line of a line-mode buffer, and all but the last line of a character mode buffer, behave as if a `<newline>` character were entered as standard input.

If an error occurs during this process, an error message shall be written, and no more characters resulting from the execution of this command shall be processed.

If a *count* is specified, behave as if that count were entered as user input before the characters from the @ buffer were entered.

Current line: As specified for the individual commands.

Current column: As specified for the individual commands.

5.35.7.2.42 ~

Synopsis: **[***count***]** **~**

Reverse the case of the current character and the next *count* – 1 characters, such that lowercase characters that have uppercase counterparts shall be changed to uppercase characters, and uppercase characters that have lowercase counterparts shall be changed to lowercase characters, as prescribed by the current locale. No other characters shall be affected by this command.

If there are less than *count* – 1 characters after the cursor in the edit buffer, *count* shall be adjusted to the number of characters after the cursor in the edit buffer minus 1.

For the purposes of this command, the next character after the last character on the line shall be the next character in the edit buffer.

Current line: Set to the line including the (*count* – 1)-th character after the cursor.

Current column: Set to the last column in which any portion of the (*count*–1)-th character after the cursor is displayed.

5.35.7.2.43 a

Synopsis: **[***count***]** `a`

Enter text input mode after the current cursor position. No characters already in the edit buffer shall be affected by this command. A *count* shall cause the input text to be appended *count* – 1 more times to the end of the input.

Current line/column: As specified for the text input commands; see 5.35.7.3.

5.35.7.2.44 A

Synopsis: **[***count***]** A

This command shall be equivalent to the vi commands $**[***count***]**a (see 5.35.7.2.43).

5.35.7.2.45 b

Synopsis: **[***count***]** b

With the exception that words are used as the delimiter instead of bigwords, this command shall be equivalent to the B command; see 5.35.7.2.46.

5.35.7.2.46 B

Synopsis: **[***count***]** B

If the edit buffer is empty or the cursor is on the first character of the edit buffer, it shall be an error. If less than *count* bigwords begin between the cursor and the start of the edit buffer, *count* shall be adjusted to the number of bigword beginnings between the cursor and the start of the edit buffer.

If used as a motion command:

(1) The text region shall be from the first character of the *count*-th previous bigword beginning up to but not including the cursor character.

(2) Any text copied to a buffer shall be in character mode.

If not used as a motion command:

Current line: Set to the line containing the *current column*.

Current column: Set to the last column upon which any part of the first character of the *count*-th previous bigword is displayed.

5.35.7.2.47 c

Synopsis: **[***buffer***]** **[***count***]** c *motion*

If the *motion* command is the c command repeated:

(1) The buffer text shall be in line mode.

(2) If there are less than *count* – 1 lines after the current line in the edit buffer, it shall be an error.

(3) The text region shall be from the current line up to and including the next *count* – 1 lines.

Otherwise, the buffer text mode and text region shall be as specified by the motion command. c

The replaced text shall be copied into buffer, if specified, and into the unnamed buffer. If the text to be replaced contains characters from more than a single line, or the buffer text is in line mode, the replaced text shall be copied into the numeric buffers as well.

If the buffer text is in line mode:

(1) Any lines that contain characters in the region shall be deleted, and the editor shall enter text input mode at the beginning of a new line which shall replace the first line deleted.

(2) If the `autoindent` edit option is set, autoindent characters equal to the autoindent characters on the first line deleted shall be inserted as if entered by the user.

Otherwise, if characters from more than one line are in the region of text:

(1) The text shall be deleted.

(2) Any text remaining in the last line in the text region shall be appended to the first line in the region, and the last line in the region shall be deleted.

(3) The editor shall enter text input mode after the last character not deleted from the first line in the text region, if any; otherwise, on the first column of the first line in the region.

Otherwise:

(1) If the glyph for `$` is smaller than the region, the end of the region shall be marked with a `$`. C

(2) The editor shall enter text input mode, overwriting the region of text. C

Current line/column: As specified for the text input commands; see 5.35.7.3.

5.35.7.2.48 C

Synopsis: **[***buffer***] [***count***]** `C`

This command shall be equivalent to the `vi` command **[***buffer***] [***count***]** `c$` (see 5.35.7.2.47).

5.35.7.2.49 d

Synopsis: **[***buffer***] [***count***]** `d` *motion*

If the *motion* command is the `d` command repeated:

(1) The buffer text shall be in line mode.

(2) If there are less than *count* – 1 lines after the current line in the edit buffer, it shall be an error.

(3) The text region shall be from the current line up to and including the next *count* – 1 lines.

Otherwise, the buffer text mode and text region shall be as specified by the motion command. C

If in open mode, and the current line is deleted, and the line remains on the display, an `@` character shall be displayed as the first glyph of that line. C

Delete the region of text into buffer, if specified, and into the unnamed buffer. If the text to be deleted contains characters from more than a single line, or the buffer text is in line mode, the deleted text shall be copied into the numeric buffers, as well.

Current line: Set to the first text region line that appears in the edit buffer, unless that line has been deleted, in which case it shall be set to the last line in the edit buffer, or line 1 if the edit buffer is empty.

Current column:

(1) If the line is empty, set to column position 1.

(2) Otherwise, if the buffer text is in line mode or the motion was from the cursor toward the end of the edit buffer:

  (a) If a character from the current line is displayed in the current column, set to the last column that displays any portion of that character. C

  (b) Otherwise, set to the last column in which any portion of any character in the line is displayed.

(3) Otherwise, if a character is displayed in the column that began the text region, set to the last column that displays any portion of that character.

(4) Otherwise, set to the last column in which any portion of any character in the line is displayed.

5.35.7.2.50 `D`

Synopsis: **[***buffer***]** `D`

This command shall be equivalent to the `vi` command **[***buffer***]** `d$` (see 5.35.7.2.49). C

5.35.7.2.51 `e`

Synopsis: **[***count***]** `e`

With the exception that words are used instead of bigwords as the delimiter, this command shall be equivalent to the `E` command; see 5.35.7.2.52.

5.35.7.2.52 `E`

Synopsis: **[***count***]** `E`

If the edit buffer is empty it shall be an error. If less than *count* bigwords end between the cursor and the end of the edit buffer, *count* shall be adjusted to the number of bigword endings between the cursor and the end of the edit buffer.

If used as a motion command:

(1) The text region shall be from the last character of the *count*-th next bigword up to and including the cursor character.

(2) Any text copied to a buffer shall be in character mode.

If not used as a motion command:

Current line: Set to the line containing the *current column*.

Current column: Set to the last column upon which any part of the last character of the *count*-th next bigword is displayed.

5.35.7.2.53 `f`

Synopsis: **[***count***]** `f` *character*

It shall be an error if *count* occurrences of the character do not occur after the cursor in the line.

If used as a motion command:

(1) The text range shall be from the cursor character up to and including the *count*-th occurrence of the specified character after the cursor.

(2) Any text copied to a buffer shall be in character mode.

If not used as a motion command:

Current line: Unchanged.

Current column: Set to the last column in which any portion of the *count*-th occurrence of the specified character after the cursor appears in the line.

5.35.7.2.54 `F`

Synopsis: **[***count***]** `F` *character*

It shall be an error if *count* occurrences of the character do not occur before the cursor in the line.

If used as a motion command:

(1) The text region shall be from the *count*-th occurrence of the specified character before the cursor, up to, but not including the cursor character.

(2) Any text copied to a buffer shall be in character mode.

If not used as a motion command:

Current line: Unchanged.

Current column: Set to the last column in which any portion of the *count*-th occurrence of the specified character before the cursor appears in the line.

5.35.7.2.55 `G`

Synopsis: **[***count***]** `G`

If *count* is not specified, it shall default to the last line of the edit buffer.

If *count* is greater than the last line of the edit buffer, it shall be an error.

If used as a motion command:

(1) The text region shall be from the cursor line up to and including the specified line.

(2) Any text copied to a buffer shall be in line mode.

If not used as a motion command:

Current line: Set to *count*.

Current column: Set to nonblank.

5.35.7.2.56 `H`

Synopsis: **[***count***]** `H`

If the beginning of the line count greater than the first line of which any portion appears on the display does not exist, it shall be an error.

If used as a motion command:

(1) If in open mode, the text region shall be the current line.

(2) Otherwise, the text region shall be from the starting line up to and including (the first line of the display + *count* – 1).

(3) Any text copied to a buffer shall be in line mode.

If not used as a motion command:

If in open mode, this command shall set the current column to nonblank and do nothing else.

Otherwise, it shall set the current line and current column as follows:

Current line: Set to (the first line of the display + *count* – 1).

Current column: Set to nonblank.

5.35.7.2.57 `i`

Synopsis: **[***count***]** `i`

Enter text input mode before the current cursor position. No characters already in the edit buffer shall be affected by this command. A *count* shall cause the input text to be appended *count* – 1 more times to the end of the input.

Current line/column: As specified for the text input commands; see 5.35.7.3.

5.35.7.2.58 `I`

Synopsis: **[***count***]** `I`

This command shall be equivalent to the `vi` commands `^`**[***count***]**`i` (see 5.35.7.2.57).

5.35.7.2.59 `J`

Synopsis: **[***count***]** `J`

If the current line is the last line in the edit buffer, it shall be an error.

This command shall be equivalent to the `ex join` command (see 5.10.7.5.12) with no addresses, and an `ex` command *count* value of 1 if *count* was not specified or if a *count* of 1 was specified, and an `ex` command *count* value of *count* − 1 for any other value of *count*, except that the current line and column shall be set as follows:

Current line: Unchanged.

Current column: The last column in which any portion of the character following the last character in the initial line is displayed, or the last character in the line if no characters were appended. C C

5.35.7.2.60 `L`

Synopsis: **[***count***]** `L`

If the beginning of the line count less than the last line of which any portion appears on the display does not exist, it shall be an error. C C

If used as a motion command:

(1) If in open mode, the text region shall be the current line.

(2) Otherwise, the text region shall include all lines from the starting cursor line to (the last line of the display − (*count* − 1)).

(3) Any text copied to a buffer shall be in line mode.

If not used as a motion command:

If in open mode, this command shall set the current column to nonblank and do nothing else.

Otherwise, it shall set the current line and current column as follows:

Current line: Set to (the last line of the display − (*count* − 1)).

Current column: Set to nonblank.

5.35.7.2.61 `m`

Synopsis: `m` *character*

This command shall be equivalent to the `ex mark` command (see 5.10.7.5.15) with the specified character as an argument.

5.35.7.2.62 `M`

Synopsis: `M`

The middle line of the display shall be calculated as follows:

(*the top line of the display*) + (((*number of lines displayed*) + 1) / 2) – 1

If used as a motion command:

(1) If in open mode, the text region shall be the current line.

(2) Otherwise, the text region shall include all lines from the starting cursor line up to and including the middle line of the display.

(3) Any text copied to a buffer shall be in line mode.

If not used as a motion command:

If in open mode, this command shall set the current column to nonblank and do nothing else.

Otherwise, it shall set the current line and current column as follows:

Current line: Set to the middle line of the display.

Current column: Set to nonblank.

5.35.7.2.63 `n`

Synopsis: `n`

If the remembered search direction was forward, the `n` command shall be equivalent to the `vi` `/` command with no characters entered by the user (see 5.35.7.2.34). Otherwise, it shall be equivalent to the `vi` `?` command with no characters entered by the user (see 5.35.7.2.40).

If the `n` command is used as a motion command for the `!` command, the editor shall not enter text input mode on the last line on the screen, and shall behave as if the user entered a single `!` character as the text input.

5.35.7.2.64 `N`

Synopsis: `N`

If the remembered search direction was forward, the `N` command shall be equivalent to the `vi` `?` command with no characters entered by the user (see 5.35.7.2.40.) Otherwise, it shall be equivalent to the `vi` `/` command with no characters entered by the user (see 5.35.7.2.34).

If the `N` command is used as a motion command for the `!` command, the editor shall not enter text input mode on the last line on the screen, and shall behave as if the user entered a single `!` character as the text input.

5.35.7.2.65 o

Synopsis: **[***count***]** o

Enter text input mode in a new line appended after the current line. A *count* shall cause the input text to be appended *count* − 1 more times to the end of the already added text, each time starting on a new, appended line.

Current line/column: As specified for the text input commands; see 5.35.7.3.

5.35.7.2.66 O

Synopsis: **[***count***]** O

Enter text input mode in a new line inserted before the current line. A *count* shall cause the input text to be appended *count* − 1 more times to the end of the already added text, each time starting on a new, appended line.

Current line/column: As specified for the text input commands; see 5.35.7.3.

5.35.7.2.67 p

Synopsis: **[***buffer***]** **[***count***]** p

If no *buffer* is specified, the unnamed buffer shall be used.

If the buffer text is in line mode, the text shall be appended below the current line, and each line of the buffer shall become a new line in the edit buffer. A *count* shall cause the buffer text to be appended *count* − 1 more times to the end of the already added text, each time starting on a new, appended line.

If the buffer text is in character mode, the text shall be appended into the current line after the cursor, and each line of the buffer other than the first and last shall become a new line in the edit buffer. A *count* shall cause the buffer text to be appended *count* − 1 more times to the end of the already added text, each time starting after the last added character.

Current line: If the buffer text is in line mode, set the line to line + 1; otherwise, unchanged.

Current column:

If the buffer text is in line mode:

(1) If there is a nonblank character in the first line of the buffer, set to the last column on which any portion of the first nonblank character in the line is displayed.

(2) If there is no nonblank character in the first line of the buffer, set to the last column on which any portion of the last character in the first line of the buffer is displayed.

If the buffer text is in character mode:

(1) If the text in the buffer is from more than a single line, then set to the last column on which any portion of the first character from the buffer is displayed.

(2) Otherwise, if the buffer is the unnamed buffer, set to the last column on which any portion of the last character from the buffer is displayed.

(3) Otherwise, set to the first column on which any portion of the first character from the buffer is displayed.

5.35.7.2.68 P

Synopsis: **[***buffer***]** **[***count***]** `P`

If no *buffer* is specified, the unnamed buffer shall be used. C

If the buffer text is in line mode, the text shall be inserted above the current line, and each line of the buffer shall become a new line in the edit buffer. A *count* shall cause the buffer text to be appended *count* − 1 more times to the end of the already added text, each time starting on a new, appended line.

If the buffer text is in character mode, the text shall be inserted into the current line before the cursor, and each line of the buffer other than the first and last shall become a new line in the edit buffer. A *count* shall cause the buffer text to be appended *count* − 1 more times to the end of the already added text, each time starting after the last added character.

Current line: Unchanged.

Current column:

If the buffer text is in line mode:

(1) If there is a nonblank character in the first line of the buffer, set to the last column on which any portion of that character is displayed.

(2) If there is no nonblank character in the first line of the buffer, set to the last column on which any portion of the last character in the first line of the buffer is displayed.

If the buffer text is in character mode:

(1) If the buffer is the unnamed buffer, set to the last column on which any portion of the last character from the buffer is displayed.

(2) Otherwise, set to the first column on which any portion of the first character from the buffer is displayed.

5.35.7.2.69 Q

Synopsis: `Q`

Leave visual or open mode and enter `ex` command mode.

Current line: Unchanged.

Current column: Unchanged.

5.35.7.2.70 r

Synopsis: **[***count***]** `r` *character*

Replace the *count* characters at and after the cursor with the specified character. If there are less than *count* characters at and after the cursor on the line, it shall be an error.

If character is `<control-V>`, any next character other than `<newline>` shall be stripped of any special meaning and used as a literal character.

If character is `<ESC>`, no replacement shall be made and the current line and current column shall be unchanged.

If character is `<carriage-return>` or `<newline>`, *count* new lines shall be appended to the current line. All but the last of these lines shall be empty. *Count* characters at and after the cursor shall be discarded, and any remaining characters after the cursor in the current line shall be moved to the last of the new lines. If the `autoindent` edit option is set, they shall be preceded by the same number of autoindent characters found on the line from which the command was executed. C C

Current line: Unchanged unless the replacement character is a `<carriage-return>` or `<newline>`, in which case it shall be set to line + *count*.

Current column: Set to the last column position on which a portion of the last replaced character is displayed, or if the replacement character caused new lines to be created, set to nonblank C

5.35.7.2.71 R

Synopsis: **[***count***]** `R`

Enter text input mode at the current cursor position. A *count* shall cause the input text to be appended *count* – 1 more times to the end of the input.

Current line/column: As specified for the text input commands; see 5.35.7.3.

5.35.7.2.72 s

Synopsis: **[***buffer***]** **[***count***]** `s`

This command shall be equivalent to the `vi` command **[***buffer***]** **[***count***]** `cl` (see 5.35.7.2.47).

5.35.7.2.73 S

Synopsis: **[***buffer***]** **[***count***]** `S`

This command shall be equivalent to the `vi` command **[***buffer***]** **[***count***]** `c_` (see 5.35.7.2.47).

5.35.7.2.74 t

Synopsis: **[***count***]** t *character*

It shall be an error if *count* occurrences of the character do not occur after the cursor in the line.

If used as a motion command:

(1) The text region shall be from the cursor up to but not including the *count*-th occurrence of the specified character after the cursor.

(2) Any text copied to a buffer shall be in character mode.

If not used as a motion command:

Current line: Unchanged.

Current column: Set to the last column in which any portion of the character before the *count*-th occurrence of the specified character after the cursor appears in the line.

5.35.7.2.75 T

Synopsis: **[***count***]** T *character*

It shall be an error if *count* occurrences of the character do not occur before the cursor in the line.

If used as a motion command:

(1) If the character before the cursor is the specified character, it shall be an error.

(2) The text region shall be from the character before the cursor up to but not including the *count*-th occurrence of the specified character before the cursor.

(3) Any text copied to a buffer shall be in character mode.

If not used as a motion command:

Current line: Unchanged.

Current column: Set to the last column in which any portion of the character after the *count*-th occurrence of the specified character before the cursor appears in the line.

5.35.7.2.76 u

Synopsis: u

This command shall be equivalent to the ex undo command (see 5.10.7.5.34), except that the current line and current column shall be set as follows:

Current line:

Set to the first line added or changed if any; otherwise, move to the line preceding any deleted text if one exists; otherwise, move to line 1. C C

Current column:

If undoing an `ex` command, set to the first nonblank.

Otherwise, if undoing a text input command:

(1) If the command was an `C`, `c`, `O`, `o`, `R`, `S`, or `s` command, the current column shall be set to the value it held when the text input command was entered.

(2) Otherwise, set to the last column in which any portion of the first character after the deleted text is displayed, or, if no characters follow the text deleted from this line, set to the last column in which any portion of the last character in the line is displayed, or 1 if the line is empty.

Otherwise, if a single line was modified (i.e., not added or deleted) by the `u` command:

(1) If text was added or changed, set to the last column in which any portion of the first character added or changed is displayed.

(2) If text was deleted, set to the last column in which any portion of the first character after the deleted text is displayed, or, if no characters follow the deleted text, set to the last column in which any portion of the last character in the line is displayed, or 1 if the line is empty.

Otherwise, set to nonblank.

5.35.7.2.77 U

Synopsis: `U`

Restore the current line to its state immediately before the most recent time that it became the current line.

Current line: Unchanged.

Current column: Set to the first column in the line in which any portion of the first character in the line is displayed.

5.35.7.2.78 w

Synopsis: **[***count***]** `w`

With the exception that words are used as the delimiter instead of bigwords, this command shall be equivalent to the `W` command; see 5.35.7.2.79.

5.35.7.2.79 W

Synopsis: **[***count***]** W

If the edit buffer is empty, it shall be an error. If there are less than *count* bigwords between the cursor and the end of the edit buffer, *count* shall be adjusted to move the cursor to the last bigword in the edit buffer.

If used as a motion command:

(1) If the associated command is c, *count* is 1, and the cursor is on a <blank> character, the region of text shall be the current character and no further action shall be taken.

(2) If there are less than *count* bigwords between the cursor and the end of the edit buffer, then the command shall succeed, and the region of text shall include the last character of the edit buffer.

(3) If there are <blank> characters or an end-of-line that precede the *count*-th bigword, and the associated command is c, the region of text shall be up to and including the last character before the preceding <blank> characters or end-of-line.

(4) If there are <blank> characters or an end-of-line that precede the bigword, and the associated command is d or y, the region of text shall be up to and including the last <blank> character before the start of the bigword or end-of-line.

(5) Any text copied to a buffer shall be in character mode.

If not used as a motion command:

If the cursor is on the last character of the edit buffer, it shall be an error.

Current line: Set to the line containing the *current column*.

Current column: Set to the last column in which any part of the first character of the *count*-th next bigword is displayed.

5.35.7.2.80 x

Synopsis: **[***buffer***]** **[***count***]** x

Delete the *count* characters at and after the current character into buffer, if specified, and into the unnamed buffer.

If the line is empty, it shall be an error. If there are less than *count* characters at and after the cursor on the current line, *count* shall be adjusted to the number of characters at and after the cursor.

Current line: Unchanged.

Current column: If the line is empty, set to column position 1. Otherwise, if there were *count* or less characters at and after the cursor on the current line, set to the last column that displays any part of the last character of the line. Otherwise, unchanged.

5.35.7.2.81 X

Synopsis: **[***buffer***] [***count***]** `X`

Delete the *count* characters before the current character into *buffer*, if specified, and into the unnamed buffer.

If there are no characters before the current character on the current line, it shall be an error. If there are less than *count* previous characters on the current line, *count* shall be adjusted to the number of previous characters on the line.

Current line: Unchanged.

Current column: Set to (*current column − the width of the deleted characters*).

5.35.7.2.82 y

Synopsis: **[***buffer***] [***count***]** `y` *motion*

Copy the region of text into buffer, if specified, and into the unnamed buffer.

If the *motion* command is the `y` command repeated:

(1) The buffer shall be in line mode.

(2) If there are less than *count* − 1 lines after the current line in the edit buffer, it shall be an error.

(3) The text region shall be from the current line up to and including the next *count* − 1 lines.

Otherwise, the buffer text mode and text region shall be as specified by the motion command. C

Current line: If the motion was from the current cursor position toward the end of the edit buffer, unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region specified by the *motion* command.

Current column:

(1) If the motion was from the current cursor position toward the end of the edit buffer, unchanged.

(2) Otherwise, if the current line is empty, set to column position 1.

(3) Otherwise, set to the last column that displays any part of the first character in the file that is part of the text region specified by the *motion* command.

5.35.7.2.83 Y

Synopsis: **[***buffer***] [***count***]** `Y`

This command shall be equivalent to the `vi` command **[***buffer***] [***count***]** `y_` (see 5.35.7.2.82).

5.35.7.2.84 `z`

If in open mode, the `z` command shall have a Synopsis of:

Synopsis: **[***count***]** `z`

If *count* is not specified, it shall default to the `window` edit option − 1. The `z` command shall be equivalent to the `ex` `z` command, with a type character of “=” and a *count* of *count*−2, except that the current line and current column shall be set as follows, and the window edit option shall not be affected. If the calculation for the *count* argument would result in a negative number, the *count* argument to the `ex` `z` command shall be zero. A blank line shall be written after the last line is written.

Current line: Unchanged.

Current column: Unchanged.

If not in open mode, the `z` command shall have a Synopsis of:

Synopsis: **[***line***]** `z` **[***count***]** *character*

If *line* is not specified, it shall default to the current line. If *line* is specified, but is greater than the number of lines in the edit buffer, it shall default to the number of lines in the edit buffer.

If *count* is specified, the value of the `window` edit option shall be set to *count* (as described in 5.10.7.8.29), and the screen shall be redrawn.

Line shall be placed as specified by the following characters:

`<newline>`
`<carriage-return>`
Place the beginning of the line on the first line of the display.

`.` Place the beginning of the line in the center of the display. The middle line of the display shall be calculated as described for the `M` command (see 5.35.7.2.62).

`-` Place an unspecified portion of the line on the last line of the display.

`+` If *line* was specified, equivalent to the `<newline>` case. If *line* was not specified, display a screen where the first line of the display shall be (current last line) + 1. If there are no lines after the last line in the display, it shall be an error.

`^` If *line* was specified, display a screen where the last line of the display shall contain an unspecified portion of the first line of a display that had an unspecified portion of the specified line on the last line of the display. If this calculation results in a line before the beginning of the edit buffer, display the first screen of the edit buffer.

Otherwise, display a screen where the last line of the display shall contain an unspecified portion of (current first line − 1). If this calculation results in a line before the beginning of the edit buffer, it shall be an error.

Current line:

If *line* and the ˆ character were specified:

(1) If the first screen was displayed as a result of the command attempting to display lines before the beginning of the edit buffer:

If the first screen was already displayed, unchanged; otherwise, set to (current first line – 1).

(2) Otherwise, set to the last line of the display.

If *line* and the + character were specified, set to the first line of the display.

Otherwise, if *line* was specified, set to *line*.

Otherwise, unchanged. *Current column*: Set to nonblank.

5.35.7.2.85 `ZZ`

Synopsis: `ZZ`

This command shall be equivalent to the `ex` `xit` command with no addresses, trailing `!`, or file name (see 5.10.7.5.39).

5.35.7.3 Input Mode Commands

In text input mode, the current line shall consist of zero or more of the following categories:

(1) Characters preceding the text input entry point:

Characters in this category shall not be modified during text input mode.

(2) Autoindent characters:

Autoindent characters shall be automatically inserted into each line that is created in text input mode, either as a result of entering a `<newline>` or `<carriage-return>` while in text input mode, or as an effect of the command itself; e.g., `O` or `o` (see 5.10.7.8.1), as if entered by the user.

It shall be possible to erase autoindent characters with the `<control-D>` command (see 5.35.7.3.2); it is unspecified if they can be erased by `<control-H>`, `<control-U>`, and `<control-W>` characters (see 5.35.7.3.3, 5.35.7.3.6, and 5.35.7.3.8). Erasing any autoindent character turns the glyph into erase-columns and deletes the character from the edit buffer, but does not change its representation on the screen.

(3) Text input characters:

Text input characters are the characters entered by the user. Erasing (see 5.35.7.3.3, 5.35.7.3.6, and 5.35.7.3.8) any text input character turns the glyph into erase-columns and deletes the character from the edit buffer, but does not change its representation on the screen.

Each text input character entered by the user (that does not have a special meaning) shall be treated as follows:

(a) The text input character shall be appended to the last character in the edit buffer from the first, second, or third categories.

(b) If there are no erase-columns on the screen, the text input command was the `R` command, and characters in the fifth category from the original line follow the cursor, the next such character shall be deleted from the edit buffer. If the `slowopen` edit option is not set, the corresponding glyph on the screen shall become erase-columns.

(c) If there are erase-columns on the screen, as many columns as they occupy, or as are necessary, shall be overwritten to display the text input character. (If only part of a multicolumn glyph is overwritten, the remainder shall be left on the screen, and continue to be treated as erase-columns; it is unspecified if the remainder of the glyph is modified in any way.)

(d) If additional screen columns are needed to display the text input character:

[1] If the `slowopen` edit option is set, the text input characters shall be displayed on subsequent screen columns, overwriting any characters displayed in those columns.

[2] Otherwise, any characters currently displayed on or after the column on the screen where the text input character is to be displayed shall be pushed ahead the number of screen columns necessary to display the rest of the text input character.

(4) Erase-columns:

Erase-columns are not logically part of the edit buffer, appearing only on the screen, and may be overwritten on the screen by subsequent text input characters. When text input mode ends, all erase-columns shall no longer appear on the screen.

Erase-columns are initially the region of text specified by the `c` command (see 5.35.7.2.47); however, erasing autoindent or text input characters causes the glyphs of the erased characters to be treated as erase-columns.

(5) Characters following the text region for the `c` command, or the text input entry point for all other commands:

Characters in this category shall not be modified during text input mode, except as specified in category (3b) for the `R` text input command, or as `<blank>` characters deleted when a `<newline>` or `<carriage-return>` is entered (see 5.35.7.3.4). C

It is unspecified if it is an error to attempt to erase past the beginning of a line that was created by the entry of a `<newline>` or `<carriage-return>` character during text input mode. If it is not an error, the editor shall behave as if the erasing character was entered immediately after the last text input character entered on the previous line, and all of the characters on the current line shall be treated as erase-columns.

When text input mode is entered, or after a text input mode character is entered (except as specified for the special characters below), the cursor shall be positioned as follows:

(1) On the first column that displays any part of the first erase-column, if one exists.

(2) Otherwise, if the `slowopen` edit option is set, on the first screen column after the last character in the first, second, or third categories, if one exists.

(3) Otherwise, the first column that displays any part of the first character in the fifth category, if one exists.

(4) Otherwise, the screen column after the last character in the first, second, or third categories, if one exists.

(5) Otherwise, on column position 1.

The characters that are updated on the screen during text input mode are unspecified, other than that the last text input character shall always be updated, and, if the `slowopen` edit option is not set, the current cursor character shall always be updated.

The following specifications are for command characters entered during text input mode.

5.35.7.3.1 NUL

Synopsis: NUL

If the first character of the text input is a NUL, the most recently input text shall be input as if entered by the user, and then text input mode shall be exited. The text shall be input literally; i.e., characters are neither macro or abbreviation expanded, nor are any characters interpreted in any special manner. It is C
unspecified if implementations shall support more than 256 bytes of remembered C
input text. C

5.35.7.3.2 `<control-D>`

Synopsis: `<control-D>`

The `<control-D>` character shall have no special meaning when in text input mode for a line-oriented command (see 5.35.7.2).

This command need not be supported on block-mode terminals.

C

If the cursor does not follow an autoindent character, or an autoindent character and a `0` or `^` character:

(1) If the cursor is in column position 1, the `<control-D>` character shall be discarded and no further action taken.

(2) Otherwise, the `<control-D>` character shall have no special meaning.

If the last input character was a `0`, the cursor shall be moved to column position 1.

Otherwise, if the last input character was a `^`, the cursor shall be moved to column position 1. In addition, the `autoindent` level for the next input line shall be derived from the same line from which the `autoindent` level for the current input line was derived.

Otherwise, the cursor shall be moved back to the column after the previous `shiftwidth` (see 5.10.7.8.19) boundary.

All of the glyphs on columns between the starting cursor position and (inclusively) the ending cursor position shall become erase-columns as described in 5.35.7.3.

Current line: Unchanged.

Current column: Set to 1 if the `<control-D>` was preceded by a `^` or `0`; otherwise, set to (column − 1) − ((column − 2) % `shiftwidth`).

5.35.7.3.3 <control-H>

Synopsis: `<control-H>`

If in text input mode for a line-oriented command, and there are no characters to erase, text input mode shall be terminated, no further action shall be done for this command, and the current line and column shall be unchanged.

If there are characters other than autoindent characters that have been input on the current line before the cursor, the cursor shall move back one character.

Otherwise, if there are autoindent characters on the current line before the cursor, it is implementation-defined if the `<control-H>` command is an error or if the cursor moves back one autoindent character.

Otherwise, if the cursor is in column position 1 and there are previous lines that have been input, it is implementation-defined if the `<control-H>` command is an error or if it is equivalent to entering `<control-H>` after the last input character on the previous input line.

Otherwise, it shall be an error.

All of the glyphs on columns between the starting cursor position and (inclusively) the ending cursor position shall become erase-columns as described in 5.35.7.3.

The current *erase* character (see `stty` in 4.59) shall cause an equivalent action to the `<control-H>` command, unless the previously inserted character was a backslash, in which case it shall be as if the literal current *erase* character had been inserted instead of the backslash.

Current line: Unchanged, unless previously input lines are erased, in which case it shall be set to line − 1.

Current column: Set to the first column that displays any portion of the character backed up over.

5.35.7.3.4 `<newline>`

Synopsis: `<newline>`
Synopsis: `<carriage-return>`
Synopsis: `<control-J>`
Synopsis: `<control-M>`

If input was part of a line-oriented command, text input mode shall be terminated and the command shall continue execution with the input provided.

Otherwise, terminate the current line. If there are no characters other than autoindent characters on the line, all characters on the line shall be discarded. Otherwise, it is unspecified if the autoindent characters in the line are modified by entering these characters.

Continue text input mode on a new line appended after the current line. If the `slowopen` edit option is set, the lines on the screen below the current line shall not be pushed down, but the first of them shall be cleared and shall appear to be overwritten. Otherwise, the lines of the screen below the current line shall be pushed down.

If the `autoindent` edit option is set, an appropriate number of `autoindent` characters shall be added as a prefix to the line as described by the `ex` `autoindent` edit option (see 5.10.7.8.1).

All columns after the cursor that are erase-columns (as described in 5.35.7.3) shall be discarded.

If the `autoindent` edit option is set, all `<blank>` characters immediately following the cursor shall be discarded.

All remaining characters after the cursor shall be transferred to the new line, positioned after any autoindent characters.

Current line: Set to current line + 1.

Current column: Set to the first column that displays any portion of the first character after the autoindent characters on the new line, if any, or the first column position after the last autoindent character, if any, or column position 1.

5.35.7.3.5 `<control-T>`

Synopsis: `<control-T>`

The `<control-T>` character shall have no special meaning when in text input mode for a line-oriented command (see 5.35.7.2).

This command need not be supported on block-mode terminals.

Behave as if the user entered the minimum number of `<blank>` characters necessary to move the cursor forward to the column position after the next `shiftwidth` (see 5.10.7.8.19) boundary.

Current line: Unchanged.

Current column: Set to column + `shiftwidth` − ((column − 1) % `shiftwidth`).

5.35.7.3.6 <control-U>

Synopsis: `<control-U>`

If there are characters other than autoindent characters that have been input on the current line before the cursor, the cursor shall move to the first character input after the autoindent characters.

Otherwise, if there are autoindent characters on the current line before the cursor, it is implementation-defined if the `<control-U>` command is an error or if the cursor moves to the first column position on the line.

Otherwise, if the cursor is in column position 1 and there are previous lines that have been input, it is implementation-defined if the `<control-U>` command is an error or if it is equivalent to entering `<control-U>` after the last input character on the previous input line.

Otherwise, it shall be an error.

All of the glyphs on columns between the starting cursor position and (inclusively) the ending cursor position shall become erase-columns as described in 5.35.7.3.

The current *kill* character (see `stty` in 4.59) shall cause an equivalent action to the `<control-U>` command, unless the previously inserted character was a backslash, in which case it shall be as if the literal current *kill* character had been inserted instead of the backslash.

Current line: Unchanged, unless previously input lines are erased, in which case it shall be set to line − 1.

Current column: Set to the first column that displays any portion of the last character backed up over.

5.35.7.3.7 <control-V>

Synopsis: `<control-V>`
Synopsis: `<control-Q>`

Allow the entry of any subsequent character, other than `<control-J>` or `<newline>`, as a literal character, removing any special meaning that it may have to the editor in text input mode. If a `<control-V>` or `<control-Q>` is entered before a `<control-J>` or `<newline>` character, the `<control-V>` or `<control-Q>` character shall be discarded, and the `<control-J>` or `<newline>` shall behave as described in 5.35.7.3.4. C

For purposes of the display only, the editor shall behave as if a ^ character was entered, and the cursor shall be positioned as if overwriting the ^ character. When a subsequent character is entered, the editor shall behave as if that character was entered instead of the original `<control-V>` or `<control-Q>` character.

Current line: Unchanged.

Current column: Unchanged.

5.35.7.3.8 `<control-W>`

Synopsis: `<control-W>`

If there are characters other than autoindent characters that have been input on the current line before the cursor, the cursor shall move back over the last word preceding the cursor (including any `<blank>` characters between the end of the last word and the current cursor); the cursor shall not move to before the first character after the end of any `autoindent` characters.

Otherwise, if there are autoindent characters on the current line before the cursor, it is implementation-defined if the `<control-W>` command is an error or if the cursor moves to the first column position on the line.

Otherwise, if the cursor is in column position 1 and there are previous lines that have been input, it is implementation-defined if the `<control-W>` command is an error or if it is equivalent to entering `<control-W>` after the last input character on the previous input line.

Otherwise, it shall be an error.

All of the glyphs on columns between the starting cursor position and (inclusively) the ending cursor position shall become erase-columns as described in 5.35.7.3.

Current line: Unchanged, unless previously input lines are erased, in which case it shall be set to line – 1.

Current column: Set to the first column that displays any portion of the last character backed up over.

5.35.7.3.9 `<ESC>`

Synopsis: `<ESC>`

If input was part of a line-oriented command:

(1) If `<interrupt>` was entered, text input mode shall be terminated and the editor shall return to command mode. The terminal shall be alerted.

(2) If `<ESC>` was entered, text input mode shall be terminated and the command shall continue execution with the input provided.

Otherwise, terminate text input mode and return to command mode.

Any autoindent characters entered on newly created lines that have no other characters shall be deleted.

Any leading autoindent and `<blank>` characters on newly created lines shall be rewritten to be the minimum number of `<blank>` characters possible.

The screen shall be redisplayed as necessary to match the contents of the edit buffer.

Current line: Unchanged.

Current column:

(1) If there are text input characters on the current line, the column shall be set to the last column where any portion of the last text input character is displayed.

(2) Otherwise, if a character is displayed in the current column, unchanged.

(3) Otherwise, set to column position 1.

5.35.8 Exit Status

The `vi` utility shall exit with one of the following values:

0	Successful completion.
>0	An error occurred.

5.35.9 Consequences of Errors

When any error is encountered and the standard input is not a terminal device file, `vi` shall not write the file or return to command or text input mode, and shall terminate with a nonzero exit status.

Otherwise, when an unrecoverable error is encountered it shall be equivalent to a SIGHUP asynchronous event.

Otherwise, when an error is encountered, the editor shall behave as specified in 5.35.7.2.

5.37 `write` – Write to another user

⇒ **5.37.5.3 `write` Environment Variables.** *Change the description of* **LC_CTYPE** *to:*

LC_CTYPE	This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files). If the locale of the recipient does not use an LC_CTYPE equivalent to that of the sender, the results are undefined.

Rationale: This change is the result of interpretation request PASC 1003.2-92 #26 submitted for IEEE Std 1003.2-1992.

Section 6: Revisions to Software Development Utilities Option

6.1 `ar` – Create and maintain library archives

⇒ **6.1.6.1 `ar` Standard Output.** *Change the two paragraphs:* If the `–r` option is used with the `–v` option, and *file* is already in the archive, the standard output format shall be

`"r – %s\n"`, <*file*>

where *file* is the operand specified on the command line.

If *file* is being added to the archive with the `–r` option, the standard output format shall be

`"a – %s\n"`, <*file*>

where *file* is the operand specified on the command line.

to:

If the `–r` option is used with the `–v` option:

(1) If *file* is already in the archive, the standard output format shall be

`"r – %s\n"`, <*file*>

where <*file*> is the operand specified on the command line.

If *file* is not already in the archive, the standard output format shall be

`"a – %s\n"`, <*file*>

where <*file*> is the operand specified on the command line.

Rationale: This change is the result of interpretation request PASC 1003.2-92 #92 submitted for IEEE Std 1003.2-1992.

6.2 make – Maintain, update, and regenerate groups of programs B

Rationale: The changes to make are the result of interpretation requests PASC 1003.2-92 #94, #100, and #113 submitted for IEEE Std 1003.2-1992. The large majority of these changes change the term “command line” to be specific in each case about whether it is a “make utility command line” or a “makefile command line.” To avoid clutter, it is not further diffmarked. B

⇒ **6.2.3 make Options.** *Change the text from option* –q *to the end of the subclause to:*

–q Return a zero exit value if the target file is up-to-date; otherwise, return an exit value of 1. Targets shall not be updated if this option is specified. However, a makefile command line (associated with the targets) with a plus-sign (+) prefix shall be executed.

–r Clear the suffix list and do not use the built-in rules.

–S Terminate make if an error occurs while executing the commands to bring a target up-to-date. This shall be the default and the opposite of –k.

–s Do not write makefile command lines or touch messages (see –t) to standard output before executing. This mode shall be the same as if the special target .SILENT were specified without prerequisites. See 6.2.7.2.

–t Update the modification time of each target as though a touch *target* had been executed. See touch in 4.63. Targets that have prerequisites but no commands (see 6.2.7.3), or that are already up-to-date, shall not be touched in this manner. Write messages to standard output for each target file, indicating the name of the file and that it was touched. Normally, the makefile command lines associated with each target are not executed. However, a makefile command line with a plus-sign (+) prefix shall be executed.

Any options specified in the **MAKEFLAGS** environment variable shall be evaluated before any options specified on the make utility command line. If the –k and –S options are both specified on the make utility command line or by the **MAKEFLAGS** environment variable, the last option specified shall take precedence. If the –f or –p options appear in the **MAKEFLAGS** environment variable, the result is undefined.

⇒ **6.2.4 `make` Operands.** *Change the final paragraph to:*

If the *target_name* and *macro=name* operands are intermixed on the `make` utility command line, the results are unspecified.

⇒ **6.2.5.3 `make` Environment Variables.** *Change the text from variable* **MAKEFLAGS** *to the end of the subclause to:*

MAKEFLAGS This variable shall be interpreted as a character string representing a series of option characters to be used as the default options. The implementation shall accept both of the following formats (but need not accept them when intermixed):

(1) The characters are option letters without the leading hyphens or `<blank>` separation used on a `make` utility command line.

(2) The characters are formatted in a manner similar to a portion of the `make` utility command line: options are preceded by hyphens and `<blank>`-separated as described in 2.10.2. The *macro=name* macro definition operands can also be included. The difference between the contents of **MAKEFLAGS** and the `make` utility command line is that the contents of the variable shall not be subjected to the word expansions (see 3.6) associated with parsing the command-line values.

The value of the **SHELL** environment variable shall not be used as a macro and shall not be modified by defining the `SHELL` macro in a makefile or on the `make` utility command line. All other environment variables, including those with null values, shall be used as macros, as defined in 6.2.7.4.

⇒ **6.2.6.1 `make` Standard Output.** *Add a new sentence to the end of the paragraph:*

If the `−t` option is present and a file is *touched*, `make` shall write to standard output a message of unspecified format indicating that the file was touched, including the filename of the file.

⇒ **6.2.6.3 `make` Output Files.** *Change this subclause to:*

Files can be created when the `−t` option is present. Additional files can also be created by the utilities invoked by `make`.

⇒ **6.2.7.1 Makefile Syntax.** *Change the first paragraph to:*

A makefile can contain rules, macro definitions (see 6.2.7.4), and comments. There are two kinds of rules: inference rules (6.2.7.5) and target rules (6.2.7.3). The `make` utility shall contain a set of built-in inference rules. If the `−r` option is present, the built-in rules shall not be used and the suffix list shall be cleared. Additional rules of both types can be specified in a makefile. If a rule is defined more than once, the value of the rule shall be that of the last one specified. Macros can also be defined more than once, and the value of the macro is specified by 6.2.7.4. Comments start with a number sign (#) and continue until an unescaped `<newline>` is reached.

⇒ **6.2.7.1 Makefile Syntax.** *Change the fourth paragraph (the one beginning* "The rules in makefiles . . . "*) to:*

The rules in makefiles shall consist of the following types of lines: target rules, including special targets (see 6.2.7.3); inference rules (see 6.2.7.5); macro definitions (see 6.2.7.4); empty lines; and comments.

⇒ **6.2.7.1 Makefile Syntax.** *Change the fifth paragraph (the one beginning* "When an escaped . . . "*) to:*

When an escaped `<newline>` (one preceded by a backslash) is found anywhere in the makefile except in a command line, it shall be replaced, along with any leading white space on the following line, with a single `<space>`. When an escaped `<newline>` is found in a command line in a makefile, the command line shall contain the backslash, the `<newline>`, and the next line, except that the first character of the next line shall not be included if it is a `<tab>`.

⇒ **6.2.7.2 Makefile Execution.** *Replace this subclause with:*

Makefile command lines shall be processed one at a time by writing the makefile command line to the standard output (unless one of the conditions listed under "@" suppresses the writing) and executing the command(s) in the line. A `<tab>` character may precede the command to standard output. Command execution shall be as if the makefile command line were the argument to the *system*() function in POSIX.1 {8}. The environment for the command being executed shall contain all of the variables in the environment of `make`.

By default, when `make` receives a nonzero status from the execution of a command, it terminates with an error message to standard error.

Makefile command lines can have one or more of the following prefixes: a hyphen (−), an at sign (@), or a plus sign (+). These modify the way in which `make` processes the command. When a command is written to standard output, the prefix shall not be included in the output.

- If the command prefix contains a hyphen, or if the `−i` option is present, or if the special target `.IGNORE` has either the current target as a prerequisite or has no prerequisites, any error found while executing the command shall be ignored.

@ If the command prefix contains an at sign and the `make` utility command-line `−n` option is not specified, or the `−s` option is present, or the special target `.SILENT` has either the current target as a prerequisite or has no prerequisites, the command shall not be written to standard output before it is executed.

+ If the command prefix contains a plus sign, this indicates a makefile command line that shall be executed even if `−n`, `−q`, or `−t` is specified on the `make` utility command line.

⇒ **6.2.7.3 Target Rules.** *In the second paragraph (the one beginning with* "Target entries . . . "*), change "command lines" to "makefile command lines."*

⇒ **6.2.7.3 Target Rules.** *Replace the list entry for* `.SUFFIXES` *with the following:* C

`.SUFFIXES` Prerequisites of `.SUFFIXES` shall be appended to the list of known suffixes and are used in conjunction with the inference rules (see 6.2.7.5). If `.SUFFIXES` does not have any prerequisites, the list of known suffixes shall be cleared. C

The special targets `.IGNORE`, `.POSIX`, `.PRECIOUS`, `.SILENT`, and `.SUFFIXES` shall be specified without commands. C

⇒ **6.2.7.4 Macros.** *Delete the following paragraph:*

Subsequent appearances of $(*string1*) or ${*string1*} shall be replaced by *string2*. The parentheses or braces are optional if *string1* is a single character. The macro $$ shall be replaced by the single character $.

⇒ **6.2.7.4 Macros.** *Change the fifth paragraph (the one beginning* "Macros can appear anywhere . . . "*) to:*

Macros can appear anywhere in the makefile. $(*string1*) or ${*string1*} shall be replaced by *string2*, as follows:

(1) Macros in target lines shall be evaluated when the target line is read. C

(2) Macros in makefile command lines shall be evaluated when the command is executed. C

(3) Macros in the string before the equals sign in a macro definition shall be evaluated when the macro assignment is made.

(4) Macros after the equals sign in a macro definition shall not be evaluated until the defined macro is used in a rule or command, or before the equals sign in a macro definition.

The parentheses or braces are optional if *string1* is a single character. The macro $$ shall be replaced by the single character $.

⇒ **6.2.7.4 Macros.** *Change the sixth through eleventh paragraphs (*“Macro assignments . . . <numbered list> . . . shall be reversed.”*) to:*

Macro definitions shall be taken from the following sources, in the following logical order, before the makefile(s) are read.

(1) Macros specified on the `make` utility command line, in the order specified on the command line. It is unspecified whether the internal macros defined in 6.2.7.7 are accepted from this source.

(2) Macros defined by the **MAKEFLAGS** environment variable, in the order specified in the environment variable. It is unspecified whether the internal macros defined in 6.2.7.7 are accepted from this source.

(3) The contents of the environment, excluding the **MAKEFLAGS** and **SHELL** variables and including the variables with null values.

(4) Macros defined in the inference rules built into `make`.

Macro definitions from these sources shall not override macro definitions from a lower-numbered source. Macro definitions from a single source (e.g., the `make` utility command line, the **MAKEFLAGS** environment variable or the other environment variables) shall override previous macro definitions from the same source.

Macros defined in the makefile(s) shall override macro definitions that occur before them in the makefile(s) and macro definitions from source (4). If the `−e` option is not specified, macros defined in the makefile(s) shall override macro definitions from source (3). Macros defined in the makefile(s) shall not override macro definitions from source (1) or source (2).

Before the makefile(s) are read, all of the `make` utility command-line options (except `−f` and `−p`) and `make` utility command-line macro definitions (except any for the `MAKEFLAGS` macro), not already included in the `MAKEFLAGS` macro, shall be added to the `MAKEFLAGS` macro. Other implementation-defined options and macros may also be added to the `MAKEFLAGS` macro. If this modifies the value `MAKEFLAGS` macro, or, if the `MAKEFLAGS` macro is modified at any subsequent time, the **MAKEFLAGS** environment variable shall be modified to match the new value of the `MAKEFLAGS` macro.

Before the makefile(s) are read, all of the `make` utility command-line macro definitions (except the **MAKEFLAGS** macro or the **SHELL** macro) shall be added to the environment of `make`. Other implementation-defined variables may also be added to the environment of `make`.

⇒ **6.2.7.7 Internal Macros.** *Change the description of* `$<` *to:*

`$<` In an inference rule, the `$<` macro shall evaluate to the file name whose existence allowed the inference rule to be chosen for the target. In the `.DEFAULT` rule, the `$<` macro shall evaluate to the current target name. The meaning of the `$<` macro macro is otherwise unspecified.

For example, in the `.c.a` inference rule, `$<` represents the prerequisite `.c` file.

Section 7: Revisions to Language-Independent System Services

Editor's Note: Remove this section. It is no longer required due to the movement of APIs from this standard to POSIX.1 {8}.

Annex A
(normative)
Revisions to C Language Development Utilities Option

A.1 c89 – Compile Standard C programs

⇒ **A.1.7.1 c89 Standard Libraries.** *Change the description of* `–l c` *to:*

`–l c` This library contains all mandatory (i.e., nonoptional) library functions referenced in `<stdlib.h>`, `<stdio.h>`, `<time.h>`, `<setjmp.h>`, `<signal.h>`, `<unistd.h>`, `<sys/types.h>`, `<string.h>`, and `<ctype.h>`, except for those functions referenced in `<math.h>`. There may be additional functions included; section 2.9.3 of POSIX.1 {8} describes constants that indicate the presence of optional facilities, and these constants can be used with `getconf` to determine whether those functions are included in the library accessed by `–l c`. For example, if an invocation of

```
getconf _POSIX_VERSION
```

exits with a status of zero, the library searched also shall include all mandatory (nonoptional) functions defined by ISO/IEC 9945-1: 1990; if the status is nonzero, it is unspecified whether these functions are available. An implementation shall not require this operand to be present to cause a search of this library.

⇒ **A.1.7.1 c89 Standard Libraries.** *Add to the end of the subclause:*

All other libraries that shall be specified when building a POSIX.1 {8} conforming application are those listed in POSIX.1 {8} subclause 2.7.3, Headers and Function Prototypes. *Editor's Note: The table referenced is in fact to be found in POSIX.1a draft 17 onwards.*

Rationale: Since Annex B is gone, all references to it have to be removed and a more generic statement of the interaction with POSIX.1 {8} has been included.

A.3 yacc – Yet another compiler compiler

⇒ **A.3.6.3.1 yacc Code File.** *Delete the second paragraph, which is:* B

The contents of the program section (see A.3.7.1.4) of the input file shall then be included. B

⇒ **A.3.7.1.4 yacc Programs Section.** *Change this subclause to:* B

The *programs* section can include the definition of the lexical analyzer *yylex*() and any other functions; for example, those used in the actions specified in the grammar rules. It is unspecified whether the programs section precedes or follows the semantic actions in the output file; therefore, if the application contains any macro definitions and declarations intended to apply to the code in the semantic actions, it shall place them within %{ ... %} in the declarations section. B

Rationale: The preceding changes are the result of interpretation request PASC 1003.2-92 #93 submitted for IEEE Std 1003.2-1992. B

⇒ **A.3.7.4 Interface to the Lexical Analyzer.** *In the third paragraph (the one beginning* "If the token numbers ..."*), change the sentence* "All assigned token numbers shall be unique and distinct from the token numbers used for literals." *to:* B

All token numbers assigned by yacc shall be unique and distinct from the token numbers used for literals and user assigned tokens. B

Rationale: This change is the result of interpretation request PASC 1003.2-92 #104 submitted for IEEE Std 1003.2-1992. B

Annex B
(normative)
Revisions to C Language Bindings Option

Editor's Note: Replace the text of this entire annex with the following. (It is no longer required due to the movement of APIs from this standard to POSIX.1 {8}. Unlike Section 7, it is not being removed because we wish to avoid renumbering all of the following annexes.)

This annex is unused.

Annex C
(normative)

Revisions to FORTRAN Development and Runtime Utilities Options

There are no revisions to Annex C.

Annex D
(informative)
Revisions to Bibliography

⇒ **D Bibliography.** *Remove the entry for ISO/IEC 10646-1.*

Rationale: The entry for this standard has been moved into the normative references.

⇒ **D Bibliography.** *Add the following entry in the proper order:*

{B90} RFC 2045, Freed, N., Borenstein, N. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*

{B91} ISO/IEC 14652: 199?, *Functionality for internationalization— Specification of cultural conventions*

{B92} ISO/IEC 15435: 199?, *Information technology—Internationalization APIs*

{B93} ISO/IEC 15897: 199?, *Information technology—Procedures for European registration of cultural elements*

Annex E
(informative)
Revisions to Rationale and Notes

⇒ **E Rationale and Notes.** *Remove all references to the C-Language Binding Option and* {POSIX2_C_BIND} *from this annex, or reword to indicate they have moved to P1003.1a. Reword all references to language-independent functions in Chapter 7 to use the POSIX.1 {8} function names.*

Rationale: Since Chapter 7 and Annex B are gone, all references to them have to be removed.

⇒ **E.2.2.2 General Terms.** *Add the following rationale text at the end of this subclause, immediately preceding E.2.2.3.*

Symbolic Links

Symbolic link support was added to the first revision of this standard to achieve synchronization with IEEE Std 1003.1-199x. This entailed a significant number of small changes to many interfaces.

Because a symbolic link and its referenced object coexist in the file system name space, confusion can arise in distinguishing between the link itself and the referenced object. Historically, utilities and system calls have adopted their own link following conventions in a somewhat ad hoc fashion. Rules for a uniform approach are outlined here, although historical practice has been adhered to as much as was possible. To promote consistent system use, user-written utilities are encouraged to follow these same rules.

Symbolic links are handled either by operating on the link itself, or by operating on the object referenced by the link. In the latter case, an application or system call is said to "follow" the link. Symbolic links may reference other symbolic links, in which case links are dereferenced until an object that is not a symbolic link is found, a symbolic link that references a file that does not exist is found, or a loop is detected. (Current implementations do not detect loops, but have a limit on the number of symbolic links that they will dereference before declaring it an error.)

There are four domains for which default symbolic link policy is established in a system. In almost all cases, there are utility options that override this default behavior. The four domains are as follows:

(1) Symbolic links specified to system calls that take file name arguments

(2) Symbolic links specified as command-line file name arguments to utilities that are not performing a traversal of a file hierarchy

(3) Symbolic links referencing files not of type directory, specified to utilities that are performing a traversal of a file hierarchy

(4) Symbolic links referencing files of type directory, specified to utilities that are performing a traversal of a file hierarchy

First Domain

The first domain is not within the scope of this standard.

Second Domain

The reason this category is restricted to utilities that are not traversing the file hierarchy is that some standard utilities take an option that specifies a hierarchical traversal, but by default operate on the arguments themselves. Generally, users specifying the option for a file hierarchy traversal wish to operate on a single, physical hierarchy, and therefore symbolic links, which may reference files outside of the hierarchy, are ignored. For example, `chown` *owner file* is a different operation from the same command with the `–R` option specified. In this example, the behavior of the command `chown` *owner file* is described here, while the behavior of the command `chown –R` *owner file* is described in the third and fourth domains.

The general rule is that the utilities in this category follow symbolic links named as arguments.

Exceptions in the second domain are:

— The `mv` and `rm` utilities do not follow symbolic links named as arguments, but respectively attempt to rename or delete them.

— The `ls` utility is also an exception to this rule. For compatibility with historical systems, when the `–R` option is not specified, the `ls` utility follows symbolic links named as arguments if the `–L` option is specified or if the `–F`, `–d`, or `–l` options are not specified. (If the `–L` option is specified, `ls` always follows symbolic links; it is the only utility where the `–L` option affects its behavior even though a tree walk is not being performed.)

All other standard utilities, when not traversing a file hierarchy, always follow symbolic links named as arguments.

Historical practice is that the `–h` option is specified if standard utilities are to act upon symbolic links instead of upon their targets. Examples of commands that have historically had a `–h` option for this purpose are the `chgrp`, `chown`, `file`, and `test` utilities.

Third Domain

The third domain is symbolic links, referencing files not of type directory, specified to utilities that are performing a traversal of a file hierarchy. (This includes symbolic links specified as command-line file name arguments or encountered during the traversal.)

The intention of POSIX.2 is that the operation that the utility is performing is applied to the symbolic link itself, if that operation is applicable to symbolic links. The reason that the operation is not required is that symbolic links in some systems do not have such attributes as a file owner, and therefore the chown operation would be meaningless. If symbolic links on the system have an owner, it is the intention that the utility chown cause the owner of the symbolic link to change. If symbolic links do not have an owner, the symbolic link should be ignored. Specifically, by default, no change should be made to the file referenced by the symbolic link.

Fourth Domain

The fourth domain is symbolic links referencing files of type directory, specified to utilities that are performing a traversal of a file hierarchy. (This includes symbolic links specified as command-line file name arguments or encountered during the traversal.)

All standard utilities do not, by default, indirect into the file hierarchy referenced by the symbolic link. (POSIX.2 uses the informal term “physical walk” to describe this case. The case where the utility does indirect through the symbolic link is termed a “logical walk.”)

There are three reasons for the default to a physical walk.

— With very few exceptions, a physical walk has been the historical default on UNIX systems supporting symbolic links. Because some utilities (i.e., rm) must default to a physical walk, regardless, changing historical practice in this regard would be confusing to users and needlessly incompatible.

— For systems where symbolic links have the historical file attributes (i.e., owner, group, mode), defaulting to a logical traversal would require the addition of a new option to the commands to modify the attributes of the link itself. This is painful and more complex than the alternatives.

— There is a security issue with defaulting to a logical walk. Historically, the command chown –R *user file* has been safe for the super-user because *setuid* and *setgid* bits were lost when the ownership of the file was changed. If the walk were logical, changing ownership would no longer be safe because a user might have inserted a symbolic link pointing to any file in the tree. Again, this would necessitate the addition of an option to the commands doing hierarchy traversal to not indirect through the symbolic links, and historical scripts doing recursive walks would instantly become security problems. While this is mostly an issue for system administrators, it is preferable to not have different defaults for different classes of users.

As consistently as possible, users may cause standard utilities performing a file hierarchy traversal to follow any symbolic links named on the command line, regardless of the type of file they reference, by specifying the −H (for "half logical") option. This option is intended to make the command-line name space look like the logical name space.

As consistently as possible, users may cause standard utilities performing a file hierarchy traversal to follow any symbolic links named on the command line as well as any symbolic links encountered during the traversal, regardless of the type of file they reference, by specifying the −L (for "logical") option. This option is intended to make the entire name space look like the logical name space.

For consistency, implementors are encouraged to use the −P (for "physical") flag to specify the physical walk in utilities that do logical walks by default for whatever reason. The only standard utilities that require the −P option are `cd` and `pwd`; see the note below.

When one or more of the −H, −L, and −P flags can be specified, the last one specified determines the behavior of the utility. This permits users to alias commands so that the default behavior is a logical walk and then override that behavior on the command line.

Exceptions in the Third and Fourth Domains

To maintain compatibility with historical systems, the `ls` and `rm` utilities are exceptions to these rules.

The `rm` utility never follows symbolic links and does not support the −H, −L, or −P options.

The `ls` utility never follows symbolic links unless the −L option is specified, when it follows all of the symbolic links, regardless of their type or if specified on the command line or encountered in the traversal. The `ls` utility does not support the −H and −P options.

POSIX.2 requires that the standard utilities `ls`, `find`, and `pax` detect infinite loops when doing logical walks; i.e., a directory, or more commonly a symbolic link, that refers to an ancestor in the current file hierarchy. If the file system itself is corrupted, causing the infinite loop, it may be impossible to recover. Because `find` and `ls` are often used in system administration and security applications, they should attempt to recover and continue as best as they can. The `pax` utility should terminate because the archive it was creating is by definition corrupted. Other, less vital, utilities should probably simply terminate as well. Implementations are strongly encouraged to detect infinite loops in all utilities.

Historical practice is shown in Table E-100. The heading SVID3 stands for the Third Edition of the System V Interface Definition {B37}.

Historically, several shells have had built-in versions of the `pwd` utility. In some of these shells, `pwd` reported the physical path, and in others, the logical path. Implementations of the shell corresponding to this standard must report the logical path by default. Earlier versions of this standard did not require the `pwd` utility to be a built-in utility. Now that `pwd` is required to set an environment

Table E-100 – Historical Practice for Symbolic Links

Utility	SVID3	4.3BSD	4.4BSD	POSIX	Comments
`cd`				`–L`	Treat “. .” logically
`cd`				`–P`	Treat “. .” physically
`chgrp`			`–H`	`–H`	Follow command–line symlinks
`chgrp`			`–h`	`–L`	Follow symlinks
`chgrp`	`–h`			`–h`	Affect the symlink
`chmod`				`–h`	Affect the symlink
`chmod`			`–H`	`–H`	Follow command–line symlinks
`chmod`			`–h`	`–L`	Follow symlinks
`chown`			`–H`	`–H`	Follow command–line symlinks
`chown`			`–h`	`–L`	Follow symlinks
`chown`	`–h`			`–h`	Affect the symlink
`cp`			`–H`	`–H`	Follow command–line symlinks
`cp`			`–h`	`–L`	Follow symlinks
`cpio`	`–L`		`–L`		Follow symlinks
`du`			`–H`	`–H`	Follow command–line symlinks
`du`			`–h`	`–L`	Follow symlinks
`file`	`–h`			`–h`	Affect the symlink
`find`			`–H`	`–H`	Follow command–line symlinks
`find`			`–h`	`–L`	Follow symlinks
`find`	`–follow`		`–follow`	`–follow`	Follow symlinks
`ln`	`–s`	`–s`	`–s`	`–s`	Create a symbolic link
`ls`	`–L`	`–L`	`–L`	`–L`	Follow symlinks
`ls`				`–H`	Follow command–line symlinks
`mv`					Operates on the symlink
`pax`			`–H`	`–H`	Follow command–line symlinks
`pax`			`–h`	`–L`	Follow symlinks
`pwd`				`–L`	Printed path may contain symlinks
`pwd`				`–P`	Printed path will not contain symlinks
`rm`					Operates on the symlink
`tar`			`–H`		Follow command–line symlinks
`tar`		`–h`	`–h`		Follow symlinks
`test`	`–h`		`–h`	`–h`	Affect the symlink

variable in the current shell execution environment, it must be a built-in utility.

The `cd` command is required, by default, to treat the string “. .” logically. Implementors are required to support the `–P` flag in `cd` so that users can have their current environment handled physically.

In 4.3BSD, `chgrp` during tree traversal changed the group of the symbolic link, not the target. Symbolic links in 4.4BSD do not have owner, group, mode, or other standard UNIX system file attributes.

The only significant work required for vendors to conform to this standard will be to add the `–H` and `–L` options to the eight standard utilities that will require them.

⇒ **E.2.5.2.2 LC_COLLATE.** *Change the second-to-last paragraph to:*

The character (and collating element) order is defined by the order in which characters and elements are specified between the `order_start` and `order_-end` keywords. This character order is used in range expressions in REs (see 2.8). Weights assigned to the characters and elements define the collation sequence; in the absence of weights, the character order is also the collation sequence. For two elements that have the same primary, secondary, and tertiary weights, the character order is also the collation sequence.

⇒ **E.3.6.2 Parameter Expansion.** *In Table E-1, change the fourth row as follows:*

	parameter **set and not null**	***parameter*** **set but null**	***parameter*** **unset**
`${`*parameter*`=`*word*`}`	substitute *parameter*	substitute null	assign *word*

Rationale: This change is the result of interpretation request PASC 1003.2-92 #48 submitted for IEEE Std 1003.2-1992.

⇒ **E.4.48 pax Rationale.** *Replace the full rationale for* `pax` *with the following.* B

E.4.48 pax – Portable archive interchange

The `pax` utility was commissioned for POSIX.2-1992. It represented a peaceful B compromise between advocates of the historical `tar` and `cpio` utilities. B

A fundamental difference between `cpio` and `tar` was in the way directories were treated. The `cpio` utility did not treat directories differently from other files, and to select a directory and its contents required that each file in the hierarchy be explicitly specified. For `tar`, a directory matched every file in the file hierarchy it rooted.

The `pax` utility offers both interfaces; by default, directories map into the file hierarchy they root. The `–d` option causes `pax` to skip any file not explicitly referenced, as `cpio` historically did. The `tar`-style behavior was chosen as the default because it was believed that this was the more common usage and because `tar` is the more commonly available interface (being provided historically on both System V and BSD implementations). B

The data interchange format specification originally published in Section 10 of B POSIX.1 {8} required that processes with "appropriate privileges" always shall B restore the ownership and permissions of extracted files exactly as archived. If viewed from the historic equivalence between super-user and "appropriate privileges," there are two problems with this requirement. First, users running as super-users may unknowingly set dangerous permissions on extracted files. Second, it is needlessly limiting in that super-users cannot extract files and own them as super-user unless the archive was created by the super-user. (It should be noted that restoration of ownerships and permissions for the super-user, by default, is historical practice in `cpio`, but not in `tar`.) In order to avoid these two problems, the `pax` specification has an additional "privilege" mechanism, the `−p` option. Only a `pax` invocation with the POSIX.1 {8} privileges needed, and which has the `−p` option set using the `e` specification character, has the "appropriate privilege" to restore full ownership and permission information.

Note also that POSIX.1 {8} Section 10.1 requires that the file ownership and access permissions shall be set, on extraction, in the same fashion as the POSIX.1 {8} *creat*() function when provided the mode stored in the archive. This means that the file creation mask of the user is applied to the file permissions.

Users should note that directories may be created by `pax` while extracting files with permissions that are different from those that existed at the time the archive was created. When extracting sensitive information into a directory hierarchy that no longer exists, users are encouraged to set their file creation mask appropriately to protect these files during extraction. C

The table of contents output is written to standard output to facilitate pipeline processing.

B

The one pathname per line format of standard input precludes pathnames containing `<newline>`s. Although such pathnames violate the portable filename guidelines, they may exist and their presence may inhibit usage of `pax` within shell scripts. This problem is inherited from historical archive programs. The problem can be avoided by listing filename arguments on the command line instead of on standard input.

A pre-1992 draft had hard links displaying for all pathnames. This was removed because it complicates the output of the case where `−v` is not specified and does not match historical `cpio` usage. The hard-link information is available in the `−v` display.

B

The archive formats inherited from POSIX.1 {8} have certain restrictions that have been brought along from historical usage. For example, there are restrictions on the length of pathnames stored in the archive. When `pax` is used in copy (`−rw`) mode (copying directory hierarchies), the ability to use extensions from the `−x pax` format overcomes these restrictions. B

The default *blocksize* value of 5120 B for `cpio` was selected because it is one of the standard block-size values for `cpio`, set when the `−B` option is specified. (The other default block-size value for `cpio` is 512 B, and this was considered to be too small.) The default block value of 10 240 B for `tar` was selected because that is the standard block-size value for BSD `tar`. The maximum block size of 32 256 B (2^{15}–512 B) is the largest multiple of 512 B that fits into a signed 16 b tape controller transfer register. There are known limitations in some historical systems that would prevent larger blocks from being accepted. Historical values were chosen to improve compatibility with historical scripts using dd or similar utilities to manipulate archives. Also, default block sizes for any file type other than character special file has been deleted from POSIX.2 as unimportant and not likely to affect the structure of the resulting archive.

Implementations are permitted to modify the block-size value based on the archive format or the device to which the archive is being written. This is to provide implementations the opportunity to take advantage of special types of devices, and it should not be used without a great deal of consideration because it will almost certainly decrease archive portability.

The intended use of the –n option was to permit extraction of one or more files from the archive without processing the entire archive. This was viewed by the standard developers as offering significant performance advantages over historical implementations. The –n option in pre-1992 drafts had three effects; the first was to cause special characters in patterns to not be treated specially. The second was to cause only the first file that matched a pattern to be extracted. The third was to cause pax to write a diagnostic message to standard error when no file was found matching a specified pattern. Only the second behavior is retained by POSIX.2, for many reasons. First, it is in general not acceptable for a single option to have multiple effects. Second, the ability to make pattern matching characters act as normal characters is useful for parts of pax other than file extraction. Third, a finer degree of control over the special characters is useful because users may wish to normalize only a single special character in a single file name. Fourth, given a more general escape mechanism, the previous behavior of the –n option can be easily obtained using the –s option or a sed script. Finally, writing a diagnostic message when a pattern specified by the user is unmatched by any file is useful behavior in all cases. In this version of POSIX.2, the –n was removed from the copy mode synopsis of pax; it is inapplicable because there are no *pattern* operands specified in this mode.

There is another method than pax for copying subtrees in POSIX.2, described as part of the cp utility (see 4.13). Both methods are historical practice: cp provides a simpler, more intuitive interface, while pax offers a finer granularity of control. Each provides additional functionality to the other; in particular, pax maintains the hard-link structure of the hierarchy while cp does not. It is the intention of the standard developers that the results be similar (using appropriate option combinations in both utilities). The results are not required to be identical; there seemed insufficient gain to applications to balance the difficulty of implementations having to guarantee that the results would be exactly identical.

A single archive may span more than one file. It is suggested that implementations provide informative messages to the user on standard error whenever the archive file is changed.

The –d option (do not create intermediate directories not listed in the archive) found in pre-1992 drafts was originally provided as a complement to the historical –d option of cpio. It has been deleted.

The −`s` option in pre-1992 drafts specified a subset of the substitution command from the `ed` utility. As there was no reason for only a subset to be supported, the −`s` option is now compatible with the current `ed` specification. Since the delimiter can be any nonnull character, the following usage with single spaces is valid:

```
pax -s " foo bar " ...
```

The −`t` option (specify an implementation-defined identifier naming an input or output device) found in pre-1992 drafts has been deleted because it is not historical practice and is of limited utility. In particular, historic versions of neither `cpio` nor `tar` had the concept of devices that were not mapped into the file system; if the devices are mapped into the file system, the −`f` option is sufficient.

The default behavior of `pax` with regard to file modification times is the same as historical implementations of `tar`. It is not the historical behavior of `cpio`.

Because the −`i` option uses `/dev/tty`, utilities without a controlling terminal will not be able to use this option. Implementations are allowed, but not required, to keep track of interactively renamed files, allowing for the processing of links to those files.

The −`y` option, found in pre-1992 drafts, has been deleted because a line containing a single period for the −`i` option has equivalent functionality. The special lines for the −`i` option (a single period and the empty line) are historical practice in `cpio`.

In pre-1992 drafts, an −`e` *charmap* option was included to increase portability of files between systems using different coded character sets. This option was omitted because it was apparent that consensus could not be formed for it. In this version of POSIX.2, the use of UTF8 should be an adequate substitute.

The −`k` option was added to address international concerns about the dangers involved in the character set transformations of −`e` (if the target character set were different than the source, the file names might be transformed into names matching existing files) and also was made more general to protect files transferred between file systems with different {NAME_MAX} values (truncating a filename on a smaller system might also inadvertently overwrite existing files). As stated, it prevents any overwriting, even if the target file is older than the source. This version of POSIX.2 adds more granularity of options to solve this problem by introducing the −`o invalid=` option—specifically the UTF8 action. (Note that an existing file that is named with a UTF8 encoding is still subject to overwriting in this case. The −`k` option closes that loophole.)

It is almost certain that appropriate privileges will be required for `pax` to accomplish parts of this specification. Specifically, creating files of type block special or character special, restoring file access times unless the files are owned by the user (the −`t` option), or preserving file owner, group, and mode (the −`p` option) will all probably require appropriate privileges.

Some of the file characteristics referenced in this standard may not be supported by some archive formats. For example, neither the `tar` nor `cpio` formats contain the file access time. For this reason, the `e` specification character has been provided, intended to cause all file characteristics specified in the archive to be retained.

It is required that extracted directories, by default, have their access and modification times and permissions set to the values specified in the archive. This has obvious problems in that the directories are almost certainly modified after being extracted and that directory permissions may not permit file creation. One possible solution is to create directories with the mode specified in the archive, as modified by the *umask* of the user, with sufficient permissions to allow file creation. After all files have been extracted, `pax` would then reset the access and modification times and permissions as necessary.

In read mode, implementations are permitted to overwrite files when the archive has multiple members with the same name. This may fail, of course, if permissions on the first version of the file do not permit it to be overwritten.

The `−p` (privileges) option was invented to reconcile differences between historical `tar` and `cpio` implementations. In particular, the two utilities used `−m` in diametrically opposed ways. The `−p` option also provides a consistent means of extending the ways in which future file attributes can be addressed, such as for enhanced security systems or high-performance files. There are two modes that will be most commonly used:

`-p e` “Preserve everything.” This would be used by the historical superuser, someone with all the appropriate privileges, to preserve all aspects of the files as they are recorded in the archive. The `e` flag is the sum of `o` and `p`, and other implementation-defined attributes.

`-p p` “Preserve” the file mode bits. This would be used by the user with regular privileges who wished to preserve aspects of the file other than the ownership. The file times are preserved by default, but two other flags are offered to disable these and use the time of extraction.

The list-mode formatting description in 4.48.3.1 borrows heavily from the one defined by the `printf` utility. However, since there is no separate operand list to get conversion arguments, the format was extended to allow specifying the name of the conversion argument as part of the conversion specification.

The `T` specifier allows time fields to be displayed in any of the date formats. B
Unlike the `ls` utility, `pax` does not adjust the format when the date is less than B
six months in the past. This makes parsing the output more predictable. B

The `M` specifier handles the ten-character prefix field for type and permissions used with `ls`.

The `D` specifier handles the ability to display the major/minor or file size, as with `ls`, by using `%-8(size)D`.

The `L` specifier handles the `ls` display for symbolic links.

Conversion specifiers were added to generate existing known types used for `ls`. B

Examples B

To copy the contents of the current directory to tape drive 1, medium density (assuming historical System V device naming procedures; the historical BSD device name would be `/dev/rmt9`):

```
pax -w -f /dev/rmt/1m .
```

To copy the *olddir* directory hierarchy to *newdir*:

```
mkdir newdir
pax -rw olddir newdir
```

To read the archive `a.pax`, with all files rooted in "`/usr`" in the archive extracted relative to the current directory:

```
pax -r -s ',^//*usr//*,,' -f a.pax
```

Using the option

```
-o listopt="%M %(atime)T %(size)D %(name)s"
```

overrides the default output description in Standard Output and instead writes

```
-rw-rw--- Jan 12 15:53 1492 /usr/foo/bar
```

Using the options

```
-o listopt='%L\t%(size)D\n%.7' \
-o listopt='(name)s\n%(ctime)T\n%T'
```

overrides the default output description in Standard Output and instead writes

```
/usr/foo/bar -> /tmp    1492
/usr/fo
Jan 12 1991
Jan 31 15:53
```

Rationale for the New pax Interchange Format B

The new POSIX data interchange format was developed primarily to satisfy international concerns that the `ustar` and `cpio` formats in POSIX.1 {8} did not provide for file, user, and group names encoded in characters outside a subset of ISO/IEC 646 {1}. The standard developers realized that this new POSIX data interchange format should be very extensible because there were other requirements they foresaw in the near future:

— Support international character encodings and locale information

— Support security information (ACLs, etc.) emerging from POSIX security working groups

— Support future file types, such as realtime or contiguous files

— Include data areas for implementation use

— Support systems with words larger than 32 b and timers with subsecond granularity

The following were not goals for this format because these are better handled by separate utilities or are inappropriate for a portable format:

— Encryption

— Compression

— Data translation between locales and codesets

— I-node storage

The format chosen to support the goals is an extension of the `ustar` format, which has been moved into this standard from its original home in POSIX.1 {8}. Of the two formats, only the `ustar` format was selected for extensions because:

— It was easier to extend in an upward compatible way. It offered version flags and header block type fields with room for future standardization. The `cpio` format, while possessing a more flexible file naming methodology, could not be extended without breaking some theoretical implementation or using a dummy file name that could be a legitimate file name.

— Industry experience since the original “tar wars” fought in developing POSIX.1 {8} has clearly been in favor of the `ustar` format, which is generally the default output format selected for `pax` implementations on new POSIX.2 systems.

The new format was designed with one additional goal in mind: reasonable behavior when an older `tar` or `pax` utility happened to read an archive. Since POSIX.1-1990 mandated that a “format-reading utility” had to treat unrecognized *typeflag* values as regular files, this allowed the format to include all the extended information in a pseudo-regular file that preceded each real file. An option is given that allows the archive creator to set up reasonable names for these files on the older systems. Also, the normative text suggests that reasonable file access values be used for this `ustar` header block. Making these header files inaccessible for convenient reading and deleting would not be reasonable. File permissions of 600 or 700 are suggested.

The `ustar` *typeflag* field was used to accommodate the additional functionality of the new format rather than *magic* or *version* because POSIX.1-1990 (and, by reference, the previous version of POSIX.2 `pax`), mandated the behavior of the format-reading utility when it encountered an unknown *typeflag*, but was silent about the other two fields.

A good deal of the complexity of this new format is found in its relation to the original `ustar` format. If the backwards compatibility goal had been abandoned, none of the text relating the precedence of `ustar` fields to extended header records would have been required. A format that consisted entirely of extended header records followed by data records could have been designed. However, the standard developers believed that the new format should have some basis in an existing format, if only to avoid yet another complete invention as part of the standardization process.

Early drafts of the first revision to this standard contained a proposed archive format that was based on compatibility with the standard for tape files (ISO 1001, similar to the format used historically on many mainframes and minicomputers). This format was overly complex and required considerable overhead in volume and header records. Furthermore, the standard developers felt that it would not be acceptable to the community of POSIX developers, so it was later changed to be a format more closely related to historical practice on POSIX systems.

The *prefix* and *name* split of pathnames in `ustar` was replaced by the single `path` extended header record for simplicity.

The concept of a global extended header (*typeflag* `g`) was controversial. If this were applied to an archive being recorded on magnetic tape, a few unreadable blocks at the beginning of the tape could be a serious problem; a utility attempting to extract as many files as possible from a damaged archive could lose a large percentage of file header information in this case. However, if the archive were on a reliable medium, such as a CD-ROM, the global extended header offers considerable potential size reductions by eliminating redundant information. Thus, the text warns against using the global method for unreliable media and provides a method for implanting global information in the extended header for each file, rather than in the *typeflag* `g` records.

No facility for data translation or filtering on a per-file basis is included because the standard developers could not invent an interface that would allow this in an efficient manner. If a filter, such as encryption or compression, is to be applied to all the files, it is more efficient to apply the filter to the entire archive as a single file. The standard developers considered interfaces that would invoke a shell script for each file going into or out of the archive, but the system overhead in this approach was considered to be too high.

One such approach would be to have `filter=` records that give a pathname for an executable. When the program is invoked, the file and archive would be open for standard input/output and all the header fields would be available as environment variables or command-line arguments. The standard developers did discuss such schemes, but they were omitted from the standard due to concerns about excessive overhead. Also, the program itself would need to be in the archive if it were to be used portably.

There is currently no portable means of identifying the character set(s) used for a file in the file system. Therefore, `pax` has not been given a mechanism to generate `charset` records automatically. The only portable means of doing this is for the user to write the archive using the `−o charset=`*string* command-line option. This assumes that all of the files in the archive use the same encoding. The "implementation defined" text is included to allow for a system that can identify the encodings used for each of its files.

The table of standards that accompanies the `charset` record description is acknowledged to be very limited. Only a limited number of character set standards is reasonable for maximal interchange. Any character set is, of course, possible by prior agreement. It was suggested that EBCDIC be listed, but it was omitted because it is not defined by a formal standard. Formal standards, and then only those with reasonably large followings, can be included here, simply as a matter of practicality. The <*value*>s represent names of officially registered character sets in the format required by ISO 2375 {B5}.

The normal comma-or-blank-separated-list rules are not followed in the case of keyword options to allow ease of argument parsing for `getopts`.

Further information on character encodings is in the following Rationale for Archive Character Set Encoding/Decoding.

The standard developers have reserved keyword name space for vendor extensions. It is suggested that the format to be used is:

VENDOR.keyword

where *VENDOR* is the name of the vendor or organization in all uppercase letters. It is further suggested that the *keyword* following the period be named differently than any of the standard keywords so that it could be used for future standardization, if appropriate, by omitting the *VENDOR* prefix.

The <*length*> field in the extended header record was included to make it simpler to step through the records, even if a record contains an unknown format (to a particular `pax`) with complex interactions of special characters. It also provides a minor integrity checkpoint within the records to aid a program attempting to recover files from a damaged archive.

There are no extended header versions of the *devmajor* and *devminor* fields because the unspecified-format `ustar` header field should be sufficient. If they are not, vendor-specific extended keywords (such as *VENDOR*`.devmajor`) should be used.

Device and i-number labeling of files was not adopted from `cpio`; files are interchanged strictly on a symbolic name basis, as in `ustar`.

This version of POSIX.2 contains only namespace placeholders for security and realtime extensions. The POSIX working groups responsible for those areas are expected to amend this standard to provide additional details. It is currently unknown whether they would prescribe a single string of text or would allocate keywords at a finer granularity, such as `realtime.`*foo* or `security.`*bar*.

The POSIX security working group has not yet populated its "`security.`" name space. When it amends this standard, the POSIX security working group will presumably define the relationship between its records [which will probably define some sort of access control list (ACL)] and the modes and permissions found in the `ustar` headers. Vendor-specific extended keywords (such as *VENDOR*`.security`) should be used for any implementation-specific security arrangements.

Just as with the `ustar` format descriptions, the new format makes no special arrangements for multivolume archives. Each of the `pax` archive types is assumed to be inside a single POSIX file and splitting that file over multiple volumes (diskettes, tape cartridges, etc.), processing their labels, and mounting each in the proper sequence are considered to be implementation details that cannot be described portably. Perhaps the POSIX system administration working group will provide portable solutions for this.

The pax format is intended for interchange, not only for backup on a single (family of) systems. It is not as densely packed as might be possible for backup:

— It contains information as coded characters that could be coded in binary.

— It identifies extended records with name fields that could be omitted in favor of a fixed-field layout.

— It translates names into a portable character set and identifies locale-related information, both of which are probably unnecessary for backup.

The requirements on restoring from an archive are slightly different from the historical wording, allowing for nonmonolithic privilege to bring forward as much as possible. In particular, attributes such as "high performance file" might be broadly but not universally granted while set-user-ID or *chown*() might be much more restricted. There is no implication in this standard that the security information be honored after it is restored to the file hierarchy, in spite of what might be improperly inferred by the silence on that topic. That is a topic for another standard.

Links are recorded in the fashion described here because a link can be to any file type. It is desirable in general to be able to restore part of an archive selectively and restore all of those files completely. If the data is not associated with each link, it is not possible to do this. However, the data associated with a file can be large, and when selective restoration is not needed, this can be a significant burden. The archive is structured so that files that have no associated data can always be restored by the name of any linkname of any link, and the user may choose whether data is recorded with each instance of a file that contains data. The format permits mixing of both types of links in a single archive; this can be done for special needs, and pax is expected to interpret such archives on input properly, despite the fact that there is no pax option that would force this mixed case on output. (When −o linkdata is used, the output must contain the duplicate data, but the implementation is free to include it or omit it when −o linkdata is not used.)

The time values are included as extended header records for those implementations needing more than the eleven octal digits allowed by the ustar format. Even though some implementations can support finer file-time granularities than seconds, the normative text requires support only for seconds since the Epoch because POSIX.1 {8} states them that way. The ustar format includes only *mtime*; the new format adds atime and ctime for symmetry. The atime access time restored to the file system will be affected by the −p a and −p e options. The ctime creation time (actually i-node modification time) is described with "appropriate privilege" so that it can be ignored when writing to the file system. POSIX does not provide a portable means to change file creation time. Nothing is intended to prevent a nonportable implementation of pax from restoring the value.

The gid, size, and uid extended header records were included to allow expansion beyond the sizes specified in the regular tar header. New file system architectures are emerging that will exhaust the 12-digit *size* field. There are probably not many systems requiring more than 8 digits for user and group IDs, but the B B B B extended header values were included for completeness, allowing overrides for all of the decimal values in the tar header. B

The standard developers intended to describe the effective results of pax with regard to file ownerships and permissions; implementations are not restricted in timing or sequencing the restoration of such, provided the results are as specified.

Much of the text describing the extended headers refers to use in "write or copy modes." The copy-mode references are due to the normative text: "The effect of the copy shall be as if the copied files were written to an archive file and then subsequently extracted ...." There is certainly no way to test whether pax is actually generating the extended headers in copy mode, but the effects must be as if it had.

Rationale for pax Archive Character Set Encoding/Decoding

There is a need to exchange archives of files between systems of different native codesets. File names, group names, and user names must be preserved to the fullest extent possible when an archive is read on the receiving platform. Translation of the contents of files is not within the scope of the pax utility.

There will also be the need to represent glyphs that are not available on the receiving platform. (A *glyph* is commonly called a character, but without any reference to a specific encoding of that character. The term glyph refers to the symbol itself.) These unsupported glyphs cannot be automatically folded to the local set of glyphs due to the chance of collisions. This could result in overwriting previous extracted files from the archive or pre-existing files on the system.

For these reasons, the codeset used to represent glyphs within the extended header records of the pax archive must be sufficiently rich to handle all commonly used character sets. The fields requiring translation include, at a minimum, file names, user names, group names, and link pathnames. The POSIX security group and other working groups may specify other extended header records requiring similar treatment and implementations may wish to have localized extended keywords that use nonportable characters.

The standard developers considered the following options:

— The archive creator specifies the well-defined name of the source codeset. The receiver must then recognize the codeset name and perform the appropriate translations to the destination codeset.

— The archive creator includes within the archive the character mapping table for the source codeset used to encode extended header records. The receiver must then read the character mapping table and perform the appropriate translations to the destination codeset.

— The archive creator translates the extended header records in the source codeset into a canonical form. The receiver must then perform the appropriate translations to the destination codeset.

The approach that incorporates the name of the source codeset poses the problem of codeset name registration, and makes the archive useless to pax archive decoders that do not recognize that codeset.

Because parts of an archive may be corrupted, the standard developers felt that including the character map of the source codeset was too fragile. The loss of this one key component could result in making the entire archive useless. (The difference between this and the global extended header decision was that the latter has a workaround—duplicating extended header records on unreliable media—but this would be too burdensome for large character set maps.)

Both of the above approaches also put an undue burden on the pax archive receiver to handle the cross-product of all source and destination codesets.

To simplify the translation from the source codeset to the canonical form and from the canonical form to the destination codeset, the standard developers decided that the internal representation should be a stateless encoding. A stateless encoding is one where each codepoint has the same meaning, without regard to the decoder being in a specific state. An example of a stateful encoding would be the Japanese Shift-JIS; an example of a stateless encoding would ISO/IEC 646 {1} (equivalent to 7 b ASCII).

For these reasons, the standard developers decided to adopt a canonical format for the representation of file information strings. The obvious, well-endorsed candidate is ISO/IEC 10646 {10} (based in part on Unicode), which can be used to represent the glyphs of virtually all standardized character sets. The standard developers initially agreed upon using UCS2 (16 b Unicode) as the internal representation. This repertoire of glyphs provides a sufficiently rich set to represent all commonly-used codesets.

However, the standard developers found that the 16 b Unicode representation had some problems. It forced the issue of standardizing byte ordering. The 2 B length of each character made the extended header records twice as long for the case of strings coded entirely from historical 7 b ASCII. For these reasons, the standard developers chose the UTF8 (File-System Safe Universal Translation Format) defined in ISO/IEC 10646 {10}. This multibyte representation encodes UCS2 or UCS4 characters reliably and deterministically, eliminating the need for a canonical byte ordering. In addition, NUL octets and other characters possibly confusing to POSIX file systems do not appear, except to represent themselves. It was realized that certain national codesets take up more space after the encoding, due to their placement within the UCS range; it was felt that the usefulness of the encoding of the names outweighs the disadvantage of size increase for file, user, and group names.

The encoding of UTF8 is as follows:

UCS4 Hex Encoding	**UTF8 Binary Encoding**
00000000-0000007F	0xxxxxxx
00000080-000007FF	110xxxxx 10xxxxxx
00000800-0000FFFF	1110xxxx 10xxxxxx 10xxxxxx
00010000-001FFFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
00200000-03FFFFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
04000000-7FFFFFFF	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

where each x represents a bit value from the character being translated.

Rationale for the ustar Interchange Format B

The description of the ustar format reflects numerous enhancements over pre-1988 versions of the historical tar utility. The goal of these changes was not only to provide the functional enhancements desired, but also to retain compatibility between new and old versions. This compatibility has been retained. Archives written using the old archive format are compatible with the new format.

Implementors should be aware that the previous file format did not include a mechanism to archive directory type files. For this reason, the convention of using a file name ending with slash was adopted to specify a directory on the archive.

The total size of the name and prefix fields have been set to meet the minimum requirements for {PATH_MAX}. If a pathname will fit within the name field, it is recommended that the pathname be stored there without the use of the prefix field. Although the name field is known to be too small to contain {PATH_MAX} characters, the value was not changed in this version of the archive file format to retain backward compatibility, and instead the *prefix* was introduced. Also, because of the earlier version of the format, there is no way to remove the restriction on the *linkname* field being limited in size to just that of the *name* field.

The *size* field is required to be meaningful in all implementation extensions, although it could be zero. This is required so that the data blocks can always be properly counted.

It is suggested that if device special files need to be represented that cannot be represented in the standard format that one of the extension types ('A'-'Z') be used, and that the additional information for the special file be represented as data and be reflected in the size field.

Attempting to restore a special file type, where it is converted to ordinary data and conflicts with an existing file name, need not be specially detected by the utility. If run as an ordinary user, pax should not be able to overwrite the entries in, for example, /dev in any case (whether the file is converted to another type or not). If run as a privileged user, it should be able to do so, and it would be considered a bug if it did not. The same is true of ordinary data files and similarly named special files; it is impossible to anticipate the needs of the user (who could really intend to overwrite the file), so the behavior should be predictable (and thus regular) and rely on the protection system as required.

The value '7' in the typeflag field is intended to define how contiguous files can be stored in a ustar archive. POSIX.1 {8} does not require the contiguous file extension, but does define a standard way of archiving such files so that all conforming systems can interpret these file types in a meaningful and consistent manner. On a system that does not support extended file types, the pax utility should do the best it can with the file and go on to the next.

The file protection modes are those conventionally used by the ls utility. This is extended beyond the usage in POSIX.2 to support the "shared text" or "sticky" bit. It is intended that the conformance document should not document anything beyond the existence of and support of such a mode. Further extensions are expected to these bits, particularly with overloading the set-user-ID and set-group-ID flags.

Rationale for the cpio Interchange Format B

The reference to appropriate privilege in the `cpio` format refers to an error on standard output; the `ustar` format does not make comparable statements.

The model for this format was the historical System V `cpio` `–c` data interchange format. This model documents the portable version of the `cpio` format and not the binary version. It has the flexibility to transfer data of any type described within POSIX.1 {8}, yet is extensible to transfer data types specific to extensions beyond POSIX.1 {8} (e.g., or contiguous files). Because it describes existing practice, there is no question of maintaining upward compatibility.

`cpio` Header

There has been some concern that the size of the *c_ino* field of the header is too small to handle those systems that have very large i-node numbers. However, the *c_ino* field in the header is used strictly as a hard link resolution mechanism for archives. It is not necessarily the same value as the i-node number of the file in the location from which that file is extracted.

The name *c_magic* is based on historical usage.

`cpio` File Name

For most historical implementations of the `cpio` utility, {PATH_MAX} octets can be used to describe the pathname without the addition of any other header fields (the NUL character would be included in this count). {PATH_MAX} is the minimum value for pathname size, documented as 256 B in Section 2 of POSIX.1 {8}. However, an implementation may use *c_namesize* to determine the exact length of the pathname. With the current description of the `cpio` header, this pathname size can be as large as a number that is described in six octal digits.

Two values are documented under the *c_mode* field values to provide for extensibility for known file types:

`0110 000`	Reserved for contiguous files. The implementation may treat the rest of the information for this archive like a regular file. If this file type is undefined, the implementation may create the file as a regular file.
`0140 000`	Reserved for sockets. If this type is undefined on the target system, the implementation may decide to ignore this file type and output a warning message.

This provides for extensibility of the `cpio` format while allowing for the ability to read old archives. Files of an unknown type may be read as “regular files” on some implementations. On a system that does not support extended file types, the `pax` utility should do the best it can with the file and go on to the next.

In POSIX.1 {8}, the symbolic link value was reserved, but this has been deleted in light of support for symbolic links elsewhere in this standard.

⇒ **E.5.10 ex Rationale.** *Replace the full rationale for* ex *with the following.* B

E.5.10 ex – Text editor

The ex/vi specification is based on the historical practice found in the 4BSD and System V implementations of ex and vi. A freely redistributable implementation of ex/vi, which is tracking this specification fairly closely, and demonstrates the intended changes between historical implementations and this specification, may be obtained from Keith Bostic (bostic@cs.berkeley.edu) or by anonymous FTP from:

```
ftp.cs.berkeley.edu:ucb/4bsd/nvi.tar.gz
```

A "restricted editor" (both the historical red utility and modifications to ex) were considered and rejected for inclusion. Neither option provided the level of security that users might expect.

E.5.10.1 Synopsis

There is no additional rationale provided for this subclause.

E.5.10.2 Description

It is recognized that ex visual mode and related features would be difficult, if not impossible, to implement satisfactorily on a block-mode terminal, or a terminal without any form of cursor addressing; thus, it is not a mandatory requirement that such features should work on all terminals. It is the intention, however, that an ex implementation should provide the full set of capabilities on all terminals capable of supporting them.

E.5.10.3 Options

The −c replacement for +*command* was inspired by the −e option of sed. Historically, all such commands (see edit and next as well) were executed from the last line of the edit buffer. This meant, for example, that +/*pattern* would fail unless the wrapscan option was set. This standard requires conformance to historical practice. Historically, some implementations restricted the ex commands that could be listed as part of the command-line arguments. For consistency, this standard does not permit these restrictions.

Historically, the ex and vi utilities accepted a −l option, which set the lisp and showmatch edit options. The −l option was omitted because it was difficult to justify the inclusion of programming-language dependent features. Similarly, the lisp edit option was omitted.

In historical implementations of the editor, the −R option (and the readonly edit option) only prevented overwriting of files; appending to files was still permitted, mapping loosely into the csh **noclobber** variable. Some implementations, however, have not followed this semantic, and readonly does not permit appending either. This standard follows the latter practice, believing that it is a more obvious and intuitive meaning of readonly.

The −s option (and its obsolescent single-hyphen form) suppresses all interactive user feedback and is useful for editing scripts in batch jobs. The list of specific effects is historical practice. The terminal type "incapable of supporting open and visual modes" has historically been named "dumb."

The −t option was required because the ctags utility appears in POSIX.2 and the option is available in all historical implementations of ex.

Historically, the ex and vi utilities accepted a −x option, which did encryption based on the algorithm found in the historical crypt utility. The −x option for encryption, and the associated crypt utility, were omitted because the algorithm used was not specifiable and the export control laws of some nations make it difficult to export cryptographic technology. In addition, it did not historically provide the level of security that users might expect.

E.5.10.4 Operands

There is no additional rationale provided for this subclause.

E.5.10.5 External Influences

E.5.10.5.1 Standard Input

An end-of-file condition is not equivalent to an end-of-file character. A common end-of-file character, <control-D>, is historically an ex command.

There was no maximum line length in historical implementations of ex. Specifically, as it was parsed in chunks, the addresses had a different maximum length than the filenames. Further, the maximum line buffer size was declared as {BUFSIZ}, which was different lengths on different systems. This version of this standard selected the value of {LINE_MAX} to impose a reasonable restriction on portable usage of ex and to aid test-suite writers in their development of realistic tests that exercise this limit.

E.5.10.5.2 Input Files

It was an explicit decision by the standard developers that a <newline> character be added to any file lacking one. It was believed that this feature of ex and vi was relied on by users in order to make text files lacking a trailing <newline> more portable. It is recognized that this will require a user specified option or extension for implementations that permit ex and vi to edit files of type other than text if such files are not otherwise identified by the system. It was agreed that the ability to edit files of arbitrary type can be useful, but it was not considered necessary to mandate that an ex or vi implementation be required to handle files other than text files.

The paragraph in the Input Files subclause, "By default, . . . ," is intended to close a long-standing security problem in ex and vi, that of the "modeline" or "modelines" edit option. This feature allows any line in the first or last five lines of the file containing the strings ex: or vi: (and, apparently, ei: or vx:) to be a line containing editor commands, and ex interprets all the text up to the next : or <newline> as a command. Consider the consequences, for example, of an unsuspecting user using ex or vi as the editor when replying to a mail message in which a line such as

```
ex:! rm -rf *:
```

appeared in the signature lines. The standard developers believed strongly that an editor should not by default interpret any lines of a file. Vendors are strongly urged to delete this feature from their implementations of ex and vi.

E.5.10.5.3 Environment Variables

There is no additional rationale provided for this subclause.

E.5.10.5.4 Asynchronous Events

The intention of the phrase "complete write" is that the entire edit buffer be written to stable storage. The note regarding temporary files is intended for implementations that use temporary files to back edit buffers unnamed by the user.

Historically, SIGQUIT was ignored by ex, but was the equivalent of Q in visual mode; i.e., it exited visual mode and entered ex mode. This standard permits, but does not require, this behavior. Historically, SIGINT was often used by vi users to terminate text input mode (<control-C> is often easier to enter than <ESC>). Some implementations of vi alerted the terminal on this event, and some did not. This standard requires that SIGINT behave identically to <ESC>, and that the terminal not be alerted.

Historically, suspending the ex editor during text input mode was similar to SIGINT, as completed lines were retained, but any partial line discarded, and the editor returned to command mode. This standard is silent on this issue; implementations are encouraged to follow historical practice, where possible.

Historically, the vi editor did not treat SIGTSTP as an asynchronous event, and it was therefore impossible to suspend the editor in visual text input mode. There are two major reasons for this. The first is that SIGTSTP is a broadcast signal on UNIX systems, and the chain of events where the shell execs an application that then execs vi usually caused confusion for the terminal state if SIGTSTP was delivered to the process group in the default manner. The second was that most implementations of the UNIX *curses* package are not reentrant, and the receipt of SIGTSTP at the wrong time will cause them to crash. This standard is silent on this issue; implementations are encouraged to treat suspension as an asynchronous event if possible.

Historically, modifications to the edit buffer made before SIGINT interrupted an operation were retained; i.e., anywhere from zero to all of the lines to be modified might have been modified by the time the SIGINT arrived. These changes were not discarded by the arrival of SIGINT. This standard permits this behavior, noting that the `undo` command is required to be able to undo these partially completed commands.

The action taken for signals other than SIGINT, SIGCONT, SIGHUP, and SIGTERM is unspecified because some implementations attempt to save the edit buffer in a useful state when other signals are received.

E.5.10.6 External Effects

E.5.10.6.1 Standard Output

There is no additional rationale provided for this subclause.

E.5.10.6.2 Standard Error

For `ex`/`vi`, diagnostic messages are those messages reported as a result of a failed attempt to invoke `ex` or `vi`, such as invalid options or insufficient resources, or an abnormal termination condition. Diagnostic messages should not be confused with the error messages generated by inappropriate or illegal user commands.

E.5.10.6.3 Output Files

There is no additional rationale provided for this subclause.

E.5.10.7 Extended Description

E.5.10.7.1 `ex` and `vi` Initialization

If an `ex` command (other than `cd`, `chdir`, or `source`) has a file name argument, one or both of the alternate and current pathnames will be set. Informally, they are set as follows:

(1) If the `ex` command is one that replaces the contents of the edit buffer, and it succeeds, the current pathname will be set to the file name argument (the first file name argument in the case of the next command) and the alternate pathname will be set to the previous current pathname, if there was one.

(2) In the case of the file read/write forms of the read and write commands, if there is no current pathname, the current pathname will be set to the file name argument.

(3) Otherwise, the alternate pathname will be set to the file name argument.

For example, `:edit foo` and `:recover foo`, when successful, set the current pathname, and, if there was a previous current pathname, the alternate pathname. The commands `:write !command` and `:edit` set neither the current or alternate pathnames. If the `:edit foo` command were to fail for some reason, the alternate pathname would be set. The read and write commands set the alternate pathname to their *file* argument, unless the current pathname is not set, in which case they set the current pathname to their *file* arguments. The alternate pathname was not historically set by the :source command. This standard requires conformance to historical practice. Implementations adding commands that take file names as arguments are encouraged to set the alternate pathname as described here. C

Historically, ex and vi read the .exrc file in the **$HOME** directory twice, if the editor was executed in the **$HOME** directory. This standard prohibits this behavior.

Historically, the historical 4BSD ex and vi read the **$HOME** and local .exrc files if they were owned by the real ID of the user, or the sourceany option was set, regardless of other considerations. This was a security problem because it is possible to put normal UNIX commands inside a .exrc file. This standard does not specify the sourceany option, and historical implementations are encouraged to delete it.

The .exrc files must be owned by the real ID of the user, and not writeable by anyone other than the owner. The appropriate privileges exception is intended to permit users to acquire special privileges, but continue to use the .exrc files in their home directories.

System V release 3.2 and later vi implementations added the option [no]exrc. The behavior is that local .exrc files are read only if the exrc option is set. The default for the exrc option was off, so by default, local .exrc files were not read. The problem this was intended to solve was that System V permitted users to give away files, so there is no possible ownership or writeability test to ensure that the file is safe. This is still a security problem on systems where users can give away files, but there is nothing additional that this standard can do. The implementation-defined exception is intended to permit groups to have local .exrc files that are shared by users, by creating pseudo-users to own the shared files.

This standard does not mention system-wide ex and vi startup files. While they exist in several implementations of ex and vi, they are not present in any implementations considered historical practice by this standard. Implementations that have such files should use them only if they are owned by the real user ID or an appropriate user (e.g., root on UNIX systems) and if they are not writeable by any user other than their owner. System-wide startup files should be read before the **EXINIT** variable, $HOME/.exrc or local .exrc files are evaluated.

Historically, any ex command could be entered in the **EXINIT** variable or the .exrc file, although ones requiring that the edit buffer already contain lines of text generally caused historical implementations of the editor to drop core. This standard requires that any ex command be permitted in the **EXINIT** variable and .exrc files, for simplicity of specification and consistency, although many of them will obviously fail under many circumstances.

The initialization of the contents of the edit buffer uses the phrase "the effect shall be" with regard to various ex commands. The intent of this phrase is that C edit buffer contents loaded during the initialization phase not be lost; i.e., loading the edit buffer should fail if the `.exrc` file read in the contents of a file and did not subsequently write the edit buffer. An additional intent of this phrase is to specify that the initial current line and column is set as specified for the individual `ex` commands.

Historically, the `−t` option behaved as if the tag search were a +*command*; i.e., it was executed from the last line of the file specified by the tag. This resulted in the search failing if the pattern was a forward search pattern and the `wrapscan` edit option was not set. This standard does not permit this behavior, requiring that the search for the tag pattern be performed on the entire file, and, if not found, that the current line be set to a more reasonable location in the file.

Historically, the empty edit buffer presented for editing when a file was not specified by the user was unnamed. This is permitted by the standard, however, implementations are encouraged to provide users a temporary file name for this buffer because it permits them the use of `ex` commands that use the current pathname during temporary edit sessions.

Historically, the file specified using the `−t` option was not part of the current argument list. This practice is permitted by the standard, however, implementations are encouraged to include its name in the current argument list for consistency.

Historically, the `−c` command (or +*command*) was generally not executed until a file that already exists was edited. This standard requires conformance to this historical practice. Commands that could cause the `−c` command to be executed include the `ex` commands `edit`, `next`, `recover`, `rewind`, and `tag`, and the `vi` commands `<control-^>` and `<control-]>`. Historically, reading a file into an edit buffer did not cause the `−c` command to be executed (even though it might set the current pathname) with the exception that it did cause the `−c` command to be executed if: the editor was in `ex` mode, the edit buffer had no current pathname, the edit buffer was empty, and no read commands had yet been attempted. For consistency and simplicity of specification, this standard does not permit this behavior.

Historically, the `−r` option was the same as a normal edit session if there was no recovery information available for the file. This allowed users to enter "`vi −r *.c`" and recover whatever files were recoverable. In some implementations, recovery was attempted only on the first file named, and the file was not entered into the argument list; in others, recovery was attempted for each file named. In addition, some historical implementations ignored `−r` if `−t` was specified or did not support command-line file arguments with the `−t` option. For consistency and simplicity of specification, this standard disallows these special cases, and requires that recovery be attempted the first time each file is edited.

Historically, `vi` initialized the ` and ' marks, but `ex` did not. This meant that if the first command in `ex` mode was "visual," or if an `ex` command was executed first (e.g., `vi +10 file`), `vi` was entered without the marks being initialized. Because the standard developers believed the marks to be generally useful, and for consistency and simplicity of specification, this standard requires that they always be initialized if in open or visual mode, or if in `ex` mode and the edit buffer is not empty. Not initializing it in `ex` mode if the edit buffer is empty is historical practice, however it has always been possible to set (and use) marks in empty edit buffers in open and visual mode edit sessions.

E.5.10.7.2 Addressing

Historically, `ex` and `vi` accepted the additional addressing forms `\/` and `\?`. They were equivalent to `//` and `??`, respectively. They are not required by this standard, mostly because nobody can remember if they ever did anything different historically or not.

Historically, `ex` and `vi` permitted an address of zero for several commands, and permitted the `%` address in empty files for others. For consistency, this standard requires support for the former in the few commands where it makes sense and disallows it otherwise. In addition, because this standard requires that `%` be logically equivalent to `1,$`, it is also supported where it makes sense and disallowed otherwise.

Historically, the `%` address could not be followed by further addresses. For consistency and simplicity of specification, this standard requires that additional addresses be supported. C C C

All of the following are valid addresses:

`+++`	Three lines after the current line
/*pattern*/–	One line before the next occurrence of *pattern*
`–2`	Two lines before the current line
`3 ---- 2`	Line one (note intermediate negative address)
`1 2 3`	Line six

Any number of addresses can be provided to commands taking addresses; e.g., `1,2,3,4,5p` prints lines 4 and 5, because two is the greatest valid number of addresses accepted by the `print` command. This, in combination with the semicolon delimiter, permits users to create commands based on ordered patterns in the file. For example, the command `3;/foo/;+2print` will display the first line after line 3 that contains the pattern `foo`, plus the next two lines. Note that the address "`3;`" must be evaluated before being discarded because the search origin for the `/foo/` command depends on this.

Historically, values could be added to addresses by including them after one or more `<blank>` characters; e.g., `3 - 5p` wrote the seventh line of the file, and `/foo/ 5` was the same as `/foo/+5`. However, only absolute values could be added; e.g., `5 /foo/` was an error. This standard requires conformance to historical practice. Address offsets are separately specified from addresses because they could historically be provided to visual mode search commands.

Historically, any missing addresses defaulted to the current line. This was true for leading and trailing comma-delimited addresses, and for trailing semicolon-delimited addresses. For consistency, this standard requires it for leading semicolon addresses as well.

Historically, ex and vi accepted the ˆ character as both an address and as a flag offset for commands. In both cases it was identical to the "-" character. This standard does not require or prohibit this behavior.

Historically, the enhancements to BREs could be used in addressing: e.g., **~**, \<, and \>. This standard requires conformance to historical practice; i.e., that RE usage be consistent, and that RE enhancements be supported wherever REs are used.

E.5.10.7.3 ex Command-Line Parsing

Historical ex command parsing was even more complex than that described by this standard. This standard requires the subset of the command parsing that the standard developers believed was documented and that users could reasonably be expected to use in a portable fashion, and that was historically consistent between implementations. (The discarded functionality is obscure, at best.) Historical implementations will require changes in order to comply with this standard; however, users are not expected to notice any of these changes. Most of the complexity in ex parsing is to handle three special termination cases:

(1) The !, global, v, and the filter versions of the read and write commands are delimited by <newline>s (they can contain vertical-line characters that are usually shell pipes).

(2) The ex, edit, next and visual in open and visual mode commands all take ex commands, optionally containing vertical-line characters, as their first arguments.

(3) The s command takes an RE as its first argument, and uses the delimiting characters to delimit the command.

Historically, vertical-line characters in the +*command* argument of the ex, edit, next, vi, and visual commands, and in the *pattern* and *replacement* parts of the s command, did not delimit the command, and in the filter cases for read and write, and the !, global, and v commands, they did not delimit the command at all. For example, the following commands are all valid: C C C C

```
:edit +25|s/abc/ABC/ file.c
:s/|/PIPE/
:read !spell % | columnate
:global/pattern/p|l
:s/a/b/|s/c/d|set
```

Historically, empty or <blank>-filled lines in .exrc files and sourced files (as well as **EXINIT** variables and ex command scripts) were treated as default commands; i.e., print commands. This standard specifically requires that they be ignored when encountered in .exrc and sourced files to eliminate a common source of new-user error. C C

Historically, `ex` commands with multiple adjacent (or `<blank>` separated) vertical lines were handled oddly when executed from `ex` mode. For example, the command “`||| <carriage-return>`”, when the cursor was on line 1, displayed lines 2, 3, and 5 of the file. In addition, the command “ `|` ” would only display the line after the next line, instead of the next two lines. The former worked more logically when executed from `vi` mode, and displayed lines 2, 3, and 4. This standard requires the `vi` behavior, i.e., a single default command and line number increment for each command separator, and trailing `<newline>` characters after vertical-line separators are discarded.

Historically, `ex` permitted a single extra colon as a leading command character; e.g., `:g/pattern/:p` was a valid command. This standard generalizes this to require that any number of leading colon characters be stripped.

Historically, any prefix of the `delete` command could be followed without intervening `<blank>`s by a flag character because in the command “`d p`”, `p` is interpreted as the buffer `p`. This standard requires conformance to historical practice.

Historically, the `k` command could be followed by the mark name without intervening `<blank>` characters. This standard requires conformance to historical practice.

Historically, the `s` command could be immediately followed by flag and option characters; e.g., `s/e/E/|s|sgc3p` was a valid command. However, flag characters could not stand alone; e.g., the commands “`sp`” and “`s l`” would fail, while the command “`sgp`” and “`s gl`” would succeed. (Obviously, the # flag character was used as a delimiter character if it followed the command.) Another issue was that option characters had to precede flag characters even when the command was fully specified; e.g., the command “`s/e/E/pg`” would fail, while the command “`s/e/E/gp`” would succeed. This standard requires conformance to historical practice.

Historically, the first command name that had a prefix matching the input from the user was the executed command; e.g., `ve`, `ver`, and `vers` all executed the `version` command. Commands were in a specific order, however, so that `a` matched `append`, not `abbreviate`. This standard requires conformance to historical practice. The restriction on command search order for implementations with extensions is to avoid the addition of commands such that the historical prefixes would fail to work portably.

Historical implementations of `ex` and `vi` did not correctly handle multiple `ex` commands, separated by vertical-line characters, that entered or exited visual mode or the editor. Because implementations of `vi` exist that do not exhibit this failure mode, this standard does not permit it.

The requirement that alphabetic command names consist of all following alphabetic characters up to the next nonalphabetic character means that alphabetic command names must be separated from their arguments by one or more nonalphabetic characters, normally a `<blank>` or `!` character, except as specified for the exceptions, the `delete`, `k`, and `s` commands.

Historically, the repeated execution of the `ex` default print commands (`<control-D>`, *eof*, `<newline>`, `<carriage-return>`) erased any prompting character and displayed the next line(s) without scrolling the terminal; i.e., immediately below any previously displayed lines. This provided a cleaner presentation of the lines in the file for the user. This standard does not require this behavior because it may be impossible in some situations; however, implementations are strongly encouraged to provide this semantic if possible.

Historically, it was possible to change files in the middle of a command, and have the rest of the command executed in the new file, e.g.,

```
     :edit +25 file.c|s/abc/ABC/|1
```

was a valid command, and the substitution was attempted in the newly edited file. This standard requires conformance to historical practice. The following commands are examples that exercise the `ex` parser:

```
     echo 'foo|bar' > file1; echo 'foo/bar' > file2;
     vi
     :edit +1|s/|/PIPE/|w file1| e file2|1 | s/\//SLASH/|wq
```

Historically, there was no protection in editor implementations to avoid `ex` `global`, `v`, `@`, or `*` commands changing edit buffers during execution of their associated commands. Because this would almost invariably result in catastrophic failure of the editor, and implementations exist that do exhibit these problems, this standard requires that changing the edit buffer during a `global` or `v` command, or during a `@` or `*` command for which there will be more than a single execution, be an error. Implementations supporting multiple edit buffers simultaneously are strongly encouraged to apply the same semantics to switching between buffers as well. c

The `ex` command quoting required by this standard is a superset of the quoting in historical implementations of the editor. For example, it was not historically possible to escape a `<blank>` character in a file name; e.g., `:edit foo\\\ bar` would report that too many file names had been entered for the `edit` command, and there was no method of escaping a `<blank>` in the first argument of an edit, `ex`, `next`, or `visual` command at all. This standard extends historical practice, requiring that quoting behavior be made consistent across all `ex` commands, except for the `map`, `unmap`, `abbreviate`, and `unabbreviate` commands, which historically used `<control-V>` instead of backslashes for quoting. For those four commands, this standard requires conformance to historical practice.

Backslash quoting in `ex` is nonintuitive. Backslash escapes are ignored unless they escape a special character; e.g., when performing file argument expansion, the string `\\%` is equivalent to `\%`, not `\`<*current pathname*>. This can be confusing for users because backslash is usually one of the characters that causes shell expansion to be performed, and therefore shell quoting rules must be taken into consideration. Generally, quoting characters are only considered if they escape a special character, and a quoting character must be provided for each layer of parsing for which the character is special. As another example, only a single backslash is necessary for the `\l` sequence in substitute replacement patterns, because the character `l` is not special to any parsing layer above it.

`<Control-V>` quoting in `ex` is slightly different from backslash quoting. In the four commands where `<control-V>` quoting applies (`abbreviate`, `unabbreviate`, `map` and `unmap`), any character may be escaped by a `<control-V>` whether it would have a special meaning or not. This standard requires conformance to historical practice.

Historical implementations of the editor did not require delimiters within character classes to be escaped; e.g., the command `:s/[/]//` on the string `xxx/yyy` would delete the / from the string. This standard disallows this historical practice for consistency and because it places a large burden on implementations by requiring that knowledge of REs be built into the editor parser.

Historically, quoting `<newline>` characters in `ex` commands was handled inconsistently. In most cases, the `<newline>` always terminated the command, regardless of any preceding escape character, because backslash characters did not escape `<newline>` characters for most `ex` commands. However, some `ex` commands (e.g., `s`, `map`, and abbreviation) permitted `<newline>`s to be escaped (although in the case of `map` and abbreviation, `<control-V>` characters escaped them instead of backslashes). This was true in not only the command line but also `.exrc` and `source`d files. For example, the command

```
map = foo<control-V><newline>bar
```

would succeed, although it was sometimes difficult to get the `<control-V>` and the inserted `<newline>` passed to the `ex` parser. For consistency and simplicity of specification, this standard requires that it be possible to escape `<newline>` characters in `ex` commands at all times, using backslashes for most `ex` commands, and using `<control-V>` characters for the `map` and abbreviation commands. For example, the command `print<newline>list` is required to be parsed as the single command `print<newline>list`. While this differs from historical practice, the standard developers believed it unlikely that any script or user depended on the historical behavior.

Historically, an error in a command specified using the `−c` option did not cause the rest of the `−c` command(s) to be discarded. This standard disallows this for consistency with mapped keys, the `@`, `global`, `source`, and `v` commands, the **EXINIT** environment variable, and the `.exrc` files.

E.5.10.7.4 ex Input Editing

One of the common uses of the historical `ex` editor is over slow network connections. Editors that run in canonical mode can require far less traffic to and from, and far less processing on, the host machine, as well as more easily supporting block-mode terminals. For these reasons, this standard requires that `ex` be implemented using canonical mode input processing, as was done historically.

The POSIX.1 {8} standard does not require the historical 4BSD input editing characters "word erase" or "literal next." For this reason, it is unspecified how they are handled by `ex`, although they must have the required effect. Implementations that resolve them after the line has been ended using a `<newline>` or `<control-M>` character, and implementations that rely on the underlying system terminal support for this processing, are both conforming. Implementations are strongly urged to use the underlying system functionality, if at all possible, for compatibility with other system text input interfaces.

Historically, when the *eof* character was used to decrement the autoindent level, the cursor moved to display the new end of the autoindent characters, but did not move the cursor to a new line, nor did it erase the `<control-D>` character from the line. This standard does not specify that the cursor remain on the same line or that the rest of the line is erased; however, implementations are strongly encouraged to provide the best possible user interface; i.e., the cursor should remain on the same line, and any `<control-D>` character on the line should be erased.

The POSIX.1 {8} standard does not require the historical 4BSD input editing character "reprint," traditionally `<control-R>`, which redisplayed the current input from the user. For this reason, and because the functionality cannot be implemented after the line has been terminated by the user, this standard makes no requirements about this functionality. Implementations are strongly urged to make this historical functionality available, if possible.

Historically, `<control-Q>` did not perform a literal next function in `ex`, as it did in `vi`. This standard requires conformance to historical practice to avoid breaking C historical `ex` scripts and `.exrc` files. C

E.5.10.7.4.1 *eof*

Whether the *eof* character immediately modifies the autoindent characters in the prompt is left unspecified so that implementations can conform in the presence of systems that do not support this functionality. Implementations are encouraged to modify the line and redisplay it immediately, if possible. C

The specification of the handling of the *eof* character differs from historical prac- C tice only in that *eof* characters are not discarded if they follow normal characters C in the text input. Historically, they were always discarded. C

E.5.10.7.4.2 `<newline>`

There is no additional rationale provided for this subclause.

E.5.10.7.4.3 `<control-V>`

There is no additional rationale provided for this subclause.

E.5.10.7.4.4 `<control-W>`

There is no additional rationale provided for this subclause.

E.5.10.7.5 `ex` Command Descriptions

Historically, several commands (e.g., `global`, `v`, `visual`, `s`, `write`, `wq`, `yank`, `!`, `<`, `>`, `&`, and `~`) were executable in empty files (i.e., the default address(es) were `0`), C or permitted explicit addresses of `0` (e.g., `0` was a valid address, or `0,0`, was a C valid range). Addresses of `0`, or command execution in an empty file, make sense C only for commands that add new text to the edit buffer or write commands (because users may wish to write empty files). This standard requires this behavior for such commands and disallows it otherwise, for consistency and simplicity of specification.

A *count* to an `ex` command has been historically corrected to be no greater than the last line in a file; e.g., in a five line file, the command `1,6print` would fail, but the command `1print300` would succeed. This standard requires conformance to historical practice.

Historically, the use of flags in `ex` commands could be obscure. General historical practice was as described by this standard, but there were some special cases. For example, the `list`, `number`, and `print` commands ignored trailing address offsets; e.g., "`3p +++#`" would display line 3, and 3 would be the current line after the execution of the command. The `open` and `visual` commands ignored both the trailing offsets and the trailing flags. Also, *flags* specified to the `open` and `visual` commands interacted badly with the `list` edit option, and setting and then unsetting it during the open/visual session would cause `vi` to stop displaying lines in the specified format. For consistency and simplicity of specification, this standard does not permit any of these exceptions to the general rule.

This standard uses the word "copy" in several places when discussing buffers. This is not intended to imply implementation.

Historically, `ex` users could not specify numeric buffers because of the ambiguity this would cause; e.g., in the command `3 delete 2`, it is unclear if 2 is a buffer name or a *count*. This standard requires conformance to historical practice by default, but does not preclude extensions.

Historically, the contents of the unnamed buffer were frequently discarded after commands that did not explicitly affect it; for example, when using the `edit` command to switch files. For consistency and simplicity of specification, this standard does not permit this behavior.

The `ex` utility did not historically have access to the numeric buffers, and, furthermore, deleting lines in `ex` did not modify their contents. For example, if, after doing a delete in `vi`, the user switched to `ex`, did another delete, and then switched back to `vi`, the contents of the numeric buffers would not have changed. This standard requires conformance to historical practice. Numeric buffers are described in the `ex` portion of this standard in order to confine the description of buffers to a single location in this standard.

The metacharacters that trigger shell expansion in *file* arguments match historical practice, as does the method for doing shell expansion. Implementations wishing to provide users the flexibility to alter the set of metacharacters are encouraged to provide a `shellmeta` string edit option.

Historically, `ex` commands executed from `vi` refreshed the screen when it did not strictly need to do so; e.g., `:!date > /dev/null` does not require a screen refresh because the output of the UNIX `date` command requires only a single line of the screen. This standard requires that the screen be refreshed if it has been overwritten, but makes no requirements as to how an implementation should make that determination. Implementations may prompt and refresh the screen regardless.

The following table is a condensed version of information contained in the normative text. It is presented here to facilitate the review of the editor options that affect, or are affected by, `ex` commands or addresses. Edit options such as `list` and `number`, which affect all commands that display lines, are not exhaustively listed.

`ex` Command	**Editor Option**	
`/`	`ignorecase, magic, wrapscan`	
`?`	`ignorecase, magic, wrapscan`	
`!`	`autoprint, autowrite, shell, warn, readonly, writeany`	C
`#, number`	`list`	
`<`	`autoprint, tabstop, shiftwidth`	
`>`	`autoprint, tabstop, shiftwidth`	
`<control-D>`	`scroll`	
`append`	`autoindent, number, shiftwidth`	
`change`	`autoindent, number, shiftwidth`	
`copy`	`autoprint`	
`delete`	`autoprint`	
`global`	`ignorecase, magic, report`	
`insert`	`autoindent, number, shiftwidth`	
`join`	`autoprint`	
`list`	`number`	
`map`	`remap`	
`move`	`autoprint`	
`next`	`autowrite, readonly, writeany`	
`print`	`list, number`	
`put`	`autoprint`	
`read`	`autoprint, shell`	
`rewind`	`autowrite, readonly, writeany`	
`s`	`autoprint, ignorecase, magic`	
`shell`	`shell`	
`stop`	`autowrite, readonly, writeany`	
`suspend`	`autowrite, readonly, writeany`	
`tag`	`autoprint, autowrite, taglength, tags, readonly, writeany`	
`undo`	`autoprint`	
`v`	`ignorecase, magic, report`	
`visual`	`window`	
`write`	`readonly, shell, writeany`	
`xit`	`readonly, writeany`	
`z`	`scroll, window`	

E.5.10.7.5.1 abbreviate

Historical practice was that characters that were entered as part of an abbreviation replacement were subject to map expansions, the showmatch edit option, further abbreviation expansions, etc.; i.e., they were logically pushed onto the terminal input queue, and were not a simple replacement. This standard requires conformance to historical practice. Historical practice was that whenever a nonword character (that had not been escaped by a <control-V>) was entered after a word character, vi would check for abbreviations. The check was based on the type of the character entered before the word character of the word/nonword pair that triggered the check. The word character of the word/nonword pair that triggered the check and all characters entered before the trigger pair that were of that type were included in the check, with the exception of <blank>s, which always delimited the abbreviation. C C

This means that, for the abbreviation to work, the *lhs* must end with a word character, there can be no transitions from word to nonword characters (or vice-versa) other than between the last and next-to-last characters in the *lhs*, and there can be no <blank> characters in the *lhs*. In addition, because of the historical quoting rules, it was impossible to enter a literal <control-V> in the *lhs*. This standard requires conformance to historical practice. Historical implementations did not inform users when abbreviations that could never be used were entered; implementations are strongly encouraged to do so.

For example, the following abbreviations will work:

```
:ab (p  REPLACE
:ab p   REPLACE
:ab ((p REPLACE
```

The following abbreviations will not work:

```
:ab (   REPLACE
:ab (pp REPLACE
```

Historical practice is that words on the vi colon command line were subject to abbreviation expansion, including the arguments to the abbrev (and more interestingly) the unabbrev command. Because there are implementations that do not do abbreviation expansion for the first argument to those commands, this is permitted, but not required, by this standard. However, the following sequence:

```
:ab foo bar
:ab foo baz
```

resulted in the addition of an abbreviation of baz for the string bar in historical ex/vi, and the sequence:

```
:ab foo1 bar
:ab foo2 bar
:unabbreviate foo2
```

deleted the abbreviation foo1, not foo2. These behaviors are not permitted by this standard because they clearly violate the expectations of the user.

It was historical practice that <control-V>, not backslash, characters be interpreted as escaping subsequent characters in the `abbreviate` command. This standard requires conformance to historical practice; however, it should be noted that an abbreviation containing a <blank> will never work.

E.5.10.7.5.2 append

Historically, any text following a vertical-line command separator after an `append`, `change`, or `insert` command became part of the insert text. For example, in the command:

```
:g/pattern/append|stuff1
```

a line containing the text `stuff1` would be appended to each line matching pattern. It was also historically valid to enter:

```
:append|stuff1
stuff2
.
```

and the text on the `ex` command line would be appended along with the text inserted after it. There was an historical bug, however, that the user had to enter two terminating lines (the "`.`" lines) to terminate text input mode in this case. This standard requires conformance to historical practice, but disallows the historical need for multiple terminating lines.

E.5.10.7.5.3 args

There is no additional rationale provided for this subclause.

E.5.10.7.5.4 change

See E.5.10.7.5.2.

Historical practice for cursor positioning after the `change` command when no text is input, is as described in this standard. However, one System V implementation (version SVR4.0) is known to have been modified such that the cursor is positioned on the first address specified, and not on the line before the first address. This standard disallows this modification for consistency.

Historically, the `change` command did not support buffer arguments, although some implementations allow the specification of an optional buffer. This behavior is neither required nor disallowed by this standard.

E.5.10.7.5.5 chdir

A common extension in `ex` implementations is to use the elements of a `cdpath` edit option as prefix directories for path arguments to `chdir` that are relative pathnames and that do not have `.` or `..` as their first component. Elements in the `cdpath` edit option are colon separated. The initial value of the `cdpath` edit option is the value of the shell **CDPATH** environment variable. This feature was not included in this standard because it does not exist in any of the implementations considered historical practice by this standard.

E.5.10.7.5.6 `copy`

Historical implementations of `ex` permitted copies to lines inside of the specified range; e.g., `:2,5copy3` was a valid command. This standard requires conformance to historical practice.

E.5.10.7.5.7 `delete`

This standard requires support for the historical parsing of a delete command followed by flags, without any intervening `<blank>`s. For example:

`1dp`	
`1delep`	Deletes the first line and prints the line that was second.
`1d p`	Deletes the first line, saving it in buffer p.
`1d p1l`	(Pee-one-ell.) Deletes the first line, saving it in buffer `p`, and listing the line that was second.

E.5.10.7.5.8 `edit`

Historically, any `ex` command could be entered as a +*command* argument to the `edit` command, although some (e.g., `insert` and `append`) were known to confuse historical implementations. For consistency and simplicity of specification, this standard requires that any command be supported as an argument to the `edit` command.

Historically, the command argument was executed with the current line set to the last line of the file, regardless of whether the `edit` command was executed from visual mode or not. This standard requires conformance to historical practice.

Historically, the +*command* specified to the `edit` and `next` commands was delimited by the first `<blank>` character, and there was no way to quote them. For consistency, this standard requires that the usual `ex` backslash quoting be provided.

Historically, specifying the +*command* argument to the `edit` command required a file name to be specified as well; e.g., `:edit +100` would always fail. For consistency and simplicity of specification, this standard does not permit this usage to fail for that reason.

Historically, only the cursor position of the last file edited was remembered by the editor. This standard requires that this be supported; however, implementations are permitted to remember and restore the cursor position for any file previously edited.

E.5.10.7.5.9 `file`

Historical versions of the `ex` editor `file` command displayed a current line and number of lines in the edit buffer of 0 when the file was empty, while the `vi` `<control-G>` command displayed a current line and number of lines in the edit buffer of 1 in the same situation. This standard does not permit this discrepancy, instead requiring that a message be displayed indicating that the file is empty.

E.5.10.7.5.10 global

The two-pass operation of the global and v commands is not intended to imply implementation, only the required result of the operation.

The current line and column are set as specified for the individual ex commands. This requirement is cumulative; i.e., the current line and column must track across all the commands executed by the global or v commands.

E.5.10.7.5.11 insert

See E.5.10.7.5.2. C

Historically, insert could not be used with an address of zero; i.e., not when the edit buffer was empty. This standard requires that this command behave consistently with the append command. C

E.5.10.7.5.12 join

The action of the join command in relation to the special characters is only defined for the POSIX Locale because the correct amount of white space after a period varies; in Japanese none is required, in French only a single space, and so on.

E.5.10.7.5.13 list

The historical output of the list command was potentially ambiguous. The standard developers believed correcting this to be more important than adhering to historical practice, and this standard requires unambiguous output.

E.5.10.7.5.14 map

Historically, command mode maps only applied to command names; e.g., if the character x was mapped to y, the command fx searched for the x character, not the y character. This standard requires this behavior. Historically, entering <control-V> as the first character of a vi command was an error. Several implementations have extended the semantics of vi such that <control-V> means that the subsequent command character is not mapped. This is permitted, but not required, by this standard. Regardless, using <control-V> to escape the second or later character in a sequence of characters that might match a command map, or any character in text input mode, is historical practice, and stops the entered keys from matching a map. This standard requires conformance to historical practice. C

Historical implementations permitted digits to be used as a command map *lhs*, but then ignored the map. This standard requires that the mapped digits not be ignored.

The historical implementation of the map command did not permit command maps that were more than a single character in length if the first character was printable. This behavior is permitted, but not required, by this standard.

Specifications of “function keys” in the `map` command were omitted because the historical specification of such was too simple to be generally useful in a portable manner. Historical practice is that a # followed by a number mapped to that number function key; e.g., `#3` was function key 3 for the current terminal, as well as being accessible using the keys # and `3`. Implementations have extended this semantic to permit users to specify things like `#up` and `#page_forward` as well. These extensions are permitted, but not required, by this standard.

Historically, mapped characters were remapped unless the `remap` edit option was not set, or the prefix of the mapped characters matched the mapping characters; e.g., in the map

```
:map ab abcd
```

the characters `ab` were used as is and were not remapped, but the characters `cd` were mapped if appropriate. This can cause infinite loops in the `vi` mapping mechanisms. This standard requires conformance to historical practice, and that such loops be interruptible.

Text input maps had the same problems with expanding the *lhs* for the `ex` `map!` and `unmap!` command as did the `ex` `abbreviate` and `unabbreviate` commands. See the Rationale for the `ex` `abbreviate` command (E.5.10.7.5.1). This standard requires similar modification of some historical practice for the `map` and `unmap` commands, as described for the `abbreviate` and `unabbreviate` commands.

Historically, maps that were subsets of other maps behaved differently depending on the order in which they were defined. For example:

```
:map! ab    short
:map! abc   long
```

would always translate the characters `ab` to `short`, regardless of how fast the characters `abc` were entered. If the entry order was reversed:

```
:map! abc   long
:map! ab    short
```

the characters `ab` would cause the editor to pause, waiting for the completing `c` character, and the characters might never be mapped to `short`. For consistency and simplicity of specification, this standard requires that the shortest match be used at all times.

The length of time the editor spends waiting for the characters to complete the *lhs* is unspecified because the timing capabilities of systems are often inexact and variable, and it may depend on other factors such as the speed of the connection. The time should be long enough for the user to be able to complete the sequence, but not long enough for the user to have to wait. Some implementations of `vi` have added a `keytime` option, which permits users to set the number of 0,1 s the editor waits for the completing characters. Because mapped terminal function and cursor keys tend to start with an <ESC> character, and <ESC> is the key ending `vi` text input mode, maps starting with <ESC> characters are generally exempted from this timeout period, or, at least timed out differently.

E.5.10.7.5.15 mark

Historically, users were able to set the “previous context” marks explicitly. In addition, the ex commands ′′ and ′‘ and the vi commands ′′, ‘‘, ‘′, and ′‘ all referred to the same mark. In addition, the previous context marks were not set if the command with which the address setting the mark was associated, failed. This standard requires conformance to historical practice. Historically, if marked lines were deleted, the mark was also deleted, but would reappear if the change was undone. This standard requires conformance to historical practice.

The description of the special events that set the ‘ and ′ marks matches historical practice. For example, historically the command /a/,/b/ did not set the ‘ and ′ marks, but the command /a/,/b/delete did.

E.5.10.7.5.16 move

There is no additional rationale provided for this subclause.

E.5.10.7.5.17 next

Historically, any ex command could be entered as a *+command* argument to the next command, although some (e.g., insert and append) were known to confuse historical implementations. This standard requires that any command be permitted and that it behave as specified. The next command can accept more than one file, so usage such as

```
next ‘ls [abc]∗‘
```

is valid; it need not be valid for the edit or read commands, for example, because they expect only one file name.

Historically, the next command behaved differently from the :rewind command in that it ignored the force flag if the autowrite flag was set. For consistency, this standard does not permit this behavior.

Historically, the next command positioned the cursor as if the file had never been edited before, regardless. This standard does not permit this behavior, for consistency with the edit command.

Implementations wanting to provide a counterpart to the next command that edited the previous file have used the command prev[ious], which takes no *file* argument. This standard does not require this command.

E.5.10.7.5.18 number

There is no additional rationale provided for this subclause.

E.5.10.7.5.19 open

Historically, the open command would fail if the open edit option was not set. This standard does not mention the open edit option and does not require this behavior. Some historical implementations do not permit entering open mode from open or visual mode, only from ex mode. For consistency, this standard does not permit this behavior.

Historically, entering open mode from the command line (i.e., `vi +open`) resulted in anomalous behaviors; e.g., the `ex` file and set commands, and the `vi` command `<control-G>` did not work. For consistency, this standard does not permit this behavior.

Historically, the `open` command only permitted `/` characters to be used as the search pattern delimiter. For consistency, this standard requires that the search delimiters used by the `s`, `global`, and `v` commands be accepted as well.

E.5.10.7.5.20 preserve

The `preserve` command does not historically cause the file to be considered unmodified for the purposes of future commands that may exit the editor. This standard requires conformance to historical practice.

Historical documentation stated that mail was not sent to the user when preserve was executed; however, historical implementations did send mail in this case. This standard requires conformance to the historical implementations.

E.5.10.7.5.21 print

The writing of NUL by the `print` command is not specified as a special case because the standard developers did not want to require `ex` to support NUL characters. Historically, characters were displayed using the ARPA standard mappings, which are as follows:

(1) Printable characters are left alone.

(2) Control characters less than \177 are represented as ^ followed by the character offset from the @ character in the ASCII map; e.g., \007 is represented as `^G`.

(3) \177 is represented as ^ followed by ?.

The display of characters having their eighth bit set was less standard. Existing implementations use hex (0x00), octal (\000) and a meta-bit display. (The latter displayed bytes that had their eighth bit set as the two characters "`M–`," followed by the seven-bit display as described above.) The latter probably has the best claim to historical practice because it was used for the `–v` option of 4BSD- and 4BSD-derived versions of the `cat` utility since 1980.

No specific display format is required by this standard.

Explicit dependence on the ASCII character set has been avoided where possible, hence the use of the phrase an "implementation-defined multicharacter sequence" for the display of nonprintable characters in preference to the historical usage of, for instance, `^I` for `<tab>`. Implementations are encouraged to conform to historical practice in the absence of any strong reason to diverge.

Historically, all `ex` commands beginning with the letter `p` could be entered using capitalized versions of the commands; e.g., `P[rint]`, `Pre[serve]`, and `Pu[t]` were all valid command names. This standard permits, but does not require, this historical practice because capital forms of the commands are used by some implementations for other purposes.

E.5.10.7.5.22 put

Historically, an ex put command, executed from open or visual mode, was the same as the open or visual mode P command, if the buffer was named and was cut in character mode, and the same as the p command if the buffer was named and cut in line mode. If the unnamed buffer was the source of the text, the entire line from which the text was taken was usually put, and the buffer was handled as if in line mode, but it was possible to get extremely anomalous behavior. In addition, using the Q command to switch into ex mode, and then doing a put often resulted in errors as well, such as appending text that was unrelated to the (supposed) contents of the buffer. For consistency and simplicity of specification, this standard does not permit these behaviors. All ex put commands are required to operate in line mode, and the contents of the buffers are not altered by changing the mode of the editor.

E.5.10.7.5.23 quit

There is no additional rationale provided for this subclause.

E.5.10.7.5.24 read

Historically, an ex read command executed from open or visual mode, executed in an empty file, left an empty line as the first line of the file. For consistency and simplicity of specification, this standard does not permit this behavior. Historically, a read in open or visual mode from a program left the cursor at the last line read in, not the first. For consistency, this standard does not permit this behavior.

Historical implementations of ex were unable to undo read commands that read from the output of a program. For consistency, this standard does not permit this behavior.

Historically, the ex and vi message after a successful read or write command specified "characters," not "bytes." This standard requires that the number of bytes be displayed, not the number of characters, because it may be difficult in multibyte implementations to determine the number of characters read. Implementations are encouraged to clarify the message displayed to the user.

Historically, reads were not permitted on files other than type regular, except that FIFO files could be read (probably only because they did not exist when ex and vi were originally written). Because the historical ex evaluated read! and read ! equivalently, there can be no optional way to force the read. This standard permits, but does not require, this behavior.

E.5.10.7.5.25 `recover`

Some historical implementations of the editor permitted users to recover the edit buffer contents from a previous edit session, and then exit without saving those contents (or explicitly discarding them). The intent of this standard in requiring that the edit buffer be treated as already modified is to prevent this user error.

E.5.10.7.5.26 `rewind`

Historical implementations supported the `rewind` command when the user was editing the first file in the list; i.e., the file that the `rewind` command would edit. This standard requires conformance to historical practice.

E.5.10.7.5.27 `s`

Historically, `ex` accepted an `r` option to the `s` command. The effect of the `r` option was to use the last RE used in any command as the pattern, the same as the `~` command. The `r` option is not required by this standard. Historically, the `c` and `g` options were toggled; e.g., the command `:s/abc/def/` was the same as `s/abc/def/ccccgggg`. For simplicity of specification, this standard does not permit this behavior. C C

Historically, the `edcompatible` edit option made the values of the `c` and `g` suffixes remembered instead of reinitializing them to "off" for each `s` command. The single special case was that they were always reinitialized to zero if the pattern and replacement strings were specified. This standard does not specify this behavior or the `edcompatible` edit option.

The tilde command is often used to replace the last search RE. For example, in the sequence

```
s/red/blue/
/green
~
```

the `~` command is equivalent to:

```
s/green/blue/
```

Historically, `ex` accepted all of the following forms:

```
s/abc/def/
s/abc/def
s/abc/
s/abc
```

This standard requires conformance to this historical practice.

The `s` command presumes that the `^` character only occupies a single column in the display. Much of the `ex` and `vi` specification presumes that the `<space>` character only occupies a single column in the display. There are no known character sets for which this is not true.

Historically, the final column position for the substitute commands was based on previous column movements; a search for a pattern followed by a substitution would leave the column position unchanged, while a `0` command followed by a substitution would change the column position to the first nonblank. For consistency and simplicity of specification, this standard requires that the final column position always be set to the first nonblank.

E.5.10.7.5.28 set

Historical implementations redisplayed all of the options for each occurrence of the `all` keyword. This standard permits, but does not require, this behavior.

E.5.10.7.5.29 shell

There is no additional rationale provided for this subclause.

E.5.10.7.5.30 source

Source commands can be nested to arbitrary depths, and should be limited only by system resources.

E.5.10.7.5.31 suspend

There is no additional rationale provided for this subclause.

E.5.10.7.5.32 tag

No requirement is made as to where `ex` and `vi` shall look for the file referenced by the tag entry. Historical practice has been to look for the path found in the tags file, based on the current directory. A useful extension found in some implementations is to look based on the directory containing the tags file that held the entry, as well. No requirement is made as to which reference for the tag in the tags file is used. This is deliberate, in order to permit extensions such as multiple entries in a tags file for a tag.

Because users often specify many different tags files, some of which need not be relevant or exist at any particular time, this standard requires that error messages about problem tags files be displayed only if the requested tag is not found, and then, only once for each time that the `tag` edit option is changed.

The requirement that the current edit buffer be unmodified is only necessary if the file indicated by the tag entry is not the same as the current file (as defined by the current pathname). Historically, the file would be reloaded if the file name had changed, as well as if the file name was different from the current pathname. For consistency and simplicity of specification, this standard does not permit this behavior, requiring that the name be the only factor in the decision.

Historically, `vi` only searched for tags in the current file from the current cursor to the end of the file, and therefore, if the `wrapscan` option was not set, tags occurring before the current cursor were not found. This standard considers this a bug, and implementations are required to search for the first occurrence in the file, regardless.

E.5.10.7.5.33 unabbreviate

There is no additional rationale provided for this subclause.

E.5.10.7.5.34 undo

The undo description deliberately uses the word "modified." The undo command is not intended to undo commands that replace the contents of the edit buffer, such as edit, next, tag, or recover.

Cursor positioning after the undo command was inconsistent in the historical vi, sometimes attempting to restore the original cursor position (global, undo, and v commands), and sometimes, in the presence of maps, placing the cursor on the last line added or changed instead of the first. This standard requires a simplified behavior for consistency and simplicity of specification.

E.5.10.7.5.35 unmap

There is no additional rationale provided for this subclause.

E.5.10.7.5.36 version

The version command cannot be exactly specified since there is no widely accepted definition of what the version information should contain. Implementations are encouraged to do something reasonably intelligent.

E.5.10.7.5.37 visual

There is no additional rationale provided for this subclause.

E.5.10.7.5.38 write

Historically, the ex and vi message after a successful read or write command specified "characters", not "bytes." This standard requires that the number of bytes be displayed, not the number of characters because it may be difficult in multibyte implementations to determine the number of characters written. Implementations are encouraged to clarify the message displayed to the user.

Implementation-defined tests are permitted so that implementations can make additional checks; e.g., for locks or file modification times.

Historically, attempting to append to a nonexistent file caused an error. It has been left unspecified in this standard to permit implementations to let the write C succeed, so that the append semantics are similar to those of the historical csh.

Historical vi permitted empty edit buffers to be written. However, since the way vi got around dealing with "empty" files was to always have a line in the edit buffer, no matter what, it wrote them as files of a single, empty line. This standard does not permit this behavior.

Historically, ex restored standard output and standard error to their values as of when ex was invoked, before writes to programs were performed. This could disturb the terminal configuration as well as be a security issue for some terminals.

This standard does not permit this, requiring that the program output be captured and displayed as if by the `ex` `print` command.

E.5.10.7.5.39 xit

There is no additional rationale provided for this subclause.

E.5.10.7.5.40 yank

There is no additional rationale provided for this subclause.

E.5.10.7.5.41 z

Historically, the line count was set to the value of the `scroll` option if the type character was end-of-file. This feature was broken on most historical implementations long ago, however, and is not documented anywhere. For this reason, this standard is resolutely silent.

Historically, the `z` command was <`blank`>-sensitive and "`z +`" and "`z -`" did different things than "`z+`" and "`z-`" because the type could not be distinguished from a flag. (The commands "`z .`" and "`z =`" were historically invalid.) This standard requires conformance to this historical practice.

Historically, the `z` command was further <`blank`>-sensitive in that the *count* could not be <`blank`>-delimited; e.g., the commands "`z= 5`" and "`z- 5`" were also invalid. Because the *count* is not ambiguous with respect to either the type character or the flags, this is not permitted by this standard.

E.5.10.7.5.42 !

Historically, `ex` filter commands only read the standard output of the commands, letting standard error appear on the terminal as usual. The `vi` utility, however, read both standard output and standard error. This standard requires the latter behavior for both `ex` and `vi`, for consistency.

E.5.10.7.5.43 <

Historically, it was possible to add shift characters to increase the effect of the command; e.g., <<< outdented (or >>> indented) the line(s) 3 levels of indentation instead of the default 1. This standard requires conformance to historical practice.

E.5.10.7.5.44 >

See E.5.10.7.5.43.

E.5.10.7.5.45 <control-D>

Historically, the <`control-D`> command erased the prompt, providing the user with an unbroken presentation of lines from the edit buffer. This is not required by this standard; implementations are encouraged to provide it if possible. Historically, the <`control-D`> command took, and then ignored, a count. This standard does not permit this behavior.

E.5.10.7.5.46 =

Historically, the `ex` = command, when executed in `ex` mode in an empty edit buffer, reported 0, and from open or visual mode, reported 1. For consistency and simplicity of specification, this standard does not permit this behavior. C C C

E.5.10.7.5.47 @

Historically, `ex` did not correctly handle the inclusion of text input commands (i.e., `append`, `insert`, and `change`) in executed buffers. This standard does not permit this exclusion for consistency.

Historically, the logical contents of the buffer being executed did not change if the buffer itself were modified by the commands being executed; i.e., buffer execution did not support self-modifying code. This standard requires conformance to historical practice.

Historically, the @ command took a range of lines, and the @ buffer was executed once per line, with the current line (.) set to each specified line. This standard requires conformance to historical practice.

Some historical implementations did not notice if errors occurred during buffer execution. This, coupled with the ability to specify a range of lines for the `ex` @ command, makes it trivial to cause them to drop core. This standard requires that implementations stop buffer execution if any error occurs, if the specified line doesn't exist, or if the contents of the edit buffer itself are replaced (e.g., the buffer executes the `ex :edit` command).

E.5.10.7.6 REs

Historical practice is that the characters in the replacement part of the last `s` command; i.e., those matched by entering a ~ in the RE were not further expanded by the RE engine. So, if the characters contained the string `a.`, they would match `a` followed by `.`, and not `a` followed by any character. This standard requires conformance to historical practice.

E.5.10.7.7 Replacement Strings

An example of case conversion with the `s` command:

```
:p
The cat sat on the mat.
:s/\<.at\>/\u&/gp
The Cat Sat on the Mat.
:s/S\(.*\)M/S\U\1\eM/p
The Cat SAT ON THE Mat.
```

E.5.10.7.8 Edit Options

The following paragraphs describe the historical behavior of some edit options that were not, for whatever reason, included in the POSIX.2 standard. Implementations are strongly encouraged to only use these names if the functionality described here is fully supported.

`beautify`
: The historical `beautify` edit option behaved as follows: In `ex` mode, keys that were not already specially handled, that were less than an ASCII space or were the `<DEL>` (\177) key, and were neither a `<tab>` nor a `<form-feed>`, and were read in from an `ex` script file, were discarded. When the first `<control-H>` was discarded a message was written to the terminal. Quoting (with a \) would keep the keys from being discarded.

  In open or visual mode, keys that were not already specially handled, that were less than an ASCII space or were the `<DEL>` (\177) key, and were neither a `<tab>` nor a `<form-feed>`, and were entered in input mode (either to the edit buffer or to the colon command line), were discarded. Quoting (using a `<control-V>`) would keep the keys from being discarded.

  For various reasons, among them internationalization concerns, this standard does not require the `beautify` option.

`directory`
: The `directory` edit option historically specified the pathname of the directory where temporary files (although not the backup file used for recovery) were created by `ex` or `vi`. This option was omitted from this standard because the default value was always implementation specific.

`edcompatible`
: The `edcompatible` edit option historically caused the `c` and `g` suffixes to the `s` command to be remembered, instead of initializing them to unset for each new `s` command. (Note that specifying both the pattern and replacement strings to the `s` command reset the `c` and `g` suffixes as well.) This option was omitted from this standard because it was not believed to be widely used, or generally useful.

`extended`
: The `extended` edit option has been used in some implementations of `vi` to provide EREs instead of BREs. This option was omitted from this standard because it is not widespread historical practice.

`flash`
: The `flash` edit option historically caused the screen to flash instead of beeping on error. This option was omitted from this standard because it is not found in some historical implementations.

`hardtabs`
: The `hardtabs` edit option historically defined the number of columns between hardware tab settings. This option was omitted from this standard because it was believed to no longer be generally useful.

`lisp`

The `lisp` edit option historically altered the behavior of the `autoindent` edit option and the (,), {, }, [[, and]] commands to match the LISP language. In addition, there was a command = (reindent) that was available only in `lisp` mode. This option was omitted from this standard because it was difficult to justify the inclusion of programming-language dependent features.

`modeline`

The `modeline` (sometimes named `modelines`) edit option(s) historically caused `ex` or `vi` to read the five first and last lines of the file for editor commands. This option is a security problem, and vendors are strongly encouraged to delete it from historical implementations.

`open`

The `open` edit option historically disallowed the `ex` `open` and `visual` commands. This edit option was omitted from this standard because these commands are required by this standard. C C C

`optimize`

The `optimize` edit option historically expedited text throughput by setting the terminal to not do automatic carriage returns when printing more than one logical line of output. This option was omitted from this standard because it was intended for terminals without addressable cursors, which are rarely, if ever, still used.

`redraw`

The `redraw` edit option historically simulated an intelligent terminal on a dumb terminal. This option was omitted from this standard because it was intended for terminals which are rarely, if ever, still used.

`ruler`

The `ruler` edit option has been used in some implementations of `vi` to present a current row/column ruler for the user. This option was omitted from this standard because it is not widespread historical practice.

`sourceany`

The `sourceany` edit option historically caused `ex` or `vi` to source startup files that were owned by users other than the user running the editor. This option is a security problem, and vendors are strongly encouraged to remove it from their implementations.

`timeout`

The `timeout` edit option historically enabled the (now standard) feature of only waiting for a short period before returning keys that could be part of a macro. This feature was omitted from this standard because its behavior is now standard, it is not widely useful, and it was rarely documented.

`verbose`

The `verbose` edit option has been used in some implementations of `vi` to cause `vi` to output error messages for common errors; e.g., attempting to move the cursor past the beginning or end of the line instead of only alerting the screen. (The historical `vi` only alerted the terminal and presented no message for such errors. The historical editor option `terse` did not select when to present error messages, it only made existing error messages more or less verbose.) This option was omitted from this standard because it is not widespread historical practice; however, implementors are encouraged to use it if they wish to provide error messages for naive users.

`wraplen`
The `wraplen` edit option has been used in some implementations of `vi` to specify an automatic margin measured from the left margin instead of from the right margin. This is useful when multiple screen sizes are being used to edit a single file. This option was omitted from this standard because it is not widespread historical practice; however, implementors are encouraged to use it if they add this functionality.

E.5.10.7.8.1 autoindent

Historically, the command `0a` did not do any autoindentation, regardless of the current indentation of line 1. This standard requires that any indentation present in line 1 be used.

E.5.10.7.8.2 autoprint

Historically, the `autoprint` edit option was not completely consistent or based solely on modifications to the edit buffer. Exceptions were the `read` command (when reading from a file, but not from a filter), the `append`, `change`, `insert`, `global`, and `v` commands, all of which were not affected by `autoprint`, and the `tag` command, which was affected by `autoprint`. This standard requires conformance to historical practice.

Historically, the `autoprint` option only applied to the last of multiple commands entered using vertical-bar delimiters; e.g. `delete<newline>` was affected by `autoprint`, but `delete|version<newline>` was not. This standard requires conformance to historical practice.

E.5.10.7.8.3 autowrite

Appending the ! character to the `ex next` command to avoid performing an automatic write was not supported in historical implementations. This standard requires that the behavior match the other `ex` commands for consistency.

E.5.10.7.8.4 errorbells

There is no additional rationale provided for this subclause.

E.5.10.7.8.5 exrc

There is no additional rationale provided for this subclause.

E.5.10.7.8.6 ignorecase

Historical implementations of case-insensitive matching (the `ignorecase` edit option) lead to counterintuitive situations when uppercase characters were used in range expressions. Historically, the process was as follows:

(1) Take a line of text from the edit buffer C

(2) Convert uppercase to lowercase in text line

(3) Convert uppercase to lowercase in REs, except in character class specifications

(4) Match REs against text

This would mean that, with `ignorecase` in effect, the text

```
The cat sat on the mat
```

would be matched by

```
/ˆthe/
```

but not by

```
/ˆ[A-Z]he/
```

For consistency with other commands implementing REs, this standard does not permit this behavior.

E.5.10.7.8.7 list

There is no additional rationale provided for this subclause.

E.5.10.7.8.8 magic

There is no additional rationale provided for this subclause.

E.5.10.7.8.9 mesg

There is no additional rationale provided for this subclause.

E.5.10.7.8.10 number

There is no additional rationale provided for this subclause.

E.5.10.7.8.11 paragraphs

Earlier versions of this standard made the default `paragraphs` and `sections` edit options implementation-defined, arguing they were historically oriented to the UNIX system `troff` text formatter, and a "portable user" could use the {, }, [[,]], (, and) commands in open or visual mode and have the cursor stop in unexpected places. This version of the standard specifies their values in the POSIX Locale because the unusual grouping (they only work when grouped into two characters at a time) means that they cannot be used for general purpose movement, regardless.

E.5.10.7.8.12 prompt

There is no additional rationale provided for this subclause.

E.5.10.7.8.13 readonly

Implementations are encouraged to provide the best possible information to the user as to the readonly status of the file, with the exception that they should not consider the current special privileges of the process. This provides users a safety net because they must force the overwrite of readonly files, even when running with additional privileges.

The `readonly` edit option specification largely conforms to historical practice. The only difference is that historical implementations did not notice that the user had set the `readonly` edit option in cases where the file was already marked readonly for some reason, and would therefore reinitialize the `readonly` edit option the next time the contents of the edit buffer were replaced. This behavior is disallowed by this standard.

E.5.10.7.8.14 remap

There is no additional rationale provided for this subclause.

E.5.10.7.8.15 report

The requirement that lines copied to a buffer interact differently than deleted lines is historical practice. For example, if the `report` edit option is set to 3, deleting 3 lines will cause a report to be written, but 4 lines must be copied before a report is written.

The requirement that the `ex` `global`, `v`, `open`, `undo`, and `visual` commands present reports based on the total number of lines added or deleted during the command execution, and, that commands executed by the `global` and `v` commands not present reports, is historical practice. This standard extends historical practice by requiring that buffer execution be treated similarly. The reasons for this are two-fold. Historically, only the report by the last command executed from the buffer would be seen by the user, as each new report would overwrite the last. In addition, the standard developers believed that buffer execution had more in common with `global` and `v` commands than it did with other `ex` commands, and should behave similarly, for consistency and simplicity of specification.

E.5.10.7.8.16 scroll

There is no additional rationale provided for this subclause.

E.5.10.7.8.17 sections

See E.5.10.7.8.11.

E.5.10.7.8.18 shell

There is no additional rationale provided for this subclause.

E.5.10.7.8.19 shiftwidth

There is no additional rationale provided for this subclause.

E.5.10.7.8.20 showmatch

The length of time the cursor spends on the matching character is unspecified because the timing capabilities of systems are often inexact and variable. The time should be long enough for the user to notice, but not long enough for the user to become annoyed. Some implementations of vi have added a matchtime option that permits users to set the number of 0,1 s intervals the cursor pauses on the matching character.

E.5.10.7.8.21 showmode

The showmode option has been used in some historical implementations of ex and vi to display the current editing mode when in open or visual mode. The editing modes have generally included "command" and "input," and sometimes other modes such as "replace" and "change." The string was usually displayed on the bottom line of the screen at the far right hand corner. In addition, a preceding * character often denoted if the contents of the edit buffer had been modified. The latter display has sometimes been part of the showmode option, and sometimes based on another option. This option was not available in the 4BSD historical implementation of vi, but was viewed as generally useful, particularly to novice users, and is required by this standard.

The smd shorthand for the showmode option was not present in all historical implementations of the editor. This standard requires it, for consistency.

Not all historical implementations of the editor displayed a mode string for command mode, differentiating command mode from text input mode(s) by the absence of a mode string. This standard permits this behavior for consistency with historical practice, but implementations are encouraged to provide a display string for both modes.

E.5.10.7.8.22 slowopen

Historically the slowopen option was automatically set if the terminal baud rate was less than 1200 baud, or if the baud rate was 1200 baud and the redraw option was not set. The slowopen option had two effects. First, when inserting characters in the middle of a line, characters after the cursor would not be pushed ahead, but would appear to be overwritten. Second, when creating a new line of text, lines after the current line would not be scrolled down, but would appear to be overwritten. In both cases, ending text input mode would cause the screen to be refreshed to match the actual contents of the edit buffer. Finally, terminals that were sufficiently intelligent caused the editor to ignore the slowopen option. This standard permits most historical behavior, extending historical practice to require `slowopen` behaviors if the edit option is set by the user.

E.5.10.7.8.23 tabstop

Tabstops are not related to the configured tabstops of the terminal hardware.

E.5.10.7.8.24 taglength

There is no additional rationale provided for this subclause.

E.5.10.7.8.25 tags

The default path for tags files is left unspecified as implementations may have their own tags implementations that do not correspond to the historical ones. The default tags option value should probably at least include the file `./tags`.

E.5.10.7.8.26 term

Historical implementations of `ex` and `vi` ignored changes to the `term` edit option after the initial terminal information was loaded. This is permitted by this standard; however, implementations are encouraged to permit the user to modify their terminal type at any time.

E.5.10.7.8.27 terse

Historically, the `terse` edit option optionally provided a shorter, less descriptive error message, for some error messages. This is permitted, but not required, by this standard. Historically, most common visual mode errors (e.g., trying to move the cursor past the end of a line) did not result in an error message, but simply alerted the terminal. Implementations wishing to provide messages for novice users are urged to do so based on the edit option `verbose`, and not `terse`.

E.5.10.7.8.28 warn

There is no additional rationale provided for this subclause.

E.5.10.7.8.29 window

In historical implementations, the default for the `window` edit option was based on the baud rate as follows:

(1) If the baud rate was less than 1200, the edit option `w300` set the `window` value; e.g., the line:

```
set w300=12
```

would set the `window` option to 12 if the baud rate was less than 1200.

(2) If the baud rate was equal to 1200, the edit option `w1200` set the `window` value.

(3) If the baud rate was greater than 1200, the edit option `w9600` set the `window` value.

The `w300`, `w1200`, and `w9600` options do not appear in this standard because of their dependence on specific baud rates.

In historical implementations, the size of the window displayed by various commands was related to, but not necessarily the same as, the `window` edit option. For example, the size of the window was set by the `ex` command `visual 10`, but it did not change the value of the `window` edit option. However, changing the value of the `window` edit option did change the number of lines that were displayed when the screen was repainted. This standard does not permit this behavior in the interests of consistency and simplicity of specification, and requires that all commands that change the number of lines that are displayed do it by setting the value of the `window` edit option.

E.5.10.7.8.30 wrapmargin

Historically, the `wrapmargin` option did not affect maps inserting characters that also had associated counts; e.g., "`:map K 5aABC DEF`." Unfortunately, there are widely used maps that depend on this behavior. For consistency and simplicity of specification, this standard does not permit this behavior.

C

Historically, `wrapmargin` was calculated using the column display width of all characters on the screen. For example, an implementation using `^I` to represent `<tab>`s when the `list` edit option was set, where `^` and `I` each took up a single column on the screen, would calculate the `wrapmargin` based on a value of 2 for each `<tab>` character. The `number` edit option similarly changed the effective length of the line as well. This standard requires conformance to historical practice.

E.5.10.7.8.31 wrapscan

There is no additional rationale provided for this subclause.

E.5.10.7.8.32 writeany

There is no additional rationale provided for this subclause.

E.5.10.8 Exit Status

There is no additional rationale provided for this subclause.

E.5.10.9 Consequences of Errors

There is no additional rationale provided for this subclause.

⇒ **E.5.7 `ctags` Rationale.** *Change the seventh paragraph (the one beginning* "Historically, . . . "*) to:*

Historically, the tags file has been used only by `ex` and `vi`. However, the format of the tags file has been published to encourage other programs to use the tags in new ways. The format allows either search patterns or line numbers to find the identifiers because the historical `vi` recognizes either. The `ctags` utility does not produce the format using line numbers because it is not useful following any source file changes that add or delete lines. The documented search patterns match historical practice. It should be noted that literal leading circumflex or trailing dollar-sign characters in the search pattern will only behave correctly if anchored to the beginning of the line or end of the line by an additional circumflex or dollar-sign character.

⇒ **E.5.18 `more` Rationale.** *Replace the full rationale for* `more` *with the following.*

Editor's Note: Only the portions changed from the 1992 standard are diff-marked.

E.5.18 `more` — Display files on a page-by-page basis

The `more` utility, available in BSD and BSD-derived systems, was chosen as the prototype for the POSIX.2 file display program since it is more widely available than either the public-domain program `less` or than `pg`, a pager provided in System V. The 4.4BSD `more` is the model for the features selected; it is almost fully upward compatible from the 4.3BSD version in wide use and has become more amenable for `vi` users. Several features originally derived from various file editors, found in both `less` and `pg`, have been added to this specification as they have proved extremely popular with users.

There are inconsistencies between `more` and `vi` that result from historical practice. For example, the single-character commands `h`, `f`, `b`, and `<space>` are screen movers in `more`, but cursor movers in `vi`. These inconsistencies were maintained because the cursor movements are not applicable to `more` and the powerful functionality achieved without the use of the control key justifies the differences.

The tags interface has been included in a program that is not a text editor because it promotes another degree of consistent operation with `vi`. It is conceivable that the paging environment of `more` would be superior for browsing source code files in some circumstances.

The operating mode referred to for block-mode terminals effectively adds a `<newline>` to each synopsis line that currently has none. So, for example, `d<newline>` would page one screenful. The mode could be triggered by a command-line option, environment variable, or some other method. The details are not imposed by POSIX.2 because there are so few systems known to support such terminals. Nevertheless, it was considered that all systems should be able to support `more` given the exception cited for this small community of terminals because, in comparison to `vi`, the cursor movements are few and the command set relatively amenable to the optional `<newline>`s.

Some versions of `more` provide a shell escaping mechanism similar to the `ex !` command. The standard developers did not consider that this was necessary in a paginator, particularly given the wide acceptance of multiple window terminals and job control features. (They chose to retain such features in the editors and `mailx` because the shell interaction also gives an opportunity to modify the editing buffer, which is not applicable to `more`).

The `−p` (position) option replaces the + command because of the Utility Syntax Guidelines. In early drafts, it took a *pattern* argument, but historical `less` provided the more general facility of a command. It would have been desirable to use the same `−c` as `ex` and `vi`, but the letter was already in use.

When the standard input is not a terminal, only the `−s` filter-modification option is effective. This is historical practice.

The text stating "from a nonrewindable stream ... implementations may limit the amount of backwards motion supported" would allow an implementation that permitted no backwards motion beyond text already on the screen. It was not possible to require a minimum amount of backwards motion that would be effective for all conceivable device types. The implementation should allow the user to back up as far as possible, within device and reasonable memory allocation constraints.

Examples

The `−p` option allows arbitrary commands to be executed at the start of each file. Examples are

`more -p G file1 file2`	Examine each file starting with its last screenful.
`more -p 100 file1 file2`	Examine each file starting with line 100 as the first line of the screen.
`more -p /100 file1 file2`	Examine each file starting with the first line containing the string `100`.

Historically, nonprintable characters were displayed using the ARPA standard mappings, which are as follows:

(1) Printable characters are left alone.

(2) Control characters less than \177 are represented as ˆ followed by the character offset from the @ character in the ASCII map; e.g., \007 is represented as `ˆG`.

(3) \177 is represented as ˆ followed by `?`.

The display of characters having their eighth bit set was less standard. Existing implementations use hex (0x00), octal (\000) and a meta-bit display. (The latter displayed characters with their eighth bit set as the two characters "`M-`", followed by the seven bit display as described previously.) The latter probably has the best claim to historical practice because it was used with the −v option of 4BSD and 4BSD derived versions of the cat utility since 1980.

No specific display format is required by this standard. Implementations are encouraged to conform to historic practice in the absence of any strong reason to diverge.

⇒ **E.5.35 vi Rationale.** *Replace the full rationale for* vi *with the following.*

E.5.35 vi – Screen-oriented (visual) display editor

Major portions of the vi clause point to the ex clause to avoid inadvertent divergence. While ex and vi have historically been implemented as a single utility, this is not required by this standard. See the rationale for the ex utility (E.5.10) for more information on vi.

E.5.35.1 Synopsis

There is no additional rationale provided for this subclause.

E.5.35.2 Description

It is recognized that portions of vi would be difficult, if not impossible, to implement satisfactorily on a block-mode terminal, or a terminal without any form of cursor addressing, thus it is not a mandatory requirement that such features should work on all terminals. It is the intention, however, that a vi implementation should provide the full set of capabilities on all terminals capable of supporting them.

E.5.35.3 Options

There is no additional rationale provided for this subclause.

E.5.35.4 Operands

There is no additional rationale provided for this subclause.

E.5.35.5 External Influences

E.5.35.5.1 Standard Input

Historically, vi exited immediately if the standard input was not a terminal. This standard permits, but does not require, this behavior.

An end-of-file condition is not equivalent to an end-of-file character. A common end-of-file character, <control-D>, is historically a vi command.

E.5.35.5.2 Input Files

There is no additional rationale provided for this subclause.

E.5.35.5.3 Environment Variables

There is no additional rationale provided for this subclause.

E.5.35.5.4 Asynchronous Events

There is no additional rationale provided for this subclause.

E.5.35.6 External Effects

E.5.35.6.1 Standard Output

The text in the standard output subclause reflects the usage of the verb "display" in this clause; some implementations of `vi` use standard output to write to the terminal, but POSIX.2 does not require that to be the case.

E.5.35.6.2 Standard Error

There is no additional rationale provided for this subclause.

E.5.35.6.3 Output Files

There is no additional rationale provided for this subclause.

E.5.35.7 Extended Description

Historically, implementations reverted to open mode if the terminal was incapable of supporting full visual mode. This standard requires this behavior. Historically, the open mode of `vi` behaved roughly equivalently to the visual mode, with the exception that only a single physical line from the edit buffer was kept current C at any time. This line was normally displayed on the next to last line of a terminal with cursor addressing (and the last line performed its normal visual functions for line-oriented commands and messages). In addition, some few commands behaved differently in open mode than in visual mode. This standard requires conformance to historical practice. The following list is a condensed version of the information contained in the normative text. It is entered here so that the basic information about open mode is available in a single place.

[*count***]**`z`, **[***count***]**`control-F`, **[***count***]**`<control-B>`
: The `z` command has a different synopsis in open mode than in visual mode. The `z`, `<control-F>`, and `<control-B>` commands all behave identically, displaying zero or more lines before and after the current line, with the current line surrounded by hyphens.

`<control-D>`
: Write the next `scroll` edit option value lines, update the current line.

`<control-U>`
Update the current line, do nothing else.

`<control-E>`, `<control-Y>`
Do nothing.

`<control-L>`
Clear the screen and redisplay the current line.

`H`, `L`, `M`
Move to the first nonblank of the current line and do nothing else.

Historically, ex and vi implementations have expected text to proceed from left to right and from top to bottom. There is no requirement in this standard that this be the case. The specification was deliberately written using words like "before," "after," "first," and "last" in order to permit implementations to support the natural text order of the language.

Historically, lines past the end of the edit buffer were marked with single tilde (**~**) characters; i.e., if the one-based display was 20 lines in length, and the last line of the file was on line one, then lines 2–20 would contain only a single **~** character.

Historically, the vi editor attempted to display only complete lines at the bottom of the screen (it did display partial lines at the top of the screen). If a line was too long to fit in its entirety at the bottom of the screen, the screen lines where the line would have been displayed were displayed as single @ characters, instead of displaying part of the line. This standard permits, but does not require, this behavior. Implementations are encouraged to attempt always to display a complete line at the bottom of the screen when doing scrolling or screen positioning by physical lines.

Historically, lines marked with @ were also used to minimize output to dumb terminals over slow lines; i.e., changes local to the cursor were updated, but changes to lines on the screen that were not close to the cursor were simply marked with an @ sign instead of being updated to match the current text. This standard permits, but does not require this feature because it is used ever less frequently as terminals become smarter and connections are faster.

E.5.35.7.1 ex and vi Initialization

Historically, vi always had a line in the edit buffer, even if the edit buffer was "empty." For example:

(1) The ex command = executed from visual mode wrote "1" when the buffer was empty.

(2) Writes from visual mode of an empty edit buffer wrote files of a single character (a `<newline>`), while writes from ex mode of an empty edit buffer wrote empty files.

(3) Put and read commands into an empty edit buffer left an empty line at the top of the edit buffer.

For consistency, this standard does not permit any of these behaviors.

Historically, vi did not always return the terminal to its original modes; for example, ICRNL was modified if it was not originally set. This standard does not permit this behavior.

E.5.35.7.2 vi Command Descriptions

Motion commands are among the most complicated aspects of vi to describe. With some exceptions, the text region and buffer type effect of a motion command on a vi command are described on a case-by-case basis. The descriptions of text regions in this standard are not intended to imply direction; i.e., an inclusive region from line *n* to line *n* + 5 is identical to a region from line *n* + 5 to line *n*. This is of more than academic interest—movements to marks can be in either direction, and, if the wrapscan option is set, so can movements to search points. Historically, lines are always stored into buffers in text order; i.e., from the start of the edit buffer to the end. This standard requires conformance to historical practice.

Historically, command counts were applied to any associated motion, and were multiplicative to any supplied motion count. For example, 2cw is the same as c2w, and 2c3w is the same as c6w. This standard requires this behavior.

Historically, vi commands that used bigwords, words, paragraphs, and sentences as objects treated groups of empty lines, or lines that contained only <blank> characters, inconsistently. Some commands treated them as a single entity, while others treated each line separately. For example, the w, W, and B commands treated groups of empty lines as individual words; i.e., the command would move the cursor to each new empty line. The e and E commands treated groups of empty lines as a single word; i.e., the first use would move past the group of lines. The b command would just beep at the user, or if done from the start of the line as a motion command, fail in unexpected ways. If the lines contained only (or ended with) <blank> characters, the w and W commands would just beep at the user, the E and e commands would treat the group as a single word, and the B and b commands would treat the lines as individual words. For consistency and simplicity of specification, this standard requires that all vi commands treat groups of empty or <blank>-filled lines as a single entity, and that movement through lines ending with <blank> characters be consistent with other movements.

Historically, vi documentation indicated that any number of double quotes were skipped after punctuation marks at sentence boundaries, however, implementations only skipped single quotes. This standard requires both to be skipped.

Historically, the first and last characters in the edit buffer were word boundaries. This historical practice is required by this standard.

Historically, vi attempted to update the minimum number of columns on the screen possible, which could lead to misleading information being displayed. This standard makes no requirements other than that the current character being entered is displayed correctly, leaving all other decisions in this area up to the implementations.

Historically, lines were arbitrarily folded between columns of any characters that required multiple column positions on the screen, with the exception of tabs, which terminated at the right-hand margin. This standard permits the former and requires the latter. Implementations that do not arbitrarily break lines between columns of characters that occupy multiple column positions should not permit the cursor to rest on a column that does not contain any part of a character.

The historical `vi` had a problem in that all movements were by physical lines, not by logical, or screen, lines. This is often the right thing to do; e.g., single line movements, such as `j` or `k`, should work on physical lines. Commands like `dj`, or `j.`, where `.` is a `change` command, only make sense for physical lines. It is not, however, the right thing to do for screen motion or scrolling commands like `<control-D>`, `<control-F>`, and `H`. If the window is fairly small, using physical lines in these cases can result in completely random motion; e.g., `1control-D` can result in a completely changed screen, without any overlap. This is clearly not what the user wanted. The problem is even worse in the case of the `H`, `L`, and `M` commands—as they position the cursor at the first nonblank of the line, they may all refer to the same location in large lines, and will result in no movement at all.

In addition, if the line is larger than the screen, using physical lines can make it impossible to display parts of the line—there are not any commands that do not display the beginning of the line in historical `vi`, and if both the beginning and end of the line cannot be on the screen at the same time, the user suffers. Finally, the page and half-page scrolling commands historically moved to the first non-`<blank>` character in the new line. If the line is approximately the same size as the screen, this is inadequate because the cursor before and after a `<control-D>` command will refer to the same location on the screen.

Implementations of `ex` and `vi` exist that do not have these problems because the relevant commands (`<control-B>`, `<control-D>`, `<control-F>`, `<control-U>`, `<control-Y>`, `<control-E>`, `H`, `L`, and `M`) commands operate on logical screen lines, not physical edit buffer lines.

This standard does not permit this behavior by default because the standard developers believed that users would find it too confusing. However, historical practice has been relaxed. For example, `ex` and `vi` historically attempted, albeit sometimes unsuccessfully, to never put part of a line on the last lines of a screen; e.g., if a line would not fit in its entirety, no part of the line was displayed, and the screen lines corresponding to the line contained single @ characters. This C
behavior is permitted, but not required by this standard, so that it is possible for implementations to support long lines in small screens more reasonably without changing the commands to be logically (instead of physically) oriented. This stan- C
dard also permits implementations to refuse to edit any edit buffer containing a line that will not fit on the screen in its entirety.

The display area (e.g., the value of the `window` edit option) has historically been "grown," or expanded, to display new text when local movements are done in displays where the number of lines displayed is less than the maximum possible. Expansion has historically been the first choice, when the target line is less than the maximum possible expansion value away. Scrolling has historically been the next choice, done when the target line is less than half a display away, and otherwise, the screen was redrawn. There were exceptions, however, in that `ex` commands generally always caused the screen to be redrawn. This standard does not specify a standard behavior because there may be external issues such as connection speed, the number of characters necessary to redraw as opposed to scroll, or terminal capabilities that implementations will have to accommodate.

The current line in this standard maps one-to-one to a physical line in the file. The current column does not. There are two different column values that are described by this standard. The first is the current column value as set by many of the `vi` commands. This value is remembered for the lifetime of the editor. The second column value is the actual position on the screen where the cursor rests. The two are not always the same. For example, when the cursor is backed by a multicolumn character, the actual cursor position on the screen has historically been the last column of the character in command mode, and the first column of the character in input mode.

Commands that set the current line, but that do not set the current cursor value, (e.g., `j` and `k`) attempt to get as close as possible to the remembered column position, so that the cursor tends to restrict itself to a vertical column as the user moves around in the edit buffer. This standard requires conformance to historical practice, requiring that the physical location of the cursor on the screen be adjusted from the current column value as necessary to support this historical behavior.

Historically, only a single line (and for some terminals, a single line minus 1 column) of characters could be entered by the user for the line oriented commands; i.e., `:`, `!`, `/`, or `?`. This standard permits, but does not require, this limitation.

Historically, "soft" errors in `vi` caused the terminal to be alerted, but no error message was displayed. As a general rule, no error message was displayed for errors in command execution in `vi`, when the error resulted from the user attempting an invalid or impossible action, or when a searched-for object was not found. Examples of soft errors included `h` at the left margin, `<control-B>` or `[[` at the beginning of the file, `2G` at the end of the file, etc. In addition, errors such as `%`, `]]`, `}`, `)`, `N`, `n`, `f`, `F`, `t`, and `T` failing to find the searched-for object were soft as well. Less consistently, `/` and `?` displayed an error message if the pattern was not found, `/`, `?`, `N`, and `n` displayed an error message if no previous RE had been specified, and `;` did not display an error message if no previous `f`, `F`, `t`, or `T` command had occurred. Also, behavior in this area might reasonably be based on a run-time evaluation of the speed of a network connection. Finally, some implementations have provided error messages for soft errors in order to assist naive users, based on the value of a `verbose` edit option. This standard does not list specific errors for which an error message shall be displayed. Implementations should conform to historical practice in the absence of any strong reason to diverge.

The following table is a condensed version of information contained in the normative text. It is presented here to facilitate the review of the editor options that affect, or are affected by, `vi` commands.

`vi` Command	**Editor Options**
`!`	`autowrite, shell, warn, writeany`
`(, ), {, }`	`paragraphs, sections`
`/, ?, N, n`	`ignorecase, magic, wrapscan`
`<, >`	`shiftwidth, tabstop`
`A, a, I, i, O, o`	`autoindent, showmatch, wrapmargin`
`C, c, R, r, S, s`	`autoindent, showmatch, wrapmargin`
`ZZ`	`readonly, writeany`
`[[, ]]`	`sections`
`<control-B>`	`window`
`<control-D>`	`scroll`
`<control-F>`	`window`
`<control-T>`	`shiftwidth, tabstop`
`<control-U>`	`scroll`
`<control-]>`	`autowrite, tag, taglength, writeany`
`z`	`window`

The = (`reindent`) command was omitted because it was LISP language-specific, and LISP language support was omitted from this standard. (See E.5.10 for more information).

E.5.35.7.2.1 `<control-B>`

The `<control-B>` and `<control-F>` commands historically considered it an error to attempt to page past the beginning or end of the file, whereas the `<control-D>` and `<control-U>` commands simply moved to the beginning or end of the file. For consistency, this standard requires the latter behavior for all four commands. All four commands still consider it an error if the current line is at the beginning (`<control-B>`, `<control-U>`) or end (`<control-F>`, `<control-D>`) of the file. Historically, the `<control-B>` and `<control-F>` commands skip two lines in order to include overlapping lines when a single command is entered. This makes less sense in the presence of a *count*, as there will be, by definition, no overlapping lines. The actual calculation used by historical implementations of the `vi` editor for `<control-B>` was:

$$((\text{current first line}) - count \times (\texttt{window}\ \text{edit option})) + 2$$

and for `<control-F>` was:

$$((\text{current first line}) + count \times (\texttt{window}\ \text{edit option})) - 2$$

This calculation does not work well when intermixing commands with and without counts; e.g., `3control-F` is not equivalent to entering the `<control-F>` command three times, and is not reversible by entering the `<control-B>` command three times. For consistency with other `vi` commands that take counts, this standard requires a different calculation.

E.5.35.7.2.2 `<control-D>`

See 5.35.7.2.1. The 4BSD and System V implementations of `vi` differed on the initial value used by the `scroll` command. 4BSD used:

((`window` edit option) + 1) / 2

while System V used the value of the `scroll` edit option. The System V version is specified by this standard because the standard developers believed that it was more intuitive and permitted the user a method of setting the scroll value initially without also setting the number of lines that are displayed.

E.5.35.7.2.3 `<control-E>`

See E.5.35.7.2.1. Historically, the `<control-E>` and `<control-Y>` commands considered it an error if the last and first lines, respectively, were already on the screen. This standard requires conformance to historical practice.

Historically, the `<control-E>` and `<control-Y>` commands had no effect in open mode. For simplicity and consistency of specification, this standard requires that they behave as usual, albeit with a single line screen.

E.5.35.7.2.4 `<control-F>`

See E.5.35.7.2.1.

E.5.35.7.2.5 `<control-G>`

There is no additional rationale provided for this subclause.

E.5.35.7.2.6 `<control-H>`

There is no additional rationale provided for this subclause.

E.5.35.7.2.7 `<newline>`

There is no additional rationale provided for this subclause.

E.5.35.7.2.8 `<control-L>`

The historical `<control-L>` command refreshed the screen exactly as it was supposed to be currently displayed, replacing any `@` characters for lines that had been deleted but not updated on the screen (see 5.35.7.2) with refreshed `@` characters. The intent of the `<control-L>` command is to refresh when the screen has been accidentally overwritten; e.g., by a write by another user, or modem noise.

E.5.35.7.2.9 `<control-P>`

There is no additional rationale provided for this subclause.

E.5.35.7.2.10 <control-R>

The historical <control-R> command redisplayed only when necessary to update lines that had been deleted but not updated on the screen and that were flagged with @ characters (see 5.35.7). There is no requirement that the screen be in any way refreshed if no lines of this form are currently displayed. This standard permits implementations to extend this command to refresh lines on the screen flagged with @ characters because they are too long to be displayed in the current framework; however, the current line and column need not be modified.

E.5.35.7.2.11 <control-U>

See E.5.35.7.2.1 and E.5.35.7.3.2.

E.5.35.7.2.12 <control-Y>

See E.5.35.7.2.1 and E.5.35.7.2.3.

E.5.35.7.2.13 <control-^>

There is no additional rationale provided for this subclause.

E.5.35.7.2.14 <ESC>

Historically, an escape character, optionally preceded by a count, in command mode alerted the terminal, but an escape character preceded by part of a command did not. For example, 33c<ESC> is a partial command and is silently cancelled, but 33<ESC> must alert the terminal.

Historically, half entered [[,]], or ZZ commands were not cancelled by <ESC>; the terminal was alerted instead. For consistency and simplicity of specification, the standard does not permit this exception.

Historically, a leading <ESC> in a vi command was not an error when it resulted C
from a map expansion, and historical macros are known to depend on this feature. C
This standard requires this behavior. C

E.5.35.7.2.15 <control-]>

Historically, the first non-<blank> character at or after the cursor was the first character, and all subsequent characters that were word characters, up to the end of the line, were included. For example, with the cursor on the leading space or on the # character in the text " #bar@", the tag was #bar. On the character b it was bar, and on the a, it was ar. This standard requires this behavior.

E.5.35.7.2.16 <space>

There is no additional rationale provided for this subclause.

E.5.35.7.2.17 !

Historically, the `<`, `>`, and `!` commands considered most cursor motions other than line oriented motions an error; for example, the command `>/foo<CR>` succeeded, while the command `>l` failed, even though the text region described by the two commands might be identical. For consistency, all three commands only consider entire lines and not partial lines, and the region is defined as any line that contains a character that was specified by the motion.

E.5.35.7.2.18 $

There is no additional rationale provided for this subclause.

E.5.35.7.2.19 %

Other matching characters have been left implementation-defined in order to permit implementations to support the historical LISP options, and to allow extensions such as matching < and > for searching HTML, or `#ifdef`, `#else`, and `#endif` for searching C source.

E.5.35.7.2.20 &

This standard requires that any `c` and `g` flags specified to the previous substitute command be ignored; however, the `r` flag may still apply, if supported by the implementation.

E.5.35.7.2.21 ’

There is no additional rationale provided for this subclause.

E.5.35.7.2.22 ‘

There is no additional rationale provided for this subclause.

E.5.35.7.2.23 [[

The `[[`, `]]`, `(`, `)`, `{`, and `}` commands are all affected by “section boundaries,” but in some historical implementations not all of the commands recognize the same section boundaries. This is a bug, not a feature, and a unique section-boundary algorithm was not described for each command. One special case that is preserved is that the sentence command moves to the end of the last line of the edit buffer while the other commands go to the beginning, in order to preserve the traditional character cut semantics of the sentence command. Historically, `vi` section boundaries at the beginning and end of the edit buffer were the first non-blank character on the first and last lines of the edit buffer if one exists; otherwise, the last character of the first and last lines of the edit buffer if one exists; otherwise, the first and last lines of the edit buffer. To increase consistency with other section locations, this has been simplified by this standard to the first character of the first and last lines of the edit buffer, or the first and the last lines of the edit buffer if they are empty.

Sentence boundaries were problematic in the historical `vi`. They were not only the boundaries as defined for the section and paragraph commands, but they were the first nonblank character that occurred after those boundaries, as well.

Historically, the `vi` section commands were documented as taking an optional window size as a count preceding the command. This was not implemented in historical versions, so this standard requires that the count repeat the command, for consistency with other `vi` commands.

E.5.35.7.2.24]]

See E.5.35.7.2.23.

E.5.35.7.2.25 ^

There is no additional rationale provided for this subclause.

E.5.35.7.2.26 _

There is no additional rationale provided for this subclause.

E.5.35.7.2.27 (

See E.5.35.7.2.23.

E.5.35.7.2.28)

See E.5.35.7.2.23.

E.5.35.7.2.29 {

See E.5.35.7.2.23.

E.5.35.7.2.30 }

See E.5.35.7.2.23.

E.5.35.7.2.31 |

There is no additional rationale provided for this subclause.

E.5.35.7.2.32 ,

There is no additional rationale provided for this subclause.

E.5.35.7.2.33 .

Historically, mapped commands other than text input commands could not be repeated using the period command. This standard requires conformance to historical practice.

The restrictions on the interpretation of special characters (e.g., `<control-H>`) in the repetition of text input mode commands is intended to match historical practice. For example, given the input sequence

```
iab<control-H><control-H><control-H>def<escape>
```

the user should be informed of an error when the sequence is first entered, but not during a command repetition. The character `<control-T>` is specifically exempted from this restriction. Historical implementations of `vi` ignored `<control-T>` characters that were input in the original command during command repetition. This standard prohibits this behavior.

E.5.35.7.2.34 /

Historically, commands did not affect the line searched to or from if the motion command was a search (`/`, `?`, `N`, `n`) and the final position was the start/end of the line. There were some special cases and `vi` was not consistent. This standard does not permit this behavior, for consistency. Historical implementations permitted, but were unable to handle searches as motion commands that wrapped (i.e., due to the edit option `wrapscan`) to the original location. This standard requires that this behavior be treated as an error.

Historically, the syntax `/RE/0` was used to force the command to cut text in line mode. This standard requires conformance to historical practice.

Historically, in open mode, a `z` specified to a search command redisplayed the current line instead of displaying the current screen with the current line highlighted. For consistency and simplicity of specification, this standard does not permit this behavior.

Historically, trailing `z` commands were permitted and ignored if entered as part of a search used as a motion command. For consistency and simplicity of specification, this standard does not permit this behavior.

E.5.35.7.2.35 0

There is no additional rationale provided for this subclause.

E.5.35.7.2.36 :

Historically, `vi` implementations restricted the commands that could be entered on the colon command line (e.g., `append` and `change`), and some other commands were known to cause them to fail catastrophically. For consistency, this standard does not permit these restrictions. When executing an `ex` command by entering `:`, it is not possible to enter a `<newline>` as part of the command because it is considered the end of the command. A different approach is to enter `ex` command mode by using the `vi` `Q` command (and later resuming visual mode with the `ex` `vi` command). In `ex` command mode, the single-line limitation does not exist. So, for example, the following is valid:

```
Q %s/break here/break\
here/ vi
```

This standard requires that, if the ex command overwrites any part of the screen that would be erased by a refresh, vi pause for a character from the user. Historically, this character could be any character; e.g., a character input by the user before the message appeared, or even a mapped character. This is probably a bug, but implementations that have tried to be more rigorous by requiring that the user enter a specific character, or that the user enter a character after the message was displayed, have been forced by user indignation back into historical behavior. This standard requires conformance to historical practice.

E.5.35.7.2.37 ;

There is no additional rationale provided for this subclause.

E.5.35.7.2.38 <

See E.5.35.7.2.17 and E.5.35.7.3.4. Historically, the < and > commands sometimes moved the cursor to the first nonblank (e.g., if the command was repeated or with _ as the motion command), and sometimes left it unchanged. This standard does not permit this inconsistency, requiring instead that the cursor always move to the first nonblank.

Historically, the < and > commands did not support buffer arguments, although some implementations allow the specification of an optional buffer. This behavior is neither required nor disallowed by this standard.

E.5.35.7.2.39 >

See E.5.35.7.2.17, E.5.35.7.2.38, and E.5.35.7.3.4.

E.5.35.7.2.40 ?

See E.5.35.7.2.34.

E.5.35.7.2.41 @

Historically, buffers could execute other buffers, and loops, infinite and otherwise, were possible. This standard requires conformance to historical practice. The **buffer* syntax of ex is not required in vi, because it is not historical practice and has been used in some vi implementations to support additional scripting languages.

Historically, vi only supported the @@ syntax for repeating the last buffer execution. This standard requires that vi support the additional ex syntax @* as well, for consistency.

E.5.35.7.2.42 ~

Historically, the ~ command ignored any associated *count*, and acted only on the characters in the current line. For consistency with other vi commands, this standard requires that an associated *count* act on the next *count* characters, and that the command move to subsequent lines if warranted by *count*, to make it possible to modify large pieces of text in a reasonably efficient manner. There exist `vi` implementations that optionally require an associated motion command for the `~` command. Implementations supporting this functionality are encouraged to base it on the `tildedop` edit option and handle the text regions and cursor positioning identically to the `yank` command.

E.5.35.7.2.43 a

Historically, *count*s specified to the `A`, `a`, `I`, and `i` commands repeated the input of the first line *count* times, and did not repeat the subsequent lines of the input text. This standard requires that the entire text input be repeated *count* times.

E.5.35.7.2.44 A

See E.5.35.7.2.43.

E.5.35.7.2.45 b

Historically, `vi` became confused if word commands were used as motion commands in empty files. This standard requires that this be an error. Historical implementations of `vi` had a large number of bugs in the word movement commands, and they varied greatly in behavior in the presence of empty lines, “words” made up of a single character, and lines containing only `<blank>` characters. For consistency and simplicity of specification, this standard does not permit this behavior.

E.5.35.7.2.46 B

See E.5.35.7.2.45.

E.5.35.7.2.47 c

There is no additional rationale provided for this subclause.

E.5.35.7.2.48 C

Some historical implementations of the `C` command did not behave as described by this standard when the `$` key was remapped because they were implemented by pushing the `$` key onto the input queue and reprocessing it. This standard does not permit this behavior.

Historically, the `C`, `S`, and `s` commands did not copy replaced text into the numeric buffers. For consistency and simplicity of specification, this standard requires that they behave like their respective `c` commands in all respects.

E.5.35.7.2.49 d

Historically, lines in open mode that were deleted were scrolled up, and an `@` glyph written over the beginning of the line. In the case of terminals that are incapable of the necessary cursor motions, the editor erased the deleted line from the screen. This standard requires conformance to historical practice; i.e., if the terminal cannot display the @ character, the line cannot remain on the screen. C

E.5.35.7.2.50 `D`

Some historical implementations of the `D` command did not behave as described by this standard when the `$` key was remapped because they were implemented by pushing the `$` key onto the input queue and reprocessing it. This standard does not permit this behavior.

E.5.35.7.2.51 `e`

See E.5.35.7.2.45.

E.5.35.7.2.52 `E`

See E.5.35.7.2.45.

E.5.35.7.2.53 `f`

There is no additional rationale provided for this subclause.

E.5.35.7.2.54 `F`

There is no additional rationale provided for this subclause.

E.5.35.7.2.55 `G`

There is no additional rationale provided for this subclause.

E.5.35.7.2.56 `H`

There is no additional rationale provided for this subclause.

E.5.35.7.2.57 `i`

See E.5.35.7.2.43.

E.5.35.7.2.58 `I`

See E.5.35.7.2.43.

E.5.35.7.2.59 `J`

An historical oddity of `vi` is that the commands `J`, `1J`, and `2J` are all equivalent. This standard requires conformance to historical practice.

The `vi` `J` command is specified in terms of the `ex` `join` command with an `ex` command *count* value. The address correction for a count that is past the end of the edit buffer is necessary for historical compatibility for both `ex` and `vi`.

E.5.35.7.2.60 `L`

There is no additional rationale provided for this subclause.

E.5.35.7.2.61 `m`

Historical practice is that only lower-case letters, plus ` and ', could be used to mark a cursor position. This standard requires conformance to historical practice, but encourages implementations to support other characters as marks as well.

E.5.35.7.2.62 `M`

There is no additional rationale provided for this subclause.

E.5.35.7.2.63 `n`

Historically, the `N` and `n` commands could not be used as motion components for the `c` command. With the exception of the "`cN`" command, which worked if the search crossed a line boundary, the text region would be discarded, and the user would not be in text input mode. For consistency and simplicity of specification, this standard does not permit this behavior.

E.5.35.7.2.64 `N`

See E.5.35.7.2.63.

E.5.35.7.2.65 `o`

Historically, *count*s to the `O` and `o` commands were used as the number of physical lines to open, if the terminal was dumb and the `slowopen` option was not set. This was intended to minimize traffic over slow connections and repainting for dumb terminals. This standard does not permit this behavior, requiring that a *count* to the `open` command behave as for other text input commands. This change to historical practice was made for consistency, and because a superset of the functionality is provided by the `slowopen` edit option.

E.5.35.7.2.66 `O`

See E.5.35.7.2.65.

E.5.35.7.2.67 `p`

Historically, *count*s to the `p` and `P` commands were ignored if the buffer was a line mode buffer, but were (mostly) implemented as described in this standard if the buffer was a character mode buffer. Because implementations exist that do not have this limitation, and because pasting lines multiple times is generally useful, this standard requires that *count* be supported for all `p` and `P` commands.

Historical implementations of `vi` were widely known to have major problems in the `p` and `P` commands, particularly when unusual regions of text were copied into the edit buffer. The standard developers viewed these as bugs, and they are not permitted for consistency and simplicity of specification.

Historically, a `P` or `p` command (or an `ex put` command executed from open or visual mode) executed in an empty file, left an empty line as the first line of the file. For consistency and simplicity of specification, this standard does not permit this behavior.

E.5.35.7.2.68 `P`

See E.5.35.7.2.67.

E.5.35.7.2.69 `Q`

There is no additional rationale provided for this subclause.

E.5.35.7.2.70 `r`

Historically, the `r` command did not correctly handle the *erase* and *word erase* characters as arguments, nor did it handle an associated *count* greater than 1 with a `<carriage-return>` argument, for which it replaced *count* characters with a single `<newline>`. This standard does not permit these inconsistencies.

Historically, the `r` command permitted the `<control-V>` escaping of entered characters, such as `<ESC>` and `<carriage-return>`; however, it required two leading `<control-V>` characters instead of one. This standard requires that this be changed for consistency with the other text input commands of `vi`.

Historically, it is an error to enter the `r` command if there are less than *count* characters at or after the cursor in the line. While a reasonable and unambiguous extension would be to permit the `r` command on empty lines, it would require that too large a *count* be adjusted to match the number of characters at or after the cursor for consistency, which is sufficiently different from historical practice to be avoided. This standard requires conformance to historical practice.

E.5.35.7.2.71 `R`

Historically, if there were autoindent characters in the line on which the `R` command was run, and `autoindent` was set, the first `<newline>` character would be properly indented and no characters would be replaced by the `<newline>` character. Each additional `<newline>` character, would replace *n* characters, where *n* was the number of characters that were needed to indent the rest of the line to the proper indentation level. This behavior is a bug and is not permitted by this standard.

E.5.35.7.2.72 `s`

See E.5.35.7.2.48.

E.5.35.7.2.73 S

See E.5.35.7.2.48.

E.5.35.7.2.74 t

There is no additional rationale provided for this subclause.

E.5.35.7.2.75 T

There is no additional rationale provided for this subclause.

E.5.35.7.2.76 u

Historical practice for cursor positioning after undoing commands was mixed. In most cases, when undoing commands that affected a single line, the cursor was moved to the start of added or changed text, or immediately after deleted text. However, if the user had moved from the line being changed, the column was either set to the first nonblank, returned to the origin of the command, or remained unchanged. When undoing commands that affected multiple lines or entire lines, the cursor was moved to the first character in the first line restored. As an example of how inconsistent this was, a search, followed by an o text input command, followed by an undo would return the cursor to the location where the o command was entered, but a cw command followed by an o command followed by an undo would return the cursor to the first nonblank of the line. This standard requires the most useful of these behaviors, and discards the least useful, in the interest of consistency and simplicity of specification.

E.5.35.7.2.77 U

There is no additional rationale provided for this subclause.

E.5.35.7.2.78 w

See E.5.35.7.2.45.

E.5.35.7.2.79 W

See E.5.35.7.2.45.

E.5.35.7.2.80 x

There is no additional rationale provided for this subclause.

E.5.35.7.2.81 X

There is no additional rationale provided for this subclause.

E.5.35.7.2.82 y

Historically, the yank command did not move to the end of the motion if the motion was in the forward direction. It moved to the end of the motion if the motion was in the backward direction, except for the _ command, or for the G and ′ commands when the end of the motion was on the current line. This was further complicated by the fact that for a number of motion commands, the yank command moved the cursor but did not update the screen; e.g., a subsequent command would move the cursor from the end of the motion, even though the cursor on the screen had not reflected the cursor movement for the yank command. This standard requires that all yank commands associated with backward motions move the cursor to the end of the motion for consistency, and specifically, to make ′ commands as motions consistent with search patterns as motions.

E.5.35.7.2.83 Y

Some historical implementations of the Y command did not behave as described by this standard when the _ key was remapped because they were implemented by pushing the _ key onto the input queue and reprocessing it. This standard does not permit this behavior.

E.5.35.7.2.84 z

Historically, the z command always redrew the screen. This is permitted but not required by this standard, because of the frequent use of the z command in macros such as "map n nz." for screen positioning, instead of its use to change the screen size. The standard developers believed that expanding or scrolling the screen offered a better interface for users. The ability to redraw the screen is preserved if the optional new window size is specified, and in the <control-L> and <control-R> commands.

The semantics of z^ are confusing at best. Historical practice is that the screen before the screen that ended with the specified line is displayed. This standard requires conformance to historical practice.

Historically, the z command would not display a partial line at the top or bottom of the screen. If the partial line would normally have been displayed at the bottom of the screen, the command worked, but the partial line was replaced with @ characters. If the partial line would normally have been displayed at the top of the screen, the command would fail. For consistency and simplicity of specification, this standard does not permit this behavior.

Historically, the z command with a line specification of 1 ignored the command. For consistency and simplicity of specification, this standard does not permit this behavior.

Historically, the z command did not set the cursor column to the first nonblank for the ^ character if the first screen was to be displayed, and was already displayed. For consistency and simplicity of specification, this standard does not permit this behavior.

E.5.35.7.2.85 `ZZ`

There is no additional rationale provided for this subclause.

E.5.35.7.3 Input Mode Commands

Historical implementations of `vi` did not permit the the user to erase more than a single line of input, or to use normal erase characters such as *line erase*, *word erase*, and *erase* to erase autoindent characters. As there exist implementations of `vi` that do not have these limitations, both behaviors are permitted, but only historical practice is required. In the case of these extensions, `vi` is required to pause at the autoindent and previous line boundaries. C

Historical implementations of `vi` updated only the portion of the screen where the current cursor character was displayed. For example, consider the `vi` input keystrokes:

```
iabcd<escape>0C<tab>
```

Historically, the `<tab>` character would overwrite the characters `abcd` when it was displayed. Other implementations replace only the `a` character with the `<tab>`, and then push the rest of the characters ahead of the cursor. Both implementations have problems. The historical implementation is probably visually nicer for the above example; however, for the keystrokes

```
iabcd<ESC>0R<tab><ESC>
```

the historical implementation results in the string `bcd` disappearing and then magically reappearing when `<ESC>` is entered. This standard requires the former behavior when overwriting erase-columns; i.e., overwriting characters that are no longer logically part of the edit buffer, and the latter behavior otherwise.

Historical implementations of `vi` discarded the `<control-D>` and `<control-T>` characters when they were entered at places where their command functionality was not appropriate. This standard requires that the `<control-T>` functionality always be available, and that `<control-D>` be treated as any other key when not operating on autoindent characters.

E.5.35.7.3.1 `NUL`

Some historical implementations of `vi` limited the number of characters entered using the NUL input character to 256 bytes. This standard permits this limitation; however, implementations are encouraged to remove this limit. C

E.5.35.7.3.2 `<control-D>`

See E.5.35.7.3.4. The hidden assumptions in the `<control-D>` command (and in the `vi` `autoindent` specification in general) is that `<space>` characters take up a single column on the screen and that `<tab>` characters are comprised of an integral number of `<space>` characters.

E.5.35.7.3.3 <control-H>

There is no additional rationale provided for this subclause.

E.5.35.7.3.4 <newline>

Implementations are permitted to rewrite autoindent characters in the line when <newline>, <carriage-return>, <control-D>, and <control-T> are entered, or when the shift commands are used, because historical implementations have both done so and found it necessary to do so. For example, a <control-D> when the cursor is preceded by a single <tab>, with tabstop set to 8, and shiftwidth set to 3, will result in the tab being replaced by several <space> characters.

E.5.35.7.3.5 <control-T>

See E.5.35.7.3.4. Historically, <control-T> only worked if no non-<blank> characters had yet been input in the current input line. In addition, the characters inserted by <control-T> were treated as autoindent characters, and could not be erased using normal user erase characters. Because implementations exist that do not have these limitations, and as moving to a column boundary is generally useful, this standard requires that both limitations be removed.

E.5.35.7.3.6 <control-U>

There is no additional rationale provided for this subclause.

E.5.35.7.3.7 <control-V>

Historically, vi used ^V, regardless of the value of the literal-next character of the terminal. This standard requires conformance to historical practice.

The uses described for <control-V> can also be accomplished with <control-Q>, which is useful on terminals that use <control-V> for the down-arrow function. However, most historical implementations use <control-Q> for the *termios* START character, so the editor will generally not receive the <control-Q> unless stty ixon mode is set to off. (In addition, some historical implementations of vi explicitly set ixon mode to on, so it was difficult for the user to set it to off.) Any of the command characters described in POSIX.2 can be made ineffective by their selection as *termios* control characters, using the stty utility or other methods described in POSIX.1 {8}.

E.5.35.7.3.8 <control-W>

There is no additional rationale provided for this subclause.

E.5.35.7.3.9 <ESC>

Historically, SIGINT alerted the terminal when used to end input mode. This behavior is permitted, but not required, by this standard.

E.5.35.8 Exit Status

There is no additional rationale provided for this subclause.

E.5.35.9 Consequences of Errors

There is no additional rationale provided for this subclause.

Annex F
(informative)
Revisions to Portability Considerations

⇒ **F Portability Considerations.** *Remove all references to the C-Language Binding Option and* {POSIX2_C_BIND} *from this annex, or reword to indicate they have moved to P1003.1a.*

Rationale: Since Annex B is gone, all references to it have to be removed.

Annex G
(informative)

Revisions to Sample National Profile

⇒ **G Sample National Profile.** *Remove all references to the C-Language Binding Option and* {POSIX2_C_BIND} *from this annex, or reword to indicate they have moved to P1003.1a.*

Rationale: Since Annex B is gone, all references to it have to be removed.

Annex H
(informative)
Balloting Instructions

This annex will not appear in the final standard. It is included in the draft to provide instructions for balloting that cannot be separated easily from the main document, as a cover letter might.

If you have received a copy of this draft before July 1999, it is important that you read this annex, whether you are an official member of the P1003.2b Balloting Group or not; comments on this draft are welcomed from all interested technical experts. **Your ballot is due to the IEEE office by ___ July 1999. This is not the date to postmark it—it is the date of receipt.**

Summary of Draft 12 Instructions

This is the second "recirculation draft" of P1003.2b. The recirculation procedure is described in this annex. For this recirculation, we are accepting objections against any normative changes that occurred from Draft 11 to Draft 12 and the contents of the Unresolved Objections List, provided as a separate document from the draft.

Send your ballot and/or comments to:

IEEE Standards Office
Computer Society Secretariat
ATTN: P1003.2b Ballot
P.O. Box 1331
445 Hoes Lane
Piscataway, NJ 08855-1331

It would also be very helpful if you sent us your ballot in machine-readable form. Your official ballot must be returned via mail to the IEEE office; if we receive only the e-mail or diskette version, that version will not count as an official document. However, the online version would be a great help to ballot resolution. Please e-mail to both of the following addresses:

Don.Cragun@eng.sun.com
nick@usenix.org

or IBM PC 3.5-inch diskette (plain text file), or Sun-style QIC-24 cartridge tapes to:

Don Cragun
Sun Microsystems, Inc.
M/S UMPK17-307
901 San Antonio Road
Palo Alto, CA 94303

Some degree of judgment is required in determining what actually changed in Draft 12. Use the diff marks as a guide, but they will frequently mark text that has no real normative changes. Please limit your objections to the actual changes: for example, if we change the `foo` −`x` option to −`y`, don't use that as an opportunity to object that we have no −`z` option. Your objection should only address why the `x` to `y` change is a problem. (We have been balloting for a long time now and it is time to tighten the consensus and finish this up.) If you find problems unrelated to changes, submit them as comments and they will be considered seriously in that category. Thanks for your cooperation on this.

Background on Balloting Procedures

The Balloting Group consists of many technical experts who are members of the IEEE or the IEEE Computer Society; enrollment of individuals in this group has already been closed. There are also a few "parties of interest" who are not members of the IEEE or the Computer Society. Members of the Balloting Group are required to return ballots within the balloting period. Other individuals who may happen to read this draft are also encouraged to submit comments concerning this draft. The only real difference between members of the Balloting Group and other individuals submitting ballots is that *affirmative* ballots are only counted from Balloting Group members who are also IEEE or Computer Society members. (There are minimum requirements for the percentages of ballots returned and for affirmative ballots out of that group.) However, objections and nonbinding comments must be resolved if received from any individual, as follows:

(1) Some objections or comments will result in changes to the standard. This will occur either by the publication of a list of changes or by the republication of an entire draft. The objections/comments are reviewed by a team from the P1003.2 working group, consisting of the Chair, Vice Chair, the Chair of PASC, and one or more Technical Reviewers. The Technical Reviewers each have subject matter expertise in a particular area and are responsible for objection resolution in one or more sections.

(2) Other objections/comments will not result in changes.

 (a) Some are misunderstandings or cover portions of the document (front matter, informative annexes, rationale, editorial matters, etc.) that are not subject to balloting.

 (b) Others are so vaguely worded that it is impossible to determine what changes would satisfy the objector. These are referred to as *Unresponsive*. (The Technical Reviewers will make a reasonable effort to contact the objector to resolve this and get a newly worded objection.) Further examples of unresponsive submittals are those not marked as either *Objection* or *Comment*; those that do not identify the portion of the document that is being objected to (each objection must be separately labeled); those that object to material in a recirculation that has not changed and do not cite an unresolved objection; those that do not provide specific or general guidance on what changes would be required to resolve the objection.

(c) Finally, others are valid technical points, but they would result in decreasing the consensus of the Balloting Group. (This judgment is made based on other ballots and on the experiences of the working group through almost five years of work and fifteen drafts preceding this one.) These are referred to as *Unresolved Objections*. Summaries of unresolved objections and their reasons for rejection are maintained throughout the balloting process, are circulated to members of the Balloting Group for their consideration, and are presented to the IEEE Standards Board when the final draft is offered for approval. Unresolved objections are only circulated to the balloting group when they are presented by members of the balloting group or by parties of interest. Unsolicited correspondence from outside these two groups may result in draft changes, but are not recirculated to the balloting group members.

Please ensure that you correctly characterize your ballot by providing one of the following:

(1) Your IEEE member number

(2) Your IEEE Computer Society affiliate number

(3) If (1) or (2) don't apply, a statement that you are a "Party of Interest"

Ballot Resolution

The general procedure for resolving ballots is:

(1) The balloting cuts off on ___ July 1999. This is a receipt date at the IEEE, not a postmark date. (Please do not telephone or FAX on ___ July 1999 and say that your specific comments will come later; late-arriving comments will not be considered as objections.) We will accept comments after that date, including direct e-mail to the working group officers or the Technical Reviewers, but they will be treated as comments only—not objections. And we don't guarantee a written response to these late submissions.

(2) The ballots are put online and distributed to the Technical Reviewers.

(3) If a ballot contains an objection, the balloter will be contacted individually by telephone, letter, or e-mail and the corrective action to be taken will be described (or negotiated). The personal contact will most likely not occur if the objection is very simple and obvious to fix or the balloter cannot be reached after a few reasonable attempts. Repeated failed attempts to elicit a response from a balloter may result in an objection being considered unresponsive, based on the judgment of the working

group chair. Once all objections in a ballot have been resolved, it becomes an affirmative ballot.

(4) If any objection cannot be resolved, the entire ballot remains negative.

(5) Once more than seventy-five percent of the ballots received (that had voted either affirmative or negative) have been turned affirmative, two lists are published to the entire balloting group: the detailed list of approved changes and the list of unresolved objections, along with our reasons for rejecting them. This is known as a *recirculation*. You have minimum of ten days (after an appropriate time to ensure the mail got through) to review these two lists and take one of the following actions:

(a) Do nothing; your ballots will continue to be counted as we have classified them, based on items (3) and (4).

(b) Explicitly change your negative ballot to affirmative by agreeing to remove all of your objections from the unresolved list.

(c) Explicitly change your affirmative ballot to negative based on your disapproval of either of the two lists you reviewed. If an issue is not on one of the two lists, new objections about this are not allowed. Negative ballots that come in on recirculations cannot be cumulative. They shall repeat any objections that the balloter considers unresolved from the previous recirculation. Ballots that simply say "and all the unresolved objections from last time" will be declared unresponsive. Ballots that are silent will be presumed to fully replace the previous ballot, and all objections not mentioned on the most current ballot will be considered as successfully resolved.

(6) The list of changes will frequently be a new draft document with the changes integrated. This is not a requirement, however, and a small number of changes may prompt merely a change list approach to recirculation.

(7) A copy of all your objections and our resolutions will be mailed to you. You can receive the full package of all resolutions from all ballots by contacting the IEEE Standards Office (who will probably charge you for the copying involved). If you don't agree with one of our resolutions and haven't been contacted personally before you receive this list, please accept our apologies and submit a new ballot against the new draft during the recirculation period.

(8) If at the end of the recirculation period there remain greater than seventy-five percent affirmative ballots, and no new objections have been received, a new draft is prepared that incorporates all the changes. This draft and the unresolved objections list go to the IEEE Standards Board for approval. If the changes cause too many ballots to slip back into negative status, another resolution and recirculation cycle begins.

Balloting Guidelines

This section consists of guidelines on how to write and submit the most effective ballot possible. The activity of resolving balloting comments is difficult and time consuming. Poorly constructed comments can make that even worse.

We have found several things that can be done to a ballot that make our job more difficult than it needs to be, and likely will result in a less than optimal response to ballots that do not follow the form below. Thus it is to your advantage, as well as ours, for you to follow these recommendations and requirements.

If a ballot that significantly violates the guidelines described in this section comes to us, we will determine that the ballot is unresponsive, and simply ignore all the material in it.

Secondly, objections that don't contain a specification so that the correction to resolve the objection "can be readily determined" are also unresponsive and will be ignored.

(If we do recognize a ballot that is generally "unresponsive," we will try to inform the balloter as soon as possible so he/she can correct it, but it is ultimately the balloter's responsibility to assure the ballot is responsive.)

Typesetting is not particularly useful to us. And please do not send handwritten ballots. Typewritten (or equivalent) is fine, and if some font information is lost it will be restored by the Technical Editor in any case. If you use `nroff`, you will include extraneous spacing and sometimes backspaces and overstrikes; if you really must use `nroff`, please turn off hyphenation and line adjusting:

```
.hy 0
.na
```

and run the output through `col -b` to remove all the overstrikes. (Also remember that backslashes and leading periods and apostrophes in your text will be treated impolitely by the `*roff` family). The ideal ballot is formatted as a "flat ASCII file," without any attempt at reproducing the typography of the draft and without embedded control characters or overstrikes; it is then printed in Courier (or some other typewriter-like) font for paper-mailing to the IEEE Standards Office and simultaneously e-mailed to the working group Chair.

Don't quote others' ballots. Cite them if you want to refer to another's ballot. If more than one person wants to endorse the same ballot, send just the cover sheets and one copy of the comments and objections. [Note to Institutional Representatives of groups like X/Open, OSF, UI, etc.: this applies to you, too. Please don't duplicate objection text with your members.] Multiple identical copies are easy to deal with, but just increase the paper volume. Multiple almost-identical ballots are a disaster because we can't tell if they are identical or not, and are likely to miss the subtle differences. Responses of the forms:

— "I agree with the item in <someone>'s ballot, but I'd like to see this done instead"

— "I am familiar with the changes to `foo` in <someone>'s ballot and I would object if this change is [or is not] included" are very useful information to us. If we resolve the objection with the original balloter (the one whose ballot you are referencing), we will also consider yours to be closed, unless you specifically include some text in your objection indicating that should not be done.

Be very careful of “Oh, by the way, this applies <here> too” items, particularly if they are in different sections of the document that are likely to be seen by different reviewers. They are probably going to be missed! Note the problem in the appropriate section, and cite the detailed description if it’s too much trouble to copy it. The reviewers don’t have time to read the whole ballot, and only read the parts that appear to apply to them. Particularly where definitions are involved, even if the change really belongs in one section but the relevant content is in another, an extra cross-reference would be indicated. If you wish to endorse someone else’s ballot, either in whole or part, be specific about whether you will be automatically satisfied if they are satisfied. If you will not necessarily be satisfied if they are, your ballot could be deemed unresponsive because it does not give achievable conditions under which your ballot could be converted to affirmative. You then must give the conditions under which you would be satisfied as well. If you would be satisfied in some areas and not in others, it is best to specifically point to each specific objection in the ballot you point to, giving the conditions for each.

Please consider this a new ballot that should stand on its own. Please do not make backward references to your ballots for previous drafts—include all the text you want considered here because the Technical Reviewer may not have your old ballot. And, the old section and line numbers won’t match up anyway. If one of your objections was not accepted exactly as you wanted, it will not be useful to send in the exact text you sent before; read the nearby Rationale section and come up with a more compelling (or clearly-stated) justification for the change.

Please be very wary about global statements, such as “all of the arithmetic functions need to be defined more clearly.” Unless you are prepared to cite specific instances of where you want changes made, with reasonably precise replacement language, your ballot will be considered unresponsive.

Ballot Form

The following form is recommended. We would greatly appreciate it if you sent the ballot in electronic form in addition to the required paper copy. Our policy is to handle all ballots online, so if you don’t send it to us that way, we have to type it in manually. For the last POSIX.2 ballot, only one or two balloters could not accommodate us on this and thus we had very little typing to do. See the first page of this Annex for the addresses and media. As you’ll see from the following, formatting a ballot that’s sent to us online is much simpler than a paper-only ballot.

The ballot should be page-numbered, and contain the name, e-mail address, and phone number(s) of the objector(s). (If you send us only a paper copy, make sure this information appears on every page; electronic ballots just need it once, in the beginning.) The lines before the first dashed line are a page header, and should only appear once on each page. Please leave adequate (at least one inch) margins on both sides. Each objection/comment/editorial comment should be sequentially numbered, not in individual ranges [i.e., not Objection #1, Comment #1]

Since we deal with the ballots online, there is no longer any requirement to put only one objection or section per page.

Don't format the ballot as a letter or document with its *own* section numbers. These are simply confusing. As shown below, it is best if you cause each objection and comment to have a sequential number that we can refer to amongst ourselves and to you over the phone. Number sequentially from 1 and count objections, comments, and editorial comments the same; don't number each in its own range. If you don't do this, we'll number them ourselves, but you won't know what numbers we're using.

Please precede each objection/comment with a little code line (if you don't, we'll have to do it ourselves):

@ <section>`.`*<clause> <code> <seqno>*

where:

@	At sign in column 1 (which means no @'s in any other column 1's).
<section>	The major section (chapter or annex) number or letter in column 3. Use zero for Global or for something, like the front matter, that has no section or annex number.
<clause>	The clause number (second-level header). Please do not go deeper than these two levels. In the text of your objection or comment, go as deep as you can in describing the location, but this code line uses two levels only.
<code>	One of the following lowercase letters, preceded and followed by spaces: o Objection. c Comment or Editorial Comment.
<seqno>	A sequence number, counting all objections and comments in a single range.

Objection:

```
Balloter Name              (202)555-1212       page  x of nn.
E-Mail Address             FAX:   Fax Number
Balloter2 Name                    (303)555-1213
E-Mail Address2            FAX:   Fax Number2
-------------------------------------------------------------------
@ x.y o seq#
<Seq#> Sect x.y OBJECTION. page xxx, line zzz:
```

Problem:

A clear statement of the problem that is observed, sufficient for others to understand the nature of the problem. Note that you should identify problems by section, page, and line numbers. This may seem redundant, but if you transpose a digit pair, we may get totally lost without a cross-check like this. Use the line number where the problem starts, not just where the section itself starts; we sometimes attempt to sort objections by line numbers to make editing more accurate. If you are referring to a range of lines, please don't say "lines 1000ff;" use a real range so we can tell where to stop looking. If you have access to the online versions of a balloting draft, please do not send in a ballot that refers to the page numbers in the `nroff` output version; use only the line and page numbers found in the printed draft or the online PostScript draft. We will really love you if you can manage to include enough context information in the problem statement (such as the name of the utility) so we can understand it without having the draft in our laps at the time. (It also helps you when we e-mail it back to you.) If you are objecting to an action in the Unresolved Objections List, use the section/page/line number reference for the appropriate place in the standard; don't refer to the UOL except to cite its number and for clarification of your points.

Action:

A precise statement of the actions to be taken on the document to resolve the objection above, which if taken verbatim will completely remove the objection.

If there is an acceptable range of actions, any of which will resolve the problem for you if taken exactly, please indicate all of them. If we accept any of these, your objection will be considered as resolved.

If the Action section is omitted or is vague in its solution, the objection will be reclassified as a nonbinding comment. The Technical Reviewers, being human, will give more attention to Actions that are well-described than ones that are vague or imprecise. The best ballots of all have very explicit directions to substitute, delete, or add text in a style consistent with the rest of the document, such as:

```
Delete the sentence on lines 101-102:
      "The implementation shall not ... or standard error."
On line 245, change "shall not" to "should not".
After line 103, add:
       -r    Reverse the order of bytes read from the file.
```

Some examples of poorly-constructed actions:

```
Remove all features of this command that are not supported by BSD.
Add -i.
Make this command more efficient and reliable.
Use some other flag that isn't so confusing.
I don't understand this section.
Specify a value--I don't care what.
```

Objection Example:

```
Hal Jespersen             (415) 364-3410            page  3 of 17.
UUCP: hlj@Posix.COM  FAX:   (415) 364-4498
-----------------------------------------------------------------
@ 2.6 o 23
23. Sect 2.6 OBJECTION. page 77, line 1217:

Problem:

The EDITOR environment variable is not used as stated
in my company.  This description would cause hundreds
of my shell scripts to break.

Action:

Change the first sentence on line 1217 to:

      The e-mail address of the editor of the user's
      favorite POSIX standard.

-----------------------
@ 3.1 o 24
24. Sect 3.1.6 OBJECTION. page 123, line 17:

Problem:

I support UO 3.01-999-6 concerning the objection to the
definition of "operator".
This definition would cause great hardship to the users
of the systems I develop.
I feel your rationale for rejection was inappropriate
because you overlooked the following technical points [etc.]...

Action:

Change the term "operator" to "operation-symbol" in this
definition and globally throughout Section 3.
```

Comment:

```
----------------------------------------------------------------
@ x.z c seq#
<Seq#> Sect x.z COMMENT. page xxx, line zzz:
```

A statement of a problem that you might want to be resolved by the reviewer, but which does not in any way affect whether your ballot is negative or positive. The form for objections is not required, but it increases the probability that your comment will have an effect on the final document.

Although there may be questions to you or responses on the topic, no changes in the drafts are required by a comment, although it will be looked at to determine whether the concern should be addressed. It is possible to abuse this rule and label all of your comments as objections, but it is a significant disservice to the individuals who are volunteering their time to address your concerns.

Remember that any issue concerning the pages preceding page 1 (the front matter), Rationale text with shaded margins, Annexes, NOTES in the text, footnotes, or examples will be treated as a nonbinding comment whether you label it that way or not, but it would help us if you'd label it correctly.

Editorial Comment:

```
----------------------------------------------------------------
@ x.z c seq#
<Seq#> Sect x.z EDITORIAL COMMENT. page xxx, line zzz:
```

These are for strictly editorial issues, where the technical meaning of the document is not changed. Examples are: typos; misspellings; English syntax or usage errors; appearances of lists or tables; arrangement of sections, clauses, and subclauses (except where the location of information changes the optionality of a feature). Marking these as comments but indicating that they are editorial speeds the process.

Please be aware that after balloting concludes the document will be subjected to more sets of editors at the IEEE and ISO who are empowered to make broad editorial changes and rewording (for example, to get the text ready for translation into French.)

Thank you for your cooperation in this important balloting process.

Don Cragun

Alphabetic Topical Index