WG14/N370

X3J11/94-055

TO: WG14 and X3J11

FROM: Kohmei Kuroda (kuroda@slab.ntt.jp)

PLP DT leader of SPIRIT

PLP DT is the abbreviation of Programming Language Portability Drafting Team

DATE: July 27,1994

SUBJECT: SPIRIT C

Attached are the following documents:

1. Aim of SPIRIT
2. Outline of SPIRIT C
3. SPIRIT C specification.

Message from SPIRIT

1. SPIRIT C will contribute to the next standard.
2. SPIRIT needs extended integers.

Aim of SPIRIT

SPIRIT

Service Providers' Integrated Requirements for Information Technology

The aim of SPIRIT is to produce a common, agreed set of specifications for a general purpose computing platform for the telecommunications industry by March '95.

Participants

SPIRIT

<u>service providers</u>:

AT&T, Bellcore, BT, ETIS, France Telecom, KDD, NTT, STET (CSELT/SIP), Telefonica , CBIS, DBP

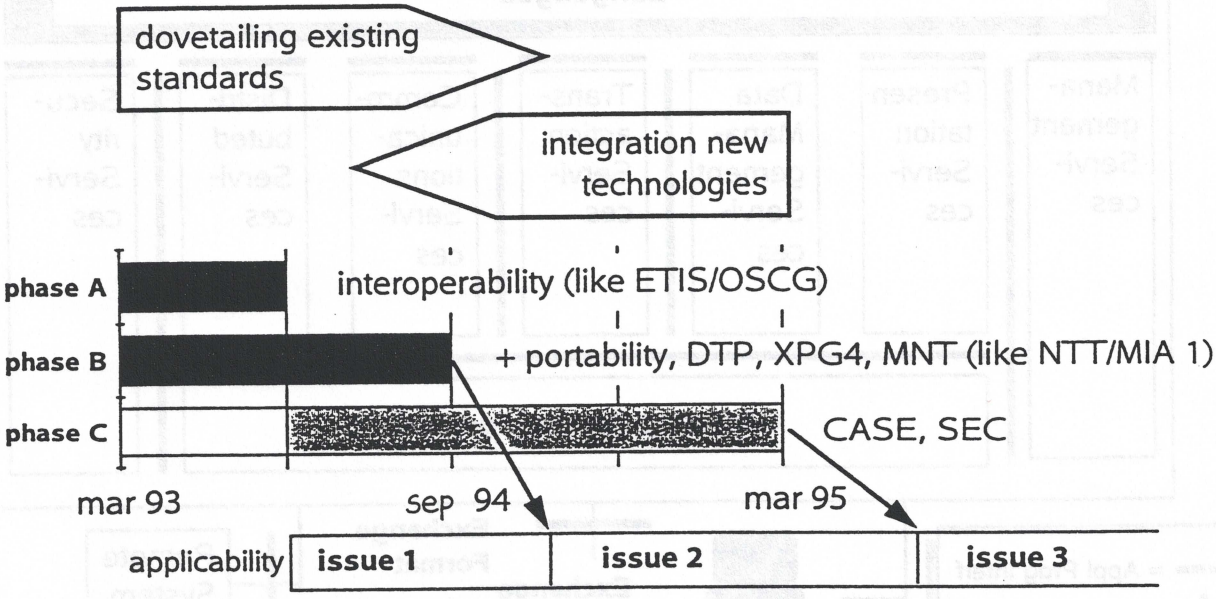
(12 European members participate via ETIS)

<u>vendors</u>:

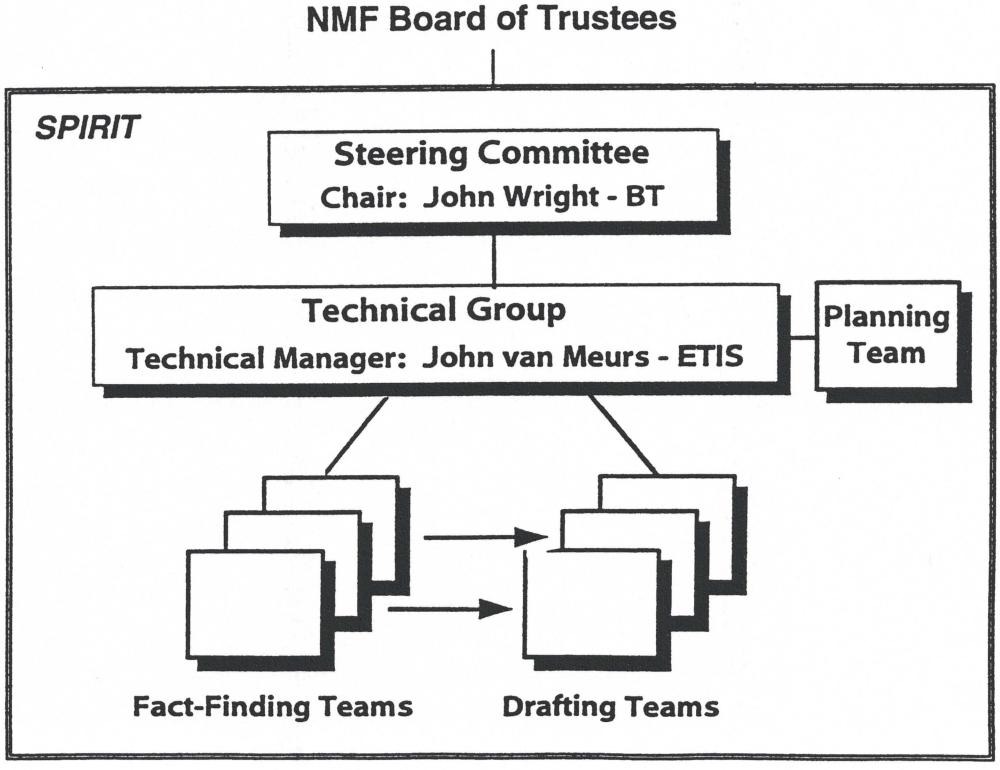
Bull, Digital, Fujitsu (incl ICL), Hitachi, HP (incl YHP), IBM, NCR, NEC, Siemens Nixdorf, Sun , Oracle

Alcatel, BNR (NT), GPT

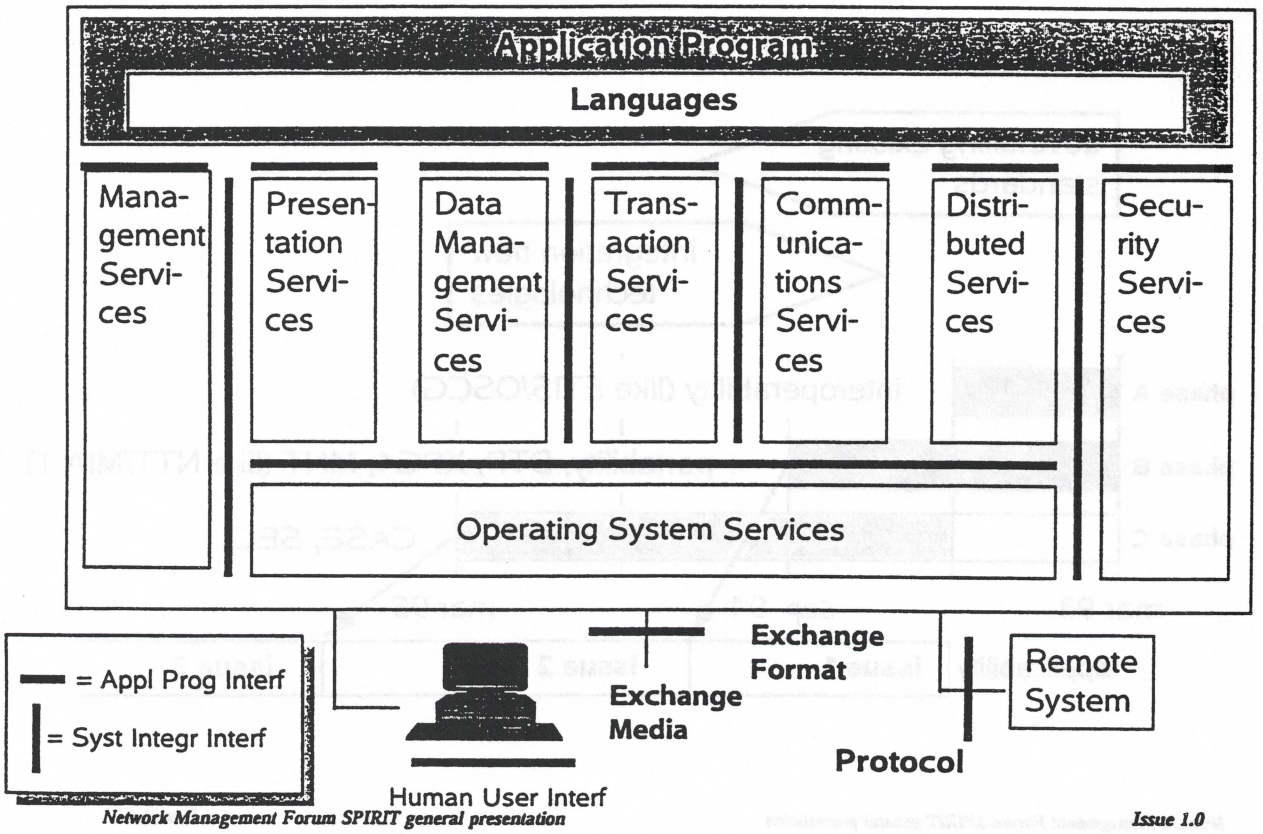
Deliverables, Timescales



Organisation



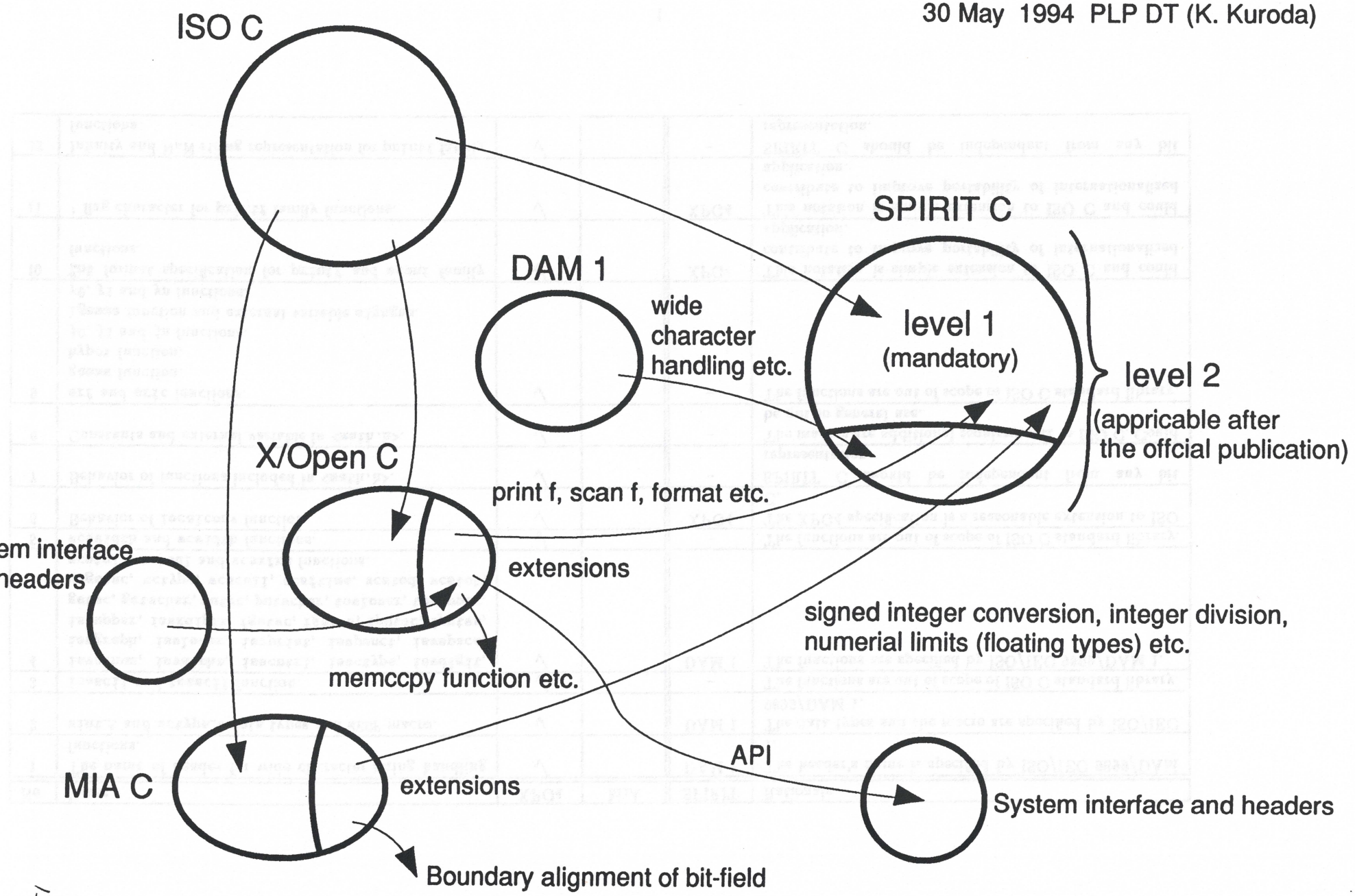
SPIRIT Interface Model



12

Outline of SPIRIT C

30 May 1994 PLP DT (K. Kuroda)



No.	Items	XPG4	MIA	SPIRIT	Rationale
1	The name of header for wide character string handling functions.	√		DAM 1	The header's name is specified by ISO/IEC 9899/DAM 1.
2	wint_t and wctype_t data types and WEOF macro.	√		DAM 1	The data types and the macro are specified by ISO/IEC 9899/DAM 1.
3	isascii and toascii function.	√		–	The functions are out of scope of ISO C standard library.
4	iswalnum, iswalpha, iswcntrl, iswctype, iswdigit, iswgraph, iswlower, iswprint, iswpunct, iswspace, iswupper, iswxdigit, fgetwc, fgetws, fputwc, fputws, getwc, getwchar, putwc, putwchar, towlower, towupper, ungetwc, wctype, wcscoll, wcsftime, wcstod, wcstok, wcstol, wcstoul and wcsxfrm functions.	√		DAM 1	The functions are specified by ISO/IEC 9899/DAM 1.
5	wcswidth and wcwidth functions.	√		–	The functions are out of scope of ISO C standard library.
6	Behavior of localconv function.	√		XPG4	The XPG4 specification is a reasonable extension to ISO C.
7	Behavior of functions included in <math.h>.	√		–	SPIRIT C should be independent from any bit representation.
8	Constants and external variable in <math.h>.	√		–	The macros are additional requirements to ISO C. Could be not so general use.
9	erf and erfc functions. gamma function. hypot function. j0, j1 and jn functions. lgamma function and external variable signgam. y0, y1 and yn functions.	√		–	The functions are out of scope of ISO C standard library.
10	%n$ format specification for printf and scanf family functions.	√		XPG4	This notation is simple extension to ISO C and could contribute to improve portability of internationalized application.
11	' flag character for printf family functions.	√		XPG4	This notation is simple extension to ISO C and could contribute to improve portability of internationalized application.
12	Infinity and NaN string representation for printf family functions.	√		–	SPIRIT C should be independent from any bit representation.

No.	Items	XPG4	MIA	SPIRIT	Rationale
13	%C and %S conversion character for printf and scanf family functions.	√		DAM 1	The equivalent notations, i.e., %wc and %ws, are specified by ISO/IEC 9899/DAM 1.
14	Parameter of fseek function.	√		–	The description is a kind of guide for user of this function. It does not specify any behavior of the function.
15	memccpy function. strfmon function. nl_langinfo function. iconv family functions. catopen, catclose and catgets functions.	√		–	The functions are out of scope of ISO C standard library.
16	Additional macros in <limits.h>.	√		–	The macros are out of scope of ISO C standard library.
17	Period of rand function.	√		XPG4	The implementation limit could be make sense and could contribute to application portability.
18	Conversion specifications for strftime function.	√		XPG4	This notation is simple extension to ISO C and could contribute to improve portability of internationalized application.
19	Requirement of full specification of ISO C v.s. Common Usage C.	√		–	XPG4 says that "Common Usage C" is obsolescent.
20	Conversion into signed integer type of the smaller or equal size.		√	MIA	To specify ISO C inplementation-defined behavior could contribute to application portability.
21	Result of the division (/,%) of negative integer.		√	MIA	To specify ISO C inplementation-defined behavior could contribute to application portability.
22	Result of the right shift when the first operand is a signed integer type and has a negative value.		√	MIA	To specify ISO C inplementation-defined behavior could contribute to application portability.
23	Behavior of freopen function.	√	√	–	MIA specification conflicts with both XPG4 and POSIX. On the other hand, specification of specific errno that is not specifiedby ISO C is out of scope.
24	Return value of the fmod function when either argument is zero.	√	√	–	Return value zero required by MIA is also allowed by XPG4. However, there is no description on errno in MIA.
25	Additional of signals.	√	√	–	These macros are out of scope of ISO C. They could be not so general use.
26	Behavior of the rename function when there exists a file with same name specified as second argument.	√	√	–	Considering XPG4 and MIA specifications are conflict each other completely different. Neither specification would not contribute to application portability so significantly.
27	Boundary alignment of bit-field.		√	–	MIA specification conflicts with ISO C when multiple 0 width bit-fields are declared continuously.
28	Data type of wcschr and wcsrchr functions.	√	√	DAM 1	The functions are specified by ISO/IEC 9899/DAM 1.

No.	Items	XPG4	MIA	SPIRIT	Rationale
29	Number of significant initial characters in an external identifier.	√	√	MIA	The limit is implementation limit of ISO C. Tough condition for conforming implementation would contribute to improve application portability.
30	Numerical value in <limits.h> (MB_LEN_MAX).	√	√	-	This implementation limit depends on locale supported by a conforming implementation. It is not general requirement.
31	Numerical value in <float.h>.	√	√	MIA	This limit is implementation limit of ISO C. Tough condition for conforming implementation would contribute to improve application portability.
32	wcscpy, wcsncpy, wcscat, wcscmp, wcsncmp, wcschr, wcscspn, wcspbrk, wcsrchr, wcsspn and wcslen functions.	√	√	DAM 1	The functions are specified by ISO/IEC 9899/DAM 1.
33	wcswcs function.	√	√	DAM 1	Equivalent function, i.e., wcsstr function, is specified by ISO/IEC 9899/DAM 1. Only function name is different.

SPIRIT

Service Providers Integrated Requirements for Information Technology

Part 2 Development Environment
Volume 2 C Language

Published by

Network Management FORUM

Issue 2.0
Draft 1.1

30 May 1994

Network Management Forum Notice

The Network Management Forum has attempted to make the contents of this document accurate; however, due to the inherent complexity in the design and implementation of protocols and interfaces, and other aspects of the information contained herein, no liability is accepted for any errors or omissions or for consequences of any use made of this document. Under no circumstances will the Network Management Forum or any of its members be liable for direct or indirect damages or any costs or losses resulting from the use of this specification. A user's sole and exclusive remedy is to provide input to the Network Management Forum for its consideration in redrafting the contents within the Forum's own discretion. Should the Forum not exist at the time the user presents its input, any claim by the user shall be deemed to be stale and unactionable. The risk of designing and implementing products in accordance with this document is taken solely by the user of this specification. This document may involve a claim of patent rights by one or more of the contributors to this document, pursuant to the agreement on Intellectual Property Rights between the Network Management Forum and its members.

Each member of the Network Management Forum reserves the right to make its own independent procurement decisions in whatever manner it deems appropriate. Nothing herein is intended as a recommendation of any product or service to anyone, nor does this document represent any commitment by anyone to purchase any product or service.

The Network Management Forum reserves the right to revise or withdraw this document for any reason.

SPIRIT Notice

This document references products for the purpose of illustration only, to be representative of the matter which SPIRIT is reviewing. Inclusion by name does not constitute a license or grant of right to use, which must be obtained from the product owner. SPIRIT has not done a technical review of the products referenced nor has SPIRIT obtained information on all functionally equivalent products. Thus, this document is not intended to constitute a recommendation or endorsement of any particular product. Questions concerning references to products should be brought to the attention of SPIRIT through the Network Management Forum office.

Inquires should be directed to the
Network Management Forum office at:

1201 Mt. Kemble Avenue
Morristown, NJ 07960
Tel. No. +1 201-425-1900

Contents

19

20

22

Figures

Tables

0 Introduction

The purpose of this C language specification is to define a standard interface between application programs and systems software for SPIRIT environment, so that any application program written to this C language interface be portable across SPIRIT conforming platforms regardless of its supply vendors. This C language interface is intended to be used in conjunction with other SPIRIT APIs, such as Base System API, that have C language binding.

This SPIRIT C language specification complies with ISO C language standard, i.e. ISO/IEC 9899:1990, with its (proposed) amendment, i.e. ISO/IEC 9899/AM1:.... . Since all the mandatory requirements of the ISO C standard are also applicable to this SPIRIT C language specification, any implementation conforms to this specification also be ISO C standard compliant. However, this SPIRIT C language specification defines some additional requirements to the ISO C standard, a conforming application to this specification not always be strictly conforming application of the ISO C standard.

This SPIRIT C language specification document describes the differences between the ISO C standard and this SPIRIT C language specification. The specification elements that are identical with ISO C standard are not restated in this document. All the phrases of "This International Standard" in the ISO C standard need to be replaced with "This specification" at the application of the technical contents to this SPIRIT C language specification.

The following typographical conventions are used throughout this document:

Sans-serif string is used for indication of additional requirements to the ISO C standard that the SPIRIT C language specifies.

— The indication of "The same as ISO/IEC 9899:1990", "The same as ISO/IEC 9899/AM1" or "The same as ISO/IEC 9899:1990 with ISO/IEC 9899/AM1" means that the technical contents of the subject clause or subclause in this SPIRIT C language specification is identical with the corresponding clause or subclause of identified ISO C standard specification.

— The indication of "N/C" means that the SPIRIT C language specification does not change any technical contents of the unnumbered subclause of corresponding ISO C standard.

The organization of ISO/IEC 9899:1990 is maintained in this specification, as far as possible, but new subclauses, introduced by ISO/IEC 9899/AM1, e.g. subclauses "Wide-character classification and mapping utilities" and "Extended multibyte and wide-character utilities", in the corresponding clauses of this specification. Also a clause of "Code conversion utilities" is added at the bottom of "Library" section.

1 Scope

The same as ISO/IEC 9899:1990

2 Normative references

The same as ISO/IEC 9899:1990

3 Definitions and conventions

In this specification, "shall" is to be interpreted as a requirement on an implementation or a program; conversely, "shall not" to be interpreted as prohibition.

For the purposes of this specification, the following definitions apply. Other terms are defined at their first appearance, indicated by *italic* type. Terms explicitly defined in this specification are not to be presumed to refer implicitly to similar terms defined elsewhere. Terms not defined in this specification are to be interpreted according to ISO 2382.

3.1 alignment: N/C

3.2 argument: N/C

3.3 bit: N/C

3.4 byte: N/C

3.5 character: N/C

3.6 constraints: N/C

3.7 diagnostic message: N/C

3.8 forward references: N/C

3.9 implementation: N/C

3.10 implementation-defined behavior: N/C

3.11 implementation limits: N/C

3.12 multibyte character: N/C

3.13 object: N/C

3.14 parameter: N/C

3.15 undefined behavior: N/C

3.16 unspecified behavior: N/C

3.17 conformance document: A document provided by an implementer that contains implementation details as described in the "Documentation" clause of "Compliance" section.

4 Compliance

4.1 Conforming implementation levels

This specification provides two levels of conformance for implementation. The implementation level chosen shall be documented in the conformance document.

4.1.1 Implementation level 1

A conforming implementation claiming level 1 conformance of this specification shall meet all the requirements specified by this specification excluding ISO/IEC 9899/AM1.

4.1.2 Implementation level 2

A conforming implementation claiming level 2 conformance of this specification shall meet all the requirements specified by this specification.

NOTE — As of April 1994, the ISO/IEC 9899/AM1 has not yet been finalized. The implementation level 2 conformance will become applicable after the official publication of the amendment to the ISO/IEC 9899:1990.

4.2 Documentation

A conformance document with the following information shall be available for an implementation claiming conformance to this specification.

The conformance document shall not contain information about extended facilities or capabilities outside the scope of this specification.

The conformance document shall contain a statement that indicates the full name and date of the SPIRIT specification that applies.

The conformance document shall contain a statement that indicates the level of conformance that the implementation claims.

28

The conformance document shall describe the limits values found in the <limits.h> and <float.h> headers, translation limits, stating values, the conditions under which those values may change, and the limits of such variations, if any.

The conformance document shall describe the behavior of the implementation for all implementation-defined features defined in this specification. The conformance document may specify the behavior of the implementation for those features where this specification states that implementation may vary or where features are identified as undefined or unspecified.

No features other than those described in this specification shall be present in the conformance document.

The phrase "shall document" or "shall be documented" in this specification mean that documentation of the feature shall appear in the conformance document, as described previously.

5 Environment

The same as ISO/IEC 9899:1990

5.1 Conceptual models

The same as ISO/IEC 9899:1990

5.1.1 Translation environment

5.1.1.1 Program structure

The same as ISO/IEC 9899:1990

5.1.1.2 Translation phases

The same as ISO/IEC 9899:1990

5.1.1.3 Diagnostics

The same as ISO/IEC 9899:1990

5.1.2 Execution environments

The same as ISO/IEC 9899:1990

5.1.2.1 Freestanding environment

The same as ISO/IEC 9899:1990

5.1.2.2 Hosted environment

The same as ISO/IEC 9899:1990

5.1.2.2.1 Program startup

The same as ISO/IEC 9899:1990

5.1.2.2.2 Program execution

The same as ISO/IEC 9899:1990

5.1.2.2.3 Program termination

The same as ISO/IEC 9899:1990

5.1.2.3 Program execution

The same as ISO/IEC 9899:1990

5.2 Environmental considerations

5.2.1 Character sets

The same as ISO/IEC 9899:1990

5.2.1.1 Trigraph sequences

The same as ISO/IEC 9899:1990

5.2.1.2 Multibyte characters

The same as ISO/IEC 9899:1990

5.2.2 Character display semantics

The same as ISO/IEC 9899:1990

5.2.3 Signals and interrupts

The same as ISO/IEC 9899:1990

5.2.4 Environmental limits

The same as ISO/IEC 9899:1990

5.2.4.1 Translation limits

The same as ISO/IEC 9899:1990

5.2.4.2 Numerical limits

The same as ISO/IEC 9899:1990

5.2.4.2.1 Size of integral type <limits.h>

The same as ISO/IEC 9899:1990

5.2.4.2.2 Characteristics of floating types <float.h>

The technical requirements of this subclause is the same as ISO/IEC 9899:1990 except the implementation limits values shown in below:

The values given in the following lists shall be replaced by implementation-defined expressions that shall be equal or grater in magnitude (absolute value) to those shown, with the same sign:

FLT_RADIX 2

FLT_MANT_DIG
DBL_MANT_DIG
DBL_MAINT_DIG

FLT_DIG 6
DBL_DIG 14
LDBL_DIG 14

30

FLT_MIN_EXP
DBL_MIN_EXP
LDBL_MIN_EXP

FLT_MIN_10_EXP	−37
DBL_MIN_10_EXP	−78
LDBL_MIN_10_EXP	−78

FLT_MAX_EXP
DBL_MAX_EXP
LDBL_MAX_EXP

FLT_MAX_10_EXP	+38
DBL_MAX_10_EXP	+75
LDBL_MAX_10_EXP	+75

The values given on the following list shall be replaced by implementation-defined expressions with values that shall be equal to or greater than those shown:

. LT_MAX	1E+38
DBL_MAX	1E+75
LDBL_MAX	1E+75

The values given in the following list shall be replaced by implementation-defined expressions with the values that shall be equal to or less than those shown:

FLT_EPSILON	1E−5
DBL_EPSILON	1E−13
LDBL_EPSILON	1E−13

FLT_MIN	1E−37
DBL_MIN	1E−78
LDBL_MIN	1E−78

6 Language

The same as ISO/IEC 9899:1990

.1 Lexical elements

The same as ISO/IEC 9899:1990

6.1.1 Keywords

The same as ISO/IEC 9899:1990

6.1.2 Identifiers

Syntax

N/C

Description

N/C

Constraints

N/C

Semantics

N/C

Implementation limits

The implementation shall treat at least first 31 characters of an *internal name* (a macro name or an identifier that does not have external linkage) as significant. Corresponding lowercase and uppercase letters are different. The implementation may further restrict the significance of an *external name* (an identifier that has external linkage) to eight characters and may ignore distinctions of alphabetical case for such names. These limitations on identifiers are all implementation-defined.

Any identifiers that differ in a significant character are different identifiers. If two identifiers differ in a nonsignificant character, the behavior is undefined.

6.1.2.1 Scopes of identifiers

The same as ISO/IEC 9899:1990

6.1.2.2 Linkages of identifiers

The same as ISO/IEC 9899:1990

6.1.2.3 Name spaces of identifiers

The same as ISO/IEC 9899:1990

6.1.2.4 Storage durations of objects

The same as ISO/IEC 9899:1990

6.1.2.5 Types

The same as ISO/IEC 9899:1990

6.1.2.6 Compatible type and composite type

The same as ISO/IEC 9899:1990

6.1.3 Constants

The same as ISO/IEC 9899:1990

6.1.3.1 Floating constants

The same as ISO/IEC 9899:1990

6.1.3.2 Integer constants

The same as ISO/IEC 9899:1990

6.1.3.3 Enumeration constants

The same as ISO/IEC 9899:1990

6.1.3.4 Character constants

The same as ISO/IEC 9899:1990

6.1.4 String literals

The same as ISO/IEC 9899:1990

6.1.5 Operators

The same as ISO/IEC 9899:1990 with ISO/IEC 9899/AM1

6.1.6 Punctuators

The same as ISO/IEC 9899:1990 with ISO/IEC 9899/AM1

6.1.7 Header names

The same as ISO/IEC 9899:1990

6.1.8 Preprocessing numbers

The same as ISO/IEC 9899:1990

6.1.9 Comments

The same as ISO/IEC 9899:1990

6.2 Conversions

The same as ISO/IEC 9899:1990

6.2.1 Arithmetic operands

6.2.1.1 Characters and integers

The same as ISO/IEC 9899:1990

6.2.1.2 Signed and unsigned integers

When a value with integral type is converted to another integral type, if the value can be represented by the new type, its value is unchanged.

When a signed integer is converted to an unsigned integer with equal or grater size, if the value of the signed integer is nonnegative, its value is unchanged. Otherwise: if the unsigned integer has grater size, the signed integer is first prompted to the signed integer corresponding to the unsigned integer; the value is converted to unsigned by adding to it one grater than the largest number that can be represented in the unsigned integer type.

When a value with integral type is demoted to an unsigned integer with smaller size, the result is the nonnegative remainder on division by the number one grater than the largest unsigned number that can be represented in the type with smaller size. When a value with integral type is demoted to a signed integer with smaller size, the least significant bits are preserved, and when an unsigned integer is converted to its corresponding signed integer, all bits are preserved.

6.2.1.3 Floating and integral

The same as ISO/IEC 9899:1990

6.2.1.4 Floating types

The same as ISO/IEC 9899:1990

6.2.1.5 Usual arithmetic conversions

The same as ISO/IEC 9899:1990

6.2.2 Other operands

6.2.2.1 Lvalues and function designators

The same as ISO/IEC 9899:1990

6.2.2.2 void

The same as ISO/IEC 9899:1990

6.2.2.3 Pointers

The same as ISO/IEC 9899:1990

6.3 Expressions

The same as ISO/IEC 9899:1990

6.3.1 Primary expressions

The same as ISO/IEC 9899:1990

6.3.2 Postfix operators

The same as ISO/IEC 9899:1990

6.3.2.1 Array subscripting

The same as ISO/IEC 9899:1990

6.3.2.2 Function calls

The same as ISO/IEC 9899:1990

6.3.2.3 Structure and union members

The same as ISO/IEC 9899:1990

6.3.2.4 Postfix increment and decrement operators

The same as ISO/IEC 9899:1990

6.3.3 Unary operators

The same as ISO/IEC 9899:1990

6.3.3.1 Prefix increment and decrement operators

The same as ISO/IEC 9899:1990

6.3.3.2 Address and indirection operators

The same as ISO/IEC 9899:1990

6.3.3.3 Unary arithmetic operators

The same as ISO/IEC 9899:1990

6.3.3.4 The sizeof operators

The same as ISO/IEC 9899:1990

6.3.4 Cast operators

The same as ISO/IEC 9899:1990

6.3.5 Multiplicative operators

Syntax

N/C

Constraints

N/C

Semantics

The usual arithmetic conversions are performed on the operands.

The result of the binary * operator is the product of the operands.

The result of the / operator is the quotient from the division of the first operand by the second; the result of the % operator is the remainder. In the both operations, if the value of the second operand is zero, the behavior is undefined.

When integers are divided and the division is inexact, if both operands are positive the result of the / operator is the largest integer less than the algebraic quotient and the result of the % operator is positive. If either operand is negative, the result of the / operator is the integer of lesser magnitude that is the nearest to the algebraic quotient. If the quotient a/b is representable, the expression $(a/b) * b + a\%b$ shall equal a.

6.3.6 Additive operators

The same as ISO/IEC 9899:1990

6.3.7 Bitwise shift operators

Syntax

N/C

Constraints

N/C

Semantics

The integral promotion are performed on each of the operands. The type of the result is that of the promoted left operand. If the value of the right operand is negative or is grater than or equal to the width in bits of the promoted left operand, the behavior is undefined.

The result of E1 << E2 is E1 left-shifted E2 bit operations; vacated bits are filled with zeros. If E1 has an unsigned type, the value of the results is E1 multiplied by the quantity, 2 raised to the power E2, reduced modulo ULONG_MAX+1 if E1 has type unsigned long, UINT_MAX+1 otherwise. (The constants ULONG_MAX and UINT_MAX are defined in the header < limits.h >.)

The result of E1 >> E2 is E1 right-shifted E2 bit positions. If E1 has an unsigned type or if E1 has a signed type and a nonnegative value, the value of the result is integral part of the quotient of E1 divided by the quantity, 2 raised to the power E2. If E1 has a signed type and a negative value, the sign is preserved; vacated bits are filled with ones in a two's-complement representation.

6.3.8 Relational operators

The same as ISO/IEC 9899:1990

6.3.9 Equality operators

The same as ISO/IEC 9899:1990

6.3.10 Bitwise AND operator

The same as ISO/IEC 9899:1990

6.3.11 Bitwise exclusive OR operator

The same as ISO/IEC 9899:1990

6.3.12 Bitwise inclusive OR operator

The same as ISO/IEC 9899:1990

6.3.13 Logical AND operator

The same as ISO/IEC 9899:1990

6.3.14 Logical OR operator

The same as ISO/IEC 9899:1990

6.3.15 Conditional operator

The same as ISO/IEC 9899:1990

6.3.16 Assignment operators

The same as ISO/IEC 9899:1990

6.3.16.1 Simple assignment

The same as ISO/IEC 9899:1990

6.3.16.2 Compound assignment

The same as ISO/IEC 9899:1990

6.3.17 Comma operator

The same as ISO/IEC 9899:1990

6.4 Constant expressions

The same as ISO/IEC 9899:1990

6.5 Declarations

The same as ISO/IEC 9899:1990

6.5.1 Storage-class specifiers

The same as ISO/IEC 9899:1990

6.5.2 Type specifiers

The same as ISO/IEC 9899:1990

6.5.2.1 Structure and union specifiers

The same as ISO/IEC 9899:1990

6.5.2.2 Enumeration specifiers

The same as ISO/IEC 9899:1990

.2.3 Tags

The same as ISO/IEC 9899:1990

6.5.3 Type qualifiers

The same as ISO/IEC 9899:1990

6.5.4 Declarators

The same as ISO/IEC 9899:1990

6.5.4.1 Pointer declarators

The same as ISO/IEC 9899:1990

6.5.4.2 Array declarators

The same as ISO/IEC 9899:1990

6.5.4.3 Function declarators (including prototypes)

The same as ISO/IEC 9899:1990

6.5.5 Type names

The same as ISO/IEC 9899:1990

6.5.6 Type definitions

The same as ISO/IEC 9899:1990

6.5.7 Initialization

The same as ISO/IEC 9899:1990

6.6 Statements

The same as ISO/IEC 9899:1990

6.6.1 Labeled statements

The same as ISO/IEC 9899:1990

6.6.2 Compound statement, or block

The same as ISO/IEC 9899:1990

6.6.3 Expression and null statements

The same as ISO/IEC 9899:1990

6.6.4 Selection statements

The same as ISO/IEC 9899:1990

6.6.4.1 The if statement

The same as ISO/IEC 9899:1990

6.6.4.2 The switch statement

The same as ISO/IEC 9899:1990

6.6.5 Iteration statements

The same as ISO/IEC 9899:1990

6.6.5.1 The while statement

The same as ISO/IEC 9899:1990

6.6.5.2 The do statement

The same as ISO/IEC 9899:1990

6.6.5.3 The for statement

The same as ISO/IEC 9899:1990

6.6.6 Jump statements

The same as ISO/IEC 9899:1990

6.6.6.1 The goto statement

The same as ISO/IEC 9899:1990

6.6.6.2 The continue statement

The same as ISO/IEC 9899:1990

6.6.6.3 The break statement

The same as ISO/IEC 9899:1990

6.6.6.4 The return statement

The same as ISO/IEC 9899:1990

6.7 External definitions

The same as ISO/IEC 9899:1990

6.7.1 Function definitions

The same as ISO/IEC 9899:1990

6.7.2 External object definitions

The same as ISO/IEC 9899:1990

6.8 Preprocessing directives

The same as ISO/IEC 9899:1990

6.8.1 Conditional inclusion

The same as ISO/IEC 9899:1990

6.8.2 Source file inclusion

The same as ISO/IEC 9899:1990

6.8.3 Macro replacement

The same as ISO/IEC 9899:1990

6.8.3.1 Argument substitution

The same as ISO/IEC 9899:1990

6.8.3.2 The # operator

The same as ISO/IEC 9899:1990

6.8.3.3 The ## operator

The same as ISO/IEC 9899:1990

6.8.3.4 Rescanning and further replacement

The same as ISO/IEC 9899:1990

6.8.3.5 Scope of macro definitions

The same as ISO/IEC 9899:1990

6.8.4 Line control

The same as ISO/IEC 9899:1990

6.8.5 Error directive

The same as ISO/IEC 9899:1990

6.8.6 Pragma directive

The same as ISO/IEC 9899:1990

6.8.7 Null directive

The same as ISO/IEC 9899:1990

6.8.8 Predefined macro names

The following macros names shall be defined the implementation:

__LINE__ The line number of the current source line (a decimal constant).

__FILE__ The presumed name of the source line (a character string literal).

__DATE__ The date of translation of the source file (a character string literal of the form "Mmm dd yyyy", where the names of the months are the same as those generated by the asctime function, and the first character of dd is a space character if the value is less than 10). If the date of translation is not available, an implementation-defined valid date shall be supplied.

__TIME__ The time of translation of the source file (a character string literal of the form "hh:mm:ss" as in the time generated by the asctime function). If the time of translation is not available, an implementation-defined valid time shall be supplied.

__STDC__ The decimal constant 1, intended to indicate a conforming implementation of ISO/IEC 9899:1990.

__STDC_VERSION__ The decimal constant 199407, intended to indicate an implementation conforming to ISO/IEC 9899/AM1:.... .

The value of the predefined macros (except for __LINE__ and __FILE__) remain constant throughout the translation unit.

None of these macro names, nor the identifier defined, shall be the subject of a #define or a #undef preprocessing directive. All predefined macro names shall being with a leading underscore followed by an uppercase letter or a second underscore.

NOTE — The decimal constant 199407 for __STDC_VERSION__ is subject to be change at the official publication of the ISO/IEC 9899/AM1.

6.9 Future language directions

6.9.1 External names

The same as ISO/IEC 9899:1990

6.9.2 Character escape sequences

The same as ISO/IEC 9899:1990

6.9.3 Storage-class specifiers

The same as ISO/IEC 9899:1990

6.9.4 Function declarators

The same as ISO/IEC 9899:1990

6.9.5 Function definitions

The same as ISO/IEC 9899:1990

6.9.6 Array parameters

The same as ISO/IEC 9899:1990

7 Library

7.1 Introduction

7.1.1 Definitions of terms

The same as ISO/IEC 9899:1990 with ISO/IEC 9899/AM1

7.1.2 Standard headers

The same as ISO/IEC 9899:1990 with ISO/IEC 9899/AM1

7.1.3 Reserved identifiers

The same as ISO/IEC 9899:1990

7.1.4 Errors <errno.h>

The same as ISO/IEC 9899:1990 with ISO/IEC 9899/AM1

7.1.5 Limits <float.h> and <limits.h>

The same as ISO/IEC 9899:1990

7.1.6 Common definitions <stddef.h>

The same as ISO/IEC 9899:1990

7.1.7 Use of library functions

The same as ISO/IEC 9899:1990

.2 Diagnostics <assert.h>

The same as ISO/IEC 9899:1990

7.2.1 Program diagnostics

7.2.1.1 The assert macro

The same as ISO/IEC 9899:1990

7.3 Character handling <ctype.h>

The same as ISO/IEC 9899:1990

7.3.1 Character testing functions

The same as ISO/IEC 9899:1990

7.3.1.1 The isalnum function

The same as ISO/IEC 9899:1990

7.3.1.2 The isalpha function

The same as ISO/IEC 9899:1990

7.3.1.3 The iscntrl function

The same as ISO/IEC 9899:1990

7.3.1.4 The isdigit function

The same as ISO/IEC 9899:1990

7.3.1.5 The isgraph function

The same as ISO/IEC 9899:1990

7.3.1.6 The islower function

The same as ISO/IEC 9899:1990

7.3.1.7 The isprint function

The same as ISO/IEC 9899:1990

7.3.1.8 The ispunct function

The same as ISO/IEC 9899:1990

7.3.1.9 The isspace function

The same as ISO/IEC 9899:1990

7.3.1.10 The isupper function

The same as ISO/IEC 9899:1990

7.3.1.11 The isxdigit function

The same as ISO/IEC 9899:1990

7.3.2 Character case mapping functions

7.3.2.1 The tolower function

The same as ISO/IEC 9899:1990

7.3.2.2 The toupper function

The same as ISO/IEC 9899:1990

7.4 Localization <locale.h>

The same as ISO/IEC 9899:1990

7.4.1 Locale control

7.4.1.1 The setlocale function

The same as ISO/IEC 9899:1990

7.4.2 Numeric formatting convention inquiry

7.4.2.1 The localeconv function

Synopsis

N/C

Description

The localeconv function sets the components of an object with type struct lconv with values appropriate for the formatting of numeric quantities (monetary and otherwise) according to the rules of the current locale.

The members of the structure with type char * are pointers to strings, any of which (except decimal_point) can point to "", to indicate that the value is not available in the current locale or is of zero length. The members with type char are nonnegative numbers, any of which can be CHAR_MAX to indicate that the value is not available in the current locale. The members include the following:

char *decimal_point

The decimal-point character used to format nonmonetary quantities.

char *thousands_sep

The character used to separate groups of digits before the decimal-point character in formatted nonmonetary quantities.

char *grouping

A string whose elements indicate the size of each group of digits in formatted nonmonetary quantities.

char *int_curr_symbol

The international currency symbol applicable to the current locale. The first three characters contain the alphabetic international currency symbol in accordance with those specified in ISO 4217:1987. The forth character (immediately preceding the null character) is the character used to separate the international currency symbol from the monetary quantity.

char *currency_symbol

The local currency symbol applicable to the current locale.

char *mon_decimal_point

The decimal-point used to format monetary quantities.

char *mon_thousands_sep

The separator for groups of digits before the decimal-point in formatted monetary quantities.

char *mon_grouping

A string whose elements indicate the size of each group of digits in formatted monetary quantities.

char *positive_sign

The string used to indicate a nonnegative-valued formatted monetary quantity.

char *negative_sign

The string used to indicate a negative-value formatted monetary quantity.

char int_frac_digits

The number of fractional digits (those after the decimal-point) to be displayed in a internationally formatted monetary quantity.

char frac_digits

The number of fractional digits (those after the decimal-point) to be displayed in a formatted monetary quantity.

char p_cs_precedes

Set to 1 or 0 if the currency_symbol or int_curr_symbol respectively preceeds or succeeds the value for a nonnegative formatted monetary quantity.

char p_sep_by_space

Set to 1 or 0 if the currency_symbol or int_curr_symbol ¿from the value for a nonnegative formatted monetary quantity. Set to 1 if a spece separates the symbol from the value; and set to 2 of a space separates the symbol and the sign string, if adjacent.

char n_cs_precedes

Set to 1 or 0 if the currency_symbol or int_curr_symbol respectively precedes or succeeds the value for a negative formatted monetary quantity.

char n_sep_by_space

Set to 1 or 0 if the currency_symbol or int_curr_symbol ¿from the value for a negative formatted monetary quantity. Set to 1 of a space separates the symbol from the value; and set to 2 of a space separates the symbol and the sign string, if adjacent.

char p_sign_posn

Set to a value indicating the positioning of the positive_sign for a nonnegative formatted monetary quantity.

char n_sign_posn

Set to a value indicating the positioning of the negative_sign for a negative formatted monetary quantity.

The elements of grouping and mon_grouping are interpreted according to the following:

CHAR_MAX No further grouping is to be performed.
0 The previous element is to be repeatedly used for the remainder of the digits.
other The integer value is the number of digits that comprise the current group. The next element is examined to determine the size of the next group of digits before the current group.

The value of p_sign_posn and n_sign_posn is interpreted according to the following:

0 Parentheses surround the quantity and currency_symbol or int_curr_symbol.
1 The sign string precedes the quantity and currency_symbol or int_curr_symbol.
2 The sign string succeeds the quantity and currency_symbol or int_curr_symbol.
3 The sign string immediately precedes the currency_symbol or int_curr_symbol.
4 The sign string immediately succeeds the currency_symbol or int_curr_symbol.

Returns

N/C

Example

N/C

7.5 Mathematics <math.h>

The same as ISO/IEC 9899:1990

44

7.5.1 Treatment of error conditions

The same as ISO/IEC 9899:1990

7.5.2 Trigonometric functions

7.5.2.1 The acos function

The same as ISO/IEC 9899:1990

7.5.2.2 The asin function

The same as ISO/IEC 9899:1990

7.5.2.3 The atan function

The same as ISO/IEC 9899:1990

7.5.2.4 The atan2 function

The same as ISO/IEC 9899:1990

7.5.2.5 The cos function

The same as ISO/IEC 9899:1990

7.5.2.6 The sin function

The same as ISO/IEC 9899:1990

7.5.2.7 The tan function

The same as ISO/IEC 9899:1990

7.5.3 Hyperbolic functions

.5.3.1 The cosh function

The same as ISO/IEC 9899:1990

7.5.3.2 The sinh function

The same as ISO/IEC 9899:1990

7.5.3.3 The tanh function

The same as ISO/IEC 9899:1990

7.5.4 Exponential and logarithmic functions

7.5.4.1 The exp function

The same as ISO/IEC 9899:1990

7.5.4.2 The frexp function

The same as ISO/IEC 9899:1990

7.5.4.3 The ldexp function

The same as ISO/IEC 9899:1990

7.5.4.4 The log function

The same as ISO/IEC 9899:1990

7.5.4.5 log10 function

The same as ISO/IEC 9899:1990

7.5.4.6 the modf function

The same as ISO/IEC 9899:1990

7.5.5 Power functions

7.5.5.1 The pow function

The same as ISO/IEC 9899:1990

7.5.5.2 The sqrt function

The same as ISO/IEC 9899:1990

7.5.6 Nearest integer, absolute value, and remainder functions

7.5.6.1 The ceil function

The same as ISO/IEC 9899:1990

7.5.6.2 The fabs function

The same as ISO/IEC 9899:1990

7.5.6.3 The floor function

The same as ISO/IEC 9899:1990

7.5.6.4 The fmod function

Synopsis

N/C

Description

N/C

Returns

The fmod function returns the value $x - i * y$, for some integer i such that, if y is nonzero, the result has the same sign as x and magnitude less than the magnitude of y. If y is zero, fmod function returns zero and errno may be set to EDOM.

7.6 Nonlocal jumps <setjump.h>

The same as ISO/IEC 9899:1990

7.6.1 Save calling environment

7.6.1.1 The setjump macro

The same as ISO/IEC 9899:1990

7.6.2 Restore calling environment

7.6.2.1 The longjmp function

The same as ISO/IEC 9899:1990

7.7 Signal handling <signal.h>

The same as ISO/IEC 9899:1990

7.7.1 Specify signal handling

..7.1.1 The signal function

The same as ISO/IEC 9899:1990

7.7.2 Send signal

7.7.2.1 The raise function

The same as ISO/IEC 9899:1990

7.8 Variable arguments <stdarg.h>

The same as ISO/IEC 9899:1990

7.8.1 Variable argument list access macros

The same as ISO/IEC 9899:1990

.8.1.1 The va_start macro

The same as ISO/IEC 9899:1990

7.8.1.2 The va_arg macro

The same as ISO/IEC 9899:1990

7.8.1.3 The va_end macro

The same as ISO/IEC 9899:1990

7.9 Input/output <stdio.h>

7.9.1 Introduction

The same as ISO/IEC 9899:1990 with ISO/IEC 9899/AM1

7.9.2 Streams

The same as ISO/IEC 9899:1990 with ISO/IEC 9899/AM1

7.9.3 Files

The same as ISO/IEC 9899:1990 with ISO/IEC 9899/AM1

7.9.4 Operations on files

7.9.4.1 The remove function

The same as ISO/IEC 9899:1990

7.9.4.2 The rename function

The same as ISO/IEC 9899:1990

7.9.4.3 The tmpfile function

The same as ISO/IEC 9899:1990

7.9.4.4 The tmpnam function

The same as ISO/IEC 9899:1990

7.9.5 File access function

7.9.5.1 The fclose function

The same as ISO/IEC 9899:1990

7.9.5.2 The fflush function

The same as ISO/IEC 9899:1990

7.9.5.3 The fopen function

The same as ISO/IEC 9899:1990

7.9.5.4 The freopen function

The same as ISO/IEC 9899:1990

7.9.5.5 The setbuf function

The same as ISO/IEC 9899:1990

7.9.5.6 The setvbuf function

The same as ISO/IEC 9899:1990

7.9.6 Formatted input/output functions

7.9.6.1 The fprintf function

Synopsis

N/C

Description

48

The fprintf function writes output to the stream pointed to by stream, under control of the string pointed to bt format that specifies how subsequent arguments are covered for output. If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated (as always) but are otherwise ignored. The fprintf function returns when the end of the format string is encountered.

The format shall be a multibyte character sequence, beginning and ending in its initial shift state. The format is composed of zero or more directives: ordinary multibyte character (not %), which are copied unchanged to the output stream; and conversion specifications, each of which results in fetching zero or more subsequent arguments.

Conversions can be applied to the nthe argument after the format in the argument list, rather than to the next unused argument. In this case, the conversion character % (see below) is replaced by the sequence %n$, where n is a decimal integer in the range [1, {NL_ARGMAX}], giving the position of the argument in the argument list. This feature provides for the definition of format strings that select arguments in an order appropriate to specific languages.

In format strings containing the %n$ form of conversion specifications, numbered arguments in the argument list can be referenced ¿from the format string as many times as required.

Each conversion specification is introduced be the character %. After the %, the following appear in sequence:

— Zero or more *flags* (in any other) that modify the meaning of the conversion specification.

An optional minimum *field width*. If the converted value has fewer characters than the field width, it will be padded with spaces (by default) on the left (or right, if the left adjustment flag, described later, has been given) to the field width. The field width takes the form of an asterisk * (described later) or a decimal integer.

— An optional *precision* that gives the minimum number of digits to appear for the d, i, o, u, x, and X conversions, the numebr of digits to appear after the decimal-point character for e. E, and f conversions, the maximum number of significant digits for the g and G conversions, or the maximum number of characters to be written from a string in s conversion. The precision takes the form of a period (.) followed either by an asterisk * (described later) or by an optional decimal integer; if only the period is specified, the precision is taken as zero. If a precision appears with any other conversion specifier, the behavior is undefined.

— An optional h specifying that a following d, i, o, u, x, or X conversion specifier applies to a short int or unsigned short int argument (The argument will have been promoted according to the integral promotions, and its value shall be converted to short int or unsigned short int before printing); an optional h specifying that a following n conversion specifier applies to a pointer to a short int argument; an optional l (ell) specifying that a following d, i, o, x, or X conversion specifier applies to a long int or unsigned long int argument; an optional l specifying that a following n conversion specifier applies to a pointer to a long int argument; an optional l specifying that a following c conversion specifier applies to a wint_t argument; an optional l specifying that a following s conversion specifier applies to a pointer to a wchar_t argument; or an optional L specifying that a following e, E, f, g, or G conversion specifier applies to a long double argument, if an h, l, or L appears with any other conversion specifier, the behavior is undefined.

— A character that specifies the type of conversion to be applied.

As noted above, a field width, or precision, or both may be indicated by an asterisk. In this case, an int argument supplies ie field width or precision. The arguments specifying field width, or precision, or both, shall appear (in that order) before the argument (if any) to be converted. A negative field width argument is taken as a – flag followed by a positive field width. A negative precision argument is taken as if the precision were omitted. In format strings containing the %n$ form of a conversion specification, a field width or precision may be indicated by the sequence *m$, where m is a decimal integer in the range [1, {NL_ARGMAX}] giving the position in the argument list (after the format argument) of an integer argument containing the field width or precision, for example:

```
printf ("%1$d:%2$.*3$d:%4$.*3$d \n",
        hour, min, precision, sec );
```

The format can contain either numbered argument specifications (that is, %n$ and *m$), or numbered argument specifications (that is, % and *), but normally not both. The only exception to this is that %% can be mixed with the %m$ form. The results of mixing numbered and unnumbered argument specifications in a format string are undefined. When numbered argument specifications are used, specifying the Nth argument requires that all the leasing arguments, from the first to the (N–1)th, are specified in the format string.

The flag characters and their meanings are

' The integer portion of the result of a decimal conversion (%1, %d, %u, %f, %g or %G) will be formatted with thousands' grouping characters. For other conversions the behavior is undefined. The non-monetary grouping character is used.

– The result if the conversion will be left-justified within the field. (It will be right-justified if this flag is not specified.)

+ The result of a signed conversion will always begin with a plus or minus sign. (It will begin with a sign only when a negative value is converted if this flag is not specified.)

space If the first character of a signed conversion is not a sign, or of a signed conversion results in no characters, a space will be prefixed to the result. If the *space* and + flags both appear, the *space* flag will be ignored.

The result is to be converted an "alternate form." For o conversion, it increases the precision to force the first digit of the result to be a zero, FOr x (or X) conversion, a nonzero result will have 0x (or 0X prefixed to it. For e, E, f, g, and G conversions, the result will always contain a decimal-point characters, even if no digits follow it. (Normally, a decimal-point character appears in the result of these conversions only if a digit follows ot.) For g and G conversions, trailing zeros will *not* be removed from the results. For other conversions, the behavior is undefined.

0 For d, i, o, u, x, X, e, E, f, g, and G conversions, leading zeros (following any indication of sign or base) are used to pad the field width; no space padding is performed. If the 0 and – flags both appear, the 0 flag will be ignored. For d, i, o, u, x, and X conversions, if a precision is specified, the 0 flag will be ignored. For other conversions, the behavior is undefined.

The conversion specifiers and their meanings are

d,i The int argument is converted to signed decimal in the style *[–]dddd*. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result if converting a zero value with a precision of zero is no character.

o,u,x,X The unsigned int argument is converted to unsigned octal (o), unsigned decimal (u), or unsigned hexadecimal notation (x or X) in the style *dddd*; the letters abcdef are used for x conversion and the letters ABCDEF for X conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is no characters.

f The double argument is converted to decimal notation in the style *[–]ddd.ddd*, where the number of digits after the decimal-point character is equal to the precision specification. If the precision is missing, it is taken as 6; if the precision is zero and the # flag is not specified, no decimal-point character appears. If a decimal-point character appears, at least one digit appears before it. The value is rounded to appropriate number of digits.

e,E The double argument is converted the style *[–]d.dde±dd*, where there is one digit before the decimal-point character (which is nonzero if the argument is nonzero) and the number of digits after it is equal to the precision; if the precision is missing, it is taken as 6; if the precision is zero and the # flag is not specified, no decimal-point character appears. The value is rounded to the appropriate number of digits. The E conversion specifier will produce a number with E instead e introducing the exponent. The exponent always contains at least two digits. If the value is zero, the exponent is zero.

g,G The double argument is converted in style f or e (or in style E in the case of a G conversion specifier), with the precision specifying the number of significant digits. If the precision is zero, it is taken as 1. The style used depends on the value converted; style e (or E) will be used only if the exponent resulting from such a conversion is less than –4 or grater than or equal to the precision. Trailing zeros are removed from the fractional portion of the result; a decimal-point character appears only if it is followed by a digit.

c If no l qualifier is present, the int argument is converted to an unsigned char, and the resulting character is written. Otherwise, the wint_t argument is converted as if by an ls conversion specification with no precision and an argument that points to a two-element array of wchar_t, the first element containing wint_t argument to the lc conversion specification and the second a null wide character.

s If no l qualifier is present, the argument shall be a pointer to an array of character type. Characters from the array are written up to (but not including) a terminating null character. If the precision is specified, no more than that many characters are written. If the precision is not specified or is grater than the size of the array, the array shall contain a null character.

If an l qualifier is present, the argument shall be a pointer to an array of wchar_t type. Wide characters from the array are converted to multibyte characters (each as if by a call to the wcrtomb function, with the conversion state described by an mbstate_t object initialized to zero before the first wide character is converted) up to and including a terminating null wide character. The resulting multibyte characters are written up to (but not including) the terminating null character (byte). If no precision is specified, the array shall contain a null wide character. If a precision is specified, no more than that many characters (bytes) are written (including shift sequences, if any), and the array shall contain a null wide character if, to equal the multibyte character sequence length given by the precision, the function would need to access a wide character one past the end of the array. In no case is partial multibyte character written.

P The argument shall be a pointer to void. The value of the pointer is converted to a sequence of printable characters, in an implementation-defined manner.

n The argument shall be a pointer to an integer into which is written the number of characters written to the output stream so far by this call to fprintf. No argument is converted.

% A % is written. No argument is converted. The complete conversion specification shall be %%.

If a conversion specification is invalid, the behavior is undefined.

If any argument is, or points to, a union or an aggregate (except for an array of character type using %s conversion, or a pointer using %p conversion), the behavior is undefined.

In no case does a nonexistent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is expanded to contain the conversion result.

Returns

N/C

Example

N/C

7.9.6.2 The fscanf function

Synopsis

N/C

Description

`e fsacnf function reads input from the stream pointed to by stream, under control of the string pointed to by format .at specifies the admissible input sequences and how they are to be converted for assignment, using subsequent arguments as pointed to the objects to receive the converted input. If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated (as always) but are otherwise ignored.

Conversions can be applied to the nth argument after the format in the argument list, rather than to the next unused argument. In this case, the conversion character % (see below) is replaced by the sequence %n$, where n is a decimal integer in the range [1,{NL_ARGMAX}]. This feature provided for the definition of format string that select arguments in an order appropriate to specific languages. In format strings containing the %n$ form of conversion specifications, it is unspecified whether numbered arguments in the argument list can be referenced from the format string more than once.

The format can contain either form of a conversion specification, that is, % or %n$, but the two forms cannot normally be mixed within a single format string. The only exception to this is that %% or %* can be mixed with the %n$ form.

The formant shall be a multibyte character sequence, beginning and ending in its initial shift state. The format is composed of zero or more directives: one or more white-space characters; an ordinary multibyte character (neither % nor a white-space character); or a conversion specification. Each conversion specification is introduced by the character %. After the %, the following appear in sequence:

— An optional assignment-suppressing character *.

— An optional nonzero decimal integer that specifies the minimum field width.

— An optional h, l (ell) or L indicating the size of the receiving object. The conversion specifiers d, i, and n shall be preceded by h if the corresponding argument is a pointer to short int rather than a pointer to int, or by l if it is a pointer to long int. Similarly, the conversion specifiers o, u, and x shall be preceded by h if the corresponding argument is a pointer to unsigned short int rather than a pointer to unsigned int, or by l if it is a pointer to unsigned long int. The conversion specifiers c, s, and [shall be preceded by l if the corresponding argument is a pointer to wchar_t rather than a pointer to a character type. Finally, the conversion specifiers e, f, and g shall be preceded by l if the corresponding argument is a pointer to double rather than a pointer to a float, or by L if it is a pointer to long double. If an h, l, or L appears with any other conversion specifier, the behavior is undefined.

— A character that specifies the type of conversion to be applied. The valid conversion specifiers are described below.

The fscanf function executes each directive of the format in turn. If a directive fails, as detailed below, the fscanf function returns. failures are described as input failures (due to the unavailability of input characters), or matching failures (due to inappropriate input).

A directive composed of white-space character(s) is executed by reading input up to the first non-white-space character (which remains unread), or until no more characters can be read.

A directive that is an ordinary multibyte character is executed by reading the next characters of the stream. If one of the characters differs from one composing the directive, the directive fails, and the differing and subsequent characters remain unread.

A directive that is a conversion specification defines a set of matching input sequences, as described below for each specifier. A conversion specification is executed in the following steps:

Input white-space characters (as specified by the isspace function) are skipped, unless the specification includes a [, c, or n specifier.

An input item is read from the stream, unless the specification includes an n specifier. An input item is defined as the longest matching sequence of input characters, unless that exceeds a specified field width, in which case it is the initial subsequence of that length in the sequence. The first character, if any, after the input item remains unread. If the length of the input item is zero, the execution of the directive fails; this condition is matching failure unless end-of-file, an encoding error, or a read error prevented input from the stream, in which case it is an input failure. Except in the case of a % specifier, the input item (or, in the case of a %n directive, the count of input characters) is converted to a type appropriate to the conversion specifier. If the input item is not a matching sequence, the execution of the directive fails: this condition is matching failure. Unless assignment suppression was indicated by a *, the result of the conversion is placed in the object pointed to by the first argument following the format argument that has not already received a conversion result if the conversion specification is introduced by %, or in the nth argument if introduced by the character sequence %n$. If the object does not have an appropriate type, or if the result of the conversion cannot be represented in the space provided, the behavior is undefined.

The following conversion specifiers are valid:

d Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of the strtol function with the value 10 for the base argument. The corresponding argument shall be a pointer to integer.

i Matches an optionally signed integer, whose format is the same as expected for the subject sequence of the strtol function with the value 0 for the base argument. The corresponding argument shall be a pointer to integer.

o Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of the strtol function with the value 8 for the base argument. The corresponding argument shall be a pointer to integer.

u Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of the strtol function with the value 10 for the base argument. The corresponding argument shall be a pointer to unsigned integer.

x Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of the strtol function with the value 16 for the base argument. The corresponding argument shall be a pointer to unsigned integer.

e,f,g Matches an optionally signed floating-point number, whose format is the same as expected for the subject string of the strtod function. The corresponding argument shall be a pointer to floating.

s Matches a sequence of non-white-space characters. If no l qualifier is present, the corresponding argument shall be a pointer to a character array large enough to accept the sequence and terminating null character, which will be added automatically. If an l qualifier is present, the input shall be a sequence of multibyte characters that begins in the initial shift state. Each multibyte character is converted to a wide character as if by a call to the mbrtowc function, with the conversion state described by an mbstate_t object initialized to zero before the first multibyte character is converted. The corresponding argument shall be a pointer to an array of wchar_t large enough to accept the sequence and the terminating null wide character, which will be added automatically.

[Matches a nonempty sequence of characters from a set of expected characters (the *scanset*). If no l qualifier is present, the corresponding argument shall be a pointer to a character array large enough to accept the sequence and a terminating null character, which will be added automatically. If an l qualifier is present, the input shall be a sequence of multibyte characters that begins in the initial shift state. Each multibyte character is converted to a wide character as of by a call to the mbrtowc function, with the conversion state described by an mbstate_t object initialized to zero before the first multibyte character is converted. The corresponding argument shall be a pointer to an array of wchar_t large enough to accept the sequence and terminating null wide character, which will be added automatically. The conversion specifier includes all subsequent characters in the format string up to and including the matching right bracket]. The characters between the brackets is a circumflex (^), in which case the scanset contains all characters that do not appear in the scanlist between the circumflex and the right bracket. If the conversion specifier begins with [] or [^], the right bracket character is in the scanlist and the next bracket character is the matching right bracket that ends the specification; otherwise the first bracket character is the one that ends the specification. If a – character is in the scanlist and is not the first, nor the second where the first character is a ^, nor the last character, the behavior is implementation-defined.

c Matches a sequence of characters of the number specified by the field width (1 if no field width is present in the directive). If no l qualifier is present, the corresponding argument shall be a pointer to a character array large enough to accept the sequence. No null character is added. If an l qualifier is present, the input shall be a sequence of multibyte character that begins in the initial shift state. Each multibyte character in the sequence is converted to a wide character as of by a call to the mbrtowc function, with the conversion state described by an mbstate_t object initialized to zero before the first multibyte character is converted. The corresponding argument shall be a pointer to the initial element of an array of wchar_t large enough to accept the reslting sequence of wide characters. No null wide character is added.

p Matches an implementation-defined set of sequences, which should be the same as the set of sequences that may be produced by the %p conversion of the fprintf function. The corresponding argument shall be a pointer to a pointer to void. The interpretation of the input item is implementation-defined. If the input item is a value converted earlier during the same program execution, the pointer that results shall compare equal to that value; otherwise the behavior of the %p conversion is undefined.

52

n No input is consumed. The corresponding argument shall be a pointer to integer into which is to be written the number of characters read from the input stream so far by this call to the fscanf function. Execution of a %n directive does not increment the assignment count returned at the completion of execution of the fscanf function.

% Matches a single %; no conversion or assignment occurs. The complete conversion specification shall be %%.

If a conversion specification is invalid, the behavior is undefined.

The conversion specifiers E, G, and X are also valid and behave the same as, respectively, e, g, and x.

If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before any characters matching the current directive have been read (other than leading white space, where permitted), execution of the current directive terminates with an input failure; otherwise, unless execution of the current directive is terminated with a matching failure, execution of the following directive (if any) is terminated with an input failure.

If conversion terminates on a conflicting input character, the offending input character is left unread in the input stream. trailing white space (including new-line characters) is left unread unless matched by a directive. The success of literal matches and suppressed assignments is not directly determinable other than via the %n directive.

Returns

N/C

Examples

N/C

7.9.6.3 The printf function

In addtion to the requirements specified by ISO/IEC 9899:1990 with ISO/IEC 9899/AM1, the same modification on conversion specification with fprintf function is applied.

7.9.6.4 The scanf function

In addtion to the requirements specified by ISO/IEC 9899:1990 with ISO/IEC 9899/AM1, the same modification on conversion specification with fscanf function is applied.

7.9.6.5 The sprintf function

In addtion to the requirements specified by ISO/IEC 9899:1990 with ISO/IEC 9899/AM1, the same modification on conversion specification with fprintf function is applied.

7.9.6.6 The sscanf function

In addtion to the requirements specified by ISO/IEC 9899:1990 with ISO/IEC 9899/AM1, the same modification on conversion specification with fscanf function is applied.

7.9.6.7 The vfprintf function

In addtion to the requirements specified by ISO/IEC 9899:1990 with ISO/IEC 9899/AM1, the same modification on conversion specification with fprintf function is applied.

7.9.6.8 The vprintf function

In addition to the requirements specified by ISO/IEC 9899:1990 with ISO/IEC 9899/AM1, the same modification on conversion specification with fprintf function is applied.

7.9.7 Character input/output functions

7.9.7.1 The fgetc function

The same as ISO/IEC 9899:1990

7.9.7.2 The fgets function

The same as ISO/IEC 9899:1990

7.9.7.3 The fputc function

The same as ISO/IEC 9899:1990

7.9.7.4 The fputs function

The same as ISO/IEC 9899:1990

7.9.7.5 The getc function

The same as ISO/IEC 9899:1990

7.9.7.6 The getchar function

The same as ISO/IEC 9899:1990

7.9.7.7 The gets function

The same as ISO/IEC 9899:1990

7.9.7.8 The putc function

The same as ISO/IEC 9899:1990

7.9.7.9 The putchar function

The same as ISO/IEC 9899:1990

7.9.7.10 The puts function

The same as ISO/IEC 9899:1990

7.9.7.11 The ungetc function

The same as ISO/IEC 9899:1990

7.9.8 Direct input/output functions

7.9.8.1 The fread function

The same as ISO/IEC 9899:1990

7.9.8.2 The fwrite function

The same as ISO/IEC 9899:1990

7.9.8.3 The fgetpos function

The same as ISO/IEC 9899:1990

7.9.8.4 The fseek function

The same as ISO/IEC 9899:1990

7.9.8.5 The fsetpos function

The same as ISO/IEC 9899:1990

7.9.8.6 The ftell function

The same as ISO/IEC 9899:1990

7.9.8.7 The rewind function

The same as ISO/IEC 9899:1990

7.9.9 Error-handling functions

7.9.9.1 The clearerr function

The same as ISO/IEC 9899:1990

7.9.9.2 The feof function

The same as ISO/IEC 9899:1990

7.9.9.3 The ferror function

The same as ISO/IEC 9899:1990

7.9.9.4 The perror function

The same as ISO/IEC 9899:1990

7.10 General utilities <stdlib.h>

The same as ISO/IEC 9899:1990

7.10.1 String conversion functions

The same as ISO/IEC 9899:1990

7.10.1.1 The atof function

The same as ISO/IEC 9899:1990

7.10.1.2 The atoi function

The same as ISO/IEC 9899:1990

7.10.1.3 The atol function

The same as ISO/IEC 9899:1990

7.10.1.4 The strtod function

The same as ISO/IEC 9899:1990

7.10.1.5 The strtol function

The same as ISO/IEC 9899:1990

7.10.1.6 The strtoul function

The same as ISO/IEC 9899:1990

7.10.2 Pseudo-random sequence generation functions

7.10.2.1 The rand function

Synopsis

N/C

Description

The rand function computes a sequence of pseudo-random integers in the range 0 to RAND_MAX, with a period at least 2^{32}.

Returns

N/C

Environmental limit

N/C

7.10.2.2 the srand function

The same as ISO/IEC 9899:1990

7.10.3 Memory management functions

The same as ISO/IEC 9899:1990

7.10.3.1 The calloc function

The same as ISO/IEC 9899:1990

7.10.3.2 The free function

The same as ISO/IEC 9899:1990

7.10.3.3 The malloc function

The same as ISO/IEC 9899:1990

7.10.3.4 The realloc function

The same as ISO/IEC 9899:1990

7.10.4 Communication with The environment

7.10.4.1 The abort function

The same as ISO/IEC 9899:1990

7.10.4.2 The atexit function

The same as ISO/IEC 9899:1990

7.10.4.3 The exit function

The same as ISO/IEC 9899:1990

7.10.4.4 The getenv function

The same as ISO/IEC 9899:1990

7.10.4.5 The system function

The same as ISO/IEC 9899:1990

7.10.5 Searching and sorting utilities

7.10.5.1 The bsearch function

The same as ISO/IEC 9899:1990

7.10.5.2 The qsort function

The same as ISO/IEC 9899:1990

7.10.6 Integer arithmetic functions

7.10.6.1 The abs function

The same as ISO/IEC 9899:1990

7.10.6.2 The div function

The same as ISO/IEC 9899:1990

7.10.6.3 The labs function

The same as ISO/IEC 9899:1990

7.10.6.4 The ldiv function

The same as ISO/IEC 9899:1990

7.10.7 Multibyte character functions

The same as ISO/IEC 9899:1990

7.10.7.1 The mblen function

The same as ISO/IEC 9899:1990

7.10.7.2 The mbtowc function

The same as ISO/IEC 9899:1990

7.10.7.3 The wctomb function

The same as ISO/IEC 9899:1990

7.10.7.4 The mbstowcs function

The same as ISO/IEC 9899:1990

7.10.7.5 The wcstombs function

The same as ISO/IEC 9899:1990

7.11 String handling <string.h>

7.11.1 String function conventions

The same as ISO/IEC 9899:1990

7.11.2 Copying functions

7.11.2.1 The memcpy function

The same as ISO/IEC 9899:1990

7.11.2.2 The memmove function

The same as ISO/IEC 9899:1990

7.11.2.3 The strcpy function

The same as ISO/IEC 9899:1990

7.11.2.4 The strncpy function

The same as ISO/IEC 9899:1990

7.11.3 Concatenation functions

7.11.3.1 The strcat function

The same as ISO/IEC 9899:1990

7.11.3.2 The strncat function

The same as ISO/IEC 9899:1990

7.11.4 Comparison functions

The same as ISO/IEC 9899:1990

7.11.4.1 The memcmp function

The same as ISO/IEC 9899:1990

7.11.4.2 The strcmp function

The same as ISO/IEC 9899:1990

7.11.4.3 The strcoll function

The same as ISO/IEC 9899:1990

7.11.4.4 The strncmp function

The same as ISO/IEC 9899:1990

7.11.4.5 The strxfrm function

The same as ISO/IEC 9899:1990

7.11.5 Search functions

7.11.5.1 The memchr function

The same as ISO/IEC 9899:1990

7.11.5.2 The strchr function

The same as ISO/IEC 9899:1990

7.11.5.3 The strcspn function

The same as ISO/IEC 9899:1990

7.11.5.4 The strpbrk function

The same as ISO/IEC 9899:1990

7.11.5.5 The strrchr function

The same as ISO/IEC 9899:1990

7.11.5.6 the strspn function

The same as ISO/IEC 9899:1990

7.11.5.7 The strstr function

The same as ISO/IEC 9899:1990

7.11.5.8 The strtok function

The same as ISO/IEC 9899:1990

7.11.6 Miscellaneous functions

7.11.6.1 The memset function

The same as ISO/IEC 9899:1990

7.11.6.2 The strerror function

The same as ISO/IEC 9899:1990

7.11.6.3 The strlen function

The same as ISO/IEC 9899:1990

7.12 Date and time <time.h>

7.12.1 Components of time

The same as ISO/IEC 9899:1990

7.12.2 Time manipulation functions

7.12.2.1 The clock function

The same as ISO/IEC 9899:1990

7.12.2.2 The difftime function

The same as ISO/IEC 9899:1990

7.12.2.3 The mktime function

The same as ISO/IEC 9899:1990

7.12.2.4 The time function

The same as ISO/IEC 9899:1990

7.12.3 Time conversion functions

The same as ISO/IEC 9899:1990

7.12.3.1 The asctime function

The same as ISO/IEC 9899:1990

7.12.3.2 The ctime function

The same as ISO/IEC 9899:1990

7.12.3.3 The gmtime function

The same as ISO/IEC 9899:1990

7.12.3.4 The localtime function

The same as ISO/IEC 9899:1990

7.12.3.5 The strftime function

Synopsis

N/C

Description

The **strtime** function places characters into the array pointed to be **s** as controlled by the string pointed to by **format**. The format shall be a multibyte character sequence, beginning and ending in its initial shift state. The **format** string consists of zero or more conversion specifiers and ordinary multibyte characters. A conversion specifier consists of a % character followed by a character that determines the behavior of the conversion specifier. All ordinary multibyte characters (including the terminating null character) are copies unchanged into the array. If copying takes place between objects that overlap, the behavior is undefined. No more than **maxsize** characters are placed into the array. Each conversion specifier is replaced by appropriate characters as described in the following list. The appropriate characters are determined by the **LC_TIME** category of the current locale and by the value contained in the structure pointed to by **timeptr**.

%a is replaced by the locale's abbreviated weekday name.

%A is replaced by the locale's full weekday name.

%b is replaced by the locale's abbreviated month name.

%B is replaced by the locale's full month name.

%c is replaced by the locale's appropriate date and time representation.

%C is replaced by the century number (the year divided by 100 and truncated to an integer) as a decimal number [00-99].

%d is replaced by the day of the month as a decimal number [01,31].

%D same as %m/%d/%y.

%e is replaced by the day of the month as a decimal number [1,31]; a single digit is replaced by a space.

%h same as %b.

%H is replaced by the hour (24-hour clock) as a decimal number [00,23].

%I is replaced by the hour (12-hour clock) as a decimal number [01,12].

%j is replaced by the day of the year as a decimal number [001,366].

n is replaced by the month as a decimal number [01,12].

%M is replaced by the minutes as a decimal number.

%n is replaced by a newline character.

%p is replaced by the locale's equivalent of either a.m. or p.m.

%r is replaced by the time in a.m. and p.m. notation; on the C locale this is equivalent to %I:%M:%S %p.

%R is replaced by the time in 24 hour notation (%H:%M).

%S is replaced by the second as a decimal number [00,61].

%t is replaced by a tab character.

%T is replaced by the time (%H:%M:%S),

%u is replaced by the weekday as a decimal number [1,7], with 1 representing Monday.

%U is replaced by the week number of the year (Sunday as the first day of the week) as a decimal number [00,53].

%V is replaced by the week number of the year (Monday as the first day of the week) as a decimal number [01,53]. If the week containing 1 January has four or more days in the new year, then it is considered week 1; otherwise, it is week 53 of the previous year, and the next week is week 1.

%w is replaced by the weekday as a decimal number [0,6], with 0 representing Sunday.

%W is replaced by the week number of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Sunday are considered to be week 0.

%x is replaced by the locale's appropriate date representation.

%X is replaced by the locale's appropriate time representation.

%y is replaced by the year without century as a decimal number [00,99].

%Y is replaced by the year with century as a decimal number.

%Z is replaced by the timezone name or abbreviation, or by no bytes if no timezone information exists.

%% is replaced by %.

If a conversion specification does not correspond to any of the above, the behavior is undefined.

Modified Conversion Specifiers

Some conversion specifiers cam be modified by the E or O modifier characters to indicate that an alternative format or specification should be used rather than one normally used by the unmodified conversion specifier. If the alternative format or specification does not exist for the current locale, the behavior will be as if the unmodified conversion specification were used.

%Ec is replaced by the locale's alternative appropriate date and time representation.

%EC is replaced by the name of the base year (period) in the locale's alternative representation.

%Ex is replaced by the locale's alternative date representation.

%EX is replaced by the locale's alternative time representation.

%Ey is replaced by the offset from %EC (year only) in the locale's alternative representation.

%EY is replaced by the full alternative year representation.

%Od is replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading zeros if there is any alternative symbol for zero, otherwise with leading spaces.

%Oe is replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading spaces.

%OH is replaced by the hour (24-hour clock) using the locale's alternative numeric symbols.

%OI is replaced by the hour (12-hour clock) using the locale's alternative numeric symbols.

%Om is replaced by the month using the locale's alternative numeric symbols.

%OM is replaced by the minutes using the locale's alternative numeric symbols.

%OS is replaced by the seconds using the locale's alternative numeric symbols.

%Ou is replaced by the weekday as a number in the locale's alternative representation (Monday = 1).

%OU is replaced by the week number of the year (Sunday as the first day of the week, rules corresponding to %U) using the locale's alternative numeric symbols.

%OV is replaced by the week number of the year (Sunday as the first day of the week, rules corresponding to %V) using the locale's alternative numeric symbols.

%Ow is replaced by the number of the weekday (Sunday = 0) using the locale's alternative numeric symbols.

%OW is replaced by the week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.

%Oy is replaced by the year (offset from %C) in the locale's alternative representation and using the locale's alternative symbols.

Returns

N/C

7.13 Alternative spelling <iso646.h>

The same as ISO/IEC 9899/AM1

7.14 Wide-character classification and mapping utilities <wctype.h>

The same as ISO/IEC 9899/AM1

7.14.1 Wide-character classification utilities

The same as ISO/IEC 9899/AM1

7.14.1.1 Wide-character classification functions

The same as ISO/IEC 9899/AM1

7.14.1.1.1 The iswalnum function

The same as ISO/IEC 9899/AM1

7.14.1.1.2 The iswalpha function

The same as ISO/IEC 9899/AM1

7.14.1.1.3 The iswcntrl function

The same as ISO/IEC 9899/AM1

62

7.14.1.1.4 The iswdigit function

The same as ISO/IEC 9899/AM1

7.14.1.1.5 The iswgraph function

The same as ISO/IEC 9899/AM1

7.14.1.1.6 The iswlower function

The same as ISO/IEC 9899/AM1

7.14.1.1.7 The iswprint function

The same as ISO/IEC 9899/AM1

7.14.1.1.8 The iswpunct function

The same as ISO/IEC 9899/AM1

7.14.1.1.9 The iswspace functions

The same as ISO/IEC 9899/AM1

7.14.1.1.10 The iswupper function

The same as ISO/IEC 9899/AM1

7.14.1.1.11 The iswxdigit function

The same as ISO/IEC 9899/AM1

7.14.1.2 Extensible wide-character classification functions

The same as ISO/IEC 9899/AM1

.14.1.2.1 The wctype function

The same as ISO/IEC 9899/AM1

7.14.1.2.2 The iswctype function

The same as ISO/IEC 9899/AM1

7.14.2 Wide-character mapping utilities

The same as ISO/IEC 9899/AM1

7.14.2.1 Wide-character case-mapping functions

7.14.2.1.1 The towlower function

The same as ISO/IEC 9899/AM1

7.14.2.1.2 The towupper function

The same as ISO/IEC 9899/AM1

%EX is replaced by the locale's alternative time representation.

%Ey is replaced by the offset from %EC (year only) in the locale's alternative representation.

%EY is replaced by the full alternative year representation.

%Od is replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading zeros if there is any alternative symbol for zero, otherwise with leading spaces.

%Oe is replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading spaces.

%OH is replaced by the hour (24-hour clock) using the locale's alternative numeric symbols.

%OI is replaced by the hour (12-hour clock) using the locale's alternative numeric symbols.

%Om is replaced by the month using the locale's alternative numeric symbols.

%OM is replaced by the minutes using the locale's alternative numeric symbols.

%OS is replaced by the seconds using the locale's alternative numeric symbols.

%Ou is replaced by the weekday as a number in the locale's alternative representation (Monday = 1).

%OU is replaced by the week number of the year (Sunday as the first day of the week, rules corresponding to %U) using the locale's alternative numeric symbols.

%OV is replaced by the week number of the year (Sunday as the first day of the week, rules corresponding to %V) using the locale's alternative numeric symbols.

%Ow is replaced by the number of the weekday (Sunday = 0) using the locale's alternative numeric symbols.

%OW is replaced by the week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.

%Oy is replaced by the year (offset from %C) in the locale's alternative representation and using the locale's alternative symbols.

Returns

N/C

7.13 Alternative spelling <iso646.h>

The same as ISO/IEC 9899/AM1

7.14 Wide-character classification and mapping utilities <wctype.h>

The same as ISO/IEC 9899/AM1

7.14.1 Wide-character classification utilities

The same as ISO/IEC 9899/AM1

7.14.1.1 Wide-character classification functions

The same as ISO/IEC 9899/AM1

7.14.1.1.1 The iswalnum function

The same as ISO/IEC 9899/AM1

7.14.1.1.2 The iswalpha function

The same as ISO/IEC 9899/AM1

7.14.1.1.3 The iswcntrl function

The same as ISO/IEC 9899/AM1

62

7.15.1.1.8 The vwprintf function

In addition to the requirements specified by ISO/IEC 9899:1990 with ISO/IEC 9899/AM1, the same modification on conversion specification with fprintf function is applied.

7.15.1.1.9 The vswprintf function

In addition to the requirements specified by ISO/IEC 9899:1990 with ISO/IEC 9899/AM1, the same modification on conversion specification with fprintf function is applied.

7.15.1.2 Wide-character input/output functions

7.15.1.2.1 The fgetwc function

The same as ISO/IEC 9899/AM1

7.15.1.2.2 The fgetws function

The same as ISO/IEC 9899/AM1

7.15.1.2.3 The fputwc function

The same as ISO/IEC 9899/AM1

7.15.1.2.4 The fputws function

The same as ISO/IEC 9899/AM1

7.15.1.2.5 The getwc function

The same as ISO/IEC 9899/AM1

7.15.1.2.6 The getwchar function

The same as ISO/IEC 9899/AM1

7.15.1.2.7 The putwc function

The same as ISO/IEC 9899/AM1

7.15.1.2.8 The putwchar function

The same as ISO/IEC 9899/AM1

7.15.1.2.9 The ungetwc function

The same as ISO/IEC 9899/AM1

7.15.1.2.10 The fwide function

The same as ISO/IEC 9899/AM1

7.15.2 General wide-string utilities

The same as ISO/IEC 9899/AM1

7.15.2.1 Wide-string numeric conversion functions

7.15.2.1.1 The wcstod function

The same as ISO/IEC 9899/AM1

7.15.2.1.2 The wcstol function

The same as ISO/IEC 9899/AM1

7.15.2.1.3 The wcstoul function

The same as ISO/IEC 9899/AM1

7.15.2.2 Wide-string copying functions

7.15.2.2.1 The wcscpy function

The same as ISO/IEC 9899/AM1

7.15.2.2.2 The wcsncpy function

The same as ISO/IEC 9899/AM1

7.15.2.3 Wide-string concatenation functions

7.15.2.3.1 The wcscat function

The same as ISO/IEC 9899/AM1

7.15.2.3.2 The wcsncat function

The same as ISO/IEC 9899/AM1

7.15.2.4 Wide-string comparison functions

The same as ISO/IEC 9899/AM1

7.15.2.4.1 The wcscmp function

The same as ISO/IEC 9899/AM1

7.15.2.4.2 The wcscoll function

The same as ISO/IEC 9899/AM1

7.15.2.4.3 The wcsncmp function

The same as ISO/IEC 9899/AM1

7.15.2.4.4 The wcsxfrm function

The same as ISO/IEC 9899/AM1

7.15.2.5 Wide-string search functions

7.15.2.5.1 The wcschr function

The same as ISO/IEC 9899/AM1

7.15.2.5.2 The wcscspn function

The same as ISO/IEC 9899/AM1

7.15.2.5.3 The wcspbrk function

The same as ISO/IEC 9899/AM1

7.15.2.5.4 The wcsrchr function

The same as ISO/IEC 9899/AM1

7.15.2.5.5 The wcsspn function

The same as ISO/IEC 9899/AM1

7.15.2.5.6 The wcsstr function

The same as ISO/IEC 9899/AM1

7.15.2.5.7 The wcstok function

The same as ISO/IEC 9899/AM1

7.15.2.5.8 The wcslen function

The same as ISO/IEC 9899/AM1

7.15.2.6 Wide-character array functions

The same as ISO/IEC 9899/AM1

.15.2.6.1 The wmemchr function

The same as ISO/IEC 9899/AM1

7.15.2.6.2 The wmemcmp function

The same as ISO/IEC 9899/AM1

7.15.2.6.3 The wmemcpy function

The same as ISO/IEC 9899/AM1

7.15.2.6.4 The wmemmove function

The same as ISO/IEC 9899/AM1

7.15.2.6.5 The wmemset function

The same as ISO/IEC 9899/AM1

7.15.3 Date and time

7.15.3.1 Wide-string time conversion functions

7.15.3.1.1 The wcsftime function

The same as ISO/IEC 9899/AM1

7.15.4 Extended multibyte and wide-character conversion utilities

The same with ISO/IEC 9899/AM1

7.15.4.1 Single-byte wide-character conversion functions

7.15.4.1.1 The btowc function

The same as ISO/IEC 9899/AM1

7.15.4.1.2 The wctob function

The same as ISO/IEC 9899/AM1

7.15.4.2 Conversion state functions

7.15.4.2.1 The mbsinit function

The same as ISO/IEC 9899/AM1

7.15.4.3 Restartable multibyte/wide-character conversion functions

The same as ISO/IEC 9899/AM1

7.15.4.3.1 The mbrlen function

The same as ISO/IEC 9899/AM1

7.15.4.3.2 The mbrtowc function

The same as ISO/IEC 9899/AM1

7.15.4.3.3 The wcrtomb function

The same as ISO/IEC 9899/AM1

7.15.4.4 Restartable multibyte/wide-string conversion functions

The same as ISO/IEC 9899/AM1

7.15.4.4.1 The mbsrtowcs function

The same as ISO/IEC 9899/AM1

7.15.4.4.2 The wcsrtombs function

The same as ISO/IEC 9899/AM1

7.16 Future library directions

The same as ISO/IEC 9899:1990

7.16.1 Errors <errno.h>

The same as ISO/IEC 9899:1990

7.16.2 Character handling <ctype.h>

The same as ISO/IEC 9899:1990

7.16.3 Localization <loclae.h>

The same as ISO/IEC 9899:1990

7.16.4 Mathematics <math.h>

The same as ISO/IEC 9899:1990

7.16.5 Signal handling <signal.h>

The same as ISO/IEC 9899:1990

7.16.6 Input/output <stdio.h>

The same as ISO/IEC 9899:1990

7.16.7 General utilities <stdlib.h>

The same as ISO/IEC 9899:1990

7.16.8 String handling <string.h>

The same as ISO/IEC 9899:1990

7.16.9 Wide-character classification and mapping utilities <wctype.h>

The same as ISO/IEC 9899/AM1

7.16.10 Extended multibyte and wide-character utilities <wchar.h>

The same as ISO/IEC 9899:1990 with ISO/IEC 9899/AM1