Draft Standard for Information Technology— Portable Operating System Interface (POSIX®)

Prepared by the Austin Group

(http://www.opengroup.org/austin/)

Chapter 1

Introduction

1.1 Scope

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

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

1. General terms, concepts, and interfaces common to all volumes of IEEE Std 1003.1-200x, including utility conventions and C language header definitions, are included in the Base Definitions volume of IEEE Std 1003.1-200x.

2. Definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery, are included in the System Interfaces volume of IEEE Std 1003.1-200x.

3. Definitions for a standard source code-level interface to command interpretation services (a ‘‘shell’’) and common utility programs for application programs are included in the Shell and Utilities volume of IEEE Std 1003.1-200x.

4. Extended rationale that did not fit well into the rest of the document structure, containing historical information concerning the contents of IEEE Std 1003.1-200x and why features were included or discarded by the standard developers, is included in the Rationale (Informative) volume of IEEE Std 1003.1-200x.

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

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

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

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

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

  - — IEEE Std 1003.1g-2000 (Protocol-Independent Interfaces (PII))
  - — IEEE Std 1003.1j-2000 (Advanced Realtime Extensions)
  - — IEEE Std 1003.1q-2000 (Tracing)

- IEEE Std 1003.2-1992 (POSIX-2) (includes IEEE Std 1003.2a-1992)
- The following amendments to the ISO POSIX-2: 1993 standard:
  - — IEEE P1003.2b draft standard (Additional Utilities)
  - — IEEE Std 1003.2d-1994 (Batch Environment)
- Open Group Technical Standard, February 1997, System Interface Definitions, Issue 5 (XBD5) (ISBN: 1-85912-186-1, C605)
- Open Group Technical Standard, February 1997, Commands and Utilities, Issue 5 (XCU5) (ISBN: 1-85912-191-8, C604)
- Open Group Technical Standard, February 1997, System Interfaces and Headers, Issue 5 (XSH5) (in 2 Volumes) (ISBN: 1-85912-181-0, C606)

  Note: XBD5, XCU5, and XSH5 are collectively referred to as the *Base Specifications*.

- Open Group Technical Standard, January 2000, Networking Services, Issue 5.2 (XNS5.2) (ISBN: 1-85912-241-8, C808)
- ISO / IEC 9899: 1999, Programming Languages — C.

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

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

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

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

> *advance*( ), *brk*( ), *chroot*( ), *compile*( ), *cuserid*( ), *gamma*( ), *getdtablesize*( ), *getpagesize*( ), *getpass*( ), *getw*( ), *putw*( ), *re_comp*( ), *re_exec*( ), *regcmp*( ), *sbrk*( ), *sigstack*( ), *step*( ), *wait3*( ), **<re_comp.h>**, **<regexp.h>**, **<varargs.h>**, *loc1* , *__loc1*, *loc2* , *locs*

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

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

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

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

Page 112, CLK_TCK
Page 197 *tcgetattr*( ) rate returned option

From the ISO POSIX-2: 1993 standard, obsolescent features within the following pages are not carried forward:

Page 75, zero-length prefix within PATH
Page 156, 159 *set*
Page 178, *awk*, use of no argument and no parentheses with length
Page 259, *ed*
Page 272, *env*
Page 282, *find* −**perm**[−]*onum*
Page 295-296, *egrep*
Page 299-300, *head*
Page 305-306, *join*
Page 309-310, *kill*
Page 431-433, 435-436, *sort*
Page 444-445, *tail*
Page 453, 455-456, *touch*
Page 464-465, *tty*
Page 472, *uniq*
Page 515-516, *ex*
Page 542-543, *expand*
Page 563-565, *more*
Page 574-576, *newgrp*
Page 578, *nice*
Page 594-596, *renice*
Page 597-598, *split*
Page 600-601, *strings*
Page 624-625, *vi*
Page 693, *lex*

The *c89* utility (which specified a compiler for the C Language specified by the ISO / IEC 9899: 1990 standard) has been replaced by a *c99* utility (which specifies a compiler for the C Language specified by the ISO / IEC 9899: 1999 standard).

From XSH5, text marked OH (Optional Header) has been reviewed on a case-by-case basis and removed where appropriate. The XCU5 text marked OF (Output Format Incompletely Specified) and UN (Possibly Unsupportable Feature) has been reviewed on a case-by-case basis and removed where appropriate

For the networking interfaces, the base document is the XNS, Issue 5.2 specification. The following parts of the XNS, Issue 5.2 specification are out of scope and not included in IEEE Std 1003.1-200x:

- Part 3 (XTI)
- Part 4 (Appendixes)

Since there is much duplication between the XNS, Issue 5.2 specification and IEEE Std 1003.1g-2000, material only from the following sections of IEEE Std 1003.1g-2000 has been included:

- General terms related to sockets (2.2.2)
- Socket concepts (5.1 through 5.3, inclusive)

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

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

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

1.2 Conformance

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

1.3 Normative References

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

ANS X3.9-1978
(Reaffirmed 1989) American National Standard for Information Systems: Standard X3.9-1978, Programming Language FORTRAN.[1]

ISO / IEC 646: 1991
ISO / IEC 646: 1991, Information Processing — ISO 7-bit Coded Character Set for Information Interchange.[2]

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

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

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

[2] ISO / IEC documents can be obtained from the ISO office: 1 Rue de Varembé, Case Postale 56, CH-1211, Genève 20, Switzerland / Suisse

ISO 8601: 2000

ISO 8601: 2000, Data Elements and Interchange Formats — Information Interchange — Representation of Dates and Times.

ISO C (1999)

ISO / IEC 9899: 1999, Programming Languages — C, including Technical Corrigendum No. 1.

ISO / IEC 10646-1: 2000

ISO / IEC 10646-1: 2000, Information Technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane.

1.4 Terminology

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

can

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

implementation-defined

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

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

legacy

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

may

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

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

shall

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

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

should

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

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

undefined

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

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

unspecified

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

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

1.5 Portability

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

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

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

1.5.1 Codes

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

ADV Advisory Information

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

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

AIO Asynchronous Input and Output

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

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

BAR Barriers

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

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

BE Batch Environment Services and Utilities

The functionality described is optional.

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

CD C-Language Development Utilities

The functionality described is optional.

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

CPT Process CPU-Time Clocks

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

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

CS Clock Selection

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

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

CX Extension to the ISO C standard

The functionality described is an extension to the ISO C standard. Application writers may make use of an extension as it is supported on all IEEE Std 1003.1-200x-conforming systems.

With each function or header from the ISO C standard, a statement to the effect that ‘‘any conflict is unintentional’’ is included. That is intended to refer to a direct conflict. IEEE Std 1003.1-200x acts in part as a profile of the ISO C standard, and it may choose to further constrain behaviors allowed to vary by the ISO C standard. Such limitations are not considered conflicts.

FD FORTRAN Development Utilities

The functionality described is optional.

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

FR FORTRAN Runtime Utilities

The functionality described is optional.

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

FSC File Synchronization

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

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

IP6 IPV6

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

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

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

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

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

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

Refer to Section 1.5.2 (on page 14).

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

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

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

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

Refer to Section 1.5.2 (on page 14).

MF Memory Mapped Files

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

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

ML Process Memory Locking

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

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

MLR Range Memory Locking

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

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

MON Monotonic Clock

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

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

MPR Memory Protection

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

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

MSG Message Passing

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

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

MX IEC 60559 Floating-Point Option

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

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

OB Obsolescent

The functionality described may be withdrawn in a future version of this volume of IEEE Std 1003.1-200x. Strictly Conforming POSIX Applications and Strictly Conforming XSI Applications shall not use obsolescent features.

OF Output Format Incompletely Specified

The functionality described is an XSI extension. The format of the output produced by the utility is not fully specified. It is therefore not possible to post-process this output in a consistent fashion. Typical problems include unknown length of strings and unspecified field delimiters.

OH Optional Header

In the SYNOPSIS section of some interfaces in the System Interfaces volume of IEEE Std 1003.1-200x an included header is marked as in the following example:

OH

```
#include <sys/types.h>
#include <grp.h>
struct group *getgrnam(const char *name);
```

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

PIO Prioritized Input and Output

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

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

PS Process Scheduling

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

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

RS Raw Sockets

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

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

RTS Realtime Signals Extension

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

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

SD Software Development Utilities

The functionality described is optional.

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

SEM Semaphores

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

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

SHM Shared Memory Objects

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

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

SIO Synchronized Input and Output

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

TMR Timers

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

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

TPI Thread Priority Inheritance

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

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

TPP Thread Priority Protection

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

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

TPS Thread Execution Scheduling

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

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

TRC Trace

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

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

TRI Trace Inherit

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

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

TRL Trace Log

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

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

TSA Thread Stack Address Attribute

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

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

TSF Thread-Safe Functions

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

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

TSH Thread Process-Shared Synchronization

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

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

TSP Thread Sporadic Server

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

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

TSS Thread Stack Address Size

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

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

TYM Typed Memory Objects

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

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

UP User Portability Utilities

The functionality described is optional.

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

XSI Extension

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

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

XSR XSI STREAMS

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

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

1.5.2 Margin Code Notation

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

A Feature Dependent on One or Two Options

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

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

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

A Feature Dependent on Either of the Options Denoted

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

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

A Feature Dependent on More than Two Options

The following shorthand notations are used:

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

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

Large Sections Dependent on an Option

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

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

Chapter 2

Conformance

2.1 Implementation Conformance

2.1.1 Requirements

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

1. The system shall support all utilities, functions, and facilities defined within IEEE Std 1003.1-200x that are required for POSIX conformance (see Section 2.1.3 (on page 16)). These interfaces shall support the functional behavior described herein.

2. The system may support one or more options as described under Section 2.1.5 (on page 20). When an implementation claims that an option is supported, all of its constituent parts shall be provided.

3. The system may support the X/Open System Interface Extension (XSI) as described under Section 2.1.4 (on page 19).

4. The system may provide additional utilities, functions, or facilities not required by IEEE Std 1003.1-200x. Non-standard extensions of the utilities, functions, or facilities specified in IEEE Std 1003.1-200x should be identified as such in the system documentation. Non-standard extensions, when used, may change the behavior of utilities, functions, or facilities defined by IEEE Std 1003.1-200x. The conformance document shall define an environment in which an application can be run with the behavior specified by IEEE Std 1003.1-200x. In no case shall such an environment require modification of a Strictly Conforming POSIX Application (see Section 2.2.1 (on page 29)).

2.1.2 Documentation

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

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

The conformance document shall describe the limit values found in the headers **<limits.h>** (on page 245) and **<unistd.h>** (on page 398), 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 IEEE Std 1003.1-200x. This requirement shall be met by listing these features and providing either a specific reference to the system documentation or providing full syntax and semantics of these features. When the value or behavior in the implementation is designed to be variable or customized on each instantiation of the system, the implementation provider shall document the nature and permissible ranges of this variation.

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

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

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

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

2.1.3 POSIX Conformance

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

2.1.3.1 POSIX System Interfaces

- The system shall support all the mandatory functions and headers defined in IEEE Std 1003.1-200x, and shall set the symbolic constant _POSIX_VERSION to the value 200xxxL.
- Although all implementations conforming to IEEE Std 1003.1-200x support all the features described below, there may be system-dependent or file system-dependent configuration procedures that can remove or modify any or all of these features. Such configurations should not be made if strict compliance is required.

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

  — _POSIX_CHOWN_RESTRICTED

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

  — _POSIX_NO_TRUNC

    Pathname components longer than {NAME_MAX} generate an error.

- The following symbolic constants shall be defined as follows:
  - _POSIX_JOB_CONTROL shall have a value greater than zero.
  - _POSIX_SAVED_IDS shall have a value greater than zero.
  - _POSIX_VDISABLE shall have a value other than −1.

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

- The system may support one or more options (see Section 2.1.6 (on page 26)) denoted by the following symbolic constants:

  — _POSIX_ADVISORY_INFO

  — _POSIX_ASYNCHRONOUS_IO

  — _POSIX_BARRIERS

  — _POSIX_CLOCK_SELECTION

  — _POSIX_CPUTIME

  — _POSIX_FSYNC

  — _POSIX_IPV6

  — _POSIX_MAPPED_FILES

  — _POSIX_MEMLOCK

  — _POSIX_MEMLOCK_RANGE

  — _POSIX_MEMORY_PROTECTION

  — _POSIX_MESSAGE_PASSING

  — _POSIX_MONOTONIC_CLOCK

  — _POSIX_PRIORITIZED_IO

  — _POSIX_PRIORITY_SCHEDULING

  — _POSIX_RAW_SOCKETS

  — _POSIX_REALTIME_SIGNALS

  — _POSIX_SEMAPHORES

  — _POSIX_SHARED_MEMORY_OBJECTS

  — _POSIX_SPAWN

  — _POSIX_SPIN_LOCKS

  — _POSIX_SPORADIC_SERVER

  — _POSIX_SYNCHRONIZED_IO

  — _POSIX_THREAD_ATTR_STACKADDR

  — _POSIX_THREAD_CPUTIME

  — _POSIX_THREAD_ATTR_STACKSIZE

  — _POSIX_THREAD_PRIO_INHERIT

  — _POSIX_THREAD_PRIO_PROTECT

  — _POSIX_THREAD_PRIORITY_SCHEDULING

  — _POSIX_THREAD_PROCESS_SHARED

  — _POSIX_THREAD_SAFE_FUNCTIONS

  — _POSIX_THREAD_SPORADIC_SERVER

  — _POSIX_THREADS

— _POSIX_TIMEOUTS

— _POSIX_TIMERS

— _POSIX_TRACE

— _POSIX_TRACE_EVENT_FILTER

— _POSIX_TRACE_INHERIT

— _POSIX_TRACE_LOG

— _POSIX_TYPED_MEMORY_OBJECTS

If any of the symbolic constants _POSIX_TRACE_EVENT_FILTER, _POSIX_TRACE_LOG, or _POSIX_TRACE_INHERIT is defined to have a value other than −1, then the symbolic constant _POSIX_TRACE shall also be defined to have a value other than −1.

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

— _XOPEN_CRYPT

— _XOPEN_LEGACY

— _XOPEN_REALTIME

— _XOPEN_REALTIME_THREADS

— _XOPEN_UNIX

2.1.3.2 *POSIX Shell and Utilities*

- The system shall provide all the mandatory utilities in the Shell and Utilities volume of IEEE Std 1003.1-200x with all the functional behavior described therein.
- The system shall support the Large File capabilities described in the Shell and Utilities volume of IEEE Std 1003.1-200x.
- The system may support one or more options (see Section 2.1.6 (on page 26)) denoted by the following symbolic constants. (The literal names below apply to the *getconf* utility.)

— POSIX2_C_DEV

— POSIX2_CHAR_TERM

— POSIX2_FORT_DEV

— POSIX2_FORT_RUN

— POSIX2_LOCALEDEF

— POSIX2_PBS

— POSIX2_PBS_ACCOUNTING

— POSIX2_PBS_LOCATE

— POSIX2_PBS_MESSAGE

— POSIX2_PBS_TRACK

— POSIX2_SW_DEV

— POSIX2_UPE

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

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

2.1.4 XSI Conformance

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

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

2.1.4.1 XSI System Interfaces

- The system shall support all the functions and headers defined in IEEE Std 1003.1-200x as part of the XSI extension denoted by the symbolic constant _XOPEN_UNIX and any extensions marked with the XSI extension marking (see Section 1.5.1 (on page 6)).
- The system shall support the *mmap*( ), *munmap*( ), and *msync*( ) functions.
- The system shall support the following options defined within IEEE Std 1003.1-200x (see Section 2.1.6 (on page 26)):

  — _POSIX_FSYNC

  — _POSIX_MAPPED_FILES

  — _POSIX_MEMORY_PROTECTION

  — _POSIX_THREAD_ATTR_STACKADDR

  — _POSIX_THREAD_ATTR_STACKSIZE

  — _POSIX_THREAD_PROCESS_SHARED

  — _POSIX_THREAD_SAFE_FUNCTIONS

  — _POSIX_THREADS

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

  — Encryption

  — Realtime

  — Advanced Realtime

  — Realtime Threads

  — Advanced Realtime Threads

  — Tracing

  — XSI STREAMS

  — Legacy

2.1.4.2 XSI Shell and Utilities Conformance

- The system shall support all the utilities defined in the Shell and Utilities volume of IEEE Std 1003.1-200x as part of the XSI extension denoted by the XSI marking in the SYNOPSIS section, and any extensions marked with the XSI extension marking (see Section 1.5.1 (on page 6)) within the text.
- The system shall support the User Portability Utilities option.
- The system shall support creation of locales (see Chapter 7 (on page 119)).
- The C-language Development utility *c99* shall be supported.
- The XSI Development Utilities option may be supported. It consists of the following software development utilities:

  admin
  cflow
  ctags
  cxref
  delta
  get
  m4
  prs
  rmdel
  sact
  sccs
  unget
  val
  what

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

2.1.5 Option Groups

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

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

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

2.1.5.1 Subprofiling Considerations

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

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

A Subprofiling Option Group may provide basic system functionality that other Subprofiling Option Groups and other options depend upon.[3] If a profile of IEEE Std 1003.1-200x does not

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

require an implementation to provide a Subprofiling Option Group that provides features utilized by a required Subprofiling Option Group (or option),[4] the profile shall specify[5] all of the following:

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

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

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

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

2.1.5.2 XSI Option Groups

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

The following Option Groups are defined.

Encryption

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

crypt( ), *encrypt*( ), *setkey*( )

These functions are marked CRYPT.

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

An implementation that claims conformance to this Option Group shall set _XOPEN_CRYPT to a value other than –1.

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

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

Realtime

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

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

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

An implementation that claims conformance to this Option Group shall set _XOPEN_REALTIME to a value other than −1.

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

_POSIX_ASYNCHRONOUS_IO
_POSIX_FSYNC
_POSIX_MAPPED_FILES
_POSIX_MEMLOCK
_POSIX_MEMLOCK_RANGE
_POSIX_MEMORY_PROTECTION
_POSIX_MESSAGE_PASSING
_POSIX_PRIORITIZED_IO
_POSIX_PRIORITY_SCHEDULING
_POSIX_REALTIME_SIGNALS
_POSIX_SEMAPHORES
_POSIX_SHARED_MEMORY_OBJECTS
_POSIX_SYNCHRONIZED_IO
_POSIX_TIMERS

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

_POSIX_ASYNCHRONOUS_IO
_POSIX_MEMLOCK
_POSIX_MEMLOCK_RANGE
_POSIX_MESSAGE_PASSING
_POSIX_PRIORITY_SCHEDULING
_POSIX_REALTIME_SIGNALS
_POSIX_SEMAPHORES
_POSIX_SHARED_MEMORY_OBJECTS
_POSIX_SYNCHRONIZED_IO
_POSIX_TIMERS

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

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

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

Advanced Realtime

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

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

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

_POSIX_ADVISORY_INFO
_POSIX_CLOCK_SELECTION
_POSIX_CPUTIME
_POSIX_MONOTONIC_CLOCK
_POSIX_SPAWN
_POSIX_SPORADIC_SERVER
_POSIX_TIMEOUTS
_POSIX_TYPED_MEMORY_OBJECTS

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

_POSIX_ADVISORY_INFO
_POSIX_CLOCK_SELECTION
_POSIX_CPUTIME
_POSIX_MONOTONIC_CLOCK
_POSIX_SPAWN
_POSIX_SPORADIC_SERVER
_POSIX_TIMEOUTS
_POSIX_TYPED_MEMORY_OBJECTS

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

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

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

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

Realtime Threads

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

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

_POSIX_THREAD_PRIO_INHERIT
_POSIX_THREAD_PRIO_PROTECT
_POSIX_THREAD_PRIORITY_SCHEDULING

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

An implementation that claims conformance to this Option Group shall set _XOPEN_REALTIME_THREADS to a value other than –1.

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

_POSIX_THREAD_PRIO_INHERIT
_POSIX_THREAD_PRIO_PROTECT
_POSIX_THREAD_PRIORITY_SCHEDULING

Advanced Realtime Threads

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

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

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

_POSIX_BARRIERS
_POSIX_SPIN_LOCKS
_POSIX_THREAD_CPUTIME
_POSIX_THREAD_SPORADIC_SERVER

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

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

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

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

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

_POSIX_BARRIERS
_POSIX_SPIN_LOCKS
_POSIX_THREAD_CPUTIME
_POSIX_THREAD_SPORADIC_SERVER

Tracing

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

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

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

_POSIX_TRACE
_POSIX_TRACE_EVENT_FILTER
_POSIX_TRACE_LOG
_POSIX_TRACE_INHERIT

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

_POSIX_TRACE
_POSIX_TRACE_EVENT_FILTER
_POSIX_TRACE_LOG
_POSIX_TRACE_INHERIT

XSI STREAMS

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

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

It includes the following functions:

fattach( ), *fdetach*( ), *getmsg*( ), *getpmsg*( ), *ioctl*( ), *isastream*( ), *putmsg*( ), *putpmsg*( )

and the **<stropts.h>** header.

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

An implementation that claims conformance to this Open Group shall set _XOPEN_STREAMS to a value other than −1.

Legacy

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

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

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

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

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

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

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

An implementation that claims conformance to this Option Group shall set _XOPEN_LEGACY to a value other than −1.

2.1.6 Options

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

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

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

2.1.6.1 System Interfaces

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

2.1.6.2 Shell and Utilities

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

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

CD POSIX2_C_DEV

The system supports the C-Language Development Utilities option.

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

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

c99
lex
yacc

POSIX2_CHAR_TERM

The system supports the Terminal Characteristics option. This value need not be present on a system not supporting the User Portability Utilities option.

Where applicable, the dependency is noted within the description of the utility.

This option applies only to systems supporting the User Portability Utilities option. If supported, then the system supports at least one terminal type capable of all operations described in IEEE Std 1003.1-200x; see Section 10.2 (on page 181).

FD POSIX2_FORT_DEV

The system supports the FORTRAN Development Utilities option.

The *fort77* FORTRAN compiler is the only utility in the FORTRAN Development Utilities option. This is used for the development of FORTRAN language applications, including compilation or translation of FORTRAN source code.

The *fort77* utility may be provided by a conforming system; however, any system claiming conformance to the FORTRAN Development Utilities option shall provide the *fort77* utility.

FR POSIX2_FORT_RUN

The system supports the FORTRAN Runtime Utilities option.

The *asa* utility is the only utility in the FORTRAN Runtime Utilities option.

The *asa* utility may be provided by a conforming system; however, any system claiming conformance to the FORTRAN Runtime Utilities option shall provide the *asa* utility.

POSIX2_LOCALEDEF

The system supports the Locale Creation Utilities option.

If supported, the system supports the creation of locales as described in the *localedef* utility.

The *localedef* utility may be provided by a conforming system; however, any system claiming conformance to the Locale Creation Utilities option shall provide the *localedef* utility.

BE POSIX2_PBS

The system supports the Batch Environment Services and Utilities option (see the Shell and Utilities volume of IEEE Std 1003.1-200x, Chapter 3, Batch Environment Services).

Note: The Batch Environment Services and Utilities option is a combination of mandatory and optional batch services and utilities. The POSIX_PBS symbolic constant implies the system supports all the mandatory batch services and utilities.

POSIX2_PBS_ACCOUNTING

The system supports the Batch Accounting option.

POSIX2_PBS_CHECKPOINT

The system supports the Batch Checkpoint/Restart option.

POSIX2_PBS_LOCATE

The system supports the Locate Batch Job Request option.

POSIX2_PBS_MESSAGE

The system supports the Batch Job Message Request option.

POSIX2_PBS_TRACK

The system supports the Track Batch Job Request option.

SD POSIX2_SW_DEV

The system supports the Software Development Utilities option.

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

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

ar
make
nm
strip

UP POSIX2_UPE
The system supports the User Portability Utilities option.

The utilities in the User Portability Utilities option shall be implemented on all systems that claim conformance to this option. Certain utilities are noted as having features that cannot be implemented on all terminal types; if the POSIX2_CHAR_TERM option is supported, the system shall support all such features on at least one terminal type; see Section 10.2 (on page 181).

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

alias
at
batch
bg
crontab
split
ctags
df
du
ex
expand
fc
fg
file
jobs
man
mesg
more
newgrp
nice
nm
patch
ps
renice
split
strings
tabs
talk
time
tput
unalias
unexpand
uudecode
uuencode
vi
who
write

2.2 Application Conformance

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

2.2.1 Strictly Conforming POSIX Application

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

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

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

2.2.2 Conforming POSIX Application

2.2.2.1 ISO/IEC Conforming POSIX Application

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

2.2.2.2 <National Body> Conforming POSIX Application

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

2.2.3 Conforming POSIX Application Using Extensions

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

2.2.4 Strictly Conforming XSI Application

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

1. Shall accept any implementation behavior that results from actions it takes in areas described in IEEE Std 1003.1-200x as *implementation-defined* or *unspecified*, or where IEEE Std 1003.1-200x indicates that implementations may vary

2. Shall not perform any actions that are described as producing *undefined* results

3. For symbolic constants, shall accept any value in the range permitted by IEEE Std 1003.1-200x, but shall not rely on any value in the range being greater than the minimums listed or being less than the maximums listed in IEEE Std 1003.1-200x

4. Shall not use facilities designated as *obsolescent*

5. Is required to tolerate and permitted to adapt to the presence or absence of optional facilities whose availability is indicated by Section 2.1.4 (on page 19)

6. For the C programming language, shall not produce any output dependent on any behavior described in the ISO C standard as *unspecified*, *undefined*, or *implementation-defined*, unless the System Interfaces volume of IEEE Std 1003.1-200x specifies the behavior

7. For the C programming language, shall not exceed any minimum implementation limit defined in the ISO C standard, unless the System Interfaces volume of IEEE Std 1003.1-200x specifies a higher minimum implementation limit

8. For the C programming language, shall define _XOPEN_SOURCE to be 600 before any header is included

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

2.2.5 Conforming XSI Application Using Extensions

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

2.3 Language-Dependent Services for the C Programming Language

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

2.4 Other Language-Related Specifications

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

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

Chapter 3

Definitions

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

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

3.1 Abortive Release

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

3.2 Absolute Pathname

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

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

3.3 Access Mode

A particular form of access permitted to a file.

3.4 Additional File Access Control Mechanism

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

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

3.5 Address Space

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

3.6 Advisory Information

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

3.7 Affirmative Response

An input string that matches one of the responses acceptable to the *LC_MESSAGES* category keyword **yesexpr**, matching an extended regular expression in the current locale.

Note: The *LC_MESSAGES* category is defined in detail in Section 7.3.6 (on page 148).

3.8 Alert

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

3.9 Alert Character (<alert>)

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

3.10 Alias Name

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

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

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

3.11 Alignment

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

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

3.12 Alternate File Access Control Mechanism

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

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

3.13 Alternate Signal Stack

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

3.14 Ancillary Data

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

3.15 Angle Brackets

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

3.16 Application

A computer program that performs some desired function.

3.17 Application Address

Endpoint address of a specific application.

3.18 Application Program Interface (API)

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

3.19 Appropriate Privileges

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

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

3.20 Argument

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

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

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

3.21 Arm (a Timer)

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

3.22 Asterisk

The character '*'.

3.23 Async-Cancel-Safe Function

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

3.24 Asynchronous Events

Events that occur independently of the execution of the application.

3.25 Asynchronous Input and Output

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

3.26 Async-Signal-Safe Function

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

3.27 Asynchronously-Generated Signal

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

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

3.28 Asynchronous I/O Operation

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

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

3.29 Asynchronous I/O Completion

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

3.30 Authentication

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

3.31 Authorization

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

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

3.32 Background Job

See *Background Process Group* in Section 3.34.

3.33 Background Process

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

3.34 Background Process Group (or Background Job)

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

3.35 Backquote

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

3.36 Backslash

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

3.37 Backspace Character (<backspace>)

A character that, in the output stream, should cause printing (or displaying) to occur one column position previous to the position about to be printed. If the position about to be printed is at the beginning of the current line, the behavior is unspecified. It is the character designated by '\b' in the C language. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the backspace function. The <backspace> defined here is not necessarily the ERASE special character.

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

3.38 Barrier

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

3.39 Base Character

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

3.40 Basename

The final, or only, filename in a pathname.

3.41 Basic Regular Expression (BRE)

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

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

3.42 Batch Access List

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

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

3.43 Batch Administrator

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

3.44 Batch Client

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

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

3.45 Batch Destination

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

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

3.46 Batch Destination Identifier

A string that identifies a specific batch destination.

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

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

3.47 Batch Directive

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

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

3.48 Batch Job

A set of computational tasks for a computing system.

Batch jobs are managed by batch servers.

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

3.49 Batch Job Attribute

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

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

3.50 Batch Job Identifier

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

3.51 Batch Job Name

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

3.52 Batch Job Owner

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

3.53 Batch Job Priority

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

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

3.54 Batch Job State

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

3.55 Batch Name Service

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

3.56 Batch Name Space

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

3.57 Batch Node

A host containing part or all of a batch system.

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

- Capable of executing a batch client
- Contains a routing batch queue
- Contains an execution batch queue

3.58 Batch Operator

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

3.59 Batch Queue

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

Note: A set of batch jobs is called a batch queue largely for historical reasons. Jobs are selected from the batch queue for execution based on attributes such as priority, resource requirements, and hold conditions.

See also the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 3.1.2, Batch Queues.

3.60 Batch Queue Attribute

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

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

3.61 Batch Queue Position

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

3.62 Batch Queue Priority

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

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

3.63 Batch Rerunability

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

3.64 Batch Restart

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

3.65 Batch Server

A computational entity that provides batch services.

3.66 Batch Server Name

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

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

3.67 Batch Service

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

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

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

3.68 Batch Service Request

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

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

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

3.69 Batch Submission

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

3.70 Batch System

A collection of one or more batch servers.

3.71 Batch Target User

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

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

3.72 Batch User

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

3.73 Bind

The process of assigning a network address to an endpoint.

3.74 Blank Character (<blank>)

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

3.75 Blank Line

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

3.76 Blocked Process (or Thread)

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

3.77 Blocking

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

3.78 Block-Mode Terminal

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

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

3.79 Block Special File

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

3.80 Braces

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

3.81 Brackets

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

3.82 Broadcast

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

3.83 Built-In Utility (or Built-In)

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

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

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

3.84 Byte

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

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

3.85 Byte Input/Output Functions

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

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

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

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

3.87 Character

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

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

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

3.88 Character Array

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

3.89 Character Class

A named set of characters sharing an attribute associated with the name of the class. The classes and the characters that they contain are dependent on the value of the *LC_CTYPE* category in the current locale.

Note: The *LC_CTYPE* category is defined in detail in Section 7.3.1 (on page 122).

3.90 Character Set

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

3.91 Character Special File

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

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

3.92 Character String

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

3.93 Child Process

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

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

3.94 Circumflex

The character '^'.

3.95 Clock

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

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

3.96 Clock Jump

The difference between two successive distinct values of a clock, as observed from the application via one of the ‘‘get time’’ operations.

3.97 Clock Tick

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

3.98 Coded Character Set

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

3.99 Codeset

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

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

3.100 Collating Element

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

3.101 Collation

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

3.102 Collation Sequence

The relative order of collating elements as determined by the setting of the *LC_COLLATE* category in the current locale. 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 the order in which collating elements are specified between **order_start** and **order_end** keywords in the *LC_COLLATE* category.

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

Note: {COLL_WEIGHTS_MAX} is defined in detail in **<limits.h>**.

3.103 Column Position

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

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

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

3.104 Command

A directive to the shell to perform a particular task.

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

3.105 Command Language Interpreter

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

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

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

3.106 Composite Graphic Symbol

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

3.107 Condition Variable

A synchronization object which allows a thread to suspend execution, repeatedly, until some associated predicate becomes true. A thread whose execution is suspended on a condition variable is said to be blocked on the condition variable.

3.108 Connection

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

3.109 Connection Mode

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

3.110 Connectionless Mode

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

3.111 Control Character

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

3.112 Control Operator

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

```
&   &&   (   )   ;   ;;   newline   |   ||
```

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

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

3.113 Controlling Process

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

3.114 Controlling Terminal

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

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

3.115 Conversion Descriptor

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

3.116 Core File

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

3.117 CPU Time (Execution Time)

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

3.118 CPU-Time Clock

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

3.119 CPU-Time Timer

A timer attached to a CPU-time clock.

3.120 Current Job

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

3.121 Current Working Directory

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

3.122 Cursor Position

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

3.123 Datagram

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

3.124 Data Segment

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

3.125 Deferred Batch Service

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

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

3.126 Device

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

3.127 Device ID

A non-negative integer used to identify a device.

3.128 Directory

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

3.129 Directory Entry (or Link)

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

3.130 Directory Stream

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

3.131 Disarm (a Timer)

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

3.132 Display

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

3.133 Display Line

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

Note: This may also be written as ‘‘line on the display’’.

3.134 Dollar Sign

The character `'$'`.

3.135 Dot

In the context of naming files, the filename consisting of a single dot character (`'.'`).

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

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

3.136 Dot-Dot

The filename consisting solely of two dot characters (`".."`).

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

3.137 Double-Quote

The character `'"'`, also known as *quotation-mark*.

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

3.138 Downshifting

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

3.139 Driver

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

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

3.140 Effective Group ID

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

3.141 Effective User ID

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

3.142 Eight-Bit Transparency

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

3.143 Empty Directory

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

3.144 Empty Line

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

3.145 Empty String (or Null String)

A string whose first byte is a null byte.

3.146 Empty Wide-Character String

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

3.147 Encoding Rule

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

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

3.148 Entire Regular Expression

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

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

3.149 Epoch

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

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

3.150 Equivalence Class

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

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

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

3.151 Era

A locale-specific method for counting and displaying years.

Note: The *LC_TIME* category is defined in detail in Section 7.3.5 (on page 142).

3.152 Event Management

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

3.153 Executable File

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

3.154 Execute

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

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

3.155 Execution Time

See *CPU Time* in Section 3.117 (on page 49).

3.156 Execution Time Monitoring

A set of execution time monitoring primitives that allow online measuring of thread and process execution times.

3.157 Expand

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

Note: Word Expansions are defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.6, Word Expansions.

3.158 Extended Regular Expression (ERE)

A regular expression (see also Section 3.316 (on page 76)) that is an alternative to the Basic Regular Expression using a more extensive syntax, occasionally used by some utilities.

Note: Extended Regular Expressions are described in detail in Section 9.4 (on page 171).

3.159 Extended Security Controls

Implementation-defined security controls allowed by the file access permission and appropriate privilege (see also Section 3.19 (on page 35)) mechanisms, through which an implementation can support different security policies from those described in IEEE Std 1003.1-200x.

Note: See also Extended Security Controls defined in Section 4.3 (on page 95).

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

3.160 Feature Test Macro

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

Note: See also the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment.

3.161 Field

In the shell command language, a unit of text that is the result of parameter expansion, arithmetic expansion, command substitution, or field splitting. During command processing, the resulting fields are used as the command name and its arguments.

Note: Parameter Expansion is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.6.2, Parameter Expansion.

Arithmetic Expansion is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.6.4, Arithmetic Expansion.

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

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

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

3.162 FIFO Special File (or FIFO)

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

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

3.163 File

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

3.164 File Description

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

3.165 File Descriptor

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

Note: {OPEN_MAX} is defined in detail in **<limits.h>**.

3.166 File Group Class

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

3.167 File Mode

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

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

3.168 File Mode Bits

A file's file permission bits, set-user-ID-on-execution bit (S_ISUID), and set-group-ID-on-execution bit (S_ISGID).

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

3.169 Filename

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

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

3.170 Filename Portability

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

3.171 File Offset

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

3.172 File Other Class

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

3.173 File Owner Class

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

3.174 File Permission Bits

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

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

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

3.175 File Serial Number

A per-file system unique identifier for a file.

3.176 File System

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

3.177 File Type

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

3.178 Filter

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

3.179 First Open (of a File)

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

3.180 Flow Control

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

3.181 Foreground Job

See *Foreground Process Group* in Section 3.183.

3.182 Foreground Process

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

3.183 Foreground Process Group (or Foreground Job)

A process group whose member processes have certain privileges, denied to processes in background process groups, when accessing their controlling terminal. Each session that has established a connection with a controlling terminal has at most one process group of the session as the foreground process group of that controlling terminal.

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

3.184 Foreground Process Group ID

The process group ID of the foreground process group.

3.185 Form-Feed Character (<form-feed>)

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

3.186 Graphic Character

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

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

3.187 Group Database

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

- Group name
- Numerical group ID
- List of users allowed in the group

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

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

3.188 Group ID

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

3.189 Group Name

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

3.190 Hard Limit

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

3.191 Hard Link

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

3.192 Home Directory

The directory specified by the *HOME* environment variable.

3.193 Host Byte Order

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

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

3.194 Incomplete Line

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

3.195 Inf

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

3.196 Instrumented Application

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

3.197 Interactive Shell

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

3.198 Internationalization

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

3.199 Interprocess Communication

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

3.200 Invoke

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

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

3.201 Job

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

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

3.202 Job Control

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

3.203 Job Control Job ID

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

Table 3-1 Job Control Job ID Formats

Job Control Job ID	Meaning
%%	Current job.
%+	Current job.
%–	Previous job.
%*n*	Job number *n*.
%*string*	Job whose command begins with *string*.
%?*string*	Job whose command contains *string*.

3.204 Last Close (of a File)

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

3.205 Line

A sequence of zero or more non-<newline>s plus a terminating <newline>.

3.206 Linger

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

3.207 Link

See *Directory Entry* in Section 3.129 (on page 50).

3.208 Link Count

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

3.209 Local Customs

The conventions of a geographical area or territory for such things as date, time, and currency formats.

3.210 Local Interprocess Communication (Local IPC)

The transfer of data between processes in the same system.

3.211 Locale

The definition of the subset of a user's environment that depends on language and cultural conventions.

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

3.212 Localization

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

3.213 Login

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

3.214 Login Name

A user name that is associated with a login.

3.215 Map

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

3.216 Marked Message

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

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

3.217 Matched

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

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

3.218 Memory Mapped Files

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

3.219 Memory Object

One of:

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

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

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

3.220 Memory-Resident

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

3.221 Message

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

3.222 Message Catalog

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

3.223 Message Catalog Descriptor

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

3.224 Message Queue

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

3.225 Mode

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

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

3.226 Monotonic Clock

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

3.227 Mount Point

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

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

3.228 Multi-Character Collating Element

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

3.229 Mutex

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

3.230 Name

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

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

3.231 Named STREAM

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

3.232 NaN (Not a Number)

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

3.233 Native Language

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

3.234 Negative Response

An input string that matches one of the responses acceptable to the *LC_MESSAGES* category keyword **noexpr**, matching an extended regular expression in the current locale.

Note: The *LC_MESSAGES* category is defined in detail in Section 7.3.6 (on page 148).

3.235 Network

A collection of interconnected hosts.

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

3.236 Network Address

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

3.237 Network Byte Order

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

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

3.238 Newline Character (<newline>)

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

3.239 Nice Value

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

3.240 Non-Blocking

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

Note: The exact semantics are dependent on the type of file associated with the open file description. For data reads from devices such as ttys and FIFOs, this property causes the read to return immediately when no data was available. Similarly, for writes, it causes the call to return immediately when the thread would otherwise be delayed in the write operation; for example, because no space was available. For networking, it causes functions not to await protocol events (for example, acknowledgements) to occur. See also the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.10.7, Socket I/O Mode.

3.241 Non-Spacing Characters

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

3.242 NUL

A character with all bits set to zero.

3.243 Null Byte

A byte with all bits set to zero.

3.244 Null Pointer

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

3.245 Null String

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

3.246 Null Wide-Character Code

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

3.247 Number Sign

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

3.248 Object File

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

3.249 Octet

Unit of data representation that consists of eight contiguous bits.

3.250 Offset Maximum

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

3.251 Opaque Address

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

3.252 Open File

A file that is currently associated with a file descriptor.

3.253 Open File Description

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

3.254 Operand

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

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

3.255 Operator

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

3.256 Option

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

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

3.257 Option-Argument

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

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

3.258 Orientation

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

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

3.259 Orphaned Process Group

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

3.260 Page

The granularity of process memory mapping or locking.

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

3.261 Page Size

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

3.262 Parameter

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

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

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

3.263 Parent Directory

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

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

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

3.264 Parent Process

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

3.265 Parent Process ID

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

3.266 Pathname

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

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

3.267 Pathname Component

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

3.268 Path Prefix

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

3.269 Pattern

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

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

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

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

3.270 Period

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

3.271 Permissions

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

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

3.272 Persistence

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

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

3.273 Pipe

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

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

3.274 Polling

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

3.275 Portable Character Set

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

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

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

3.276 Portable Filename Character Set

The set of characters from which portable filenames are constructed.

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

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

3.277 Positional Parameter

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

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

3.278 Preallocation

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

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

3.279 Preempted Process (or Thread)

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

3.280 Previous Job

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

3.281 Printable Character

One of the characters included in the **print** character classification of the *LC_CTYPE* category in the current locale.

Note: The *LC_CTYPE* category is defined in detail in Section 7.3.1 (on page 122).

3.282 Printable File

A text file consisting only of the characters included in the **print** and **space** character classifications of the *LC_CTYPE* category and the <backspace>, all in the current locale.

Note: The *LC_CTYPE* category is defined in detail in Section 7.3.1 (on page 122).

3.283 Priority

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

3.284 Priority Band

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

3.285 Priority Inversion

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

3.286 Priority Scheduling

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

3.287 Priority-Based Scheduling

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

3.288 Privilege

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

3.289 Process

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

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

3.290 Process Group

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

3.291 Process Group ID

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

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

3.292 Process Group Leader

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

3.293 Process Group Lifetime

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

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

3.294 Process ID

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

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

3.295 Process Lifetime

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

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

3.296 Process Memory Locking

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

3.297 Process Termination

There are two kinds of process termination:

1. Normal termination occurs by a return from *main*( ) or when requested with the *exit*( ) or *_exit*( ) functions.

2. Abnormal termination occurs when requested by the *abort*( ) function or when some signals are received.

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

3.298 Process-To-Process Communication

The transfer of data between processes.

3.299 Process Virtual Time

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

3.300 Program

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

3.301 Protocol

A set of semantic and syntactic rules for exchanging information.

3.302 Pseudo-Terminal

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

3.303 Radix Character

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

3.304 Read-Only File System

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

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

3.305 Read-Write Lock

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

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

3.306 Real Group ID

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

3.307 Real Time

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

3.308 Realtime Signal Extension

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

3.309 Real User ID

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

3.310 Record

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

3.311 Redirection

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

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

3.312 Redirection Operator

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

```
<   >   >|   <<   >>   <&   >&   <<–   <>
```

3.313 Reentrant Function

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

3.314 Referenced Shared Memory Object

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

3.315 Refresh

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

3.316 Regular Expression

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

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

3.317 Region

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

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

3.318 Regular File

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

3.319 Relative Pathname

A pathname not beginning with a slash.

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

3.320 Relocatable File

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

3.321 Relocation

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

3.322 Requested Batch Service

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

3.323 (Time) Resolution

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

3.324 Root Directory

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

3.325 Runnable Process (or Thread)

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

3.326 Running Process (or Thread)

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

3.327 Saved Resource Limits

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

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

3.328 Saved Set-Group-ID

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

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

3.329 Saved Set-User-ID

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

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

3.330 Scheduling

The application of a policy to select a runnable process or thread to become a running process or thread, or to alter one or more of the thread lists.

3.331 Scheduling Allocation Domain

The set of processors on which an individual thread can be scheduled at any given time.

3.332 Scheduling Contention Scope

A property of a thread that defines the set of threads against which that thread competes for resources.

For example, in a scheduling decision, threads sharing scheduling contention scope compete for processor resources. In IEEE Std 1003.1-200x, a thread has scheduling contention scope of either PTHREAD_SCOPE_SYSTEM or PTHREAD_SCOPE_PROCESS.

3.333 Scheduling Policy

A set of rules that is used to determine the order of execution of processes or threads to achieve some goal.

Note: Scheduling Policy is defined in detail in Section 4.13 (on page 99).

3.334 Screen

A rectangular region of columns and lines on a terminal display. A screen may be a portion of a physical display device or may occupy the entire physical area of the display device.

3.335 Scroll

To move the representation of data vertically or horizontally relative to the terminal screen. There are two types of scrolling:

1. The cursor moves with the data.
2. The cursor remains stationary while the data moves.

3.336 Semaphore

A minimum synchronization primitive to serve as a basis for more complex synchronization mechanisms to be defined by the application program.

Note: Semaphores are defined in detail in Section 4.15 (on page 100).

3.337 Session

A collection of process groups established for job control purposes. Each process group is a member of a session. A process is considered to be a member of the session of which its process group is a member. A newly created process joins the session of its creator. A process can alter its session membership; see *setsid*( ). There can be multiple process groups in the same session.

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

3.338 Session Leader

A process that has created a session.

Note: For further information, see the *setsid*( ) function defined in the System Interfaces volume of IEEE Std 1003.1-200x.

3.339 Session Lifetime

The period between when a session is created and the end of the lifetime of all the process groups that remain as members of the session.

3.340 Shared Memory Object

An object that represents memory that can be mapped concurrently into the address space of more than one process.

3.341 Shell

A program that interprets sequences of text input as commands. It may operate on an input stream or it may interactively prompt and read commands from a terminal.

3.342 Shell, the

The Shell Command Language Interpreter; a specific instance of a shell.

Note: For further information, see the *sh* utility defined in the Shell and Utilities volume of IEEE Std 1003.1-200x.

3.343 Shell Script

A file containing shell commands. If the file is made executable, it can be executed by specifying its name as a simple command. Execution of a shell script causes a shell to execute the commands within the script. Alternatively, a shell can be requested to execute the commands in a shell script by specifying the name of the shell script as the operand to the *sh* utility.

Note: Simple Commands are defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.9.1, Simple Commands.

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

3.344 Signal

A mechanism by which a process or thread may be notified of, or affected by, an event occurring in the system. Examples of such events include hardware exceptions and specific actions by processes. The term signal is also used to refer to the event itself.

3.345 Signal Stack

Memory established for a thread, in which signal handlers catching signals sent to that thread are executed.

3.346 Single-Quote

The character `'''`, also known as *apostrophe*.

3.347 Slash

The character `'/'`, also known as *solidus*.

3.348 Socket

A file of a particular type that is used as a communications endpoint for process-to-process communication as described in the System Interfaces volume of IEEE Std 1003.1-200x.

3.349 Socket Address

An address associated with a socket or remote endpoint, including an address family identifier and addressing information specific to that address family. The address may include multiple parts, such as a network address associated with a host system and an identifier for a specific endpoint.

3.350 Soft Limit

A resource limitation established for each process that the process may set to any value less than or equal to the hard limit.

3.351 Source Code

When dealing with the Shell Command Language, input to the command language interpreter. The term shell script is synonymous with this meaning.

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

Source code also refers to the input statements prepared for the following standard utilities: *awk*, *bc*, *ed*, *lex*, *localedef*, *make*, *sed*, and *yacc*.

Source code can also refer to a collection of sources meeting any or all of these meanings.

Note: The *awk*, *bc*, *ed*, *lex*, *localedef*, *make*, *sed*, and *yacc* utilities are defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-200x.

3.352 Space Character (<space>)

The character defined in the portable character set as <space>. The <space> is a member of the **space** character class of the current locale, but represents the single character, and not all of the possible members of the class; see also Section 3.431 (on page 92).

3.353 Spawn

A process creation primitive useful for systems that have difficulty with *fork*( ) and as an efficient replacement for *fork*( )/*exec*.

3.354 Special Built-In

See *Built-In Utility* in Section 3.83 (on page 44).

3.355 Special Parameter

In the shell command language, a parameter named by a single character from the following list:

```
* @ # ? ! - $ 0
```

Note: For further information, see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.5.2, Special Parameters.

3.356 Spin Lock

A synchronization object used to allow multiple threads to serialize their access to shared data.

3.357 Sporadic Server

A scheduling policy for threads and processes that reserves a certain amount of execution capacity for processing aperiodic events at a given priority level.

3.358 Standard Error

An output stream usually intended to be used for diagnostic messages.

3.359 Standard Input

An input stream usually intended to be used for primary data input.

3.360 Standard Output

An output stream usually intended to be used for primary data output.

3.361 Standard Utilities

The utilities described in the Shell and Utilities volume of IEEE Std 1003.1-200x.

3.362 Stream

Appearing in lowercase, a stream is a file access object that allows access to an ordered sequence of characters, as described by the ISO C standard. Such objects can be created by the *fdopen*( ), *fopen*( ), or *popen*( ) functions, and are associated with a file descriptor. A stream provides the additional services of user-selectable buffering and formatted input and output; see also Section 3.363.

Note: For further information, see the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.5, Standard I/O Streams.

The *fdopen*( ), *fopen*( ), or *popen*( ) functions are defined in detail in the System Interfaces volume of IEEE Std 1003.1-200x.

3.363 STREAM

Appearing in uppercase, STREAM refers to a full duplex connection between a process and an open device or pseudo-device. It optionally includes one or more intermediate processing modules that are interposed between the process end of the STREAM and the device driver (or pseudo-device driver) end of the STREAM; see also Section 3.362.

Note: For further information, see the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.6, STREAMS.

3.364 STREAM End

The STREAM end is the driver end of the STREAM and is also known as the downstream end of the STREAM.

3.365 STREAM Head

The STREAM head is the beginning of the STREAM and is at the boundary between the system and the application process. This is also known as the upstream end of the STREAM.

3.366 STREAMS Multiplexor

A driver with multiple STREAMS connected to it. Multiplexing with STREAMS connected above is referred to as N-to-1, or *upper multiplexing*. Multiplexing with STREAMS connected below is referred to as 1-to-N or *lower multiplexing*.

3.367 String

A contiguous sequence of bytes terminated by and including the first null byte.

3.368 Subshell

A shell execution environment, distinguished from the main or current shell execution environment.

Note: For further information, see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.12, Shell Execution Environment.

3.369 Successfully Transferred

For a write operation to a regular file, when the system ensures that all data written is readable on any subsequent open of the file (even one that follows a system or power failure) in the absence of a failure of the physical storage medium.

For a read operation, when an image of the data on the physical storage medium is available to the requesting process.

3.370 Supplementary Group ID

An attribute of a process used in determining file access permissions. A process has up to {NGROUPS_MAX} supplementary group IDs in addition to the effective group ID. The supplementary group IDs of a process are set to the supplementary group IDs of the parent process when the process is created.

3.371 Suspended Job

A job that has received a SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU signal that caused the process group to stop. A suspended job is a background job, but a background job is not necessarily a suspended job.

3.372 Symbolic Link

A type of file with the property that when the file is encountered during pathname resolution, a string stored by the file is used to modify the pathname resolution. The stored string has a length of {SYMLINK_MAX} bytes or fewer.

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

3.373 Synchronized Input and Output

A determinism and robustness improvement mechanism to enhance the data input and output mechanisms, so that an application can ensure that the data being manipulated is physically present on secondary mass storage devices.

3.374 Synchronized I/O Completion

The state of an I/O operation that has either been successfully transferred or diagnosed as unsuccessful.

3.375 Synchronized I/O Data Integrity Completion

For read, when the operation has been completed or diagnosed if unsuccessful. The read is complete only when an image of the data has been successfully transferred to the requesting process. If there were any pending write requests affecting the data to be read at the time that the synchronized read operation was requested, these write requests are successfully transferred prior to reading the data.

For write, when the operation has been completed or diagnosed if unsuccessful. The write is complete only when the data specified in the write request is successfully transferred and all file system information required to retrieve the data is successfully transferred.

File attributes that are not necessary for data retrieval (access time, modification time, status change time) need not be successfully transferred prior to returning to the calling process.

3.376 Synchronized I/O File Integrity Completion

Identical to a synchronized I/O data integrity completion with the addition that all file attributes relative to the I/O operation (including access time, modification time, status change time) are successfully transferred prior to returning to the calling process.

3.377 Synchronized I/O Operation

An I/O operation performed on a file that provides the application assurance of the integrity of its data and files.

3.378 Synchronous I/O Operation

An I/O operation that causes the thread requesting the I/O to be blocked from further use of the processor until that I/O operation completes.

Note: A synchronous I/O operation does not imply synchronized I/O data integrity completion or synchronized I/O file integrity completion.

3.379 Synchronously-Generated Signal

A signal that is attributable to a specific thread.

For example, a thread executing an illegal instruction or touching invalid memory causes a synchronously-generated signal. Being synchronous is a property of how the signal was generated and not a property of the signal number.

3.380 System

An implementation of IEEE Std 1003.1-200x.

3.381 System Crash

An interval initiated by an unspecified circumstance that causes all processes (possibly other than special system processes) to be terminated in an undefined manner, after which any changes to the state and contents of files created or written to by an application prior to the interval are undefined, except as required elsewhere in IEEE Std 1003.1-200x.

3.382 System Console

An implementation-defined device that receives messages sent by the *syslog*( ) function, and the *fmtmsg*( ) function when the MM_CONSOLE flat is set.

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

3.383 System Databases

An implementation provides two system databases.

The *group database* contains the following information for each group:

1. Group name
2. Numerical group ID
3. List of all users allowed in the group

The *user database* contains the following information for each user:

1. User name
2. Numerical user ID
3. Numerical group ID
4. Initial working directory
5. Initial user program

If the initial user program field is null, the system default is used. If the initial working directory field is null, the interpretation of that field is implementation-defined. These databases may contain other fields that are unspecified by IEEE Std 1003.1-200x.

3.384 System Documentation

All documentation provided with an implementation except for the conformance document. Electronically distributed documents for an implementation are considered part of the system documentation.

3.385 System Process

An implementation-defined object, other than a process executing an application, that has a process ID.

3.386 System Reboot

An implementation-defined sequence of events that may result in the loss of transitory data; that is, data that is not saved in permanent storage. For example, message queues, shared memory, semaphores, and processes.

3.387 System Trace Event

A trace event that is generated by the implementation, in response either to a system-initiated action or to an application-requested action, except for a call to *posix_trace_event*( ). When supported by the implementation, a system-initiated action generates a process-independent system trace event and an application-requested action generates a process-dependent system trace event. For a system trace event not defined by IEEE Std 1003.1-200x, the associated trace event type identifier is derived from the implementation-defined name for this trace event, and the associated data is of implementation-defined content and length.

3.388 System-Wide

Pertaining to events occurring in all processes existing in an implementation at a given point in time.

3.389 Tab Character (<tab>)

A character that in the output stream indicates that printing or displaying should start at the next horizontal tabulation position on the current line. It is the character designated by `'\t'` in the C language. If the current position is at or past the last defined horizontal tabulation position, the behavior is unspecified. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the tabulation.

3.390 Terminal (or Terminal Device)

A character special file that obeys the specifications of the general terminal interface.

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

3.391 Text Column

A roughly rectangular block of characters capable of being laid out side-by-side next to other text columns on an output page or terminal screen. The widths of text columns are measured in column positions.

3.392 Text File

A file that contains characters organized into one or more lines. The lines do not contain NUL characters and none can exceed {LINE_MAX} bytes in length, including the <newline>. Although IEEE Std 1003.1-200x does not distinguish between text files and binary files (see the ISO C standard), many utilities only produce predictable or meaningful output when operating on text files. The standard utilities that have such restrictions always specify *text files* in their

STDIN or INPUT FILES sections.

3.393 Thread

A single flow of control within a process. Each thread has its own thread ID, scheduling priority and policy, *errno* value, thread-specific key/value bindings, and the required system resources to support a flow of control. Anything whose address may be determined by a thread, including but not limited to static variables, storage obtained via *malloc*( ), directly addressable storage obtained through implementation-defined functions, and automatic variables, are accessible to all threads in the same process.

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

3.394 Thread ID

Each thread in a process is uniquely identified during its lifetime by a value of type **pthread_t** called a thread ID.

3.395 Thread List

An ordered set of runnable threads that all have the same ordinal value for their priority.

The ordering of threads on the list is determined by a scheduling policy or policies. The set of thread lists includes all runnable threads in the system.

3.396 Thread-Safe

A function that may be safely invoked concurrently by multiple threads. Each function defined in the System Interfaces volume of IEEE Std 1003.1-200x is thread-safe unless explicitly stated otherwise. Examples are any ‘‘pure’’ function, a function which holds a mutex locked while it is accessing static storage, or objects shared among threads.

3.397 Thread-Specific Data Key

A process global handle of type **pthread_key_t** which is used for naming thread-specific data.

Although the same key value may be used by different threads, the values bound to the key by *pthread_setspecific*( ) and accessed by *pthread_getspecific*( ) are maintained on a per-thread basis and persist for the life of the calling thread.

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

3.398 Tilde

The character '~'.

3.399 Timeouts

A method of limiting the length of time an interface will block; see also Section 3.76 (on page 43).

3.400 Timer

A mechanism that can notify a thread when the time as measured by a particular clock has reached or passed a specified value, or when a specified amount of time has passed.

3.401 Timer Overrun

A condition that occurs each time a timer, for which there is already an expiration signal queued to the process, expires.

3.402 Token

In the shell command language, a sequence of characters that the shell considers as a single unit when reading input. A token is either an operator or a word.

Note: The rules for reading input are defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.3, Token Recognition.

3.403 Trace Analyzer Process

A process that extracts trace events from a trace stream to retrieve information about the behavior of an application.

3.404 Trace Controller Process

A process that creates a trace stream for tracing a process.

3.405 Trace Event

A data object that represents an action executed by the system, and that is recorded in a trace stream.

3.406 Trace Event Type

A data object type that defines a class of trace event.

3.407 Trace Event Type Mapping

A one-to-one mapping between trace event types and trace event names.

3.408 Trace Filter

A filter that allows the trace controller process to specify those trace event types that are to be ignored; that is, not generated.

3.409 Trace Generation Version

A data object that is an implementation-defined character string, generated by the trace system and describing the origin and version of the trace system.

3.410 Trace Log

The flushed image of a trace stream, if the trace stream is created with a trace log.

3.411 Trace Point

An action that may cause a trace event to be generated.

3.412 Trace Stream

An opaque object that contains trace events plus internal data needed to interpret those trace events.

3.413 Trace Stream Identifier

A handle to manage tracing operations in a trace stream.

3.414 Trace System

A system that allows both system and user trace events to be generated into a trace stream. These trace events can be retrieved later.

3.415 Traced Process

A process for which at least one trace stream has been created. A traced process is also called a target process.

3.416 Tracing Status of a Trace Stream

A status that describes the state of an active trace stream. The tracing status of a trace stream can be retrieved from the trace stream attributes. An active trace stream can be in one of two states: running or suspended.

3.417 Typed Memory Name Space

A system-wide name space that contains the names of the typed memory objects present in the system. It is configurable for a given implementation.

3.418 Typed Memory Object

A combination of a typed memory pool and a typed memory port. The entire contents of the pool are accessible from the port. The typed memory object is identified through a name that belongs to the typed memory name space.

3.419 Typed Memory Pool

An extent of memory with the same operational characteristics. Typed memory pools may be contained within each other.

3.420 Typed Memory Port

A hardware access path to one or more typed memory pools.

3.421 Unbind

Remove the association between a network address and an endpoint.

3.422 Unit Data

See *Datagram* in Section 3.123 (on page 49).

3.423 Upshifting

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

3.424 User Database

A system database of implementation-defined format that contains at least the following information for each user ID:

- User name
- Numerical user ID
- Initial numerical group ID
- Initial working directory
- Initial user program

The initial numerical group ID is used by the *newgrp* utility. Any other circumstances under which the initial values are operative are implementation-defined.

If the initial user program field is null, an implementation-defined program is used.

If the initial working directory field is null, the interpretation of that field is implementation-defined.

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

3.425 User ID

A non-negative integer that is used to identify a system user. When the identity of a user is associated with a process, a user ID value is referred to as a real user ID, an effective user ID, or a saved set-user-ID.

3.426 User Name

A string that is used to identify a user; see also Section 3.424 (on page 90). To be portable across systems conforming to IEEE Std 1003.1-200x, the value is composed of characters from the portable filename character set. The hyphen should not be used as the first character of a portable user name.

3.427 User Trace Event

A trace event that is generated explicitly by the application as a result of a call to *posix_trace_event*( ).

3.428 Utility

A program, excluding special built-in utilities provided as part of the Shell Command Language, that can be called by name from a shell to perform a specific task, or related set of tasks.

Note: For further information on special built-in utilities, see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.14, Special Built-In Utilities.

3.429 Variable

In the shell command language, a named parameter.

Note: For further information, see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.5, Parameters and Variables.

3.430 Vertical-Tab Character (<vertical-tab>)

A character that in the output stream indicates that printing should start at the next vertical tabulation position. It is the character designated by '\v' in the C language. If the current position is at or past the last defined vertical tabulation position, the behavior is unspecified. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the tabulation.

3.431 White Space

A sequence of one or more characters that belong to the **space** character class as defined via the *LC_CTYPE* category in the current locale.

In the POSIX locale, white space consists of one or more <blank>s (<space>s and <tab>s), <newline>s, <carriage-return>s, <form-feed>s, and <vertical-tab>s.

3.432 Wide-Character Code (C Language)

An integer value corresponding to a single graphic symbol or control code.

Note: C Language Wide-Character Codes are defined in detail in Section 6.3 (on page 115).

3.433 Wide-Character Input/Output Functions

The functions that perform wide-oriented input from streams or wide-oriented output to streams: *fgetwc*( ), *fputwc*( ), *fputws*( ), *fwprintf*( ), *fwscanf*( ), *getwc*( ), *getwchar*( ), *getws*( ), *putwc*( ), *putwchar*( ), *ungetwc*( ), *vfwprintf*( ), *vwprintf*( ), *wprintf*( ), and *wscanf*( ).

Note: These functions are defined in detail in the System Interfaces volume of IEEE Std 1003.1-200x.

3.434 Wide-Character String

A contiguous sequence of wide-character codes terminated by and including the first null wide-character code.

3.435 Word

In the shell command language, a token other than an operator. In some cases a word is also a portion of a word token: in the various forms of parameter expansion, such as ${*name*–*word*}, and variable assignment, such as *name*=*word*, the word is the portion of the token depicted by *word*. The concept of a word is no longer applicable following word expansions—only fields remain.

Note: For further information, see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.6.2, Parameter Expansion and the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.6, Word Expansions.

3.436 Working Directory (or Current Working Directory)

A directory, associated with a process, that is used in pathname resolution for pathnames that do not begin with a slash.

3.437 Worldwide Portability Interface

Functions for handling characters in a codeset-independent manner.

3.438 Write

To output characters to a file, such as standard output or standard error. Unless otherwise stated, standard output is the default output destination for all uses of the term write; see the distinction between display and write in Section 3.132 (on page 50).

3.439 XSI

The X/Open System Interface is the core application programming interface for C and *sh* programming for systems conforming to the Single UNIX Specification. This is a superset of the mandatory requirements for conformance to IEEE Std 1003.1-200x.

3.440 XSI-Conformant

A system which allows an application to be built using a set of services that are consistent across all systems that conform to IEEE Std 1003.1-200x and that support the XSI extension.

Note: See also Chapter 2 (on page 15).

3.441 Zombie Process

A process that has terminated and that is deleted when its exit status has been reported to another process which is waiting for that process to terminate.

3.442 ±0

The algebraic sign provides additional information about any variable that has the value zero when the representation allows the sign to be determined.

Chapter 4

General Concepts

For the purposes of IEEE Std 1003.1-200x, the general concepts given in Chapter 4 apply.

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

4.1 Concurrent Execution

Functions that suspend the execution of the calling thread shall not cause the execution of other threads to be indefinitely suspended.

4.2 Directory Protection

If a directory is writable and the mode bit S_ISVTX is set on the directory, a process may remove or rename files within that directory only if one or more of the following is true:

- The effective user ID of the process is the same as that of the owner ID of the file.
- The effective user ID of the process is the same as that of the owner ID of the directory.
- The process has appropriate privileges.

If the S_ISVTX bit is set on a non-directory file, the behavior is unspecified.

4.3 Extended Security Controls

An implementation may provide implementation-defined extended security controls (see Section 3.159 (on page 54)). These permit an implementation to provide security mechanisms to implement different security policies than those described in IEEE Std 1003.1-200x. These mechanisms shall not alter or override the defined semantics of any of the interfaces in IEEE Std 1003.1-200x.

4.4 File Access Permissions

The standard file access control mechanism uses the file permission bits, as described below.

Implementations may provide *additional* or *alternate* file access control mechanisms, or both. An additional access control mechanism shall only further restrict the access permissions defined by the file permission bits. An alternate file access control mechanism shall:

- Specify file permission bits for the file owner class, file group class, and file other class of that file, corresponding to the access permissions.
- Be enabled only by explicit user action, on a per-file basis by the file owner or a user with the appropriate privilege.
- Be disabled for a file after the file permission bits are changed for that file with *chmod*( ). The disabling of the alternate mechanism need not disable any additional mechanisms supported

by an implementation.

Whenever a process requests file access permission for read, write, or execute/search, if no additional mechanism denies access, access shall be determined as follows:

- If a process has the appropriate privilege:
  - — If read, write, or directory search permission is requested, access shall be granted.
  - — If execute permission is requested, access shall be granted if execute permission is granted to at least one user by the file permission bits or by an alternate access control mechanism; otherwise, access shall be denied.
- Otherwise:
  - — The file permission bits of a file contain read, write, and execute/search permissions for the file owner class, file group class, and file other class.
  - — Access shall be granted if an alternate access control mechanism is not enabled and the requested access permission bit is set for the class (file owner class, file group class, or file other class) to which the process belongs, or if an alternate access control mechanism is enabled and it allows the requested access; otherwise, access shall be denied.

4.5 File Hierarchy

Files in the system are organized in a hierarchical structure in which all of the non-terminal nodes are directories and all of the terminal nodes are any other type of file. Since multiple directory entries may refer to the same file, the hierarchy is properly described as a *directed graph*.

4.6 Filenames

For a filename to be portable across implementations conforming to IEEE Std 1003.1-200x, it shall consist only of the portable filename character set as defined in Section 3.276 (on page 70).

The hyphen character shall not be used as the first character of a portable filename. Uppercase and lowercase letters shall retain their unique identities between conforming implementations. In the case of a portable pathname, the slash character may also be used.

4.7 File Times Update

Each file has three distinct associated time values: *st_atime*, *st_mtime*, and *st_ctime*. The *st_atime* field is associated with the times that the file data is accessed; *st_mtime* is associated with the times that the file data is modified; and *st_ctime* is associated with the times that the file status is changed. These values are returned in the file characteristics structure, as described in **<sys/stat.h>**.

Each function or utility in IEEE Std 1003.1-200x that reads or writes data or changes file status indicates which of the appropriate time-related fields shall be ''marked for update''. If an implementation of such a function or utility marks for update a time-related field not specified by IEEE Std 1003.1-200x, this shall be documented, except that any changes caused by pathname resolution need not be documented. For the other functions or utilities in IEEE Std 1003.1-200x (those that are not explicitly required to read or write file data or change file status, but that in some implementations happen to do so), the effect is unspecified.

An implementation may update fields that are marked for update immediately, or it may update such fields periodically. At an update point in time, any marked fields shall be set to the current time and the update marks shall be cleared. All fields that are marked for update shall be updated when the file ceases to be open by any process, or when a *stat*( ), *fstat*( ), or *lstat*( ) is performed on the file. Other times at which updates are done are unspecified. Marks for update, and updates themselves, are not done for files on read-only file systems; see Section 3.304 (on page 74).

4.8 Host and Network Byte Orders

When data is transmitted over the network, it is sent as a sequence of octets (8-bit unsigned values). If an entity (such as an address or a port number) can be larger than 8 bits, it needs to be stored in several octets. The convention is that all such values are stored with 8 bits in each octet, and with the first (lowest-addressed) octet holding the most-significant bits. This is called ‘‘network byte order’’.

Network byte order may not be convenient for processing actual values. For this, it is more sensible for values to be stored as ordinary integers. This is known as ‘‘host byte order’’. In host byte order:

- The most significant bit might not be stored in the first byte in address order.
- Bits might not be allocated to bytes in any obvious order at all.

8-bit values stored in **uint8_t** objects do not require conversion to or from host byte order, as they have the same representation. 16 and 32-bit values can be converted using the *htonl*( ), *htons*( ), *ntohl*( ), and *ntohs*( ) functions. When reading data that is to be converted to host byte order, it should either be received directly into a **uint16_t** or **uint32_t** object or should be copied from an array of bytes using *memcpy*( ) or similar. Passing the data through other types could cause the byte order to be changed. Similar considerations apply when sending data.

4.9 Measurement of Execution Time

The mechanism used to measure execution time shall be implementation-defined. The implementation shall also define to whom the CPU time that is consumed by interrupt handlers and system services on behalf of the operating system will be charged. See Section 3.117 (on page 49).

4.10 Memory Synchronization

Applications shall ensure that access to any memory location by more than one thread of control (threads or processes) is restricted such that no thread of control can read or modify a memory location while another thread of control may be modifying it. Such access is restricted using functions that synchronize thread execution and also synchronize memory with respect to other threads. The following functions synchronize memory with respect to other threads:

fork( )
pthread_barrier_wait( )
pthread_cond_broadcast( )
pthread_cond_signal( )
pthread_cond_timedwait( )
pthread_cond_wait( )
pthread_create( )
pthread_join( )
pthread_mutex_lock( )
pthread_mutex_timedlock( )
pthread_mutex_trylock( )
pthread_mutex_unlock( )
pthread_spin_lock( )
pthread_spin_trylock( )
pthread_spin_unlock( )
pthread_rwlock_rdlock( )
pthread_rwlock_timedrdlock( )
pthread_rwlock_timedwrlock( )
pthread_rwlock_tryrdlock( )
pthread_rwlock_trywrlock( )
pthread_rwlock_unlock( )
pthread_rwlock_wrlock( )
sem_post( )
sem_trywait( )
sem_wait( )
wait( )
waitpid( )

Unless explicitly stated otherwise, if one of the above functions returns an error, it is unspecified whether the invocation causes memory to be synchronized.

Applications may allow more than one thread of control to read a memory location simultaneously.

4.11 Pathname Resolution

Pathname resolution is performed for a process to resolve a pathname to a particular file in a file hierarchy. There may be multiple pathnames that resolve to the same file.

Each filename in the pathname is located in the directory specified by its predecessor (for example, in the pathname fragment **a/b**, file **b** is located in directory **a**). Pathname resolution shall fail if this cannot be accomplished. If the pathname begins with a slash, the predecessor of the first filename in the pathname shall be taken to be the root directory of the process (such pathnames are referred to as *absolute pathnames*). If the pathname does not begin with a slash, the predecessor of the first filename of the pathname shall be taken to be the current working directory of the process (such pathnames are referred to as *relative pathnames*).

The interpretation of a pathname component is dependent on the value of {NAME_MAX} and _POSIX_NO_TRUNC associated with the path prefix of that component. If any pathname component is longer than {NAME_MAX}, the implementation shall consider this an error.

A pathname that contains at least one non-slash character and that ends with one or more trailing slashes shall be resolved as if a single dot character (`'.'`) were appended to the pathname.

If a symbolic link is encountered during pathname resolution, the behavior shall depend on whether the pathname component is at the end of the pathname and on the function being performed. If all of the following are true, then pathname resolution is complete:

1. This is the last pathname component of the pathname.
2. The pathname has no trailing slash.
3. The function is required to act on the symbolic link itself, or certain arguments direct that the function act on the symbolic link itself.

In all other cases, the system shall prefix the remaining pathname, if any, with the contents of the symbolic link. If the combined length exceeds {PATH_MAX}, and the implementation considers this to be an error, *errno* shall be set to [ENAMETOOLONG] and an error indication shall be returned. Otherwise, the resolved pathname shall be the resolution of the pathname just created. If the resulting pathname does not begin with a slash, the predecessor of the first filename of the pathname is taken to be the directory containing the symbolic link.

If the system detects a loop in the pathname resolution process, it shall set *errno* to [ELOOP] and return an error indication. The same may happen if during the resolution process more symbolic links were followed than the implementation allows. This implementation-defined limit shall not be smaller than {SYMLOOP_MAX}.

The special filename dot shall refer to the directory specified by its predecessor. The special filename dot-dot shall refer to the parent directory of its predecessor directory. As a special case, in the root directory, dot-dot may refer to the root directory itself.

A pathname consisting of a single slash shall resolve to the root directory of the process. A null pathname shall not be successfully resolved. A pathname that begins with two successive slashes may be interpreted in an implementation-defined manner, although more than two leading slashes shall be treated as a single slash.

4.12 Process ID Reuse

A process group ID shall not be reused by the system until the process group lifetime ends.

A process ID shall not be reused by the system until the process lifetime ends. In addition, if there exists a process group whose process group ID is equal to that process ID, the process ID shall not be reused by the system until the process group lifetime ends. A process that is not a system process shall not have a process ID of 1.

4.13 Scheduling Policy

A scheduling policy affects process or thread ordering:

- When a process or thread is a running thread and it becomes a blocked thread
- When a process or thread is a running thread and it becomes a preempted thread
- When a process or thread is a blocked thread and it becomes a runnable thread
- When a running thread calls a function that can change the priority or scheduling policy of a process or thread
- In other scheduling policy-defined circumstances

Conforming implementations shall define the manner in which each of the scheduling policies may modify the priorities or otherwise affect the ordering of processes or threads at each of the occurrences listed above. Additionally, conforming implementations shall define in what other circumstances and in what manner each scheduling policy may modify the priorities or affect the ordering of processes or threads.

4.14 Seconds Since the Epoch

A value that approximates the number of seconds that have elapsed since the Epoch. A Coordinated Universal Time name (specified in terms of seconds (*tm_sec*), minutes (*tm_min*), hours (*tm_hour*), days since January 1 of the year (*tm_yday*), and calendar year minus 1900 (*tm_year*)) is related to a time represented as seconds since the Epoch, according to the expression below.

If the year is <1970 or the value is negative, the relationship is undefined. If the year is ≥1970 and the value is non-negative, the value is related to a Coordinated Universal Time name according to the C-language expression, where *tm_sec*, *tm_min*, *tm_hour*, *tm_yday*, and *tm_year* are all integer types:

```
tm_sec + tm_min*60 + tm_hour*3600 + tm_yday*86400 +
    (tm_year–70)*31536000 + ((tm_year–69)/4)*86400 –
    ((tm_year–1)/100)*86400 + ((tm_year+299)/400)*86400
```

The relationship between the actual time of day and the current value for seconds since the Epoch is unspecified.

How any changes to the value of seconds since the Epoch are made to align to a desired relationship with the current actual time are made is implementation-defined. As represented in seconds since the Epoch, each and every day shall be accounted for by exactly 86 400 seconds.

Note: The last three terms of the expression add in a day for each year that follows a leap year starting with the first leap year since the Epoch. The first term adds a day every 4 years starting in 1973, the second subtracts a day back out every 100 years starting in 2001, and the third adds a day back in every 400 years starting in 2001. The divisions in the formula are integer divisions; that is, the remainder is discarded leaving only the integer quotient. |

4.15 Semaphore

A minimum synchronization primitive to serve as a basis for more complex synchronization mechanisms to be defined by the application program.

For the semaphores associated with the Semaphores option, a semaphore is represented as a shareable resource that has a non-negative integer value. When the value is zero, there is a (possibly empty) set of threads awaiting the availability of the semaphore.

For the semaphores associated with the X/Open System Interface Extension (XSI), a semaphore is a positive integer (0 through 32767). The *semget*( ) function can be called to create a set or array of semaphores. A semaphore set can contain one or more semaphores up to an implementation-defined value.

Semaphore Lock Operation

An operation that is applied to a semaphore. If, prior to the operation, the value of the semaphore is zero, the semaphore lock operation shall cause the calling thread to be blocked and added to the set of threads awaiting the semaphore; otherwise, the value shall be decremented. |

Semaphore Unlock Operation

An operation that is applied to a semaphore. If, prior to the operation, there are any threads in the set of threads awaiting the semaphore, then some thread from that set shall be removed from the set and becomes unblocked; otherwise, the semaphore value shall be incremented.

4.16 Thread-Safety

Refer to the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.9, Threads.

4.17 Tracing

The trace system allows a traced process to have a selection of events created for it. Traces consist of streams of trace event types.

A trace event type is identified on the one hand by a trace event type name, also referenced as a trace event name, and on the other hand by a trace event type identifier. A trace event name is a human-readable string. A trace event type identifier is an opaque identifier used by the trace system. There shall be a one-to-one relationship between trace event type identifiers and trace event names for a given trace stream and also for a given traced process. The trace event type identifier shall be generated automatically from a trace event name by the trace system either when a trace controller process invokes *posix_trace_trid_eventid_open*( ) or when an instrumented application process invokes *posix_trace_eventid_open*( ). Trace event type identifiers are used to filter trace event types, to allow interpretation of user data, and to identify the kind of trace point that generated a trace event.

Each trace event shall be of a particular trace event type, and associated with a trace event type identifier. The execution of a trace point shall generate a trace event if a trace stream has been created and started for the process that executed the trace point and if the corresponding trace event type identifier is not ignored by filtering.

A generated trace event shall be recorded in a trace stream, and optionally also in a trace log if a trace log is associated with the trace stream, except that:

- For a trace stream, if no resources are available for the event, the event is lost.
- For a trace log, if no resources are available for the event, or a flush operation does not succeed, the event is lost.

A trace event recorded in an active trace stream may be retrieved by an application having the appropriate privileges.

A trace event recorded in a trace log may be retrieved by an application having the appropriate privileges after opening the trace log as a pre-recorded trace stream, with the function *posix_trace_open*( ).

When a trace event is reported it is possible to retrieve the following:

- A trace event type identifier
- A timestamp
- The process ID of the traced process, if the trace event is process-dependent
- Any optional trace event data including its length

- If the Threads option is supported, the thread ID, if the trace event is process-dependent
- The program address at which the trace point was invoked

Trace events may be mapped from trace event types to trace event names. One such mapping shall be associated with each trace stream. An active trace stream is associated with a traced process, and also with its children if the Trace Inherit option is supported and also the inheritance policy is set to _POSIX_TRACE_INHERIT. Therefore each traced process has a mapping of the trace event names to trace event type identifiers that have been defined for that process.

Traces can be recorded into either trace streams or trace logs.

The implementation and format of a trace stream are unspecified. A trace stream need not be and generally is not persistent. A trace stream may be either active or pre-recorded:

- An active trace stream is a trace stream that has been created and has not yet been shut down. It can be of one of the two following classes:
  1. An active trace stream without a trace log that was created with the *posix_trace_create*( ) function
  2. If the Trace Log option is supported, an active trace stream with a trace log that was created with the *posix_trace_create_withlog*( ) function
- A pre-recorded trace stream is a trace stream that was opened from a trace log object using the *posix_trace_open*( ) function.

An active trace stream can loop. This behavior means that when the resources allocated by the trace system for the trace stream are exhausted, the trace system reuses the resources associated with the oldest recorded trace events to record new trace events.

If the Trace Log option is supported, an active trace stream with a trace log can be flushed. This operation causes the trace system to write trace events from the trace stream to the associated trace log, following the defined policies or using an explicit function call. After this operation, the trace system may reuse the resources associated with the flushed trace events.

An active trace stream with or without a trace log can be cleared. This operation shall cause all the resources associated with this trace stream to be reinitialized. The trace stream shall behave as if it was returning from its creation, except that the mapping of trace event type identifiers to trace event names shall not be cleared. If a trace log was associated with this trace stream, the trace log shall also be reinitialized.

A trace log shall be recorded when the *posix_trace_shutdown*( ) operation is invoked or during tracing, depending on the tracing strategy which is defined by a log policy. After the trace stream has been shut down, the trace information can be retrieved from the associated trace log using the same interface used to retrieve information from an active trace stream.

For a traced process, if the Trace Inherit option is supported and the trace stream's inheritance attribute is _POSIX_TRACE_INHERIT, the initial targeted traced process shall be traced together with all of its future children. The *posix_pid* member of each trace event in a trace stream shall be the process ID of the traced process.

Each trace point may be an implementation-defined action such as a context switch, or an application-programmed action such as a call to a specific operating system service (for example, *fork*( )) or a call to *posix_trace_event*( ).

Trace points may be filtered. The operation of the filter is to filter out (ignore) selected trace events. By default, no trace events are filtered.

The results of the tracing operations can be analyzed and monitored by a trace controller process or a trace analyzer process.

Only the trace controller process has control of the trace stream it has created. The control of the operation of a trace stream is done using its corresponding trace stream identifier. The trace controller process is able to:

- Initialize the attributes of a trace stream
- Create the trace stream
- Start and stop tracing
- Know the mapping of the traced process
- If the Trace Event Filter option is supported, filter the type of trace events to be recorded
- Shut the trace stream down

A traced process may also be a trace controller process. Only the trace controller process can control its trace stream(s). A trace stream created by a trace controller process shall be shut down if its controller process terminates or executes another file.

A trace controller process may also be a trace analyzer process. Trace analysis can be done concurrently with the traced process or can be done off-line, in the same or in a different platform.

4.18 Treatment of Error Conditions for Mathematical Functions

For all the functions in the **<math.h>** header, an application wishing to check for error situations should set *errno* to 0 and call *feclearexcept*(FE_ALL_EXCEPT) before calling the function. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

The following error conditions are defined for all functions in the **<math.h>** header.

4.18.1 Domain Error

A *domain error* shall occur if an input argument is outside the domain over which the mathematical function is defined. The description of each function lists any required domain errors; an implementation may define additional domain errors, provided that such errors are consistent with the mathematical definition of the function.

On a domain error, the function shall return an implementation-defined value; if the integer expression (math_errhandling & MATH_ERRNO) is non-zero, *errno* shall be set to [EDOM]; if the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, the ‘‘invalid’’ floating-point exception shall be raised.

4.18.2 Pole Error

A *pole error* occurs if the mathematical result of the function is an exact infinity (for example, log(0.0)).

On a pole error, the function shall return the value of the macro HUGE_VAL, HUGE_VALF, or HUGE_VALL according to the return type, with the same sign as the correct value of the function; if the integer expression (math_errhandling & MATH_ERRNO) is non-zero, *errno* shall be set to [ERANGE]; if the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, the ‘‘divide-by-zero’’ floating-point exception shall be raised.

4.18.3 Range Error

A *range error* shall occur if the finite mathematical result of the function cannot be represented in an object of the specified type, due to extreme magnitude.

4.18.3.1 Result Overflows

A floating result overflows if the magnitude of the mathematical result is finite but so large that the mathematical result cannot be represented without extraordinary roundoff error in an object of the specified type. If a floating result overflows and default rounding is in effect, then the function shall return the value of the macro HUGE_VAL, HUGE_VALF, or HUGE_VALL according to the return type, with the same sign as the correct value of the function; if the integer expression (math_errhandling & MATH_ERRNO) is non-zero, *errno* shall be set to [ERANGE]; if the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, the ‘‘overflow’’ floating-point exception shall be raised.

4.18.3.2 Result Underflows

The result underflows if the magnitude of the mathematical result is so small that the mathematical result cannot be represented, without extraordinary roundoff error, in an object of the specified type. If the result underflows, the function shall return an implementation-defined value whose magnitude is no greater than the smallest normalized positive number in the specified type; if the integer expression (math_errhandling & MATH_ERRNO) is non-zero, whether *errno* is set to [ERANGE] is implementation-defined; if the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, whether the ‘‘underflow’’ floating-point exception is raised is implementation-defined.

4.19 Treatment of NaN Arguments for the Mathematical Functions

For functions called with a NaN argument, no errors shall occur and a NaN shall be returned, except where stated otherwise.

If a function with one or more NaN arguments returns a NaN result, the result should be the same as one of the NaN arguments (after possible type conversion), except perhaps for the sign.

On implementations that support the IEC 60559:1989 standard floating point, functions with signaling NaN argument(s) shall be treated as if the function were called with an argument that is a required domain error and shall return a quiet NaN result, except where stated otherwise.

Note: The function might never see the signaling NaN, since it might trigger when the arguments are evaluated during the function call.

On implementations that support the IEC 60559:1989 standard floating point, for those functions that do not have a documented domain error, the following shall apply:

These functions shall fail if:

Domain Error Any argument is a signaling NaN.

Either, the integer expression (math_errhandling & MATH_ERRNO) is non-zero and *errno* shall be set to [EDOM], or the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero and the invalid floating-point exception shall be raised.

4.20 Utility

A utility program shall be either an executable file, such as might be produced by a compiler or linker system from computer source code, or a file of shell source code, directly interpreted by the shell. The program may have been produced by the user, provided by the system implementor, or acquired from an independent distributor.

The system may implement certain utilities as shell functions (see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.9.5, Function Definition Command) or built-in utilities, but only an application that is aware of the command search order described in the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.9.1.1, Command Search and Execution or of performance characteristics can discern differences between the behavior of such a function or built-in utility and that of an executable file.

4.21 Variable Assignment

In the shell command language, a word consisting of the following parts:

```
varname=value
```

When used in a context where assignment is defined to occur and at no other time, the *value* (representing a word or field) shall be assigned as the value of the variable denoted by *varname*.

Note: For further information, see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.9.1, Simple Commands.

The *varname* and *value* parts shall meet the requirements for a name and a word, respectively, except that they are delimited by the embedded unquoted equals-sign, in addition to other delimiters.

Note: Additional delimiters are described in the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.3, Token Recognition.

When a variable assignment is done, the variable shall be created if it did not already exist. If *value* is not specified, the variable shall be given a null value.

Note: An alternative form of variable assignment:

```
symbol=value
```

(where *symbol* is a valid word delimited by an equals-sign, but not a valid name) produces unspecified results. The form *symbol*=*value* is used by the KornShell *name*[*expression*]=*value* syntax.

Chapter 5

File Format Notation

The STDIN, STDOUT, STDERR, INPUT FILES, and OUTPUT FILES sections of the utility descriptions use a syntax to describe the data organization within the files, when that organization is not otherwise obvious. The syntax is similar to that used by the System Interfaces volume of IEEE Std 1003.1-200x *printf*( ) function, as described in this chapter. When used in STDIN or INPUT FILES sections of the utility descriptions, this syntax describes the format that could have been used to write the text to be read, not a format that could be used by the System Interfaces volume of IEEE Std 1003.1-200x *scanf*( ) function to read the input file.

The description of an individual record is as follows:

```
"<format>", [<arg1>, <arg2>,..., <argn>]
```

The *format* is a character string that contains three types of objects defined below:

1. *Characters* that are not *escape sequences* or *conversion specifications*, as described below, shall be copied to the output.
2. *Escape Sequences* represent non-graphic characters.
3. *Conversion Specifications* specify the output format of each argument; (see below).

The following characters have the following special meaning in the format string:

' ' (An empty character position.) Represents one or more <blank>s.

Δ Represents exactly one <space>.

Table 5-1 lists escape sequences and associated actions on display devices capable of the action.

Table 5-1 Escape Sequences and Associated Actions

Escape Sequence	Represents Character	Terminal Action
`'\\'`	backslash	Print the character `'\'`.
`'\a'`	alert	Attempt to alert the user through audible or visible notification.
`'\b'`	backspace	Move the printing position to one column before the current position, unless the current position is the start of a line.
`'\f'`	form-feed	Move the printing position to the initial printing position of the next logical page.
`'\n'`	newline	Move the printing position to the start of the next line.
`'\r'`	carriage-return	Move the printing position to the start of the current line.
`'\t'`	tab	Move the printing position to the next tab position on the current line. If there are no more tab positions remaining on the line, the behavior is undefined.
`'\v'`	vertical-tab	Move the printing position to the start of the next vertical tab position. If there are no more vertical tab positions left on the page, the behavior is undefined.

Each conversion specification is introduced by the percent-sign character (`'%'`). After the character `'%'`, the following shall appear in sequence:

flags Zero or more *flags*, in any order, that modify the meaning of the conversion specification.

field width An optional string of decimal digits to specify a minimum *field width*. For an output field, if the converted value has fewer bytes than the field width, it shall be padded on the left (or right, if the left-adjustment flag (`'–'`), described below, has been given) to the field width.

precision Gives the minimum number of digits to appear for the `d`, `o`, `i`, `u`, `x`, or `X` conversion specifiers (the field is padded with leading zeros), the number of digits to appear after the radix character for the `e` and `f` conversion specifiers, the maximum number of significant digits for the `g` conversion specifier; or the maximum number of bytes to be written from a string in the `s` conversion specifier. The precision shall take the form of a period (`'.'`) followed by a decimal digit string; a null digit string is treated as zero.

conversion specifier characters
A conversion specifier character (see below) that indicates the type of conversion to be applied.

The *flag* characters and their meanings are:

– The result of the conversion shall be left-justified within the field.

+ The result of a signed conversion shall always begin with a sign (`'+'` or `'–'`).

<space> If the first character of a signed conversion is not a sign, a <space> shall be prefixed to the result. This means that if the <space> and `'+'` flags both appear, the <space> flag shall be ignored.

# The value shall be converted to an alternative form. For `c`, `d`, `i`, `u`, and `s` conversion specifiers, the behavior is undefined. For the `o` conversion specifier, it shall increase the precision to force the first digit of the result to be a zero. For `x` or `X` conversion specifiers, a non-zero result has 0x or 0X prefixed to it, respectively. For e, E, f, g, and G conversion specifiers, the result shall always contain a radix character, even if no digits follow the radix character. For g and G conversion specifiers, trailing zeros shall not be removed from the result as they usually are.

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

Each conversion specifier character shall result in fetching zero or more arguments. The results are undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments shall be ignored.

The *conversion specifiers* and their meanings are:

d,i,o,u,x,X The integer argument shall be written as signed decimal (d or i), unsigned octal (o), unsigned decimal (u), or unsigned hexadecimal notation (x and X). The d and i specifiers shall convert to signed decimal in the style "[−]*dddd*". The x conversion specifier shall use the numbers and letters "0123456789abcdef" and the X conversion specifier shall use the numbers and letters "0123456789ABCDEF". The *precision* component of the argument shall specify the minimum number of digits to appear. If the value being converted can be represented in fewer digits than the specified minimum, it shall be expanded with leading zeros. The default precision shall be 1. The result of converting a zero value with a precision of 0 shall be no characters. If both the field width and precision are omitted, the implementation may precede, follow, or precede and follow numeric arguments of types d, i, and u with <blank>s; arguments of type o (octal) may be preceded with leading zeros.

f The floating-point number argument shall be written in decimal notation in the style **[–]***ddd.ddd*, where the number of digits after the radix character (shown here as a decimal point) shall be equal to the *precision* specification. The *LC_NUMERIC* locale category shall determine the radix character to use in this format. If the *precision* is omitted from the argument, six digits shall be written after the radix character; if the *precision* is explicitly 0, no radix character shall appear.

e,E The floating-point number argument shall be written in the style **[–]***d.ddd*e±*dd* (the symbol '±' indicates either a plus or minus sign), where there is one digit before the radix character (shown here as a decimal point) and the number of digits after it is equal to the precision. The *LC_NUMERIC* locale category shall determine the radix character to use in this format. When the precision is missing, six digits shall be written after the radix character; if the precision is 0, no radix character shall appear. The E conversion specifier shall produce a number with E instead of e introducing the exponent. The exponent shall always contain at least two digits. However, if the value to be written requires an exponent greater than two digits, additional exponent digits shall be written as necessary.

g,G The floating-point number argument shall be written in style f or e (or in style F or E in the case of a G conversion specifier), with the precision specifying the number of significant digits. The style used depends on the value converted: style e (or E) shall be used only if the exponent resulting from the conversion is less than –4 or greater than or equal to the precision. Trailing zeros are removed from the result. A radix character shall appear only if it is followed by a digit.

`c` The integer argument shall be converted to an **unsigned char** and the resulting byte shall be written.

`s` The argument shall be taken to be a string and bytes from the string shall be written until the end of the string or the number of bytes indicated by the *precision* specification of the argument is reached. If the precision is omitted from the argument, it shall be taken to be infinite, so all bytes up to the end of the string shall be written.

`%` Write a `'%'` character; no argument is converted.

In no case does a nonexistent or insufficient *field width* cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. The term *field width* should not be confused with the term *precision* used in the description of `%s`.

Examples

To represent the output of a program that prints a date and time in the form Sunday, July 3, 10:02, where *weekday* and *month* are strings:

```
"%s,Δ%sΔ%d,Δ%d:%.2d\n" <weekday>, <month>, <day>, <hour>, <min>
```

To show `'π'` written to 5 decimal places:

```
"piΔ=Δ%.5f\n",<value of π>
```

To show an input file format consisting of five colon-separated fields:

```
"%s:%s:%s:%s:%s\n", <arg1>, <arg2>, <arg3>, <arg4>, <arg5>
```

Chapter 6

Character Set

6.1 Portable Character Set

Conforming implementations shall support one or more coded character sets. Each supported locale shall include the *portable character set*, which is the set of symbolic names for characters in Table 6-1. This is used to describe characters within the text of IEEE Std 1003.1-200x. The first eight entries in Table 6-1 are defined in the ISO/IEC 6429:1992 standard and the rest of the characters are defined in the ISO/IEC 10646-1:2000 standard.

Table 6-1 Portable Character Set

Symbolic Name	Glyph	UCS	Description
<NUL>		<U0000>	NULL (NUL)
<alert>		<U0007>	BELL (BEL)
<backspace>		<U0008>	BACKSPACE (BS)
<tab>		<U0009>	CHARACTER TABULATION (HT)
<carriage-return>		<U000D>	CARRIAGE RETURN (CR)
<newline>		<U000A>	LINE FEED (LF)
<vertical-tab>		<U000B>	LINE TABULATION (VT)
<form-feed>		<U000C>	FORM FEED (FF)
<space>		<U0020>	SPACE
<exclamation-mark>	!	<U0021>	EXCLAMATION MARK
<quotation-mark>	"	<U0022>	QUOTATION MARK
<number-sign>	#	<U0023>	NUMBER SIGN
<dollar-sign>	$	<U0024>	DOLLAR SIGN
<percent-sign>	%	<U0025>	PERCENT SIGN
<ampersand>	&	<U0026>	AMPERSAND
<apostrophe>	’	<U0027>	APOSTROPHE
<left-parenthesis>	(	<U0028>	LEFT PARENTHESIS
<right-parenthesis>	)	<U0029>	RIGHT PARENTHESIS
<asterisk>	*	<U002A>	ASTERISK
<plus-sign>	+	<U002B>	PLUS SIGN
<comma>	,	<U002C>	COMMA
<hyphen-minus>	–	<U002D>	HYPHEN-MINUS
<hyphen>	-	<U002D>	HYPHEN-MINUS
<full-stop>	.	<U002E>	FULL STOP
<period>	.	<U002E>	FULL STOP
<slash>	/	<U002F>	SOLIDUS
<solidus>	/	<U002F>	SOLIDUS
<zero>	0	<U0030>	DIGIT ZERO
<one>	1	<U0031>	DIGIT ONE
<two>	2	<U0032>	DIGIT TWO
<three>	3	<U0033>	DIGIT THREE

Symbolic Name	Glyph	UCS	Description
<four>	4	<U0034>	DIGIT FOUR
<five>	5	<U0035>	DIGIT FIVE
<six>	6	<U0036>	DIGIT SIX
<seven>	7	<U0037>	DIGIT SEVEN
<eight>	8	<U0038>	DIGIT EIGHT
<nine>	9	<U0039>	DIGIT NINE
<colon>	:	<U003A>	COLON
<semicolon>	;	<U003B>	SEMICOLON
<less-than-sign>	<	<U003C>	LESS-THAN SIGN
<equals-sign>	=	<U003D>	EQUALS SIGN
<greater-than-sign>	>	<U003E>	GREATER-THAN SIGN
<question-mark>	?	<U003F>	QUESTION MARK
<commercial-at>	@		<U0040>
<A>	A	<U0041>	LATIN CAPITAL LETTER A
<B>	B	<U0042>	LATIN CAPITAL LETTER B
<C>	C	<U0043>	LATIN CAPITAL LETTER C
<D>	D	<U0044>	LATIN CAPITAL LETTER D
<E>	E	<U0045>	LATIN CAPITAL LETTER E
<F>	F	<U0046>	LATIN CAPITAL LETTER F
<G>	G	<U0047>	LATIN CAPITAL LETTER G
<H>	H	<U0048>	LATIN CAPITAL LETTER H
<I>	I	<U0049>	LATIN CAPITAL LETTER I
<J>	J	<U004A>	LATIN CAPITAL LETTER J
<K>	K	<U004B>	LATIN CAPITAL LETTER K
<L>	L	<U004C>	LATIN CAPITAL LETTER L
<M>	M	<U004D>	LATIN CAPITAL LETTER M
<N>	N	<U004E>	LATIN CAPITAL LETTER N
<O>	O	<U004F>	LATIN CAPITAL LETTER O
<P>	P	<U0050>	LATIN CAPITAL LETTER P
<Q>	Q	<U0051>	LATIN CAPITAL LETTER Q
<R>	R	<U0052>	LATIN CAPITAL LETTER R
<S>	S	<U0053>	LATIN CAPITAL LETTER S
<T>	T	<U0054>	LATIN CAPITAL LETTER T
<U>	U	<U0055>	LATIN CAPITAL LETTER U
<V>	V	<U0056>	LATIN CAPITAL LETTER V
<W>	W	<U0057>	LATIN CAPITAL LETTER W
<X>	X	<U0058>	LATIN CAPITAL LETTER X
<Y>	Y	<U0059>	LATIN CAPITAL LETTER Y
<Z>	Z	<U005A>	LATIN CAPITAL LETTER Z
<left-square-bracket>	[	<U005B>	LEFT SQUARE BRACKET
<backslash>	\	<U005C>	REVERSE SOLIDUS
<reverse-solidus>	\	<U005C>	REVERSE SOLIDUS
<right-square-bracket>	]	<U005D>	RIGHT SQUARE BRACKET
<circumflex-accent>	^	<U005E>	CIRCUMFLEX ACCENT
<circumflex>	^	<U005E>	CIRCUMFLEX ACCENT
<low-line>	_	<U005F>	LOW LINE
<underscore>	_	<U005F>	LOW LINE

Symbolic Name	Glyph	UCS	Description
<grave-accent>	‘	<U0060>	GRAVE ACCENT
<a>	a	<U0061>	LATIN SMALL LETTER A
<b>	b	<U0062>	LATIN SMALL LETTER B
<c>	c	<U0063>	LATIN SMALL LETTER C
<d>	d	<U0064>	LATIN SMALL LETTER D
<e>	e	<U0065>	LATIN SMALL LETTER E
<f>	f	<U0066>	LATIN SMALL LETTER F
<g>	g	<U0067>	LATIN SMALL LETTER G
<h>	h	<U0068>	LATIN SMALL LETTER H
<i>	i	<U0069>	LATIN SMALL LETTER I
<j>	j	<U006A>	LATIN SMALL LETTER J
<k>	k	<U006B>	LATIN SMALL LETTER K
<l>	l	<U006C>	LATIN SMALL LETTER L
<m>	m	<U006D>	LATIN SMALL LETTER M
<n>	n	<U006E>	LATIN SMALL LETTER N
<o>	o	<U006F>	LATIN SMALL LETTER O
<p>	p	<U0070>	LATIN SMALL LETTER P
<q>	q	<U0071>	LATIN SMALL LETTER Q
<r>	r	<U0072>	LATIN SMALL LETTER R
<s>	s	<U0073>	LATIN SMALL LETTER S
<t>	t	<U0074>	LATIN SMALL LETTER T
<u>	u	<U0075>	LATIN SMALL LETTER U
<v>	v	<U0076>	LATIN SMALL LETTER V
<w>	w	<U0077>	LATIN SMALL LETTER W
<x>	x	<U0078>	LATIN SMALL LETTER X
<y>	y	<U0079>	LATIN SMALL LETTER Y
<z>	z	<U007A>	LATIN SMALL LETTER Z
<left-brace>	{	<U007B>	LEFT CURLY BRACKET
<left-curly-bracket>	{	<U007B>	LEFT CURLY BRACKET
<vertical-line>	|	<U007C>	VERTICAL LINE
<right-brace>	}	<U007D>	RIGHT CURLY BRACKET
<right-curly-bracket>	}	<U007D>	RIGHT CURLY BRACKET
<tilde>	~	<U007E>	TILDE

IEEE Std 1003.1-200x uses character names other than the above, but only in an informative way; for example, in examples to illustrate the use of characters beyond the portable character set with the facilities of IEEE Std 1003.1-200x.

Table 6-1 (on page 111) defines the characters in the portable character set and the corresponding symbolic character names used to identify each character in a character set description file. The table contains more than one symbolic character name for characters whose traditional name differs from the chosen name. Characters defined in Table 6-2 (on page 116) may also be used in character set description files.

IEEE Std 1003.1-200x places only the following requirements on the encoded values of the characters in the portable character set:

- If the encoded values associated with each member of the portable character set are not invariant across all locales supported by the implementation, if an application accesses any pair of locales where the character encodings differ, or accesses data from an application running in a locale which has different encodings from the application’s current locale, the results are unspecified.

- The encoded values associated with the digits 0 to 9 shall be such that the value of each character after 0 shall be one greater than the value of the previous character.
- A null character, NUL, which has all bits set to zero, shall be in the set of characters.
- The encoded values associated with the members of the portable character set are each represented in a single byte. Moreover, if the value is stored in an object of C-language type **char**, it is guaranteed to be positive (except the NUL, which is always zero).

Conforming implementations shall support certain character and character set attributes, as defined in Section 7.2 (on page 120).

6.2 Character Encoding

The POSIX locale contains the characters in Table 6-1 (on page 111), which have the properties listed in Section 7.3.1 (on page 122). In other locales, the presence, meaning, and representation of any additional characters is locale-specific.

In locales other than the POSIX locale, a character may have a state-dependent encoding. There are two types of these encodings:

- A single-shift encoding (where each character not in the initial shift state is preceded by a shift code) can be defined if each shift-code and character sequence is considered a multibyte character. This is done using the concatenated-constant format in a character set description file, as described in Section 6.4 (on page 115). If the implementation supports a character encoding of this type, all of the standard utilities in the Shell and Utilities volume of IEEE Std 1003.1-200x shall support it. Use of a single-shift encoding with any of the functions in the System Interfaces volume of IEEE Std 1003.1-200x that do not specifically mention the effects of state-dependent encoding is implementation-defined.
- 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) cannot be defined with the current character set description file format. Use of a locking-shift encoding with any of the standard utilities in the Shell and Utilities volume of IEEE Std 1003.1-200x or with any of the functions in the System Interfaces volume of IEEE Std 1003.1-200x that do not specifically mention the effects of state-dependent encoding is implementation-defined.

While in the initial shift state, all characters in the portable character set shall retain their usual interpretation and shall not alter the shift state. The interpretation for subsequent bytes in the sequence shall be a function of the current shift state. A byte with all bits zero shall be interpreted as the null character independent of shift state. Thus a byte with all bits zero shall never occur in the second or subsequent bytes of a character.

The maximum allowable number of bytes in a character in the current locale shall be indicated by {MB_CUR_MAX}, defined in the **<stdlib.h>** header and by the **<mb_cur_max>** value in a character set description file; see Section 6.4 (on page 115). The implementation's maximum number of bytes in a character shall be defined by the C-language macro {MB_LEN_MAX}.

6.3 C Language Wide-Character Codes

In the shell, the standard utilities are written so that the encodings of characters are described by the locale's *LC_CTYPE* definition (see Section 7.3.1 (on page 122)) and there is no differentiation between characters consisting of single octets (8-bit bytes) or multiple bytes. However, in the C language, a differentiation is made. To ease the handling of variable length characters, the C language has introduced the concept of wide-character codes.

All wide-character codes in a given process consist of an equal number of bits. This is in contrast to characters, which can consist of a variable number of bytes. The byte or byte sequence that represents a character can also be represented as a wide-character code. Wide-character codes thus provide a uniform size for manipulating text data. A wide-character code having all bits zero is the null wide-character code (see Section 3.246 (on page 66)), and terminates wide-character strings (see Section 3.432 (on page 92)). The wide-character value for each member of the portable character set shall equal its value when used as the lone character in an integer character constant. Wide-character codes for other characters are locale and implementation-defined. State shift bytes shall not have a wide-character code representation.

6.4 Character Set Description File

Implementations shall provide a character set description file for at least one coded character set supported by the implementation. These files are referred to elsewhere in IEEE Std 1003.1-200x as *charmap* files. It is implementation-defined whether or not users or applications can provide additional character set description files.

IEEE Std 1003.1-200x does not require that multiple character sets or codesets be supported. Although multiple charmap files are supported, it is the responsibility of the implementation to provide the file or files; if only one is provided, only that one is accessible using the *localedef* utility's –**f** option.

Each character set description file, except those that use the ISO/IEC 10646-1:2000 standard position values as the encoding values, shall define characteristics for the coded character set and the encoding for the characters specified in Table 6-1 (on page 111), 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.

Each symbolic name specified in Table 6-1 (on page 111) shall be included in the file and shall be mapped to a unique coding value, except as noted below. The glyphs ′{′, ′}′, ′_′, ′-′, ′/′, ′\′, ′.′, and ′^′ have more than one symbolic name; all symbolic names for each such glyph shall be included, each with identical encoding. If some or all of the control characters identified in Table 6-2 (on page 116) are supported by the implementation, the symbolic names and their corresponding encoding values shall be included in the file. Some of the encodings associated with the symbolic names in Table 6-2 (on page 116) may be the same as characters found in Table 6-1 (on page 111); both names shall be provided for each encoding.

Table 6-2 Control Character Set

<ACK>	<DC2>	<ENQ>	<FS>	<IS4>	<SOH>
<BEL>	<DC3>	<EOT>	<GS>	<LF>	<STX>
<BS>	<DC4>	<ESC>	<HT>	<NAK>	<SUB>
<CAN>	<DEL>	<ETB>	<IS1>	<RS>	<SYN>
<CR>	<DLE>	<ETX>	<IS2>	<SI>	<US>
<DC1>	<EM>	<FF>	<IS3>	<SO>	<VT>

The following declarations can precede the character definitions. Each shall consist of the symbol shown in the following list, starting in column 1, including the surrounding brackets, followed by one or more <blank>s, followed by the value to be assigned to the symbol.

<code_set_name> The name of the coded character set for which the character set description file is defined. The characters of the name shall be taken from the set of characters with visible glyphs defined in Table 6-1 (on page 111).

<mb_cur_max> The maximum number of bytes in a multi-byte character. This shall default to 1.

<mb_cur_min> An unsigned positive integer value that defines the minimum number of bytes in a character for the encoded character set. XSI On XSI-conformant systems, **<mb_cur_min>** shall always be 1.

<escape_char> The character used to indicate that the characters following shall be interpreted in a special way, as defined later in this section. This shall default to backslash (`'\'`), which is the character used in all the following text and examples, unless otherwise noted.

<comment_char> The character that, when placed in column 1 of a charmap line, is used to indicate that the line shall be ignored. The default character shall be the number sign (`'#'`).

The character set mapping definitions shall be all the lines immediately following an identifier line containing the string `"CHARMAP"` starting in column 1, and preceding a trailer line containing the string `"END CHARMAP"` starting in column 1. Empty lines and lines containing a **<comment_char>** in the first column shall be ignored. Each non-comment line of the character set mapping definition (that is, between the `"CHARMAP"` and `"END CHARMAP"` lines of the file) shall be in either of two forms:

```
"%s %s %s\n", <symbolic-name>, <encoding>, <comments>
```

or:

```
"%s...%s %s %s\n", <symbolic-name>, <symbolic-name>,
    <encoding>, <comments>
```

In the first format, the line in the character set mapping definition shall define a single symbolic name and a corresponding encoding. A symbolic name is one or more characters from the set shown with visible glyphs in Table 6-1 (on page 111), enclosed between angle brackets. A character following an escape character is interpreted as itself; for example, the sequence `"<\\\>>"` represents the symbolic name `"\>"` enclosed between angle brackets.

In the second format, the line in the character set mapping definition shall define a range of one or more symbolic names. In this form, the symbolic names shall consist of zero or more non-numeric characters from the set shown with visible glyphs in Table 6-1 (on page 111), followed by an integer formed by one or more decimal digits. Both integers shall contain the same number of digits. The characters preceding the integer shall be identical in the two symbolic names, and the integer formed by the digits in the second symbolic name shall be equal to or greater than the integer formed by the digits in the first name. This shall be interpreted as a series of symbolic names formed from the common part and each of the integers between the first and the second integer, inclusive. As an example, <j0101>...<j0104> is interpreted as the symbolic names <j0101>, <j0102>, <j0103>, and <j0104>, in that order.

A character set mapping definition line shall exist for all symbolic names specified in Table 6-1 (on page 111), and shall define the coded character value that corresponds to the character indicated in the table, or the coded character value that corresponds to the control character symbolic name. If the control characters commonly associated with the symbolic names in Table 6-2 (on page 116) are supported by the implementation, the symbolic name and the corresponding encoding value shall be included in the file. Additional unique symbolic names may be included. A coded character value can be represented by more than one symbolic name.

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

```
"%cd%u", <escape_char>, <decimal byte value>
"%cx%x", <escape_char>, <hexadecimal byte value>
"%c%o", <escape_char>, <octal byte value>
```

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 three octal digits, preceded by the escape character; for example, `"\05"`, `"\141"`, or `"\217"`. In a portable charmap file, each constant represents an 8-bit byte. When constants are concatenated for multi-byte 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 multi-byte character specified by the last constant. The manner in which these constants are represented in the character stored in the system is implementation-defined. (This notation was chosen for reasons of portability. There is no requirement that the internal representation in the computer memory be in this same order.) Omitting bytes from a multi-byte character definition produces undefined results.

In lines defining ranges of symbolic names, the encoded value shall be the value for the first symbolic name in the range (the symbolic name preceding the ellipsis). Subsequent symbolic names defined by the range shall have encoding values in increasing order. Bytes shall be treated as unsigned octets, and carry shall be propagated between the bytes as necessary to represent the range. For example, the line:

```
<j0101>...<j0104>  \d129\d254
```

is interpreted as:

```
<j0101>            \d129\d254
<j0102>            \d129\d255
<j0103>            \d130\d0
<j0104>            \d130\d1
```

Note that this line is interpreted as the example even on systems with bytes larger than 8 bits.

The comment is optional.

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.

WIDTH An unsigned positive integer value defining the column width (see Section 3.103 (on page 47)) for the printable characters in the coded character set specified in Table 6-1 (on page 111) and Table 6-2 (on page 116). Coded character set character values shall be defined using symbolic character names followed by column width values. Defining a character with more than one **WIDTH** produces undefined results. The **END WIDTH** keyword shall be used to terminate the **WIDTH** definitions. Specifying the width of a non-printable character in a **WIDTH** declaration produces undefined results.

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

In this example, the numerical code point values represented by the symbols **<A>** and **<B>** are assigned a width of 1. The code point values **<C>** to **<Z>** inclusive (**<C>**, **<D>**, **<E>**, and so on) are also assigned a width of 1. Using **<A>**...**<Z>** would have required fewer lines, but the alternative was shown to demonstrate flexibility. The keyword **WIDTH_DEFAULT** could have been added as appropriate.

6.4.1 State-Dependent Character Encodings

This section addresses the use of state-dependent character encodings (that is, 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 multi-byte character, defined using the concatenated-constant format described in Section 6.4 (on page 115). 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 Section 6.4 (on page 115). 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:

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 shall produce output that contains locking shifts at the beginning or end of the resulting data, if appropriate, to retain correct state information.

Chapter 7

Locale

7.1 General

A *locale* is the definition of the subset of a user's environment that depends on language and cultural conventions. It is made up from one or more categories. Each category is identified by its name and controls specific aspects of the behavior of components of the system. Category names correspond to the following environment variable names:

LC_CTYPE Character classification and case conversion.

LC_COLLATE Collation order.

LC_MONETARY Monetary formatting.

LC_NUMERIC Numeric, non-monetary formatting.

LC_TIME Date and time formats.

LC_MESSAGES Formats of informative and diagnostic messages and interactive responses.

The standard utilities in the Shell and Utilities volume of IEEE Std 1003.1-200x shall base their behavior on the current locale, as defined in the ENVIRONMENT VARIABLES section for each utility. The behavior of some of the C-language functions defined in the System Interfaces volume of IEEE Std 1003.1-200x shall also be modified based on the current locale, as defined by the last call to *setlocale*( ).

Locales other than those supplied by the implementation can be created via the *localedef* utility, provided that the _POSIX2_LOCALEDEF symbol is defined on the system. Even if *localedef* is not provided, all implementations conforming to the System Interfaces volume of IEEE Std 1003.1-200x shall provide one or more locales that behave as described in this chapter. The input to the utility is described in Section 7.3 (on page 120). The value that is used to specify a locale when using environment variables shall be the string specified as the *name* operand to the *localedef* utility when the locale was created. The strings "C" and "POSIX" are reserved as identifiers for the POSIX locale (see Section 7.2 (on page 120)). When the value of a locale environment variable begins with a slash ('/'), it shall be interpreted as the pathname of the locale definition; the type of file (regular, directory, and so on) used to store the locale definition is implementation-defined. If the value does not begin with a slash, the mechanism used to locate the locale is implementation-defined.

If different character sets are used by the locale categories, the results achieved by an application utilizing these categories are undefined. Likewise, if different codesets are used for the data being processed by interfaces whose behavior is dependent on the current locale, or the codeset is different from the codeset assumed when the locale was created, the result is also undefined.

Applications can select the desired locale by invoking the *setlocale*( ) function (or equivalent) with the appropriate value. If the function is invoked with an empty string, such as:

```
setlocale(LC_ALL, "");
```

the value of the corresponding environment variable is used. If the environment variable is unset or is set to the empty string, the implementation shall set the appropriate environment as defined in Chapter 8 (on page 157).

7.2 POSIX Locale

Conforming systems shall provide a *POSIX locale*, also known as the C locale. The behavior of standard utilities and functions in the POSIX locale shall be as if the locale was defined via the *localedef* utility with input data from the POSIX locale tables in Section 7.3.

The tables in Section 7.3 describe the characteristics and behavior of the POSIX locale for data consisting entirely of characters from the portable character set and the control character set. For other characters, the behavior is unspecified. For C-language programs, the POSIX locale shall be the default locale when the *setlocale*( ) function is not called.

The POSIX locale can be specified by assigning to the appropriate environment variables the values `"C"` or `"POSIX"`.

All implementations shall define a locale as the default locale, to be invoked when no environment variables are set, or set to the empty string. This default locale can be the POSIX locale or any other implementation-defined locale. Some implementations may provide facilities for local installation administrators to set the default locale, customizing it for each location. IEEE Std 1003.1-200x does not require such a facility.

7.3 Locale Definition

The capability to specify additional locales to those provided by an implementation is optional, denoted by the _POSIX2_LOCALEDEF symbol. If the option is not supported, only implementation-supplied locales are available. Such locales shall be documented using the format specified in this section.

Locales can be described with the file format presented in this section. The file format is that accepted by the *localedef* utility. For the purposes of this section, the file is referred to as the *locale definition file*, but no locales shall be affected by this file unless it is processed by *localedef* or some similar mechanism. Any requirements in this section imposed upon the utility shall apply to *localedef* or to any other similar utility used to install locale information using the locale definition file format described here.

The locale definition file shall contain one or more locale category source definitions, and shall not contain more than one definition for the same locale category. If the file contains source definitions for more than one category, implementation-defined categories, if present, shall appear after the categories defined by Section 7.1 (on page 119). A category source definition contains either the definition of a category or a **copy** directive. For a description of the **copy** directive, see *localedef*. In the event that some of the information for a locale category, as specified in this volume of IEEE Std 1003.1-200x, is missing from the locale source definition, the behavior of that category, if it is referenced, is unspecified.

A category source definition shall consist of a category header, a category body, and a category trailer. A category header shall consist of the character string naming of the category, beginning with the characters *LC_*. The category trailer shall consist of the string `"END"`, followed by one or more <blank>s and the string used in the corresponding category header.

The category body shall consist of one or more lines of text. Each line shall contain an identifier, optionally followed by one or more operands. Identifiers shall be either keywords, identifying a particular locale element, or collating elements. In addition to the keywords defined in this volume of IEEE Std 1003.1-200x, the source can contain implementation-defined keywords. Each keyword within a locale shall have a unique name (that is, two categories cannot have a commonly-named keyword); no keyword shall start with the characters *LC_*. Identifiers shall be separated from the operands by one or more <blank>s.

Operands shall be characters, collating elements, or strings of characters. Strings shall be enclosed in double-quotes. Literal double-quotes within strings shall be preceded by the <*escape character*>, described below. When a keyword is followed by more than one operand, the operands shall be separated by semicolons; <blank>s shall be allowed both before and after a semicolon.

The first category header in the file can be preceded by a line modifying the comment character. It shall have the following format, starting in column 1:

```
"comment_char %c\n", <comment character>
```

The comment character shall default to the number sign (`'#'`). Blank lines and lines containing the <*comment character*> in the first position shall be ignored.

The first category header in the file can be preceded by a line modifying the escape character to be used in the file. It shall have the following format, starting in column 1:

```
"escape_char %c\n", <escape character>
```

The escape character shall default to backslash, which is the character used in all examples shown in this volume of IEEE Std 1003.1-200x.

A line can be continued by placing an escape character as the last character on the line; this continuation character shall be discarded from the input. Although the implementation need not accept any one portion of a continued line with a length exceeding {LINE_MAX} bytes, it shall place no limits on the accumulated length of the continued line. Comment lines shall not be continued on a subsequent line using an escaped newline character.

Individual characters, characters in strings, and collating elements shall be represented using symbolic names, as defined below. In addition, characters can be represented using the characters themselves or as octal, hexadecimal, or decimal constants. When non-symbolic notation is used, the resultant locale definitions are in many cases not portable between systems. The left angle bracket (`'<'`) is a reserved symbol, denoting the start of a symbolic name; when used to represent itself it shall be preceded by the escape character. The following rules apply to character representation:

1. A character can be represented via a symbolic name, enclosed within angle brackets `'<'` and `'>'`. The symbolic name, including the angle brackets, shall exactly match a symbolic name defined in the charmap file specified via the *localedef* **−f** option, and it shall be replaced by a character value determined from the value associated with the symbolic name in the charmap file. The use of a symbolic name not found in the charmap file shall constitute an error, unless the category is *LC_CTYPE* or *LC_COLLATE*, in which case it shall constitute a warning condition (see *localedef* for a description of actions resulting from errors and warnings). The specification of a symbolic name in a **collating-element** or **collating-symbol** section that duplicates a symbolic name in the charmap file (if present) shall be an error. Use of the escape character or a right angle bracket within a symbolic name is invalid unless the character is preceded by the escape character.

   For example:

   ```
   <c>;<c−cedilla>  "<M><a><y>"
   ```

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.) Within a string, the double-quote character, the escape character, and the right angle bracket character shall be escaped (preceded by the escape character) to be interpreted as the character itself. Outside strings, the characters:

```
, ; < > escape_char
```

shall be escaped to be interpreted as the character itself.

For example:

```
c "May"
```

3. A character can be represented as an octal constant. An octal constant shall be specified as the escape character followed by two or three octal digits. Each constant shall represent a byte value. Multi-byte values can be represented by concatenated constants specified in byte order with the last constant specifying the least significant byte of the character.

   For example:

   ```
   \143;\347;\143\150   "\115\141\171"
   ```

4. A character can be represented as a hexadecimal constant. A hexadecimal constant shall be specified as the escape character followed by an `'x'` followed by two hexadecimal digits. Each constant shall represent a byte value. Multi-byte values can be represented by concatenated constants specified in byte order with the last constant specifying the least significant byte of the character.

   For example:

   ```
   \x63;\xe7;\x63\x68   "\x4d\x61\x79"
   ```

5. A character can be represented as a decimal constant. A decimal constant shall be specified as the escape character followed by a `'d'` followed by two or three decimal digits. Each constant represents a byte value. Multi-byte values can be represented by concatenated constants specified in byte order with the last constant specifying the least significant byte of the character.

   For example:

   ```
   \d99;\d231;\d99\d104  "\d77\d97\d121"
   ```

Implementations may accept single-digit octal, decimal, or hexadecimal constants following the escape character. Only characters existing in the character set for which the locale definition is created shall be specified, whether using symbolic names, the characters themselves, or octal, decimal, or hexadecimal constants. If a charmap file is present, only characters defined in the charmap can be specified using octal, decimal, or hexadecimal constants. Symbolic names not present in the charmap file can be specified and shall be ignored, as specified under item 1 above.

7.3.1 LC_CTYPE

The *LC_CTYPE* category shall define character classification, case conversion, and other character attributes. In addition, a series of characters can be represented by three adjacent periods representing an ellipsis symbol (`"..."`). The ellipsis specification shall be interpreted as meaning that all values between the values preceding and following it represent valid characters. The ellipsis specification shall be valid only within a single encoded character set; that is, within a group of characters of the same size. An ellipsis shall be interpreted as including in the list all characters with an encoded value higher than the encoded value of the character preceding the ellipsis and lower than the encoded value of the character following the ellipsis.

For example:

```
\x30;...;\x39;
```

includes in the character class all characters with encoded values between the endpoints.

The following keywords shall be recognized. In the descriptions, the term ''automatically included'' means that it shall not be an error either to include or omit any of the referenced characters; the implementation provides them if missing (even if the entire keyword is missing) and accepts them silently if present. When the implementation automatically includes a missing character, it shall have an encoded value dependent on the charmap file in effect (see the description of the *localedef* **–f** option); otherwise, it shall have a value derived from an implementation-defined character mapping.

The character classes **digit**, **xdigit**, **lower**, **upper**, and **space** have a set of automatically included characters. These only need to be specified if the character values (that is, encoding) differ from the implementation default values. It is not possible to define a locale without these automatically included characters unless some implementation extension is used to prevent their inclusion. Such a definition would not be a proper superset of the C or POSIX locale and thus, it might not be possible for conforming applications to work properly.

copy Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.

upper Define characters to be classified as uppercase letters.

In the POSIX locale, the 26 uppercase letters shall be included:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

In a locale definition file, no character specified for the keywords **cntrl**, **digit**, **punct**, or **space** shall be specified. The uppercase letters <A> to <Z>, as defined in Section 6.4 (on page 115) (the portable character set), are automatically included in this class.

lower Define characters to be classified as lowercase letters.

In the POSIX locale, the 26 lowercase letters shall be included:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

In a locale definition file, no character specified for the keywords **cntrl**, **digit**, **punct**, or **space** shall be specified. The lowercase letters <a> to <z> of the portable character set are automatically included in this class.

alpha Define characters to be classified as letters.

In the POSIX locale, all characters in the classes **upper** and **lower** shall be included.

In a locale definition file, no character specified for the keywords **cntrl**, **digit**, **punct**, or **space** shall be specified. Characters classified as either **upper** or **lower** are automatically included in this class.

digit Define the characters to be classified as numeric digits.

In the POSIX locale, only:

```
0 1 2 3 4 5 6 7 8 9
```

shall be included.

In a locale definition file, only the digits <zero>, <one>, <two>, <three>, <four>, <five>, <six>, <seven>, <eight>, and <nine> shall be specified, and in contiguous ascending sequence by numerical value. The digits <zero> to <nine> of the portable character set are automatically included in this class.

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.

space Define characters to be classified as white-space characters.

In the POSIX locale, at a minimum, the <space>, <form-feed>, <newline>, <carriage-return>, <tab>, and <vertical-tab> shall be included.

In a locale definition file, no character specified for the keywords **upper**, **lower**, **alpha**, **digit**, **graph**, or **xdigit** shall be specified. The <space>, <form-feed>, <newline>, <carriage-return>, <tab>, and <vertical-tab> of the portable character set, and any characters included in the class **blank** are automatically included in this class.

cntrl Define characters to be classified as control characters.

In the POSIX locale, no characters in classes **alpha** or **print** shall be included.

In a locale definition file, no character specified for the keywords **upper**, **lower**, **alpha**, **digit**, **punct**, **graph**, **print**, or **xdigit** shall be specified.

punct Define characters to be classified as punctuation characters.

In the POSIX locale, neither the <space> nor any characters in classes **alpha**, **digit**, or **cntrl** shall be included.

In a locale definition file, no character specified for the keywords **upper**, **lower**, **alpha**, **digit**, **cntrl**, **xdigit**, or as the <space> shall be specified.

graph Define characters to be classified as printable characters, not including the <space>.

In the POSIX locale, all characters in classes **alpha**, **digit**, and **punct** shall be included; no characters in class **cntrl** shall be included.

In a locale definition file, characters specified for the keywords **upper**, **lower**, **alpha**, **digit**, **xdigit**, and **punct** are automatically included in this class. No character specified for the keyword **cntrl** shall be specified.

print Define characters to be classified as printable characters, including the <space>.

In the POSIX locale, all characters in class **graph** shall be included; no characters in class **cntrl** shall be included.

In a locale definition file, characters specified for the keywords **upper**, **lower**, **alpha**, **digit**, **xdigit**, **punct, graph**, and the <space> are automatically included in this class. No character specified for the keyword **cntrl** shall be specified.

xdigit Define the characters to be classified as hexadecimal digits.

In the POSIX locale, only:

```
0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f
```

shall be included.

In a locale definition file, only the characters defined for the class **digit** shall be specified, in contiguous ascending sequence by numerical value, followed by one or more sets of six characters representing the hexadecimal digits 10 to 15 inclusive, with each set in ascending order (for example, <A>, <B>, <C>, <D>, <E>, <F>, <a>, <b>, <c>, <d>, <e>, <f>). The digits <zero> to <nine>, the uppercase letters <A> to <F>, and the lowercase letters <a> to <f> of the portable character set are automatically included in this class.

blank Define characters to be classified as <blank>s.

In the POSIX locale, only the <space> and <tab> shall be included.

In a locale definition file, the <space> and <tab> are automatically included in this class. LI **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. A character class name shall consist of at least one and at most {CHARCLASS_NAME_MAX} bytes of alphanumeric characters from the portable filename character set. The first character of a character class name shall not be a digit. The name shall not match any of the *LC_CTYPE* keywords defined in this volume of IEEE Std 1003.1-200x. Future revisions of IEEE Std 1003.1-200x will not specify any *LC_CTYPE* keywords containing uppercase letters.

charclass-name Define characters to be classified as belonging to the named locale-specific character class. In the POSIX locale, 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 represents a class without any characters belonging to it.

The *charclass-name* can be used as the *property* argument to the *wctype*( ) function, in regular expression and shell pattern-matching bracket expressions, and by the *tr* command.

toupper Define the mapping of lowercase letters to uppercase letters.

In the POSIX locale, at a minimum, the 26 lowercase characters:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

shall be mapped to the corresponding 26 uppercase characters:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

In a locale definition file, the operand shall consist of character pairs, separated by semicolons. The characters in each character pair shall be separated by a comma and the pair enclosed by parentheses. The first character in each pair is the lowercase letter, the second the corresponding uppercase letter. Only characters specified for the keywords **lower** and **upper** shall be specified. The lowercase letters <a> to <z>, and their corresponding uppercase letters <A> to <Z>, of the portable character set are automatically included in this mapping, but only when the **toupper** keyword is omitted from the locale definition.

tolower Define the mapping of uppercase letters to lowercase letters.

In the POSIX locale, at a minimum, the 26 uppercase characters:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

shall be mapped to the corresponding 26 lowercase characters:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

In a locale definition file, the operand shall consist of character pairs, separated by semicolons. The characters in each character pair shall be separated by a comma and the pair enclosed by parentheses. The first character in each pair is the uppercase letter, the second the corresponding lowercase letter. Only characters specified for the keywords **lower** and **upper** shall be specified. If the **tolower** keyword is omitted from the locale definition, the mapping is the reverse mapping of the one specified for **toupper**.

The following table shows the character class combinations allowed:

Table 7-1 Valid Character Class Combinations

	Can Also Belong To										
In Class	**upper**	**lower**	**alpha**	**digit**	**space**	**cntrl**	**punct**	**graph**	**print**	**xdigit**	**blank**
upper		—	A	x	x	x	x	A	A	—	x
lower	—		A	x	x	x	x	A	A	—	x
alpha	—	—		x	x	x	x	A	A	—	x
digit	x	x	x		x	x	x	A	A	A	x
space	x	x	x	x		—	*	*	*	x	—
cntrl	x	x	x	x	—		x	x	x	x	—
punct	x	x	x	x	—	x		A	A	x	—
graph	—	—	—	—	—	x	—		A	—	—
print	—	—	—	—	—	x	—	—		—	—
xdigit	—	—	—	—	x	x	x	A	A		x
blank	x	x	x	x	A	—	*	*	*	x	

Notes:

1. Explanation of codes:

   A Automatically included; see text.

   — Permitted.

   x Mutually-exclusive.

   * See note 2.

2. The <space>, which is part of the **space** and **blank** classes, cannot belong to **punct** or **graph**, but shall automatically belong to the **print** class. Other **space** or **blank** characters can be classified as any of **punct**, **graph**, or **print**.

7.3.1.1 *LC_CTYPE Category in the POSIX Locale*

The character classifications for the POSIX locale follow; the code listing depicts the *localedef* input, the table represents the same information, sorted by character.

```
LC_CTYPE
# The following is the POSIX locale LC_CTYPE.
# "alpha" is by default "upper" and "lower"
# "alnum" is by definition "alpha" and "digit"
# "print" is by default "alnum", "punct" and the <space>
# "graph" is by default "alnum" and "punct"
#
upper    <A>;<B>;<C>;<D>;<E>;<F>;<G>;<H>;<I>;<J>;<K>;<L>;<M>;\
         <N>;<O>;<P>;<Q>;<R>;<S>;<T>;<U>;<V>;<W>;<X>;<Y>;<Z>
#
```

```
lower    <a>;<b>;<c>;<d>;<e>;<f>;<g>;<h>;<i>;<j>;<k>;<l>;<m>;\
         <n>;<o>;<p>;<q>;<r>;<s>;<t>;<u>;<v>;<w>;<x>;<y>;<z>
#
digit    <zero>;<one>;<two>;<three>;<four>;<five>;<six>;\
         <seven>;<eight>;<nine>
#
space    <tab>;<newline>;<vertical-tab>;<form-feed>;\
         <carriage-return>;<space>
#
cntrl    <alert>;<backspace>;<tab>;<newline>;<vertical-tab>;\
         <form-feed>;<carriage-return>;\
         <NUL>;<SOH>;<STX>;<ETX>;<EOT>;<ENQ>;<ACK>;<SO>;\
         <SI>;<DLE>;<DC1>;<DC2>;<DC3>;<DC4>;<NAK>;<SYN>;\
         <ETB>;<CAN>;<EM>;<SUB>;<ESC>;<IS4>;<IS3>;<IS2>;\
         <IS1>;<DEL>
#
punct    <exclamation-mark>;<quotation-mark>;<number-sign>;\
         <dollar-sign>;<percent-sign>;<ampersand>;<apostrophe>;\
         <left-parenthesis>;<right-parenthesis>;<asterisk>;\
         <plus-sign>;<comma>;<hyphen>;<period>;<slash>;\
         <colon>;<semicolon>;<less-than-sign>;<equals-sign>;\
         <greater-than-sign>;<question-mark>;<commercial-at>;\
         <left-square-bracket>;<backslash>;<right-square-bracket>;\
         <circumflex>;<underscore>;<grave-accent>;<left-curly-bracket>;\
         <vertical-line>;<right-curly-bracket>;<tilde>
#
xdigit   <zero>;<one>;<two>;<three>;<four>;<five>;<six>;<seven>;\
         <eight>;<nine>;<A>;<B>;<C>;<D>;<E>;<F>;<a>;<b>;<c>;<d>;<e>;<f>
#
blank    <space>;<tab>
#
toupper (<a>,<A>);(<b>,<B>);(<c>,<C>);(<d>,<D>);(<e>,<E>);\
        (<f>,<F>);(<g>,<G>);(<h>,<H>);(<i>,<I>);(<j>,<J>);\
        (<k>,<K>);(<l>,<L>);(<m>,<M>);(<n>,<N>);(<o>,<O>);\
        (<p>,<P>);(<q>,<Q>);(<r>,<R>);(<s>,<S>);(<t>,<T>);\
        (<u>,<U>);(<v>,<V>);(<w>,<W>);(<x>,<X>);(<y>,<Y>);(<z>,<Z>)
#
tolower (<A>,<a>);(<B>,<b>);(<C>,<c>);(<D>,<d>);(<E>,<e>);\
        (<F>,<f>);(<G>,<g>);(<H>,<h>);(<I>,<i>);(<J>,<j>);\
        (<K>,<k>);(<L>,<l>);(<M>,<m>);(<N>,<n>);(<O>,<o>);\
        (<P>,<p>);(<Q>,<q>);(<R>,<r>);(<S>,<s>);(<T>,<t>);\
        (<U>,<u>);(<V>,<v>);(<W>,<w>);(<X>,<x>);(<Y>,<y>);(<Z>,<z>)
END LC_CTYPE
```

Symbolic Name	Other Case	Character Classes
<NUL>		**cntrl**
<SOH>		**cntrl**
<STX>		**cntrl**
<ETX>		**cntrl**
<EOT>		**cntrl**
<ENQ>		**cntrl**
<ACK>		**cntrl**
<alert>		**cntrl**
<backspace>		**cntrl**
<tab>		**cntrl, space, blank**
<newline>		**cntrl, space**
<vertical-tab>		**cntrl, space**
<form-feed>		**cntrl, space**
<carriage-return>		**cntrl, space**
<SO>		**cntrl**
<SI>		**cntrl**
<DLE>		**cntrl**
<DC1>		**cntrl**
<DC2>		**cntrl**
<DC3>		**cntrl**
<DC4>		**cntrl**
<NAK>		**cntrl**
<SYN>		**cntrl**
<ETB>		**cntrl**
<CAN>		**cntrl**
<EM>		**cntrl**
<SUB>		**cntrl**
<ESC>		**cntrl**
<IS4>		**cntrl**
<IS3>		**cntrl**
<IS2>		**cntrl**
<IS1>		**cntrl**
<space>		**space, print, blank**
<exclamation-mark>		**punct, print, graph**
<quotation-mark>		**punct, print, graph**
<number-sign>		**punct, print, graph**
<dollar-sign>		**punct, print, graph**
<percent-sign>		**punct, print, graph**
<ampersand>		**punct, print, graph**
<apostrophe>		**punct, print, graph**
<left-parenthesis>		**punct, print, graph**
<right-parenthesis>		**punct, print, graph**
<asterisk>		**punct, print, graph**
<plus-sign>		**punct, print, graph**
<comma>		**punct, print, graph**
<hyphen>		**punct, print, graph**
<period>		**punct, print, graph**

Symbolic Name	Other Case	Character Classes
<slash>		**punct, print, graph**
<zero>		**digit, xdigit, print, graph**
<one>		**digit, xdigit, print, graph**
<two>		**digit, xdigit, print, graph**
<three>		**digit, xdigit, print, graph**
<four>		**digit, xdigit, print, graph**
<five>		**digit, xdigit, print, graph**
<six>		**digit, xdigit, print, graph**
<seven>		**digit, xdigit, print, graph**
<eight>		**digit, xdigit, print, graph**
<nine>		**digit, xdigit, print, graph**
<colon>		**punct, print, graph**
<semicolon>		**punct, print, graph**
<less-than-sign>		**punct, print, graph**
<equals-sign>		**punct, print, graph**
<greater-than-sign>		**punct, print, graph**
<question-mark>		**punct, print, graph**
<commercial-at>		**punct, print, graph**
<A>	<a>	**upper, xdigit, alpha, print, graph**
<B>	<b>	**upper, xdigit, alpha, print, graph**
<C>	<c>	**upper, xdigit, alpha, print, graph**
<D>	<d>	**upper, xdigit, alpha, print, graph**
<E>	<e>	**upper, xdigit, alpha, print, graph**
<F>	<f>	**upper, xdigit, alpha, print, graph**
<G>	<g>	**upper, alpha, print, graph**
<H>	<h>	**upper, alpha, print, graph**
<I>	<i>	**upper, alpha, print, graph**
<J>	<j>	**upper, alpha, print, graph**
<K>	<k>	**upper, alpha, print, graph**
<L>	<l>	**upper, alpha, print, graph**
<M>	<m>	**upper, alpha, print, graph**
<N>	<n>	**upper, alpha, print, graph**
<O>	<o>	**upper, alpha, print, graph**
<P>	<p>	**upper, alpha, print, graph**
<Q>	<q>	**upper, alpha, print, graph**
<R>	<r>	**upper, alpha, print, graph**
<S>	<s>	**upper, alpha, print, graph**
<T>	<t>	**upper, alpha, print, graph**
<U>	<u>	**upper, alpha, print, graph**
<V>	<v>	**upper, alpha, print, graph**
<W>	<w>	**upper, alpha, print, graph**
<X>	<x>	**upper, alpha, print, graph**
<Y>	<y>	**upper, alpha, print, graph**
<Z>	<z>	**upper, alpha, print, graph**
<left-square-bracket>		**punct, print, graph**
<backslash>		**punct, print, graph**
<right-square-bracket>		**punct, print, graph**

Symbolic Name	Other Case	Character Classes
<circumflex>		**punct, print, graph**
<underscore>		**punct, print, graph**
<grave-accent>		**punct, print, graph**
<a>	<A>	**lower, xdigit, alpha, print, graph**
<b>	<B>	**lower, xdigit, alpha, print, graph**
<c>	<C>	**lower, xdigit, alpha, print, graph**
<d>	<D>	**lower, xdigit, alpha, print, graph**
<e>	<E>	**lower, xdigit, alpha, print, graph**
<f>	<F>	**lower, xdigit, alpha, print, graph**
<g>	<G>	**lower, alpha, print, graph**
<h>	<H>	**lower, alpha, print, graph**
<i>	<I>	**lower, alpha, print, graph**
<j>	<J>	**lower, alpha, print, graph**
<k>	<K>	**lower, alpha, print, graph**
<l>	<L>	**lower, alpha, print, graph**
<m>	<M>	**lower, alpha, print, graph**
<n>	<N>	**lower, alpha, print, graph**
<o>	<O>	**lower, alpha, print, graph**
<p>	<P>	**lower, alpha, print, graph**
<q>	<Q>	**lower, alpha, print, graph**
<r>	<R>	**lower, alpha, print, graph**
<s>	<S>	**lower, alpha, print, graph**
<t>	<T>	**lower, alpha, print, graph**
<u>	<U>	**lower, alpha, print, graph**
<v>	<V>	**lower, alpha, print, graph**
<w>	<W>	**lower, alpha, print, graph**
<x>	<X>	**lower, alpha, print, graph**
<y>	<Y>	**lower, alpha, print, graph**
<z>	<Z>	**lower, alpha, print, graph**
<left-curly-bracket>		**punct, print, graph**
<vertical-line>		**punct, print, graph**
<right-curly-bracket>		**punct, print, graph**
<tilde>		**punct, print, graph**
<DEL>		**cntrl**

7.3.2 LC_COLLATE

The *LC_COLLATE* category provides a collation sequence definition for numerous utilities in the Shell and Utilities volume of IEEE Std 1003.1-200x (*sort*, *uniq*, and so on), regular expression matching (see Chapter 9 (on page 165)) and the *strcoll*( ), *strxfrm*( ), *wcscoll*( ), and *wcsxfrm*( ) functions in the System Interfaces volume of IEEE Std 1003.1-200x.

A collation sequence definition shall define the relative order between collating elements (characters and multi-character collating elements) in the locale. This order is expressed in terms of collation values; that is, by assigning each element one or more collation values (also known as collation weights). This does not imply that implementations shall assign such values, but that ordering of strings using the resultant collation definition in the locale behaves as if such assignment is done and used in the collation process. At least the following capabilities are provided:

1. **Multi-character collating elements**. Specification of multi-character collating elements (that is, sequences of two or more characters to be collated as an entity).

2. **User-defined ordering of collating elements**. Each collating element shall be assigned a collation value defining its order in the character (or basic) collation sequence. This ordering is used by regular expressions and pattern matching and, unless collation weights are explicitly specified, also as the collation weight to be used in sorting.

3. **Multiple weights and equivalence classes**. Collating elements can be assigned one or more (up to the limit {COLL_WEIGHTS_MAX}, as defined in **<limits.h>**) collating weights for use in sorting. The first weight is hereafter referred to as the primary weight.

4. **One-to-many mapping**. A single character is mapped into a string of collating elements.

5. **Equivalence class definition**. Two or more collating elements have the same collation value (primary weight).

6. **Ordering by weights**. When two strings are compared to determine their relative order, the two strings are first broken up into a series of collating elements; the elements in each successive pair of elements are then compared according to the relative primary weights for the elements. If equal, and more than one weight has been assigned, then the pairs of collating elements are recompared according to the relative subsequent weights, until either a pair of collating elements compare unequal or the weights are exhausted.

The following keywords shall be recognized in a collation sequence definition. They are described in detail in the following sections.

copy Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.

collating-element Define a collating-element symbol representing a multi-character collating element. This keyword is optional.

collating-symbol Define a collating symbol for use in collation order statements. This keyword is optional.

order_start Define collation rules. This statement shall be followed by one or more collation order statements, assigning character collation values and collation weights to collating elements.

order_end Specify the end of the collation-order statements.

7.3.2.1 *The collating-element Keyword*

In addition to the collating elements in the character set, the **collating-element** keyword can be used to define multi-character collating elements. The syntax is as follows:

```
"collating-element %s from \"%s\"\n", <collating-symbol>, <string>
```

The <*collating-symbol*> operand shall be a symbolic name, enclosed between angle brackets ('<' and '>'), and shall not duplicate any symbolic name in the current charmap file (if any), or any other symbolic name defined in this collation definition. The string operand is a string of two or more characters that collates as an entity. A <*collating-element*> defined via this keyword is only recognized with the *LC_COLLATE* category.

For example:

```
collating-element <ch> from "<c><h>"
collating-element <e-acute> from "<acute><e>"
collating-element <ll> from "ll"
```

7.3.2.2 The collating-symbol Keyword

This keyword shall be used to define symbols for use in collation sequence statements; that is, between the **order_start** and the **order_end** keywords. The syntax is as follows:

```
"collating-symbol %s\n", <collating-symbol>
```

The <*collating-symbol*> shall be a symbolic name, enclosed between angle brackets (`'<'` and `'>'`), and shall not duplicate any symbolic name in the current charmap file (if any), or any other symbolic name defined in this collation definition. A <*collating-symbol*> defined via this keyword is only recognized within the *LC_COLLATE* category.

For example:

```
collating-symbol <UPPER_CASE>
collating-symbol <HIGH>
```

The **collating-symbol** keyword defines a symbolic name that can be associated with a relative position in the character order sequence. While such a symbolic name does not represent any collating element, it can be used as a weight.

7.3.2.3 The order_start Keyword

The **order_start** keyword shall precede collation order entries and also define the number of weights for this collation sequence definition and other collation rules. The syntax is as follows:

```
"order_start %s;%s;...;%s\n", <sort-rules>, <sort-rules> ...
```

The operands to the **order_start** keyword are optional. If present, the operands define rules to be applied when strings are compared. The number of operands define how many weights each element is assigned; if no operands are present, one **forward** operand is assumed. If present, the first operand defines rules to be applied when comparing strings using the first (primary) weight; the second when comparing strings using the second weight, and so on. Operands shall be separated by semicolons (`';'`). Each operand shall consist of one or more collation directives, separated by commas (`','`). If the number of operands exceeds the {COLL_WEIGHTS_MAX} limit, the utility shall issue a warning message. The following directives shall be supported:

forward Specifies that comparison operations for the weight level shall proceed from start of string towards the end of string.

backward Specifies that comparison operations for the weight level shall proceed from end of string towards the beginning of string.

position Specifies that comparison operations for the weight level shall consider the relative position of elements in the strings not subject to **IGNORE**. The string containing an element not subject to **IGNORE** after the fewest collating elements subject to **IGNORE** from the start of the compare shall collate first. If both strings contain a character not subject to **IGNORE** in the same relative position, the collating values assigned to the elements shall determine the ordering. In case of equality, subsequent characters not subject to **IGNORE** shall be considered in the same manner.

The directives **forward** and **backward** are mutually-exclusive.

If no operands are specified, a single **forward** operand shall be assumed.

For example:

```
order_start    forward;backward
```

7.3.2.4 Collation Order

The **order_start** keyword shall be followed by collating identifier entries. The syntax for the collating element entries is as follows:

```
"%s %s;%s;...;%s\n", <collating-identifier>, <weight>, <weight>, ...
```

Each *collating-identifier* shall consist of either a character (in any of the forms defined in Section 7.3 (on page 120)), a <*collating-element*>, a <*collating-symbol*>, an ellipsis, or the special symbol **UNDEFINED**. The order in which collating elements are specified determines the character order sequence, such that each collating element shall compare less than the elements following it.

A <*collating-element*> shall be used to specify multi-character collating elements, and indicates that the character sequence specified via the <*collating-element*> is to be collated as a unit and in the relative order specified by its place.

A <*collating-symbol*> can be used to define a position in the relative order for use in weights. No weights shall be specified with a <*collating-symbol*>.

The ellipsis symbol specifies that a sequence of characters shall collate according to their encoded character values. It shall be interpreted as indicating that all characters with a coded character set value higher than the value of the character in the preceding line, and lower than the coded character set value for the character in the following line, in the current coded character set, shall be placed in the character collation order between the previous and the following character in ascending order according to their coded character set values. An initial ellipsis shall be interpreted as if the preceding line specified the NUL character, and a trailing ellipsis as if the following line specified the highest coded character set value in the current coded character set. An ellipsis shall be treated as invalid if the preceding or following lines do not specify characters in the current coded character set. The use of the ellipsis symbol ties the definition to a specific coded character set and may preclude the definition from being portable between implementations.

The symbol **UNDEFINED** shall be interpreted as including all coded character set values not specified explicitly or via the ellipsis symbol. Such characters shall be inserted in the character collation order at the point indicated by the symbol, and in ascending order according to their coded character set values. If no **UNDEFINED** symbol is specified, and the current coded character set contains characters not specified in this section, the utility shall issue a warning message and place such characters at the end of the character collation order.

The optional operands for each collation-element shall be used to define the primary, secondary, or subsequent weights for the collating element. The first operand specifies the relative primary weight, the second the relative secondary weight, and so on. Two or more collation-elements can be assigned the same weight; they belong to the same *equivalence class* if they have the same primary weight. Collation shall behave as if, for each weight level, elements subject to **IGNORE** are removed, unless the **position** collation directive is specified for the corresponding level with the **order_start** keyword. Then each successive pair of elements shall be compared according to the relative weights for the elements. If the two strings compare equal, the process shall be repeated for the next weight level, up to the limit {COLL_WEIGHTS_MAX}.

Weights shall be expressed as characters (in any of the forms specified in Section 7.3 (on page 120)), <*collating-symbol*>s, <*collating-element*>s, an ellipsis, or the special symbol **IGNORE**. A single character, a <*collating-symbol*>, or a <*collating-element*> shall represent the relative position in the character collating sequence of the character or symbol, rather than the character or characters themselves. Thus, rather than assigning absolute values to weights, a particular weight is expressed using the relative order value assigned to a collating element based on its order in the character collation sequence.

One-to-many mapping is indicated by specifying two or more concatenated characters or symbolic names. For example, if the <eszet> is given the string `"<s><s>"` as a weight, comparisons are performed as if all occurrences of the <eszet> are replaced by `"<s><s>"` (assuming that `"<s>"` has the collating weight `"<s>"`). If it is necessary to define <eszet> and `"<s><s>"` as an equivalence class, then a collating element must be defined for the string `"ss"`.

All characters specified via an ellipsis shall by default be assigned unique weights, equal to the relative order of characters. Characters specified via an explicit or implicit **UNDEFINED** special symbol shall by default be assigned the same primary weight (that is, they belong to the same equivalence class). An ellipsis symbol as a weight shall be interpreted to mean that each character in the sequence shall have unique weights, equal to the relative order of their character in the character collation sequence. The use of the ellipsis as a weight shall be treated as an error if the collating element is neither an ellipsis nor the special symbol **UNDEFINED**.

The special keyword **IGNORE** as a weight shall indicate that when strings are compared using the weights at the level where **IGNORE** is specified, the collating element shall be ignored; that is, as if the string did not contain the collating element. In regular expressions and pattern matching, all characters that are subject to **IGNORE** in their primary weight form an equivalence class.

An empty operand shall be interpreted as the collating element itself.

For example, the order statement:

```
<a>    <a>;<a>
```

is equal to:

```
<a>
```

An ellipsis can be used as an operand if the collating element was an ellipsis, and shall be interpreted as the value of each character defined by the ellipsis.

The collation order as defined in this section affects the interpretation of bracket expressions in regular expressions (see Section 9.3.5 (on page 168)).

For example:

```
order_start  forward;backward
UNDEFINED    IGNORE;IGNORE
<LOW>
<space>      <LOW>;<space>
...          <LOW>;...
<a>          <a>;<a>
<a-acute>    <a>;<a-acute>
<a-grave>    <a>;<a-grave>
<A>          <a>;<A>
<A-acute>    <a>;<A-acute>
<A-grave>    <a>;<A-grave>
<ch>         <ch>;<ch>
<Ch>         <ch>;<Ch>
<s>          <s>;<s>
<eszet>      "<s><s>";"<eszet><eszet>"
order_end
```

This example is interpreted as follows:

1. The **UNDEFINED** means that all characters not specified in this definition (explicitly or via the ellipsis) shall be ignored for collation purposes.
2. All characters between <space> and `'a'` shall have the same primary equivalence class and individual secondary weights based on their ordinal encoded values.
3. All characters based on the uppercase or lowercase character `'a'` belong to the same primary equivalence class.
4. The multi-character collating element <ch> is represented by the collating symbol <ch> and belongs to the same primary equivalence class as the multi-character collating element <Ch>.

7.3.2.5 The order_end Keyword

The collating order entries shall be terminated with an **order_end** keyword.

7.3.2.6 LC_COLLATE Category in the POSIX Locale

The collation sequence definition of the POSIX locale follows; the code listing depicts the *localedef* input.

```
LC_COLLATE
# This is the POSIX locale definition for the LC_COLLATE category.
# The order is the same as in the ASCII codeset.
order_start forward
<NUL>
<SOH>
<STX>
<ETX>
<EOT>
<ENQ>
<ACK>
<alert>
<backspace>
<tab>
<newline>
<vertical-tab>
<form-feed>
<carriage-return>
<SO>
<SI>
<DLE>
<DC1>
<DC2>
<DC3>
<DC4>
<NAK>
<SYN>
<ETB>
<CAN>
<EM>
<SUB>
<ESC>
```

```
<IS4>
<IS3>
<IS2>
<IS1>
<space>
<exclamation-mark>
<quotation-mark>
<number-sign>
<dollar-sign>
<percent-sign>
<ampersand>
<apostrophe>
<left-parenthesis>
<right-parenthesis>
<asterisk>
<plus-sign>
<comma>
<hyphen>
<period>
<slash>
<zero>
<one>
<two>
<three>
<four>
<five>
<six>
<seven>
<eight>
<nine>
<colon>
<semicolon>
<less-than-sign>
<equals-sign>
<greater-than-sign>
<question-mark>
<commercial-at>
<A>
<B>
<C>
<D>
<E>
<F>
<G>
<H>
<I>
<J>
<K>
<L>
<M>
<N>
<O>
```

```
<P>
<Q>
<R>
<S>
<T>
<U>
<V>
<W>
<X>
<Y>
<Z>
<left-square-bracket>
<backslash>
<right-square-bracket>
<circumflex>
<underscore>
<grave-accent>
<a>
<b>
<c>
<d>
<e>
<f>
<g>
<h>
<i>
<j>
<k>
<l>
<m>
<n>
<o>
<p>
<q>
<r>
<s>
<t>
<u>
<v>
<w>
<x>
<y>
<z>
<left-curly-bracket>
<vertical-line>
<right-curly-bracket>
<tilde>
<DEL>
order_end
#
END LC_COLLATE
```

7.3.3 LC_MONETARY

The *LC_MONETARY* category shall define the rules and symbols that are used to format monetary numeric information. This information is available through the *localeconv*( ) function and is used by the *strfmon*( ) function.

Some of the information is also available in an alternative form via the *nl_langinfo*( ) function (see CRNCYSTR in **<langinfo.h>**).

The following items are defined in this category of the locale. The item names are the keywords recognized by the *localedef* utility when defining a locale. They are also similar to the member names of the **lconv** structure defined in **<locale.h>**; see **<locale.h>** for the exact symbols in the header. The *localeconv*( ) function returns {CHAR_MAX} for unspecified integer items and the empty string ("") for unspecified or size zero string items.

In a locale definition file, the operands are strings, formatted as indicated by the grammar in Section 7.4 (on page 149). For some keywords, the strings can contain only integers. Keywords that are not provided, string values set to the empty string (""), or integer keywords set to −1, are used to indicate that the value is not available in the locale. The following keywords shall be recognized:

copy Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.

Note: This is a *localedef* utility keyword, unavailable through *localeconv*( ).

int_curr_symbol The international currency symbol. The operand shall be a four-character string, with the first three characters containing the alphabetic international currency symbol in accordance with those specified in the ISO 4217:1995 standard. The fourth character shall be the character used to separate the international currency symbol from the monetary quantity.

currency_symbol The string that shall be used as the local currency symbol.

mon_decimal_point The operand is a string containing the symbol that shall be used as the decimal delimiter (radix character) in monetary formatted quantities.

mon_thousands_sep The operand is a string containing the symbol that shall be used as a separator for groups of digits to the left of the decimal delimiter in formatted monetary quantities.

mon_grouping Define the size of each group of digits in formatted monetary quantities. The operand is a sequence of integers separated by semicolons. Each integer specifies the number of digits in each group, with the initial integer defining the size of the group immediately preceding the decimal delimiter, and the following integers defining the preceding groups. If the last integer is not −1, then the size of the previous group (if any) shall be repeatedly used for the remainder of the digits. If the last integer is −1, then no further grouping shall be performed.

positive_sign A string that shall be used to indicate a non-negative-valued formatted monetary quantity.

negative_sign A string that shall be used to indicate a negative-valued formatted monetary quantity.

int_frac_digits An integer representing the number of fractional digits (those to the right of the decimal delimiter) to be written in a formatted monetary quantity using **int_curr_symbol**.

frac_digits

An integer representing the number of fractional digits (those to the right of the decimal delimiter) to be written in a formatted monetary quantity using **currency_symbol**.

p_cs_precedes

An integer set to 1 if the **currency_symbol** precedes the value for a monetary quantity with a non-negative value, and set to 0 if the symbol succeeds the value.

p_sep_by_space

An integer set to 0 if no space separates the **currency_symbol** from the value for a monetary quantity with a non-negative value, set to 1 if a space separates the symbol from the value, and set to 2 if a space separates the symbol and the sign string, if adjacent.

n_cs_precedes

An integer set to 1 if the **currency_symbol** precedes the value for a monetary quantity with a negative value, and set to 0 if the symbol succeeds the value.

n_sep_by_space

An integer set to 0 if no space separates the **currency_symbol** from the value for a monetary quantity with a negative value, set to 1 if a space separates the symbol from the value, and set to 2 if a space separates the symbol and the sign string, if adjacent.

p_sign_posn

An integer set to a value indicating the positioning of the **positive_sign** for a monetary quantity with a non-negative value. The following integer values shall be recognized for **int_n_sign_posn**, **int_p_sign_posn**, **n_sign_posn**, and **p_sign_posn**:

0 Parentheses enclose the quantity and the **currency_symbol**.

1 The sign string precedes the quantity and the **currency_symbol**.

2 The sign string succeeds the quantity and the **currency_symbol**.

3 The sign string precedes the **currency_symbol**.

4 The sign string succeeds the **currency_symbol**.

n_sign_posn

An integer set to a value indicating the positioning of the **negative_sign** for a negative formatted monetary quantity.

int_p_cs_precedes

An integer set to 1 if the **int_curr_symbol** precedes the value for a monetary quantity with a non-negative value, and set to 0 if the symbol succeeds the value.

int_n_cs_precedes

An integer set to 1 if the **int_curr_symbol** precedes the value for a monetary quantity with a negative value, and set to 0 if the symbol succeeds the value.

int_p_sep_by_space

An integer to set 0 if no space separates the **int_curr_symbol** from the value for a monetary quantity with a non-negative value, set to 1 if a space separates the symbol from the value, and set to 2 if a space separates the symbol and the sign string, if adjacent.

int_n_sep_by_space

An integer set to 0 if no space separates the **int_curr_symbol** from the value for a monetary quantity with a negative value, set to 1 if a space separates the symbol from the value, and set to 2 if a space separates the symbol and the sign string, if adjacent.

int_p_sign_posn An integer set to a value indicating the positioning of the **positive_sign** for a positive monetary quantity formatted with the international format.

int_n_sign_posn An integer set to a value indicating the positioning of the **negative_sign** for a negative monetary quantity formatted with the international format.

7.3.3.1 LC_MONETARY Category in the POSIX Locale

XSI The monetary formatting definitions for the POSIX locale follow; the code listing depicting the *localedef* input, the table representing the same information with the addition of *localeconv*( ) and *nl_langinfo*( )formats. All values are unspecified in the POSIX locale.

```
LC_MONETARY
# This is the POSIX locale definition for
# the LC_MONETARY category.
#
int_curr_symbol      ""
currency_symbol      ""
mon_decimal_point    ""
mon_thousands_sep    ""
mon_grouping         -1
positive_sign        ""
negative_sign        ""
int_frac_digits      -1
frac_digits          -1
p_cs_precedes        -1
p_sep_by_space       -1
n_cs_precedes        -1
n_sep_by_space       -1
p_sign_posn          -1
n_sign_posn          -1
#
END LC_MONETARY
```

Item	langinfo Constant	POSIX Locale Value	localeconv( ) Value	localedef Value
int_curr_symbol	—	N/A	""	""
currency_symbol	CRNCYSTR	N/A	""	""
mon_decimal_point	—	N/A	""	""
mon_thousands_sep	—	N/A	""	""
mon_grouping	—	N/A	""	""
positive_sign	—	N/A	""	""
negative_sign	—	N/A	""	""
int_frac_digits	—	N/A	{CHAR_MAX}	–1
frac_digits	—	N/A	{CHAR_MAX}	–1
p_cs_precedes	CRNCYSTR	N/A	{CHAR_MAX}	–1
p_sep_by_space	—	N/A	{CHAR_MAX}	–1
n_cs_precedes	CRNCYSTR	N/A	{CHAR_MAX}	–1
n_sep_by_space	—	N/A	{CHAR_MAX}	–1
p_sign_posn	—	N/A	{CHAR_MAX}	–1
n_sign_posn	—	N/A	{CHAR_MAX}	–1

XSI In the preceding table, the **langinfo Constant** column represents an XSI-conformant extension. The entry N/A indicates that the value is not available in the POSIX locale.

7.3.4 LC_NUMERIC

XSI The *LC_NUMERIC* category shall define the rules and symbols that are used to format non-monetary numeric information. This information is available through the *localeconv*( ) function. Some of the information is also available in an alternative form via the *nl_langinfo*( ) function.

The following items are defined in this category of the locale. The item names are the keywords recognized by the *localedef* utility when defining a locale. They are also similar to the member names of the **lconv** structure defined in **<locale.h>**; see **<locale.h>** for the exact symbols in the header. The *localeconv*( ) function returns {CHAR_MAX} for unspecified integer items and the empty string ("") for unspecified or size zero string items.

In a locale definition file, the operands are strings, formatted as indicated by the grammar in Section 7.4 (on page 149). For some keywords, the strings can only contain integers. Keywords that are not provided, string values set to the empty string (""), or integer keywords set to −1, shall be used to indicate that the value is not available in the locale. The following keywords shall be recognized:

copy Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.

Note: This is a *localedef* utility keyword, unavailable through *localeconv*( ).

decimal_point The operand is a string containing the symbol that shall be used as the decimal delimiter (radix character) in numeric, non-monetary formatted quantities. This keyword cannot be omitted and cannot be set to the empty string. In contexts where standards limit the **decimal_point** to a single byte, the result of specifying a multi-byte operand shall be unspecified.

thousands_sep The operand is a string containing the symbol that shall be used as a separator for groups of digits to the left of the decimal delimiter in numeric, non-monetary formatted monetary quantities. In contexts where standards limit the **thousands_sep** to a single byte, the result of specifying a multi-byte operand shall be unspecified.

grouping Define the size of each group of digits in formatted non-monetary quantities. The operand is a sequence of integers separated by semicolons. Each integer specifies the number of digits in each group, with the initial integer defining the size of the group immediately preceding the decimal delimiter, and the following integers defining the preceding groups. If the last integer is not −1, then the size of the previous group (if any) shall be repeatedly used for the remainder of the digits. If the last integer is −1, then no further grouping shall be performed.

7.3.4.1 LC_NUMERIC Category in the POSIX Locale

The non-monetary numeric formatting definitions for the POSIX locale follow; the code listing depicting the *localedef* input, the table representing the same information with the addition of XSI *localeconv*( ) values, and *nl_langinfo*( ) constants.

```
LC_NUMERIC
# This is the POSIX locale definition for
# the LC_NUMERIC category.
#
decimal_point    "<period>"
thousands_sep    ""
grouping         -1
#
```

```
END LC_NUMERIC
```

Item	langinfo Constant	POSIX Locale Value	localeconv() Value	localedef Value
decimal_point	RADIXCHAR	"."	"."	.
thousands_sep	THOUSEP	N/A	""	""
grouping	—	N/A	""	−1

XSI In the preceding table, the **langinfo Constant** column represents an XSI-conforming extension. The entry N/A indicates that the value is not available in the POSIX locale.

7.3.5 LC_TIME

The *LC_TIME* category shall define the interpretation of the conversion specifications supported XSI by the *date* utility and shall affect the behavior of the *strftime*(), *wcsftime*(), *strptime*(), and *nl_langinfo*() functions. Since the interfaces for C-language access and locale definition differ significantly, they are described separately.

7.3.5.1 LC_TIME Locale Definition

For locale definition, the following mandatory keywords shall be recognized:

copy Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.

abday Define the abbreviated weekday names, corresponding to the `%a` conversion specification (conversion specification in the *strftime*(), *wcsftime*(), and *strptime*() functions). The operand shall consist of seven semicolon-separated strings, each surrounded by double-quotes. The first string shall be the abbreviated name of the day corresponding to Sunday, the second the abbreviated name of the day corresponding to Monday, and so on.

day Define the full weekday names, corresponding to the `%A` conversion specification. The operand shall consist of seven semicolon-separated strings, each surrounded by double-quotes. The first string is the full name of the day corresponding to Sunday, the second the full name of the day corresponding to Monday, and so on.

abmon Define the abbreviated month names, corresponding to the `%b` conversion specification. The operand shall consist of twelve semicolon-separated strings, each surrounded by double-quotes. The first string shall be the abbreviated name of the first month of the year (January), the second the abbreviated name of the second month, and so on.

mon Define the full month names, corresponding to the `%B` conversion specification. The operand shall consist of twelve semicolon-separated strings, each surrounded by double-quotes. The first string shall be the full name of the first month of the year (January), the second the full name of the second month, and so on.

d_t_fmt Define the appropriate date and time representation, corresponding to the `%c` conversion specification. The operand shall consist of a string containing any combination of characters and conversion specifications. In addition, the string can contain escape sequences defined in the table in Table 5-1 (on page 108) (`'\\'`, `'\a'`, `'\b'`, `'\f'`, `'\n'`, `'\r'`, `'\t'`, `'\v'`).

d_fmt Define the appropriate date representation, corresponding to the `%x` conversion specification. The operand shall consist of a string containing any combination of characters and conversion specifications. In addition, the string can contain escape sequences defined in the table in Table 5-1 (on page 108).

t_fmt Define the appropriate time representation, corresponding to the `%X` conversion specification. The operand shall consist of a string containing any combination of characters and conversion specifications. In addition, the string can contain escape sequences defined in the table in Table 5-1 (on page 108).

am_pm Define the appropriate representation of the *ante meridiem* and *post meridiem* strings, corresponding to the `%p` conversion specification. The operand shall consist of two strings, separated by a semicolon, each surrounded by double-quotes. The first string shall represent the *ante meridiem* designation, the last string the *post meridiem* designation.

t_fmt_ampm Define the appropriate time representation in the 12-hour clock format with **am_pm**, corresponding to the `%r` conversion specification. The operand shall consist of a string and can contain any combination of characters and conversion specifications. If the string is empty, the 12-hour format is not supported in the locale.

era Define how years are counted and displayed for each era in a locale. The operand shall consist of semicolon-separated strings. Each string shall be an era description segment with the format:

direction:*offset*:*start_date*:*end_date*:*era_name*:*era_format*

according to the definitions below. There can be as many era description segments as are necessary to describe the different eras.

Note: The start of an era might not be the earliest point in the era—it may be the latest. For example, the Christian era BC starts on the day before January 1, AD 1, and increases with earlier time.

direction Either a `'+'` or a `'−'` character. The `'+'` character shall indicate that years closer to the *start_date* have lower numbers than those closer to the *end_date*. The `'−'` character shall indicate that years closer to the *start_date* have higher numbers than those closer to the *end_date*.

offset The number of the year closest to the *start_date* in the era, corresponding to the `%Ey` conversion specification.

start_date A date in the form *yyyy*/*mm*/*dd*, where *yyyy*, *mm*, and *dd* are the year, month, and day numbers respectively of the start of the era. Years prior to AD 1 shall be represented as negative numbers.

end_date The ending date of the era, in the same format as the *start_date*, or one of the two special values `"−*"` or `"+*"`. The value `"−*"` shall indicate that the ending date is the beginning of time. The value `"+*"` shall indicate that the ending date is the end of time.

era_name A string representing the name of the era, corresponding to the `%EC` conversion specification.

era_format A string for formatting the year in the era, corresponding to the %EY conversion specification.

era_d_fmt Define the format of the date in alternative era notation, corresponding to the %Ex conversion specification.

era_t_fmt Define the locale's appropriate alternative time format, corresponding to the %EX conversion specification.

era_d_t_fmt Define the locale's appropriate alternative date and time format, corresponding to the %Ec conversion specification.

alt_digits Define alternative symbols for digits, corresponding to the %O modified conversion specification. The operand shall consist of semicolon-separated strings, each surrounded by double-quotes. The first string shall be the alternative symbol corresponding with zero, the second string the symbol corresponding with one, and so on. Up to 100 alternative symbol strings can be specified. The %O modifier shall indicate that the string corresponding to the value specified via the conversion specification shall be used instead of the value.

7.3.5.2 LC_TIME C-Language Access

XSI This section describes extensions to access information in the *LC_TIME* category using the *nl_langinfo*( ) function. This functionality is dependent on support of the XSI extension (and the rest of this section is not further shaded for this option).

The following constants used to identify items of *langinfo* data can be used as arguments to the *nl_langinfo*( ) function to access information in the *LC_TIME* category. These constants are defined in the **<langinfo.h>** header.

ABDAY_*x* The abbreviated weekday names (for example Sun), where *x* is a number from 1 to 7.

DAY_*x* The full weekday names (for example Sunday), where *x* is a number from 1 to 7.

ABMON_*x* The abbreviated month names (for example Jan), where *x* is a number from 1 to 12.

MON_*x* The full month names (for example January), where *x* is a number from 1 to 12.

D_T_FMT The appropriate date and time representation.

D_FMT The appropriate date representation.

T_FMT The appropriate time representation.

AM_STR The appropriate ante-meridiem affix.

PM_STR The appropriate post-meridiem affix.

T_FMT_AMPM The appropriate time representation in the 12-hour clock format with AM_STR and PM_STR.

ERA The era description segments, which describe how years are counted and displayed for each era in a locale. Each era description segment shall have the format:

`direction:offset:start_date:end_date:era_name:era_format`

according to the definitions below. There can be as many era description segments as are necessary to describe the different eras. Era description segments are separated by semicolons.

direction Either a '+' or a '−' character. The '+' character shall indicate that years closer to the *start_date* have lower numbers than those closer to the *end_date*. The '−' character shall indicate that years closer to the *start_date* have higher numbers than those closer to the *end_date*.

offset The number of the year closest to the *start_date* in the era.

start_date A date in the form *yyyy*/*mm*/*dd*, where *yyyy*, *mm*, and *dd* are the year, month, and day numbers respectively of the start of the era. Years prior to AD 1 shall be represented as negative numbers.

end_date The ending date of the era, in the same format as the *start_date*, or one of the two special values "−*" or "+*". The value "−*" shall indicate that the ending date is the beginning of time. The value "+*" shall indicate that the ending date is the end of time.

era_name The era, corresponding to the %EC conversion specification.

era_format The format of the year in the era, corresponding to the %EY conversion specification.

ERA_D_FMT The era date format.

ERA_T_FMT The locale's appropriate alternative time format, corresponding to the %EX conversion specification.

ERA_D_T_FMT The locale's appropriate alternative date and time format, corresponding to the %Ec conversion specification.

ALT_DIGITS The alternative symbols for digits, corresponding to the %O conversion specification modifier. The value consists of semicolon-separated symbols. The first is the alternative symbol corresponding to zero, the second is the symbol corresponding to one, and so on. Up to 100 alternative symbols may be specified.

7.3.5.3 *LC_TIME Category in the POSIX Locale*

The *LC_TIME* category definition of the POSIX locale follows; the code listing depicts the *localedef* input; the table represents the same information with the addition of *localedef* keywords, conversion specifiers used by the *date* utility and the *strftime*(), *wcsftime*(), and *strptime*() functions, XSI and *nl_langinfo*() constants.

```
LC_TIME
# This is the POSIX locale definition for
# the LC_TIME category.
#
# Abbreviated weekday names (%a)
abday      "<S><u><n>";"<M><o><n>";"<T><u><e>";"<W><e><d>";\
           "<T><h><u>";"<F><r><i>";"<S><a><t>"
#
# Full weekday names (%A)
```

```
day        "<S><u><n><d><a><y>";"<M><o><n><d><a><y>";\
           "<T><u><e><s><d><a><y>";"<W><e><d><n><e><s><d><a><y>";\
           "<T><h><u><r><s><d><a><y>";"<F><r><i><d><a><y>";\
           "<S><a><t><u><r><d><a><y>"
#
# Abbreviated month names (%b)
abmon      "<J><a><n>";"<F><e><b>";"<M><a><r>";\
           "<A><p><r>";"<M><a><y>";"<J><u><n>";\
           "<J><u><l>";"<A><u><g>";"<S><e><p>";\
           "<O><c><t>";"<N><o><v>";"<D><e><c>"
#
# Full month names (%B)
mon        "<J><a><n><u><a><r><y>";"<F><e><b><r><u><a><r><y>";\
           "<M><a><r><c><h>";"<A><p><r><i><l>";\
           "<M><a><y>";"<J><u><n><e>";\
           "<J><u><l><y>";"<A><u><g><u><s><t>";\
           "<S><e><p><t><e><m><b><e><r>";"<O><c><t><o><b><e><r>";\
           "<N><o><v><e><m><b><e><r>";"<D><e><c><e><m><b><e><r>"
#
# Equivalent of AM/PM (%p)      "AM";"PM"
am_pm      "<A><M>";"<P><M>"
#
# Appropriate date and time representation (%c)
#    "%a %b %e %H:%M:%S %Y"
d_t_fmt    "<percent-sign><a><space><percent-sign><b>\
            <space><percent-sign><e><space><percent-sign><H>\
            <colon><percent-sign><M><colon><percent-sign><S>\
            <space><percent-sign><Y>"
#
# Appropriate date representation (%x)   "%m/%d/%y"
d_fmt      "<percent-sign><m><slash><percent-sign><d>\
            <slash><percent-sign><y>"
#
# Appropriate time representation (%X)   "%H:%M:%S"
t_fmt      "<percent-sign><H><colon><percent-sign><M>\
            <colon><percent-sign><S>"
#
# Appropriate 12-hour 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>"
#
END LC_TIME
```

localedef Keyword	**langinfo Constant**	**Conversion Specification**	**POSIX Locale Value**
d_t_fmt	D_T_FMT	%c	"%a %b %e %H:%M:%S %Y"
d_fmt	D_FMT	%x	"%m/%d/%y"
t_fmt	T_FMT	%X	"%H:%M:%S"

localedef Keyword	**langinfo Constant**	**Conversion Specification**	**POSIX Locale Value**
am_pm	AM_STR	%p	"AM"
am_pm	PM_STR	%p	"PM"
t_fmt_ampm	T_FMT_AMPM	%r	"%I:%M:%S %p"
day	DAY_1	%A	"Sunday"
day	DAY_2	%A	"Monday"
day	DAY_3	%A	"Tuesday"
day	DAY_4	%A	"Wednesday"
day	DAY_5	%A	"Thursday"
day	DAY_6	%A	"Friday"
day	DAY_7	%A	"Saturday"
abday	ABDAY_1	%a	"Sun"
abday	ABDAY_2	%a	"Mon"
abday	ABDAY_3	%a	"Tue"
abday	ABDAY_4	%a	"Wed"
abday	ABDAY_5	%a	"Thu"
abday	ABDAY_6	%a	"Fri"
abday	ABDAY_7	%a	"Sat"
mon	MON_1	%B	"January"
mon	MON_2	%B	"February"
mon	MON_3	%B	"March"
mon	MON_4	%B	"April"
mon	MON_5	%B	"May"
mon	MON_6	%B	"June"
mon	MON_7	%B	"July"
mon	MON_8	%B	"August"
mon	MON_9	%B	"September"
mon	MON_10	%B	"October"
mon	MON_11	%B	"November"
mon	MON_12	%B	"December"
abmon	ABMON_1	%b	"Jan"
abmon	ABMON_2	%b	"Feb"
abmon	ABMON_3	%b	"Mar"
abmon	ABMON_4	%b	"Apr"
abmon	ABMON_5	%b	"May"
abmon	ABMON_6	%b	"Jun"
abmon	ABMON_7	%b	"Jul"
abmon	ABMON_8	%b	"Aug"
abmon	ABMON_9	%b	"Sep"
abmon	ABMON_10	%b	"Oct"
abmon	ABMON_11	%b	"Nov"
abmon	ABMON_12	%b	"Dec"
era	ERA	%EC, %Ey, %EY	N/A
era_d_fmt	ERA_D_FMT	%Ex	N/A
era_t_fmt	ERA_T_FMT	%EX	N/A
era_d_t_fmt	ERA_D_T_FMT	%Ec	N/A
alt_digits	ALT_DIGITS	%O	N/A

XSI In the preceding table, the **langinfo Constant** column represents an XSI-conformant extension.

The entry ‘‘N / A’’ indicates the value is not available in the POSIX locale.

7.3.6 LC_MESSAGES

The *LC_MESSAGES* category shall define the format and values used by various utilities for affirmative and negative responses. XSI This information is available through the *nl_langinfo*( ) function.

XSI The message catalog used by the standard utilities and selected by the *catopen*( ) function shall be determined by the setting of *NLSPATH*; see Chapter 8 (on page 157). The *LC_MESSAGES* category can be specified as part of an *NLSPATH* substitution field.

The following keywords shall be recognized as part of the locale definition file.

copy Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.

Note: This is a *localedef* keyword, unavailable through *nl_langinfo*( ).

yesexpr The operand consists of an extended regular expression (see Section 9.4 (on page 171)) that describes the acceptable affirmative response to a question expecting an affirmative or negative response.

noexpr The operand consists of an extended regular expression that describes the acceptable negative response to a question expecting an affirmative or negative response.

7.3.6.1 LC_MESSAGES Category for the POSIX Locale

The format and values for affirmative and negative responses of the POSIX locale follow; the code listing depicting the *localedef* input, the table representing the same information XSI with the addition of *nl_langinfo*( ) constants.

```
LC_MESSAGES
# This is the POSIX locale definition for
# the LC_MESSAGES category.
#
yesexpr "<circumflex><left-square-bracket><y><Y><right-square-bracket>"
#
noexpr  "<circumflex><left-square-bracket><n><N><right-square-bracket>"
#
END LC_MESSAGES
```

localedef Keyword	langinfo Constant	POSIX Locale Value
yesexpr	YESEXPR	`"^[yY]"`
noexpr	NOEXPR	`"^[nN]"`

XSI In the preceding table, the **langinfo Constant** column represents an XSI-conformant extension.

7.4 Locale Definition Grammar

The grammar and lexical conventions in this section shall together describe the syntax for the locale definition source. The general conventions for this style of grammar are described in the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 1.10, Grammar Conventions. The grammar shall take precedence over the text in this chapter.

7.4.1 Locale Lexical Conventions

The lexical conventions for the locale definition grammar are described in this section.

The following tokens shall be processed (in addition to those string constants shown in the grammar):

LOC_NAME A string of characters representing the name of a locale.

CHAR Any single character.

NUMBER A decimal number, represented by one or more decimal digits.

COLLSYMBOL A symbolic name, enclosed between angle brackets. The string cannot duplicate any charmap symbol defined in the current charmap (if any), or a **COLLELEMENT** symbol.

COLLELEMENT A symbolic name, enclosed between angle brackets, which cannot duplicate either any charmap symbol or a **COLLSYMBOL** symbol.

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

CHARSYMBOL A symbolic name, enclosed between angle brackets, from the current charmap (if any).

OCTAL_CHAR One or more octal representations of the encoding of each byte in a single character. The octal representation consists of an escape character (normally a backslash) followed by two or more octal digits.

HEX_CHAR One or more hexadecimal representations of the encoding of each byte in a single character. The hexadecimal representation consists of an escape character followed by the constant *x* and two or more hexadecimal digits.

DECIMAL_CHAR One or more decimal representations of the encoding of each byte in a single character. The decimal representation consists of an escape character followed by a character `'d'` and two or more decimal digits.

ELLIPSIS The string `"..."`.

EXTENDED_REG_EXP An extended regular expression as defined in the grammar in Section 9.5 (on page 175).

EOL The line termination character newline.

7.4.2 Locale Grammar

This section presents the grammar for the locale definition.

```
%token               LOC_NAME
%token               CHAR
%token               NUMBER
%token               COLLSYMBOL COLLELEMENT
%token               CHARSYMBOL OCTAL_CHAR HEX_CHAR DECIMAL_CHAR
%token               ELLIPSIS
%token               EXTENDED_REG_EXP
%token               EOL

%start               locale_definition

%%

locale_definition    : global_statements locale_categories
                     |                   locale_categories
                     ;

global_statements    : global_statements symbol_redefine
                     | symbol_redefine
                     ;

symbol_redefine      : 'escape_char'  CHAR EOL
                     | 'comment_char' CHAR EOL
                     ;

locale_categories    : locale_categories locale_category
                     | locale_category
                     ;

locale_category      : lc_ctype | lc_collate | lc_messages
                     | lc_monetary | lc_numeric | lc_time
                     ;

/* The following grammar rules are common to all categories */

char_list            : char_list char_symbol
                     | char_symbol
                     ;

char_symbol          : CHAR | CHARSYMBOL
                     | OCTAL_CHAR | HEX_CHAR | DECIMAL_CHAR
                     ;

elem_list            : elem_list char_symbol
                     | elem_list COLLSYMBOL
                     | elem_list COLLELEMENT
                     | char_symbol
                     | COLLSYMBOL
                     | COLLELEMENT
                     ;

symb_list            : symb_list COLLSYMBOL
                     | COLLSYMBOL
                     ;
```

```
locale_name         : LOC_NAME
                    | ’"’ LOC_NAME ’"’
                    ;
/* The following is the LC_CTYPE category grammar */
lc_ctype            : ctype_hdr ctype_keywords         ctype_tlr
                    | ctype_hdr ’copy’ locale_name EOL ctype_tlr
                    ;
ctype_hdr           : ’LC_CTYPE’ EOL
                    ;
ctype_keywords      : ctype_keywords ctype_keyword
                    | ctype_keyword
                    ;
ctype_keyword       : charclass_keyword charclass_list EOL
                    | charconv_keyword charconv_list EOL
                    | ’charclass’ charclass_namelist EOL
                    ;
charclass_namelist  : charclass_namelist ’;’ CHARCLASS
                    | CHARCLASS
                    ;
charclass_keyword   : ’upper’ | ’lower’ | ’alpha’ | ’digit’
                    | ’punct’ | ’xdigit’ | ’space’ | ’print’
                    | ’graph’ | ’blank’ | ’cntrl’ | ’alnum’
                    | CHARCLASS
                    ;
charclass_list      : charclass_list ’;’ char_symbol
                    | charclass_list ’;’ ELLIPSIS ’;’ char_symbol
                    | char_symbol
                    ;
charconv_keyword    : ’toupper’
                    | ’tolower’
                    ;
charconv_list       : charconv_list ’;’ charconv_entry
                    | charconv_entry
                    ;
charconv_entry      : ’(’ char_symbol ’,’ char_symbol ’)’
                    ;
ctype_tlr           : ’END’ ’LC_CTYPE’ EOL
                    ;
/* The following is the LC_COLLATE category grammar */
lc_collate          : collate_hdr collate_keywords       collate_tlr
                    | collate_hdr ’copy’ locale_name EOL collate_tlr
                    ;
collate_hdr         : ’LC_COLLATE’ EOL
                    ;
```

```
collate_keywords    :                order_statements
                    | opt_statements order_statements
                    ;

opt_statements      : opt_statements collating_symbols
                    | opt_statements collating_elements
                    | collating_symbols
                    | collating_elements
                    ;

collating_symbols   : 'collating-symbol' COLLSYMBOL EOL
                    ;

collating_elements  : 'collating-element' COLLELEMENT
                    | 'from' '"' elem_list '"' EOL
                    ;

order_statements    : order_start collation_order order_end
                    ;

order_start         : 'order_start' EOL
                    | 'order_start' order_opts EOL
                    ;

order_opts          : order_opts ';' order_opt
                    | order_opt
                    ;

order_opt           : order_opt ',' opt_word
                    | opt_word
                    ;

opt_word            : 'forward' | 'backward' | 'position'
                    ;

collation_order     : collation_order collation_entry
                    | collation_entry
                    ;

collation_entry     : COLLSYMBOL EOL
                    | collation_element weight_list EOL
                    | collation_element             EOL
                    ;

collation_element   : char_symbol
                    | COLLELEMENT
                    | ELLIPSIS
                    | 'UNDEFINED'
                    ;

weight_list         : weight_list ';' weight_symbol
                    | weight_list ';'
                    | weight_symbol
                    ;

weight_symbol       : /* empty */
                    | char_symbol
                    | COLLSYMBOL
                    | '"' elem_list '"'
                    | ’"’ symb_list ’"’
                    | ELLIPSIS
                    | ’IGNORE’
                    ;

order_end           : ’order_end’ EOL
                    ;

collate_tlr         : ’END’ ’LC_COLLATE’ EOL
                    ;

/* The following is the LC_MESSAGES category grammar */

lc_messages         : messages_hdr messages_keywords      messages_tlr
                    | messages_hdr ’copy’ locale_name EOL messages_tlr
                    ;

messages_hdr        : ’LC_MESSAGES’ EOL
                    ;

messages_keywords   : messages_keywords messages_keyword
                    | messages_keyword
                    ;

messages_keyword    : ’yesexpr’ ’"’ EXTENDED_REG_EXP ’"’ EOL
                    | ’noexpr’  ’"’ EXTENDED_REG_EXP ’"’ EOL
                    ;

messages_tlr        : ’END’ ’LC_MESSAGES’ EOL
                    ;

/* The following is the LC_MONETARY category grammar */

lc_monetary         : monetary_hdr monetary_keywords       monetary_tlr
                    | monetary_hdr ’copy’ locale_name EOL  monetary_tlr
                    ;

monetary_hdr        : ’LC_MONETARY’ EOL
                    ;

monetary_keywords   : monetary_keywords monetary_keyword
                    | monetary_keyword
                    ;

monetary_keyword    : mon_keyword_string mon_string EOL
                    | mon_keyword_char NUMBER EOL
                    | mon_keyword_char ’-1’   EOL
                    | mon_keyword_grouping mon_group_list EOL
                    ;

mon_keyword_string  : ’int_curr_symbol’ | ’currency_symbol’
                    | ’mon_decimal_point’ | ’mon_thousands_sep’
                    | ’positive_sign’ | ’negative_sign’
                    ;

mon_string          : ’"’ char_list ’"’
                    | ’""’
                    ;
```

```
mon_keyword_char    : ’int_frac_digits’ | ’frac_digits’
                    | ’p_cs_precedes’ | ’p_sep_by_space’
                    | ’n_cs_precedes’ | ’n_sep_by_space’
                    | ’p_sign_posn’ | ’n_sign_posn’
                    ;

mon_keyword_grouping : ’mon_grouping’
                    ;

mon_group_list      : NUMBER
                    | mon_group_list ’;’ NUMBER
                    ;

monetary_tlr        : ’END’ ’LC_MONETARY’ EOL
                    ;

/* The following is the LC_NUMERIC category grammar */

lc_numeric          : numeric_hdr numeric_keywords       numeric_tlr
                    | numeric_hdr ’copy’ locale_name EOL numeric_tlr
                    ;

numeric_hdr         : ’LC_NUMERIC’ EOL
                    ;

numeric_keywords    : numeric_keywords numeric_keyword
                    | numeric_keyword
                    ;

numeric_keyword     : num_keyword_string num_string EOL
                    | num_keyword_grouping num_group_list EOL
                    ;

num_keyword_string  : ’decimal_point’
                    | ’thousands_sep’
                    ;

num_string          : ’"’ char_list ’"’
                    | ’""’
                    ;

num_keyword_grouping: ’grouping’
                    ;

num_group_list      : NUMBER
                    | num_group_list ’;’ NUMBER
                    ;

numeric_tlr         : ’END’ ’LC_NUMERIC’ EOL
                    ;

/* The following is the LC_TIME category grammar */

lc_time             : time_hdr time_keywords          time_tlr
                    | time_hdr ’copy’ locale_name EOL time_tlr
                    ;

time_hdr            : ’LC_TIME’ EOL
                    ;
```

```
time_keywords       : time_keywords time_keyword
                    | time_keyword
                    ;

time_keyword        : time_keyword_name time_list EOL
                    | time_keyword_fmt time_string EOL
                    | time_keyword_opt time_list EOL
                    ;

time_keyword_name   : ’abday’ | ’day’ | ’abmon’ | ’mon’
                    ;

time_keyword_fmt    : ’d_t_fmt’ | ’d_fmt’ | ’t_fmt’
                    | ’am_pm’ | ’t_fmt_ampm’
                    ;

time_keyword_opt    : ’era’ | ’era_d_fmt’ | ’era_t_fmt’
                    | ’era_d_t_fmt’ | ’alt_digits’
                    ;

time_list           : time_list ’;’ time_string
                    | time_string
                    ;

time_string         : ’"’ char_list ’"’
                    ;

time_tlr            : ’END’ ’LC_TIME’ EOL
                    ;
```

Chapter 8

Environment Variables

8.1 Environment Variable Definition

Environment variables defined in this chapter affect the operation of multiple utilities, functions, and applications. There are other environment variables that are of interest only to specific utilities. Environment variables that apply to a single utility only are defined as part of the utility description. See the ENVIRONMENT VARIABLES section of the utility descriptions in the Shell and Utilities volume of IEEE Std 1003.1-200x for information on environment variable usage.

The value of an environment variable is a string of characters. For a C-language program, an array of strings called the environment shall be made available when a process begins. The array is pointed to by the external variable *environ*, which is defined as:

```
extern char **environ;
```

These strings have the form *name*=*value*; *name*s shall not contain the character `'='`. For values to be portable across systems conforming to IEEE Std 1003.1-200x, the value shall be composed of characters from the portable character set (except NUL and as indicated below). There is no meaning associated with the order of strings in the environment. If more than one string in a process' environment has the same *name*, the consequences are undefined. |

Environment variable names used by the utilities in the Shell and Utilities volume of IEEE Std 1003.1-200x consist solely of uppercase letters, digits, and the `'_'` (underscore) from the characters defined in Table 6-1 (on page 111) and do not begin with a digit. Other characters may be permitted by an implementation; applications shall tolerate the presence of such names. Uppercase and lowercase letters shall retain their unique identities and shall not be folded together. The name space of environment variable names containing lowercase letters is reserved for applications. Applications can define any environment variables with names from this name space without modifying the behavior of the standard utilities. |

Note: Other applications may have difficulty dealing with environment variable names that start with a digit. For this reason, use of such names is not recommended anywhere. |

The *values* that the environment variables may be assigned are not restricted except that they are considered to end with a null byte and the total space used to store the environment and the arguments to the process is limited to {ARG_MAX} bytes.

Other *name*=*value* pairs may be placed in the environment by, for example, calling any of the XSI *setenv*( ), *unsetenv*( ), or *putenv*( ) functions, manipulating the *environ* variable, or by using *envp* arguments when creating a process; see *exec* in the System Interfaces volume of IEEE Std 1003.1-200x.

It is unwise to conflict with certain variables that are frequently exported by widely used command interpreters and applications:

ARFLAGS
CC
CDPATH
CFLAGS
CHARSET
COLUMNS
DATEMSK
DEAD
EDITOR
ENV
EXINIT
FC
FCEDIT
FFLAGS
GET
GFLAGS
HISTFILE
HISTORY
HISTSIZE
HOME
IFS
LANG
LC_ALL
LC_COLLATE
LC_CTYPE
LC_MESSAGES
LC_MONETARY
LC_NUMERIC
LC_TIME
LDFLAGS
LEX
LFLAGS
LINENO
LINES
LISTER
LOGNAME
LPDEST
MAIL
MAILCHECK
MAILER
MAILPATH
MAILRC
MAKEFLAGS
MAKESHELL
MANPATH
MBOX
MORE
MSGVERB
NLSPATH
NPROC
OLDPWD
OPTARG
OPTERR
OPTIND
PAGER
PATH
PPID
PRINTER
PROCLANG
PROJECTDIR
PS1
PS2
PS3
PS4
PWD
RANDOM
SECONDS
SHELL
TERM
TERMCAP
TERMINFO
TMPDIR
TZ
USER
VISUAL
YACC
YFLAGS

If the variables in the following two sections are present in the environment during the execution of an application or utility, they shall be given the meaning described below. Some are placed into the environment by the implementation at the time the user logs in; all can be added or changed by the user or any ancestor of the current process. The implementation adds or changes environment variables named in IEEE Std 1003.1-200x only as specified in IEEE Std 1003.1-200x. If they are defined in the application's environment, the utilities in the Shell and Utilities volume of IEEE Std 1003.1-200x and the functions in the System Interfaces volume of IEEE Std 1003.1-200x assume they have the specified meaning. Conforming applications shall not set these environment variables to have meanings other than as described. See *getenv*( ) and the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.12, Shell Execution Environment for methods of accessing these variables.

8.2 Internationalization Variables

This section describes environment variables that are relevant to the operation of internationalized interfaces described in IEEE Std 1003.1-200x.

Users may use the following environment variables to announce specific localization requirements to applications. Applications can retrieve this information using the *setlocale*( ) function to initialize the correct behavior of the internationalized interfaces. The descriptions of the internationalization environment variables describe the resulting behavior only when the application locale is initialized in this way. The use of the internationalization variables by utilities described in the Shell and Utilities volume of IEEE Std 1003.1-200x are described in the ENVIRONMENT VARIABLES section for those utilities in addition to the global effects described in this section.

LANG This variable shall determine the locale category for native language, local customs, and coded character set in the absence of the *LC_ALL* and other *LC_** (*LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*, *LC_TIME*) environment variables. This can be used by applications to determine the language to use for error messages and instructions, collating sequences, date formats, and so on.

LC_ALL This variable shall determine the values for all locale categories. The value of the *LC_ALL* environment variable has precedence over any of the other environment variables starting with *LC_*(*LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*, *LC_TIME*) and the *LANG* environment variable.

LC_COLLATE This variable shall determine the locale category for character collation. It determines collation information for regular expressions and sorting, including equivalence classes and multi-character collating elements, in various utilities and the *strcoll*( ) and *strxfrm*( ) functions. Additional semantics of this variable, if any, are implementation-defined.

LC_CTYPE This variable shall determine the locale category for character handling functions, such as *tolower*( ), *toupper*( ), and *isalpha*( ). This environment variable determines the interpretation of sequences of bytes of text data as characters (for example, single as opposed to multi-byte characters), the classification of characters (for example, alpha, digit, graph), and the behavior of character classes. Additional semantics of this variable, if any, are implementation-defined.

LC_MESSAGES This variable shall determine the locale category for processing affirmative and negative responses and the language and cultural conventions in which messages should be written. XSI It also affects the behavior of the *catopen*( ) function in determining the message catalog. Additional semantics of this variable, if any, are implementation-defined. The language and cultural conventions of diagnostic and informative messages whose format is unspecified by IEEE Std 1003.1-200x should be affected by the setting of *LC_MESSAGES*.

LC_MONETARY This variable shall determine the locale category for monetary-related numeric formatting information. Additional semantics of this variable, if any, are implementation-defined.

LC_NUMERIC This variable shall determine the locale category for numeric formatting (for example, thousands separator and radix character) information in various utilities as well as the formatted I/O operations in *printf*( ) and *scanf*( ) and the string conversion functions in *strtod*( ). Additional semantics of this variable, if any, are implementation-defined.

LC_TIME This variable shall determine the locale category for date and time formatting information. It affects the behavior of the time functions in *strftime*( ). Additional semantics of this variable, if any, are implementation-defined.

XSI *NLSPATH* This variable shall contain a sequence of templates that the *catopen*( ) function uses when attempting to locate message catalogs. Each template consists of an optional prefix, one or more conversion specifications, a filename, and an optional suffix.

For example:

```
NLSPATH="/system/nlslib/%N.cat"
```

defines that *catopen*( ) should look for all message catalogs in the directory **/system/nlslib**, where the catalog name should be constructed from the *name* parameter passed to *catopen*( ) (%N), with the suffix **.cat**.

Conversion specifications consist of a '%' symbol, followed by a single-letter keyword. The following keywords are currently defined:

`%N` The value of the *name* parameter passed to *catopen*( ).

`%L` The value of the *LC_MESSAGES* category.

`%l` The *language* element from the *LC_MESSAGES* category.

`%t` The *territory* element from the *LC_MESSAGES* category.

`%c` The *codeset* element from the *LC_MESSAGES* category.

`%%` A single `'%'` character.

An empty string is substituted if the specified value is not currently defined. The separators underscore (`'_'`) and period (`'.'`) are not included in the `%t` and `%c` conversion specifications.

Templates defined in *NLSPATH* are separated by colons (`':'`). A leading or two adjacent colons `"::"` is equivalent to specifying `%N`. For example:

```
    NLSPATH=":%N.cat:/nlslib/%L/%N.cat"
```

indicates to *catopen*( ) that it should look for the requested message catalog in *name*, *name*.**cat**, and **/nlslib/***category*/*name*.**cat**, where *category* is the value of the *LC_MESSAGES* category of the current locale.

Users should not set the *NLSPATH* variable unless they have a specific reason to override the default system path. Setting *NLSPATH* to override the default system path produces undefined results in the standard utilities and in applications with appropriate privileges.

XSI

The environment variables *LANG*, *LC_ALL*, *LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*, *LC_TIME*, and *NLSPATH* provide for the support of internationalized applications. The standard utilities shall make use of these environment variables as described in this section and the individual ENVIRONMENT VARIABLES sections for the utilities. If these variables specify locale categories that are not based upon the same underlying codeset, the results are unspecified.

The values of locale categories shall be determined by a precedence order; the first condition met below determines the value:

1. If the *LC_ALL* environment variable is defined and is not null, the value of *LC_ALL* shall be used.

2. If the *LC_** environment variable (*LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*, *LC_TIME*) is defined and is not null, the value of the environment variable shall be used to initialize the category that corresponds to the environment variable.

3. If the *LANG* environment variable is defined and is not null, the value of the *LANG* environment variable shall be used.

4. If the *LANG* environment variable is not set or is set to the empty string, the implementation-defined default locale shall be used.

If the locale value is `"C"` or `"POSIX"`, the POSIX locale shall be used and the standard utilities behave in accordance with the rules in Section 7.2 (on page 120) for the associated category.

If the locale value begins with a slash, it shall be interpreted as the pathname of a file that was created in the output format used by the *localedef* utility; see OUTPUT FILES under *localedef*. Referencing such a pathname shall result in that locale being used for the indicated category.

XSI If the locale value has the form:

```
language[_territory][.codeset]
```

it refers to an implementation-provided locale, where settings of language, territory, and codeset are implementation-defined.

LC_COLLATE, *LC_CTYPE*, *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*, and *LC_TIME* are defined to accept an additional field *@modifier*, which allows the user to select a specific instance of localization data within a single category (for example, for selecting the dictionary as opposed to the character ordering of data). The syntax for these environment variables is thus defined as:

```
[language[_territory][.codeset][@modifier]]
```

For example, if a user wanted to interact with the system in French, but required to sort German text files, *LANG* and *LC_COLLATE* could be defined as:

```
LANG=Fr_FR
LC_COLLATE=De_DE
```

This could be extended to select dictionary collation (say) by use of the *@modifier* field; for example:

```
LC_COLLATE=De_DE@dict
```

An implementation may support other formats.

If the locale value is not recognized by the implementation, the behavior is unspecified.

At runtime, these values are bound to a program's locale by calling the *setlocale*( ) function.

Additional criteria for determining a valid locale name are implementation-defined.

8.3 Other Environment Variables

COLUMNS This variable shall represent a decimal integer >0 used to indicate the user's preferred width in column positions for the terminal screen or window; see Section 3.103 (on page 47). If this variable is unset or null, the implementation determines the number of columns, appropriate for the terminal or window, in an unspecified manner. When *COLUMNS* is set, any terminal-width information implied by *TERM* is overridden. Users and conforming applications should not set *COLUMNS* unless they wish to override the system selection and produce output unrelated to the terminal characteristics.

Users should not need to set this variable in the environment unless there is a specific reason to override the implementation's default behavior, such as to display data in an area arbitrarily smaller than the terminal or window.

XSI *DATEMSK* Indicates the pathname of the template file used by *getdate*( ).

HOME The system shall initialize this variable at the time of login to be a pathname of the user's home directory. See **<pwd.h>**.

LINES This variable shall represent a decimal integer >0 used to indicate the user's preferred number of lines on a page or the vertical screen or window size in lines. A line in this case is a vertical measure large enough to hold the tallest character in the character set being displayed. If this variable is unset or null, the implementation determines the number of lines, appropriate for the terminal or window (size, terminal baud rate, and so on), in an unspecified manner. When *LINES* is set, any terminal-height information implied by *TERM* is overridden. Users and conforming applications should not set *LINES* unless they wish to override the system selection and produce output unrelated to the terminal characteristics.

Users should not need to set this variable in the environment unless there is a specific reason to override the implementation's default behavior, such as to display data in an area arbitrarily smaller than the terminal or window.

LOGNAME The system shall initialize this variable at the time of login to be the user's login name. See **<pwd.h>**. For a value of *LOGNAME* to be portable across implementations of IEEE Std 1003.1-200x, the value should be composed of characters from the portable filename character set.

XSI *MSGVERB* Describes which message components shall be used in writing messages by *fmtmsg*( ).

PATH This variable shall represent the sequence of path prefixes that certain functions and utilities apply in searching for an executable file known only by a filename. The prefixes shall be separated by a colon (`':'`). When a non-zero-length prefix is applied to this filename, a slash shall be inserted between the prefix and the filename. A zero-length prefix is a legacy feature that indicates the current working directory. It appears as two adjacent colons (`"::"`), as an initial colon preceding the rest of the list, or as a trailing colon following the rest of the list. A strictly conforming application shall use an actual pathname (such as **.**) to represent the current working directory in *PATH*. The list shall be searched from beginning to end, applying the filename to each prefix, until an executable file with the specified name and appropriate execution permissions is found. If the pathname being sought contains a slash, the search through the path prefixes shall not be performed. If the pathname begins with a slash, the specified path is resolved (see Section 4.11 (on page 98)). If *PATH* is unset or is set to null, the path search is implementation-defined.

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.

SHELL This variable shall represent a pathname of the user's preferred command language interpreter. If this interpreter does not conform to the Shell Command Language in the Shell and Utilities volume of IEEE Std 1003.1-200x, Chapter 2, Shell Command Language, utilities may behave differently from those described in IEEE Std 1003.1-200x.

TMPDIR This variable shall represent a pathname of a directory made available for programs that need a place to create temporary files.

TERM This variable shall represent the terminal type for which output is to be prepared. This information is used by utilities and application programs wishing to exploit special capabilities specific to a terminal. The format and allowable values of this environment variable are unspecified.

TZ This variable shall represent timezone information. The contents of the environment variable named *TZ* shall be used by the *ctime*( ), *localtime*( ), *strftime*( ), and *mktime*( ) functions, and by various utilities, to override the default timezone. The value of *TZ* has one of the two forms (spaces inserted for clarity):

`:`*`characters`*

or:

`std offset dst offset, rule`

If *TZ* is of the first format (that is, if the first character is a colon), the characters following the colon are handled in an implementation-defined manner.

The expanded format (for all *TZ*s whose value does not have a colon as the first character) is as follows:

`stdoffset``[`*`dst`*`[`*`offset`*`][,`*`start`*`[/`*`time`*`],`*`end`*`[/`*`time`*`]]]`

Where:

std and *dst* Indicate no less than three, nor more than {TZNAME_MAX}, bytes that are the designation for the standard (*std*) or the alternative (*dst*—such as Daylight Savings Time) timezone. Only *std* is required; if *dst* is missing, then the alternative time does not apply in this locale.

Each of these fields may occur in either of two formats quoted or unquoted:

— In the quoted form, the first character shall be the less-than (`'<'`) character and the last character shall be the greater-than (`'>'`) character. All characters between these quoting characters shall be alphanumeric characters in the current locale, the plus-sign (`'+'`) character, or the minus-sign (`'−'`) character. The *std* and *dst* fields in this case shall not include the quoting characters.

— In the unquoted form, all characters in these fields shall be alphabetic characters in the current locale.

The interpretation of these fields is unspecified if either field is less than three bytes (except for the case when *dst* is missing), more than {TZNAME_MAX} bytes, or if they contain characters other than those specified.

offset Indicates the value added to the local time to arrive at Coordinated Universal Time. The *offset* has the form:

`hh``[:`*`mm`*`[:`*`ss`*`]]`

The minutes (*mm*) and seconds (*ss*) are optional. The hour (*hh*) shall be required and may be a single digit. The *offset* following *std* shall be required. If no *offset* follows *dst*, the alternative time is assumed to be one hour ahead of standard time. One or more digits may be used; the value is always interpreted as a decimal number. The hour shall be between zero and 24, and the minutes (and seconds)—if present—between zero and 59. The result of using values outside of this range is unspecified. If preceded by a `'−'`, the timezone shall be east of the Prime Meridian; otherwise, it shall be west (which may be indicated by an optional preceding `'+'`).

rule Indicates when to change to and back from the alternative time. The *rule* has the form:

```
date[/time],date[/time]
```

where the first *date* describes when the change from standard to alternative time occurs and the second *date* describes when the change back happens. Each *time* field describes when, in current local time, the change to the other time is made.

The format of *date* is one of the following:

J*n* The Julian day *n* (1 ≤ *n* ≤ 365). Leap days shall not be counted. That is, in all years—including leap years—February 28 is day 59 and March 1 is day 60. It is impossible to refer explicitly to the occasional February 29.

n The zero-based Julian day (0 ≤ *n* ≤ 365). Leap days shall be counted, and it is possible to refer to February 29.

M*m.n.d* The *d*'th day (0 ≤ *d* ≤ 6) of week *n* of month *m* of the year (1 ≤ *n* ≤ 5, 1 ≤ *m* ≤ 12, where week 5 means ‘‘the last *d* day in month *m*’’ which may occur in either the fourth or the fifth week). Week 1 is the first week in which the *d*'th day occurs. Day zero is Sunday.

The *time* has the same format as *offset* except that no leading sign (`'−'` or `'+'`) is allowed. The default, if *time* is not given, shall be 02:00:00.

Chapter 9

Regular Expressions

Regular Expressions (REs) provide a mechanism to select specific strings from a set of character strings.

Regular expressions are a context-independent syntax that can represent a wide variety of character sets and character set orderings, where these character sets are interpreted according to the current locale. While many regular expressions can be interpreted differently depending on the current locale, many features, such as character class expressions, provide for contextual invariance across locales.

The Basic Regular Expression (BRE) notation and construction rules in Section 9.3 (on page 167) shall apply to most utilities supporting regular expressions. Some utilities, instead, support the Extended Regular Expressions (ERE) described in Section 9.4 (on page 171); any exceptions for both cases are noted in the descriptions of the specific utilities using regular expressions. Both BREs and EREs are supported by the Regular Expression Matching interface in the System Interfaces volume of IEEE Std 1003.1-200x under *regcomp*( ), *regexec*( ), and related functions.

9.1 Regular Expression Definitions

For the purposes of this section, the following definitions shall apply:

entire regular expression

The concatenated set of one or more BREs or EREs that make up the pattern specified for string selection.

matched

A sequence of zero or more characters shall be said to be matched by a BRE or ERE when the characters in the sequence correspond to a sequence of characters defined by the pattern.

Matching shall be based on the bit pattern used for encoding the character, not on the graphic representation of the character. This means that if a character set contains two or more encodings for a graphic symbol, or if the strings searched contain text encoded in more than one codeset, no attempt is made to search for any other representation of the encoded symbol. If that is required, the user can specify equivalence classes containing all variations of the desired graphic symbol.

The search for a matching sequence starts at the beginning of a string and stops when the first sequence matching the expression is found, where *first* is defined to mean ‘‘begins earliest in the string’’. If the pattern permits a variable number of matching characters and thus there is more than one such sequence starting at that point, the longest such sequence is matched. For example: the BRE `"bb*"` matches the second to fourth characters of *abbbc*, and the ERE (*wee* | *week*)(*knights* | *night*) matches all ten characters of *weeknights*.

Consistent with the whole match being the longest of the leftmost matches, each subpattern, from left to right, shall match the longest possible string. For this purpose, a null string shall be considered to be longer than no match at all. For example, matching the BRE `"\(.*\).*"` against `"abcdef"`, the subexpression `"(\1)"` is `"abcdef"`, and matching the BRE `"\(a*\)*"` against `"bc"`, the subexpression `"(\1)"` is the null string.

When a multi-character collating element in a bracket expression (see Section 9.3.5 (on page 168)) is involved, the longest sequence shall be measured in characters consumed from the string to be matched; that is, the collating element counts not as one element, but as the number of characters it matches.

BRE (ERE) matching a single character

A BRE or ERE that shall match either a single character or a single collating element.

Only a BRE or ERE of this type that includes a bracket expression (see Section 9.3.5 (on page 168)) can match a collating element.

BRE (ERE) matching multiple characters

A BRE or ERE that shall match a concatenation of single characters or collating elements.

Such a BRE or ERE is made up from a BRE (ERE) matching a single character and BRE (ERE) special characters.

invalid

This section uses the term *invalid* for certain constructs or conditions. Invalid REs shall cause the utility or function using the RE to generate an error condition. When *invalid* is not used, violations of the specified syntax or semantics for REs produce undefined results: this may entail an error, enabling an extended syntax for that RE, or using the construct in error as literal characters to be matched. For example, the BRE construct `"\{1,2,3\}"` does not comply with the grammar. A conforming application cannot rely on it producing an error nor matching the literal characters `"\{1,2,3\}"`.

9.2 Regular Expression General Requirements

The requirements in this section shall apply to both basic and extended regular expressions.

The use of regular expressions is generally associated with text processing. REs (BREs and EREs) operate on text strings; that is, zero or more characters followed by an end-of-string delimiter (typically NUL). Some utilities employing regular expressions limit the processing to lines; that is, zero or more characters followed by a <newline>. In the regular expression processing described in IEEE Std 1003.1-200x, the <newline> is regarded as an ordinary character and both a period and a non-matching list can match one. The Shell and Utilities volume of IEEE Std 1003.1-200x specifies within the individual descriptions of those standard utilities employing regular expressions whether they permit matching of <newline>s; if not stated otherwise, the use of literal <newline>s or any escape sequence equivalent produces undefined results. Those utilities (like *grep*) that do not allow <newline>s to match are responsible for eliminating any <newline> from strings before matching against the RE. The *regcomp*( ) function in the System Interfaces volume of IEEE Std 1003.1-200x, however, can provide support for such processing without violating the rules of this section.

The interfaces specified in IEEE Std 1003.1-200x do not permit the inclusion of a NUL character in an RE or in the string to be matched. If during the operation of a standard utility a NUL is included in the text designated to be matched, that NUL may designate the end of the text string for the purposes of matching.

When a standard utility or function that uses regular expressions specifies that pattern matching shall be performed without regard to the case (uppercase or lowercase) of either data or patterns, then when each character in the string is matched against the pattern, not only the character, but also its case counterpart (if any), shall be matched. This definition of case-insensitive processing is intended to allow matching of multi-character collating elements as well as characters, as each character in the string is matched using both its cases. For example, in a locale where `"Ch"` is a multi-character collating element and where a matching list expression matches such elements, the RE `"[[.Ch.]]"` when matched against the string `"char"`, is in reality matched against "ch", "Ch", "cH", and "CH".

The implementation shall support any regular expression that does not exceed 256 bytes in length.

9.3 Basic Regular Expressions

9.3.1 BREs Matching a Single Character or Collating Element

A BRE ordinary character, a special character preceded by a backslash or a period, shall match a single character. A bracket expression shall match a single character or a single collating element.

9.3.2 BRE Ordinary Characters

An ordinary character is a BRE that matches itself: any character in the supported character set, except for the BRE special characters listed in Section 9.3.3.

The interpretation of an ordinary character preceded by a backslash ('\') is undefined, except for:

- The characters ')', '(', '{', and '}'
- The digits 1 to 9 inclusive (see Section 9.3.6 (on page 170))
- A character inside a bracket expression

9.3.3 BRE Special Characters

A *BRE special character* has special properties in certain contexts. Outside those contexts, or when preceded by a backslash, such a character is a BRE that matches the special character itself. The BRE special characters and the contexts in which they have their special meaning are as follows:

. [\ The period, left-bracket, and backslash shall be special except when used in a bracket expression (see Section 9.3.5 (on page 168)). An expression containing a '[' that is not preceded by a backslash and is not part of a bracket expression produces undefined results.

* The asterisk shall be special except when used:

- In a bracket expression
- As the first character of an entire BRE (after an initial '^', if any)
- As the first character of a subexpression (after an initial '^', if any); see Section 9.3.6 (on page 170)

^ The circumflex shall be special when used as:

- An anchor (see Section 9.3.8 (on page 171))
- The first character of a bracket expression (see Section 9.3.5 (on page 168))

$ The dollar sign shall be special when used as an anchor.

9.3.4 Periods in BREs

A period ('.'), when used outside a bracket expression, is a BRE that shall match any character in the supported character set except NUL.

9.3.5 RE Bracket Expression

A bracket expression (an expression enclosed in square brackets, "[ ]") is an RE that shall match a single collating element contained in the non-empty set of collating elements represented by the bracket expression.

The following rules and definitions apply to bracket expressions:

1. A *bracket expression* is either a matching list expression or a non-matching list expression. It consists of one or more expressions: collating elements, collating symbols, equivalence classes, character classes, or range expressions. The right-bracket (']') shall lose its special meaning and represents itself in a bracket expression if it occurs first in the list (after an initial circumflex ('^'), if any). Otherwise, it shall terminate the bracket expression, unless it appears in a collating symbol (such as "[.].]") or is the ending right-bracket for a collating symbol, equivalence class, or character class. The special characters '.', '*', '[', and '\' (period, asterisk, left-bracket, and backslash, respectively) shall lose their special meaning within a bracket expression.

   The character sequences "[.", "[=", and "[:" (left-bracket followed by a period, equals-sign, or colon) shall be special inside a bracket expression and are used to delimit collating symbols, equivalence class expressions, and character class expressions. These symbols shall be followed by a valid expression and the matching terminating sequence ".]", "=]", or ":]", as described in the following items.

2. A *matching list expression* specifies a list that shall match any single-character collating element in any of the expressions represented in the list. The first character in the list shall not be the circumflex; for example, "[abc]" is an RE that matches any of the characters 'a', 'b', or 'c'. It is unspecified whether a matching list expression matches a multi-character collating element that is matched by one of the expressions.

3. A *non-matching list expression* begins with a circumflex ('^'), and specifies a list that shall match any single-character collating element except for the expressions represented in the list after the leading circumflex. For example, "[^abc]" is an RE that matches any character except the characters 'a', 'b', or 'c'. It is unspecified whether a non-matching list expression matches a multi-character collating element that is not matched by any of the expressions. The circumflex shall have this special meaning only when it occurs first in the list, immediately following the left-bracket.

4. A *collating symbol* is a collating element enclosed within bracket-period ("[." and ".]") delimiters. Collating elements are defined as described in Section 7.3.2.4 (on page 133). Conforming applications shall represent multi-character collating elements as collating symbols when it is necessary to distinguish them from a list of the individual characters that make up the multi-character collating element. For example, if the string "ch" is a collating element defined using the line:

   ```
   collating-element <ch-digraph> from "<c><h>"
   ```

   in the locale definition, the expression "[[.ch.]]" shall be treated as an RE containing the collating symbol 'ch', while "[ch]" shall be treated as an RE matching 'c' or 'h'. Collating symbols are recognized only inside bracket expressions. If the string is not a collating element in the current locale, the expression is invalid.

5. An *equivalence class expression* shall represent the set of collating elements belonging to an equivalence class, as described in Section 7.3.2.4 (on page 133). Only primary equivalence classes shall be recognized. The class shall be expressed by enclosing any one of the collating elements in the equivalence class within bracket-equal (`"[="` and `"=]"`) delimiters. For example, if `'a'`, `'à'`, and `'â'` belong to the same equivalence class, then `"[[=a=]b]"`, `"[[=à=]b]"`, and `"[[=â=]b]"` are each equivalent to `"[aàâb]"`. If the collating element does not belong to an equivalence class, the equivalence class expression shall be treated as a *collating symbol*.

6. A *character class expression* shall represent the union of two sets:

   a. The set of single-character collating elements whose characters belong to the character class, as defined in the *LC_CTYPE* category in the current locale.

   b. An unspecified set of multi-character collating elements.

   All character classes specified in the current locale shall be recognized. A character class expression is expressed as a character class name enclosed within bracket-colon (`"[:"` and `":]"`) delimiters.

   The following character class expressions shall be supported in all locales:

   ```
   [:alnum:]    [:cntrl:]    [:lower:]    [:space:]
   [:alpha:]    [:digit:]    [:print:]    [:upper:]
   [:blank:]    [:graph:]    [:punct:]    [:xdigit:]
   ```

   XSI In addition, character class expressions of the form:

   `[:`*name*`:]`

   are recognized in those locales where the *name* keyword has been given a **charclass** definition in the *LC_CTYPE* category.

7. In the POSIX locale, a range expression represents the set of collating elements that fall between two elements in the collation sequence, inclusive. In other locales, a range expression has unspecified behavior: strictly conforming applications shall not rely on whether the range expression is valid, or on the set of collating elements matched. A range expression shall be expressed as the starting point and the ending point separated by a hyphen (`'-'`).

   In the following, all examples assume the POSIX locale.

   The starting range point and the ending range point shall be a collating element or collating symbol. An equivalence class expression used as a starting or ending point of a range expression produces unspecified results. An equivalence class can be used portably within a bracket expression, but only outside the range. If the represented set of collating elements is empty, it is unspecified whether the expression matches nothing, or is treated as invalid.

   The interpretation of range expressions where the ending range point is also the starting range point of a subsequent range expression (for example, `"[a-m-o]"`) is undefined.

   The hyphen character shall be treated as itself if it occurs first (after an initial `'^'`, if any) or last in the list, or as an ending range point in a range expression. As examples, the expressions `"[-ac]"` and `"[ac-]"` are equivalent and match any of the characters `'a'`, `'c'`, or `'-'`; `"[^-ac]"` and `"[^ac-]"` are equivalent and match any characters except `'a'`, `'c'`, or `'-'`; the expression `"[%--]"` matches any of the characters between `'%'` and `'-'` inclusive; the expression `"[--@]"` matches any of the characters between `'-'` and `'@'` inclusive; and the expression `"[a--@]"` is either invalid or equivalent to `'@'`, because the letter 'a' follows the symbol '-' in the POSIX locale. To use a hyphen as the starting range point, it shall either come first in the bracket expression or be specified as a collating symbol; for example, "[][.-.]-0]", which matches either a right bracket or any character or collating element that collates between hyphen and 0, inclusive.

If a bracket expression specifies both '-' and ']', the ']' shall be placed first (after the '^', if any) and the '-' last within the bracket expression.

9.3.6 BREs Matching Multiple Characters

The following rules can be used to construct BREs matching multiple characters from BREs matching a single character:

1. The concatenation of BREs shall match the concatenation of the strings matched by each component of the BRE.

2. A *subexpression* can be defined within a BRE by enclosing it between the character pairs "\(" and "\)". Such a subexpression shall match whatever it would have matched without the "\(" and "\)", except that anchoring within subexpressions is optional behavior; see Section 9.3.8 (on page 171). Subexpressions can be arbitrarily nested.

3. The *back-reference 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 "\(" from the beginning of the pattern and ends with the corresponding paired "\)"). 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 back-referenced expression shall refer to the last matched string. If the subexpression referenced by the back-reference matches more than one string because of an asterisk ('*') or an interval expression (see item (5)), the back-reference shall match the last (rightmost) of these strings.

4. When a BRE matching a single character, a subexpression, or a back-reference is followed by the special character asterisk ('*'), together with that asterisk it shall match what zero or more consecutive occurrences of the BRE would match. For example, "[ab]*" and "[ab][ab]" are equivalent when matching the string "ab".

5. When a BRE matching a single character, a subexpression, or a back-reference is followed by an *interval expression* of the format "\{m\}", "\{m,\}", or "\{m,n\}", together with that interval expression it shall match what repeated consecutive occurrences of the BRE would match. The values of *m* and *n* are decimal integers in the range $0 \leq m \leq n \leq${RE_DUP_MAX}, where *m* specifies the exact or minimum number of occurrences and *n* specifies the maximum number of occurrences. The expression "\{m\}" shall match exactly *m* occurrences of the preceding BRE, "\{m,\}" shall match at least *m* occurrences, and "\{m,n\}" shall match any number of occurrences between *m* and *n*, inclusive.

   For example, in the string "abababccccccd" the BRE "c\{3\}" is matched by characters '7' to '9', the BRE "\(ab\)\{4,\}" is not matched at all, and the BRE "c\{1,3\}d" is matched by characters ten to thirteen.

The behavior of multiple adjacent duplication symbols ('*' and intervals) produces undefined results.

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.

9.3.7 BRE Precedence

The order of precedence shall be as shown in the following table:

BRE Precedence (from high to low)	
Collation-related bracket symbols	`[==] [::] [..]`
Escaped characters	`\<special character>`
Bracket expression	`[]`
Subexpressions / back-references	`\(\) \n`
Single-character-BRE duplication	`* \{m,n\}`
Concatenation	
Anchoring	`^ $`

9.3.8 BRE Expression Anchoring

A BRE can be limited to matching strings that begin or end a line; this is called *anchoring*. The circumflex and dollar sign special characters shall be considered BRE anchors in the following contexts:

1. A circumflex (`'^'`) shall be an anchor when used as the first character of an entire BRE. The implementation may treat the circumflex as an anchor when used as the first character of a subexpression. The circumflex shall anchor the expression (or optionally subexpression) to the beginning of a string; only sequences starting at the first character of a string shall be matched by the BRE. For example, the BRE `"^ab"` matches `"ab"` in the string `"abcdef"`, but fails to match in the string `"cdefab"`. The BRE `"\(^ab\)"` may match the former string. A portable BRE shall escape a leading circumflex in a subexpression to match a literal circumflex.

2. A dollar sign (`'$'`) shall be an anchor when used as the last character of an entire BRE. The implementation may treat a dollar sign as an anchor when used as the last character of a subexpression. The dollar sign shall anchor the expression (or optionally subexpression) to the end of the string being matched; the dollar sign can be said to match the end-of-string following the last character.

3. A BRE anchored by both `'^'` and `'$'` shall match only an entire string. For example, the BRE `"^abcdef$"` matches strings consisting only of `"abcdef"`.

9.4 Extended Regular Expressions

The *extended regular expression* (ERE) notation and construction rules shall apply to utilities defined as using extended regular expressions; any exceptions to the following rules are noted in the descriptions of the specific utilities using EREs.

9.4.1 EREs Matching a Single Character or Collating Element

An ERE ordinary character, a special character preceded by a backslash, or a period shall match a single character. A bracket expression shall match a single character or a single collating element. An *ERE matching a single character* enclosed in parentheses shall match the same as the ERE without parentheses would have matched.

9.4.2 ERE Ordinary Characters

An *ordinary character* is an ERE that matches itself. An ordinary character is any character in the supported character set, except for the ERE special characters listed in Section 9.4.3. The interpretation of an ordinary character preceded by a backslash (`'\'`) is undefined.

9.4.3 ERE Special Characters

An *ERE special character* has special properties in certain contexts. Outside those contexts, or when preceded by a backslash, such a character shall be an ERE that matches the special character itself. The extended regular expression special characters and the contexts in which they shall have their special meaning are as follows:

`. [ \ (` The period, left-bracket, backslash, and left-parenthesis shall be special except when used in a bracket expression (see Section 9.3.5 (on page 168)). Outside a bracket expression, a left-parenthesis immediately followed by a right-parenthesis produces undefined results.

`)` The right-parenthesis shall be special when matched with a preceding left-parenthesis, both outside a bracket expression.

`* + ? {` The asterisk, plus-sign, question-mark, and left-brace shall be special except when used in a bracket expression (see Section 9.3.5 (on page 168)). Any of the following uses produce undefined results:

- If these characters appear first in an ERE, or immediately following a vertical-line, circumflex, or left-parenthesis
- If a left-brace is not part of a valid interval expression (see Section 9.4.6 (on page 173))

`|` The vertical-line is special except when used in a bracket expression (see Section 9.3.5 (on page 168)). A vertical-line appearing first or last in an ERE, or immediately following a vertical-line or a left-parenthesis, or immediately preceding a right-parenthesis, produces undefined results.

`^` The circumflex shall be special when used as:

- An anchor (see Section 9.4.9 (on page 174))
- The first character of a bracket expression (see Section 9.3.5 (on page 168))

`$` The dollar sign shall be special when used as an anchor.

9.4.4 Periods in EREs

A period ('.'), when used outside a bracket expression, is an ERE that shall match any character in the supported character set except NUL.

9.4.5 ERE Bracket Expression

The rules for ERE Bracket Expressions are the same as for Basic Regular Expressions; see Section 9.3.5 (on page 168).

9.4.6 EREs Matching Multiple Characters

The following rules shall be used to construct EREs matching multiple characters from EREs matching a single character:

1. A *concatenation of EREs* shall match the concatenation of the character sequences matched by each component of the ERE. A concatenation of EREs enclosed in parentheses shall match whatever the concatenation without the parentheses matches. For example, both the ERE "cd" and the ERE "(cd)" are matched by the third and fourth character of the string "abcdefabcdef".

2. When an ERE matching a single character or an ERE enclosed in parentheses is followed by the special character plus-sign ('+'), together with that plus-sign it shall match what one or more consecutive occurrences of the ERE would match. For example, the ERE "b+(bc)" matches the fourth to seventh characters in the string "acabbbcde". And, "[ab]+" and "[ab][ab]*" are equivalent.

3. When an ERE matching a single character or an ERE enclosed in parentheses is followed by the special character asterisk ('*'), together with that asterisk it shall match what zero or more consecutive occurrences of the ERE would match. For example, the ERE "b*c" matches the first character in the string "cabbbcde", and the ERE "b*cd" matches the third to seventh characters in the string "cabbbcdebbbbbbcdbc". And, "[ab]*" and [ab][ab] are equivalent when matching the string "ab".

4. When an ERE matching a single character or an ERE enclosed in parentheses is followed by the special character question-mark ('?'), together with that question-mark it shall match what zero or one consecutive occurrences of the ERE would match. For example, the ERE "b?c" matches the second character in the string "acabbbcde".

5. When an ERE matching a single character or an ERE enclosed in parentheses is followed by an *interval expression* of the format "{m}", "{m,}", or "{m,n}", together with that interval expression it shall match what repeated consecutive occurrences of the ERE would match. The values of *m* and *n* are decimal integers in the range $0 \le m \le n \le$ {RE_DUP_MAX}, where *m* specifies the exact or minimum number of occurrences and *n* specifies the maximum number of occurrences. The expression "{m}" matches exactly *m* occurrences of the preceding ERE, "{m,}" matches at least *m* occurrences, and "{m,n}" matches any number of occurrences between *m* and *n*, inclusive.

   For example, in the string "abababccccccd" the ERE "c{3}" is matched by characters '7' to '9' and the ERE "(ab){2,}" is matched by characters one to six.

The behavior of multiple adjacent duplication symbols ('+', '*', '?', and intervals) produces undefined results.

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.

9.4.7 ERE Alternation

Two EREs separated by the special character vertical-line (`'|'`) shall match a string that is matched by either. For example, the ERE `"a((bc)|d)"` matches the string `"abc"` and the string `"ad"`. Single characters, or expressions matching single characters, separated by the vertical bar and enclosed in parentheses, shall be treated as an ERE matching a single character.

9.4.8 ERE Precedence

The order of precedence shall be as shown in the following table:

ERE Precedence (from high to low)	
Collation-related bracket symbols	`[==] [::] [..]`
Escaped characters	`\<special character>`
Bracket expression	`[]`
Grouping	`()`
Single-character-ERE duplication	`* + ? {m,n}`
Concatenation	
Anchoring	`^ $`
Alternation	`|`

For example, the ERE `"abba|cde"` matches either the string `"abba"` or the string `"cde"` (rather than the string `"abbade"` or `"abbcde"`, because concatenation has a higher order of precedence than alternation).

9.4.9 ERE Expression Anchoring

An ERE can be limited to matching strings that begin or end a line; this is called *anchoring*. The circumflex and dollar sign special characters shall be considered ERE anchors when used anywhere outside a bracket expression. This shall have the following effects:

1. A circumflex (`'ˆ'`) outside a bracket expression shall anchor the expression or subexpression it begins to the beginning of a string; such an expression or subexpression can match only a sequence starting at the first character of a string. For example, the EREs `"ˆab"` and `"(ˆab)"` match `"ab"` in the string `"abcdef"`, but fail to match in the string `"cdefab"`, and the ERE `"aˆb"` is valid, but can never match because the `'a'` prevents the expression `"ˆb"` from matching starting at the first character.

2. A dollar sign (`'$'`) outside a bracket expression shall anchor the expression or subexpression it ends to the end of a string; such an expression or subexpression can match only a sequence ending at the last character of a string. For example, the EREs `"ef$"` and `"(ef$)"` match `"ef"` in the string `"abcdef"`, but fail to match in the string `"cdefab"`, and the ERE `"e$f"` is valid, but can never match because the `'f'` prevents the expression `"e$"` from matching ending at the last character.

9.5 Regular Expression Grammar

Grammars describing the syntax of both basic and extended regular expressions are presented in this section. The grammar takes precedence over the text. See the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 1.10, Grammar Conventions.

9.5.1 BRE/ERE Grammar Lexical Conventions

The lexical conventions for regular expressions are as described in this section.

Except as noted, the longest possible token or delimiter beginning at a given point is recognized.

The following tokens are processed (in addition to those string constants shown in the grammar):

COLL_ELEM_SINGLE
Any single-character collating element, unless it is a META_CHAR.

COLL_ELEM_MULTI Any multi-character collating element.

BACKREF Applicable only to basic regular expressions. The character string consisting of `'\'` followed by a single-digit numeral, `'1'` to `'9'`.

DUP_COUNT Represents a numeric constant. It shall be an integer in the range 0 ≤DUP_COUNT ≤{RE_DUP_MAX}. This token is only recognized when the context of the grammar requires it. At all other times, digits not preceded by `'\'` are treated as ORD_CHAR.

META_CHAR One of the characters:

^ When found first in a bracket expression

– When found anywhere but first (after an initial `'^'`, if any) or last in a bracket expression, or as the ending range point in a range expression

] When found anywhere but first (after an initial `'^'`, if any) in a bracket expression

L_ANCHOR Applicable only to basic regular expressions. The character `'^'` when it appears as the first character of a basic regular expression and when not QUOTED_CHAR. The `'^'` may be recognized as an anchor elsewhere; see Section 9.3.8 (on page 171).

ORD_CHAR A character, other than one of the special characters in SPEC_CHAR.

QUOTED_CHAR In a BRE, one of the character sequences:

```
\^    \.    \*    \[    \$    \\
```

In an ERE, one of the character sequences:

```
\^    \.    \[    \$    \(    \)    \|
\*    \+    \?    \{    \\
```

R_ANCHOR (Applicable only to basic regular expressions.) The character `'$'` when it appears as the last character of a basic regular expression and when not QUOTED_CHAR. The `'$'` may be recognized as an anchor elsewhere; see Section 9.3.8 (on page 171).

SPEC_CHAR For basic regular expressions, one of the following special characters:

`.`	Anywhere outside bracket expressions
`\`	Anywhere outside bracket expressions
`[`	Anywhere outside bracket expressions
`^`	When used as an anchor (see Section 9.3.8 (on page 171)) or when first in a bracket expression
`$`	When used as an anchor
`*`	Anywhere except first in an entire RE, anywhere in a bracket expression, directly following `"\("`, directly following an anchoring `'^'`

For extended regular expressions, shall be one of the following special characters found anywhere outside bracket expressions:

```
^    .    [    $    (    )    |
*    +    ?    {    \
```

(The close-parenthesis shall be considered special in this context only if matched with a preceding open-parenthesis.)

9.5.2 RE and Bracket Expression Grammar

This section presents the grammar for basic regular expressions, including the bracket expression grammar that is common to both BREs and EREs.

```
%token    ORD_CHAR QUOTED_CHAR DUP_COUNT

%token    BACKREF L_ANCHOR R_ANCHOR

%token    Back_open_paren  Back_close_paren
/*          '\('             '\)'         */

%token    Back_open_brace  Back_close_brace
/*          '\{'             '\}'          */

/* The following tokens are for the Bracket Expression
   grammar common to both REs and EREs. */

%token    COLL_ELEM_SINGLE COLL_ELEM_MULTI META_CHAR

%token    Open_equal Equal_close Open_dot Dot_close Open_colon Colon_close
/*           '[='       '=]'        '[.'     '.]'      '[:'       ':]'  */

%token    class_name
/* class_name is a keyword to the LC_CTYPE locale category */
/* (representing a character class) in the current locale */
/* and is only recognized between [: and :] */

%start    basic_reg_exp
%%

/* --------------------------------------------
   Basic Regular Expression
   --------------------------------------------
*/
basic_reg_exp  :          RE_expression
               | L_ANCHOR
               |                      R_ANCHOR
               | L_ANCHOR               R_ANCHOR
               | L_ANCHOR RE_expression
               |          RE_expression R_ANCHOR
               | L_ANCHOR RE_expression R_ANCHOR
               ;
RE_expression  :               simple_RE
               | RE_expression simple_RE
               ;
simple_RE      : nondupl_RE
               | nondupl_RE RE_dupl_symbol
               ;
nondupl_RE     : one_char_or_coll_elem_RE
               | Back_open_paren RE_expression Back_close_paren
               | BACKREF
               ;
one_char_or_coll_elem_RE : ORD_CHAR
               | QUOTED_CHAR
               | ’.’
               | bracket_expression
               ;
RE_dupl_symbol : ’*’
               | Back_open_brace DUP_COUNT               Back_close_brace
               | Back_open_brace DUP_COUNT ’,’           Back_close_brace
               | Back_open_brace DUP_COUNT ’,’ DUP_COUNT Back_close_brace
               ;

/* --------------------------------------------
   Bracket Expression
   -------------------------------------------
*/
bracket_expression : ’[’ matching_list ’]’
               | ’[’ nonmatching_list ’]’
               ;
matching_list  : bracket_list
               ;
nonmatching_list : ’ˆ’ bracket_list
               ;
bracket_list   : follow_list
               | follow_list ’-’
               ;
follow_list    :             expression_term
               | follow_list expression_term
               ;
expression_term : single_expression
               | range_expression
               ;
single_expression : end_range
               | character_class
               | equivalence_class
               ;
range_expression : start_range end_range
               | start_range ’-’
               ;
```

```
start_range    : end_range '-'
               ;
end_range      : COLL_ELEM_SINGLE
               | collating_symbol
               ;
collating_symbol : Open_dot COLL_ELEM_SINGLE Dot_close
               | Open_dot COLL_ELEM_MULTI Dot_close
               | Open_dot META_CHAR Dot_close
               ;
equivalence_class : Open_equal COLL_ELEM_SINGLE Equal_close
               | Open_equal COLL_ELEM_MULTI Equal_close
               ;
character_class : Open_colon class_name Colon_close
               ;
```

The BRE grammar does not permit L_ANCHOR or R_ANCHOR inside "\(" and "\)" (which implies that '^' and '$' are ordinary characters). This reflects the semantic limits on the application, as noted in Section 9.3.8 (on page 171). Implementations are permitted to extend the language to interpret '^' and '$' as anchors in these locations, and as such, conforming applications cannot use unescaped '^' and '$' in positions inside "\(" and "\)" that might be interpreted as anchors.

9.5.3 ERE Grammar

This section presents the grammar for extended regular expressions, excluding the bracket expression grammar.

Note: The bracket expression grammar and the associated **%token** lines are identical between BREs and EREs. It has been omitted from the ERE section to avoid unnecessary editorial duplication.

```
%token  ORD_CHAR QUOTED_CHAR DUP_COUNT
%start  extended_reg_exp
%%

/* --------------------------------------------
   Extended Regular Expression
   --------------------------------------------
*/
extended_reg_exp   :                          ERE_branch
                   | extended_reg_exp '|' ERE_branch
                   ;
ERE_branch         :           ERE_expression
                   | ERE_branch ERE_expression
                   ;
ERE_expression     : one_char_or_coll_elem_ERE
                   | '^'
                   | '$'
                   | '(' extended_reg_exp ')'
                   | ERE_expression ERE_dupl_symbol
                   ;
one_char_or_coll_elem_ERE  : ORD_CHAR
                   | QUOTED_CHAR
                   | '.'
                   | bracket_expression
                   ;
```

```
ERE_dupl_symbol    : '*'
                   | '+'
                   | '?'
                   | '{' DUP_COUNT               '}'
                   | '{' DUP_COUNT ','           '}'
                   | '{' DUP_COUNT ',' DUP_COUNT '}'
                   ;
```

The ERE grammar does not permit several constructs that previous sections specify as having undefined results:

- ORD_CHAR preceded by '\'
- One or more *ERE_dupl_symbol*s appearing first in an ERE, or immediately following '|', '^', or '('
- '{' not part of a valid *ERE_dupl_symbol*
- '|' appearing first or last in an ERE, or immediately following '|' or '(', or immediately preceding ')'

Implementations are permitted to extend the language to allow these. Conforming applications cannot use such constructs.

Chapter 10

Directory Structure and Devices

10.1 Directory Structure and Files

The following directories shall exist on conforming systems and conforming applications shall make use of them only as described. Strictly conforming applications shall not assume the ability to create files in any of these directories, unless specified below.

/ The root directory.

/dev Contains **/dev/console**, **/dev/null**, and **/dev/tty**, described below.

The following directory shall exist on conforming systems and shall be used as described.

/tmp A directory made available for programs that need a place to create temporary files. Applications shall be allowed to create files in this directory, but shall not assume that such files are preserved between invocations of the application.

The following files shall exist on conforming systems and shall be both readable and writable.

/dev/null An infinite data source and data sink. Data written to **/dev/null** shall be discarded. Reads from **/dev/null** shall always return end-of-file (EOF).

/dev/tty In each process, a synonym for the controlling terminal associated with the process group of that process, if any. It is useful for programs or shell procedures that wish to be sure of writing messages to or reading data from the terminal no matter how output has been redirected. It can also be used for programs that demand the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

The following file shall exist on conforming systems and need not be readable or writable:

/dev/console The **/dev/console** file is a generic name given to the system console (see Section 3.382 (on page 85)). It is usually linked to an implementation-defined special file. It shall provide an interface to the system console conforming to the requirements of the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

10.2 Output Devices and Terminal Types

The utilities in the Shell and Utilities volume of IEEE Std 1003.1-200x historically have been implemented on a wide range of terminal types, but a conforming implementation need not support all features of all utilities on every conceivable terminal. IEEE Std 1003.1-200x states which features are optional for certain classes of terminals in the individual utility description sections. The implementation shall document which terminal types it supports and which of these features and utilities are not supported by each terminal.

When a feature or utility is not supported on a specific terminal type, as allowed by IEEE Std 1003.1-200x, and the implementation considers such a condition to be an error preventing use of the feature or utility, the implementation shall indicate such conditions through diagnostic messages or exit status values or both (as appropriate to the specific utility description) that inform the user that the terminal type lacks the appropriate capability.

IEEE Std 1003.1-200x uses a notational convention based on historical practice that identifies some of the control characters defined in Section 7.3.1 (on page 122) in a manner easily remembered by users on many terminals. The correspondence between this ‘‘<control>-*char*’’ notation and the actual control characters is shown in the following table. When IEEE Std 1003.1-200x refers to a character by its <control>- *name*, it is referring to the actual control character shown in the Value column of the table, which is not necessarily the exact control key sequence on all terminals. Some terminals have keyboards that do not allow the direct transmission of all the non-alphanumeric characters shown. In such cases, the system documentation shall describe which data sequences transmitted by the terminal are interpreted by the system as representing the special characters.

Table 10-1 Control Character Names

Name	Value	Symbolic Name	Name	Value	Symbolic Name
<control>-A	<SOH>	<SOH>	<control>-Q	<DC1>	<DC1>
<control>-B	<STX>	<STX>	<control>-R	<DC2>	<DC2>
<control>-C	<ETX>	<ETX>	<control>-S	<DC3>	<DC3>
<control>-D	<EOT>	<EOT>	<control>-T	<DC4>	<DC4>
<control>-E	<ENQ>	<ENQ>	<control>-U	<NAK>	<NAK>
<control>-F	<ACK>	<ACK>	<control>-V	<SYN>	<SYN>
<control>-G	<BEL>	<alert>	<control>-W	<ETB>	<ETB>
<control>-H	<BS>	<backspace>	<control>-X	<CAN>	<CAN>
<control>-I	<HT>	<tab>	<control>-Y	<EM>	<EM>
<control>-J	<LF>	<linefeed>	<control>-Z	<SUB>	<SUB>
<control>-K	<VT>	<vertical-tab>	<control>-[	<ESC>	<ESC>
<control>-L	<FF>	<form-feed>	<control>-\	<FS>	<FS>
<control>-M	<CR>	<carriage-return>	<control>-]	<GS>	<GS>
<control>-N	<SO>	<SO>	<control>-ˆ	<RS>	<RS>
<control>-O	<SI>	<SI>	<control>-_	<US>	<US>
<control>-P	<DLE>	<DLE>	<control>-?	<DEL>	<DEL>

Note: The notation uses uppercase letters for arbitrary editorial reasons. There is no implication that the keystrokes represent control-shift-letter sequences.

Chapter 11

General Terminal Interface

This chapter describes a general terminal interface that shall be provided. It shall be supported on any asynchronous communications ports if the implementation provides them. It is implementation-defined whether it supports network connections or synchronous ports, or both.

11.1 Interface Characteristics

11.1.1 Opening a Terminal Device File

When a terminal device file is opened, it normally causes the thread to wait until a connection is established. In practice, application programs seldom open these files; they are opened by special programs and become an application's standard input, output, and error files.

As described in *open*( ), opening a terminal device file with the O_NONBLOCK flag clear shall cause the thread to block until the terminal device is ready and available. If CLOCAL mode is not set, this means blocking until a connection is established. If CLOCAL mode is set in the terminal, or the O_NONBLOCK flag is specified in the *open*( ), the *open*( ) function shall return a file descriptor without waiting for a connection to be established.

11.1.2 Process Groups

A terminal may have a foreground process group associated with it. This foreground process group plays a special role in handling signal-generating input characters, as discussed in Section 11.1.9 (on page 187).

A command interpreter process supporting job control can allocate the terminal to different jobs, or process groups, by placing related processes in a single process group and associating this process group with the terminal. A terminal's foreground process group may be set or examined by a process, assuming the permission requirements are met; see *tcgetpgrp*( ) and *tcsetpgrp*( ). The terminal interface aids in this allocation by restricting access to the terminal by processes that are not in the current process group; see Section 11.1.4 (on page 184).

When there is no longer any process whose process ID or process group ID matches the process group ID of the foreground process group, the terminal shall have no foreground process group. It is unspecified whether the terminal has a foreground process group when there is a process whose process ID matches the foreground process ID, but whose process group ID does not. No actions defined in IEEE Std 1003.1-200x, other than allocation of a controlling terminal or a successful call to *tcsetpgrp*( ), cause a process group to become the foreground process group of the terminal.

11.1.3 The Controlling Terminal

A terminal may belong to a process as its controlling terminal. Each process of a session that has a controlling terminal has the same controlling terminal. A terminal may be the controlling terminal for at most one session. The controlling terminal for a session is allocated by the session leader in an implementation-defined manner. If a session leader has no controlling terminal, and opens a terminal device file that is not already associated with a session without using the O_NOCTTY option (see *open*( )), it is implementation-defined whether the terminal becomes the controlling terminal of the session leader. If a process which is not a session leader opens a terminal file, or the O_NOCTTY option is used on *open*( ), then that terminal shall not become the controlling terminal of the calling process. When a controlling terminal becomes associated with a session, its foreground process group shall be set to the process group of the session leader.

The controlling terminal is inherited by a child process during a *fork*( ) function call. A process relinquishes its controlling terminal when it creates a new session with the *setsid*( ) function; other processes remaining in the old session that had this terminal as their controlling terminal continue to have it. Upon the close of the last file descriptor in the system (whether or not it is in the current session) associated with the controlling terminal, it is unspecified whether all processes that had that terminal as their controlling terminal cease to have any controlling terminal. Whether and how a session leader can reacquire a controlling terminal after the controlling terminal has been relinquished in this fashion is unspecified. A process does not relinquish its controlling terminal simply by closing all of its file descriptors associated with the controlling terminal if other processes continue to have it open.

When a controlling process terminates, the controlling terminal is dissociated from the current session, allowing it to be acquired by a new session leader. Subsequent access to the terminal by other processes in the earlier session may be denied, with attempts to access the terminal treated as if a modem disconnect had been sensed.

11.1.4 Terminal Access Control

If a process is in the foreground process group of its controlling terminal, read operations shall be allowed, as described in Section 11.1.5 (on page 185). Any attempts by a process in a background process group to read from its controlling terminal cause its process group to be sent a SIGTTIN signal unless one of the following special cases applies: if the reading process is ignoring or blocking the SIGTTIN signal, or if the process group of the reading process is orphaned, the *read*( ) shall return −1, with *errno* set to [EIO] and no signal shall be sent. The default action of the SIGTTIN signal shall be to stop the process to which it is sent. See **<signal.h>**.

If a process is in the foreground process group of its controlling terminal, write operations shall be allowed as described in Section 11.1.8 (on page 187). Attempts by a process in a background process group to write to its controlling terminal shall cause the process group to be sent a SIGTTOU signal unless one of the following special cases applies: if TOSTOP is not set, or if TOSTOP is set and the process is ignoring or blocking the SIGTTOU signal, the process is allowed to write to the terminal and the SIGTTOU signal is not sent. If TOSTOP is set, and the process group of the writing process is orphaned, and the writing process is not ignoring or blocking the SIGTTOU signal, the *write*( ) shall return −1, with *errno* set to [EIO] and no signal shall be sent.

Certain calls that set terminal parameters are treated in the same fashion as *write*( ), except that TOSTOP is ignored; that is, the effect is identical to that of terminal writes when TOSTOP is set (see Section 11.2.5 (on page 193), *tcdrain*( ), *tcflow*( ), *tcflush*( ), *tcsendbreak*( ), *tcsetattr*( ), and *tcsetpgrp*( )).

11.1.5 Input Processing and Reading Data

A terminal device associated with a terminal device file may operate in full-duplex mode, so that data may arrive even while output is occurring. Each terminal device file has an *input queue*, associated with it, into which incoming data is stored by the system before being read by a process. The system may impose a limit, {MAX_INPUT}, on the number of bytes that may be stored in the input queue. The behavior of the system when this limit is exceeded is implementation-defined.

Two general kinds of input processing are available, determined by whether the terminal device file is in canonical mode or non-canonical mode. These modes are described in Section 11.1.6 and Section 11.1.7 (on page 186). Additionally, input characters are processed according to the **c_iflag** (see Section 11.2.2 (on page 189)) and **c_lflag** (see Section 11.2.5 (on page 193)) fields. Such processing can include *echoing*, which in general means transmitting input characters immediately back to the terminal when they are received from the terminal. This is useful for terminals that can operate in full-duplex mode.

The manner in which data is provided to a process reading from a terminal device file is dependent on whether the terminal file is in canonical or non-canonical mode, and on whether or not the O_NONBLOCK flag is set by *open*( ) or *fcntl*( ).

If the O_NONBLOCK flag is clear, then the read request shall be blocked until data is available or a signal has been received. If the O_NONBLOCK flag is set, then the read request shall be completed, without blocking, in one of three ways:

1. If there is enough data available to satisfy the entire request, the *read*( ) shall complete successfully and shall return the number of bytes read.

2. If there is not enough data available to satisfy the entire request, the *read*( ) shall complete successfully, having read as much data as possible, and shall return the number of bytes it was able to read.

3. If there is no data available, the *read*( ) shall return –1, with *errno* set to [EAGAIN].

When data is available depends on whether the input processing mode is canonical or non-canonical. The following sections, Section 11.1.6 and Section 11.1.7 (on page 186), describe each of these input processing modes.

11.1.6 Canonical Mode Input Processing

In canonical mode input processing, terminal input is processed in units of lines. A line is delimited by a newline character (NL), an end-of-file character (EOF), or an end-of-line (EOL) character. See Section 11.1.9 (on page 187) for more information on EOF and EOL. This means that a read request shall not return until an entire line has been typed or a signal has been received. Also, no matter how many bytes are requested in the *read*( ) call, at most one line shall be returned. It is not, however, necessary to read a whole line at once; any number of bytes, even one, may be requested in a *read*( ) without losing information.

If {MAX_CANON} is defined for this terminal device, it shall be a limit on the number of bytes in a line. The behavior of the system when this limit is exceeded is implementation-defined. If {MAX_CANON} is not defined, there shall be no such limit; see *pathconf*( ).

Erase and kill processing occur when either of two special characters, the ERASE and KILL characters (see Section 11.1.9 (on page 187)), is received. This processing shall affect data in the input queue that has not yet been delimited by a newline (NL), EOF, or EOL character. This undelimited data makes up the current line. The ERASE character shall delete the last character in the current line, if there is one. The KILL character shall delete all data in the current line, if there are any. The ERASE and KILL characters shall have no effect if there is no data in the current line. The ERASE and KILL characters themselves shall not be placed in the input queue.

11.1.7 Non-Canonical Mode Input Processing

In non-canonical mode input processing, input bytes are not assembled into lines, and erase and kill processing shall not occur. The values of the MIN and TIME members of the **c_cc** array are used to determine how to process the bytes received. The IEEE Std 1003.1-200x does not specify whether the setting of O_NONBLOCK takes precedence over MIN or TIME settings. Therefore, if O_NONBLOCK is set, *read*( ) may return immediately, regardless of the setting of MIN or TIME. Also, if no data is available, *read*( ) may either return 0, or return −1 with *errno* set to [EAGAIN].

MIN represents the minimum number of bytes that should be received when the *read*( ) function returns successfully. TIME is a timer of 0.1 second granularity that is used to time out bursty and short-term data transmissions. If MIN is greater than {MAX_INPUT}, the response to the request is undefined. The four possible values for MIN and TIME and their interactions are described below.

Case A: MIN>0, TIME>0

In case A, TIME serves as an inter-byte timer which shall be activated after the first byte is received. Since it is an inter-byte timer, it shall be reset after a byte is received. The interaction between MIN and TIME is as follows. As soon as one byte is received, the inter-byte timer shall be started. If MIN bytes are received before the inter-byte timer expires (remember that the timer is reset upon receipt of each byte), the read shall be satisfied. If the timer expires before MIN bytes are received, the characters received to that point shall be returned to the user. Note that if TIME expires at least one byte shall be returned because the timer would not have been enabled unless a byte was received. In this case (MIN>0, TIME>0) the read shall block until the MIN and TIME mechanisms are activated by the receipt of the first byte, or a signal is received. If data is in the buffer at the time of the *read*( ), the result shall be as if data has been received immediately after the *read*( ).

Case B: MIN>0, TIME=0

In case B, since the value of TIME is zero, the timer plays no role and only MIN is significant. A pending read shall not be satisfied until MIN bytes are received (that is, the pending read shall block until MIN bytes are received), or a signal is received. A program that uses case B to read record-based terminal I/O may block indefinitely in the read operation.

Case C: MIN=0, TIME>0

In case C, since MIN=0, TIME no longer represents an inter-byte timer. It now serves as a read timer that shall be activated as soon as the *read*( ) function is processed. A read shall be satisfied as soon as a single byte is received or the read timer expires. Note that in case C if the timer expires, no bytes shall be returned. If the timer does not expire, the only way the read can be satisfied is if a byte is received. If bytes are not received, the read shall not block indefinitely waiting for a byte; if no byte is received within TIME*0.1 seconds after the read is initiated, the *read*( ) shall return a value of zero, having read no data. If data is in the buffer at the time of the *read*( ), the timer shall be started as if data has been received immediately after the *read*( ).

Case D: MIN=0, TIME=0

The minimum of either the number of bytes requested or the number of bytes currently available shall be returned without waiting for more bytes to be input. If no characters are available, *read*( ) shall return a value of zero, having read no data.

11.1.8 Writing Data and Output Processing

When a process writes one or more bytes to a terminal device file, they are processed according to the **c_oflag** field (see Section 11.2.3 (on page 190)). The implementation may provide a buffering mechanism; as such, when a call to *write*( ) completes, all of the bytes written have been scheduled for transmission to the device, but the transmission has not necessarily completed. See *write*( ) for the effects of O_NONBLOCK on *write*( ).

11.1.9 Special Characters

Certain characters have special functions on input or output or both. These functions are summarized as follows:

INTR Special character on input, which is recognized if the ISIG flag is set. Generates a SIGINT signal which is sent to all processes in the foreground process group for which the terminal is the controlling terminal. If ISIG is set, the INTR character shall be discarded when processed.

QUIT Special character on input, which is recognized if the ISIG flag is set. Generates a SIGQUIT signal which is sent to all processes in the foreground process group for which the terminal is the controlling terminal. If ISIG is set, the QUIT character shall be discarded when processed.

ERASE Special character on input, which is recognized if the ICANON flag is set. Erases the last character in the current line; see Section 11.1.6 (on page 185). It shall not erase beyond the start of a line, as delimited by an NL, EOF, or EOL character. If ICANON is set, the ERASE character shall be discarded when processed.

KILL Special character on input, which is recognized if the ICANON flag is set. Deletes the entire line, as delimited by an NL, EOF, or EOL character. If ICANON is set, the KILL character shall be discarded when processed.

EOF Special character on input, which is recognized if the ICANON flag is set. When received, all the bytes waiting to be read are immediately passed to the process without waiting for a newline, and the EOF is discarded. Thus, if there are no bytes waiting (that is, the EOF occurred at the beginning of a line), a byte count of zero shall be returned from the *read*( ), representing an end-of-file indication. If ICANON is set, the EOF character shall be discarded when processed.

NL Special character on input, which is recognized if the ICANON flag is set. It is the line delimiter newline. It cannot be changed.

EOL Special character on input, which is recognized if the ICANON flag is set. It is an additional line delimiter, like NL.

SUSP If the ISIG flag is set, receipt of the SUSP character shall cause a SIGTSTP signal to be sent to all processes in the foreground process group for which the terminal is the controlling terminal, and the SUSP character shall be discarded when processed.

STOP Special character on both input and output, which is recognized if the IXON (output control) or IXOFF (input control) flag is set. Can be used to suspend output temporarily. It is useful with CRT terminals to prevent output from disappearing before it can be read. If IXON is set, the STOP character shall be discarded when processed.

START Special character on both input and output, which is recognized if the IXON (output control) or IXOFF (input control) flag is set. Can be used to resume output that has been suspended by a STOP character. If IXON is set, the START character shall be discarded when processed.

CR Special character on input, which is recognized if the ICANON flag is set; it is the carriage-return character. When ICANON and ICRNL are set and IGNCR is not set, this character shall be translated into an NL, and shall have the same effect as an NL character.

The NL and CR characters cannot be changed. It is implementation-defined whether the START and STOP characters can be changed. The values for INTR, QUIT, ERASE, KILL, EOF, EOL, and SUSP shall be changeable to suit individual tastes. Special character functions associated with changeable special control characters can be disabled individually.

If two or more special characters have the same value, the function performed when that character is received is undefined.

A special character is recognized not only by its value, but also by its context; for example, an implementation may support multi-byte sequences that have a meaning different from the meaning of the bytes when considered individually. Implementations may also support additional single-byte functions. These implementation-defined multi-byte or single-byte functions shall be recognized only if the IEXTEN flag is set; otherwise, data is received without interpretation, except as required to recognize the special characters defined in this section.

XSI If IEXTEN is set, the ERASE, KILL, and EOF characters can be escaped by a preceding '\' character, in which case no special function shall occur.

11.1.10 Modem Disconnect

If a modem disconnect is detected by the terminal interface for a controlling terminal, and if CLOCAL is not set in the **c_cflag** field for the terminal (see Section 11.2.4 (on page 192)), the SIGHUP signal shall be sent to the controlling process for which the terminal is the controlling terminal. Unless other arrangements have been made, this shall cause the controlling process to terminate (see *exit*( )). Any subsequent read from the terminal device shall return the value of zero, indicating end-of-file; see *read*( ). Thus, processes that read a terminal file and test for end-of-file can terminate appropriately after a disconnect. If the EIO condition as specified in *read*( ) also exists, it is unspecified whether on EOF condition or the [EIO] is returned. Any subsequent *write*( ) to the terminal device shall return −1, with *errno* set to [EIO], until the device is closed.

11.1.11 Closing a Terminal Device File

The last process to close a terminal device file shall cause any output to be sent to the device and any input to be discarded. If HUPCL is set in the control modes and the communications port supports a disconnect function, the terminal device shall perform a disconnect.

11.2 Parameters that Can be Set

11.2.1 The termios Structure

Routines that need to control certain terminal I/O characteristics shall do so by using the **termios** structure as defined in the **<termios.h>** header. The members of this structure include (but are not limited to):

Member Type	Array Size	Member Name	Description
tcflag_t		**c_iflag**	Input modes.
tcflag_t		**c_oflag**	Output modes.
tcflag_t		**c_cflag**	Control modes.
tcflag_t		**c_lflag**	Local modes.
cc_t	NCCS	**c_cc**[ ]	Control characters.

The types **tcflag_t** and **cc_t** are defined in the **<termios.h>** header. They shall be unsigned integer types.

11.2.2 Input Modes

Values of the **c_iflag** field describe the basic terminal input control, and are composed of the bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name symbols in this table are defined in **<termios.h>**: |

Mask Name	Description
BRKINT	Signal interrupt on break.
ICRNL	Map CR to NL on input.
IGNBRK	Ignore break condition.
IGNCR	Ignore CR.
IGNPAR	Ignore characters with parity errors.
INLCR	Map NL to CR on input.
INPCK	Enable input parity check.
ISTRIP	Strip character.
XSI IXANY	Enable any character to restart output.
IXOFF	Enable start/stop input control.
IXON	Enable start/stop output control.
PARMRK	Mark parity errors.

In the context of asynchronous serial data transmission, a break condition shall be defined as a sequence of zero-valued bits that continues for more than the time to send one byte. The entire sequence of zero-valued bits is interpreted as a single break condition, even if it continues for a time equivalent to more than one byte. In contexts other than asynchronous serial data transmission, the definition of a break condition is implementation-defined.

If IGNBRK is set, a break condition detected on input shall be ignored; that is, not put on the input queue and therefore not read by any process. If IGNBRK is not set and BRKINT is set, the break condition shall flush the input and output queues, and if the terminal is the controlling terminal of a foreground process group, the break condition shall generate a single SIGINT signal to that foreground process group. If neither IGNBRK nor BRKINT is set, a break condition shall be read as a single 0x00, or if PARMRK is set, as 0xff 0x00 0x00.

If IGNPAR is set, a byte with a framing or parity error (other than break) shall be ignored.

If PARMRK is set, and IGNPAR is not set, a byte with a framing or parity error (other than break) shall be given to the application as the three-byte sequence 0xff 0x00 X, where 0xff 0x00 is a two-byte flag preceding each sequence and X is the data of the byte received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid byte of 0xff is given to the application as 0xff 0xff. If neither PARMRK nor IGNPAR is set, a framing or parity error (other than break) shall be given to the application as a single byte 0x00.

If INPCK is set, input parity checking shall be enabled. If INPCK is not set, input parity checking shall be disabled, allowing output parity generation without input parity errors. Note that whether input parity checking is enabled or disabled is independent of whether parity detection is enabled or disabled (see Section 11.2.4 (on page 192)). If parity detection is enabled but input parity checking is disabled, the hardware to which the terminal is connected shall recognize the parity bit, but the terminal special file shall not check whether or not this bit is correctly set.

If ISTRIP is set, valid input bytes shall first be stripped to seven bits; otherwise, all eight bits shall be processed.

If INLCR is set, a received NL character shall be translated into a CR character. If IGNCR is set, a received CR character shall be ignored (not read). If IGNCR is not set and ICRNL is set, a received CR character shall be translated into an NL character.

XSI If IXANY is set, any input character shall restart output that has been suspended.

If IXON is set, start/stop output control shall be enabled. A received STOP character shall suspend output and a received START character shall restart output. When IXON is set, START and STOP characters are not read, but merely perform flow control functions. When IXON is not set, the START and STOP characters shall be read.

If IXOFF is set, start/stop input control shall be enabled. The system shall transmit STOP characters, which are intended to cause the terminal device to stop transmitting data, as needed to prevent the input queue from overflowing and causing implementation-defined behavior, and shall transmit START characters, which are intended to cause the terminal device to resume transmitting data, as soon as the device can continue transmitting data without risk of overflowing the input queue. The precise conditions under which STOP and START characters are transmitted are implementation-defined.

The initial input control value after *open*( ) is implementation-defined.

11.2.3 Output Modes

The **c_oflag** field specifies the terminal interface's treatment of output, and is composed of the bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name symbols in this table are defined in **<termios.h>**:

Mask Name	Description
OPOST	Perform output processing.
ONLCR	Map NL to CR-NL on output.
OCRNL	Map CR to NL on output.
ONOCR	No CR output at column 0.
ONLRET	NL performs CR function.
OFILL	Use fill characters for delay.
OFDEL	Fill is DEL, else NUL.
NLDLY	Select newline delays:
NL0	Newline character type 0.
NL1	Newline character type 1.
CRDLY	Select carriage-return delays:
CR0	Carriage-return delay type 0.
CR1	Carriage-return delay type 1.
CR2	Carriage-return delay type 2.
CR3	Carriage-return delay type 3.
TABDLY	Select horizontal-tab delays:
TAB0	Horizontal-tab delay type 0.
TAB1	Horizontal-tab delay type 1.
TAB2	Horizontal-tab delay type 2.
TAB3	Expand tabs to spaces.
BSDLY	Select backspace delays:
BS0	Backspace-delay type 0.
BS1	Backspace-delay type 1.
VTDLY	Select vertical-tab delays:
VT0	Vertical-tab delay type 0.
VT1	Vertical-tab delay type 1.
FFDLY	Select form-feed delays:
FF0	Form-feed delay type 0.
FF1	Form-feed delay type 1.

If OPOST is set, output data shall be post-processed as described below, so that lines of text are modified to appear appropriately on the terminal device; otherwise, characters shall be transmitted without change.

XSI If ONLCR is set, the NL character shall be transmitted as the CR-NL character pair. If OCRNL is set, the CR character shall be transmitted as the NL character. If ONOCR is set, no CR character shall be transmitted when at column 0 (first position). If ONLRET is set, the NL character is assumed to do the carriage-return function; the column pointer shall be set to 0 and the delays specified for CR shall be used. Otherwise, the NL character is assumed to do just the line-feed function; the column pointer remains unchanged. The column pointer shall also be set to 0 if the CR character is actually transmitted.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 shall indicate no delay. If OFILL is set, fill characters shall be transmitted for delay instead of a timed delay. This is useful for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character shall be DEL; otherwise, NUL.

If a form-feed or vertical-tab delay is specified, it shall last for about 2 seconds.

New-line delay shall last about 0.10 seconds. If ONLRET is set, the carriage-return delays shall be used instead of the newline delays. If OFILL is set, two fill characters shall be transmitted.

Carriage-return delay type 1 shall be dependent on the current column position, type 2 shall be about 0.10 seconds, and type 3 shall be about 0.15 seconds. If OFILL is set, delay type 1 shall transmit two fill characters, and type 2, four fill characters.

Horizontal-tab delay type 1 shall be dependent on the current column position. Type 2 shall be about 0.10 seconds. Type 3 specifies that tabs shall be expanded into spaces. If OFILL is set, two fill characters shall be transmitted for any delay.

Backspace delay shall last about 0.05 seconds. If OFILL is set, one fill character shall be transmitted.

The actual delays depend on line speed and system load.

The initial output control value after *open*( ) is implementation-defined.

11.2.4 Control Modes

The **c_cflag** field describes the hardware control of the terminal, and is composed of the bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name symbols in this table are defined in **<termios.h>**; not all values specified are required to be supported by the underlying hardware:

Mask Name	Description
CLOCAL	Ignore modem status lines.
CREAD	Enable receiver.
CSIZE	Number of bits transmitted or received per byte:
CS5	5 bits
CS6	6 bits
CS7	7 bits
CS8	8 bits.
CSTOPB	Send two stop bits, else one.
HUPCL	Hang up on last close.
PARENB	Parity enable.
PARODD	Odd parity, else even.

In addition, the input and output baud rates are stored in the **termios** structure. The symbols in the following table are defined in **<termios.h>**. Not all values specified are required to be supported by the underlying hardware.

Name	Description	Name	Description
B0	Hang up	B600	600 baud
B50	50 baud	B1200	1200 baud
B75	75 baud	B1800	1800 baud
B110	110 baud	B2400	2400 baud
B134	134.5 baud	B4800	4800 baud
B150	150 baud	B9600	9600 baud
B200	200 baud	B19200	19200 baud
B300	300 baud	B38400	38400 baud

The following functions are provided for getting and setting the values of the input and output baud rates in the **termios** structure: *cfgetispeed*( ), *cfgetospeed*( ), *cfsetispeed*( ), and *cfsetospeed*( ). The effects on the terminal device shall not become effective and not all errors need be detected until the *tcsetattr*( ) function is successfully called.

The CSIZE bits shall specify the number of transmitted or received bits per byte. If ISTRIP is not set, the value of all the other bits is unspecified. If ISTRIP is set, the value of all but the 7 low-order bits shall be zero, but the value of any other bits beyond CSIZE is unspecified when read. CSIZE shall not include the parity bit, if any. If CSTOPB is set, two stop bits shall be used; otherwise, one stop bit. For example, at 110 baud, two stop bits are normally used.

If CREAD is set, the receiver shall be enabled; otherwise, no characters shall be received.

If PARENB is set, parity generation and detection shall be enabled and a parity bit is added to each byte. If parity is enabled, PARODD shall specify odd parity if set; otherwise, even parity shall be used.

If HUPCL is set, the modem control lines for the port shall be lowered when the last process with the port open closes the port or the process terminates. The modem connection shall be broken.

If CLOCAL is set, a connection shall not depend on the state of the modem status lines. If CLOCAL is clear, the modem status lines shall be monitored.

Under normal circumstances, a call to the *open*( ) function shall wait for the modem connection to complete. However, if the O_NONBLOCK flag is set (see *open*( )) or if CLOCAL has been set, the *open*( ) function shall return immediately without waiting for the connection.

If the object for which the control modes are set is not an asynchronous serial connection, some of the modes may be ignored; for example, if an attempt is made to set the baud rate on a network connection to a terminal on another host, the baud rate need not be set on the connection between that terminal and the machine to which it is directly connected.

The initial hardware control value after *open*( ) is implementation-defined.

11.2.5 Local Modes

The **c_lflag** field of the argument structure is used to control various functions. It is composed of the bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name symbols in this table are defined in **<termios.h>**; not all values specified are required to be supported by the underlying hardware:

Mask Name	Description
ECHO	Enable echo.
ECHOE	Echo ERASE as an error correcting backspace.
ECHOK	Echo KILL.
ECHONL	Echo <newline>.
ICANON	Canonical input (erase and kill processing).
IEXTEN	Enable extended (implementation-defined) functions.
ISIG	Enable signals.
NOFLSH	Disable flush after interrupt, quit or suspend.
TOSTOP	Send SIGTTOU for background output.

If ECHO is set, input characters shall be echoed back to the terminal. If ECHO is clear, input characters shall not be echoed.

If ECHOE and ICANON are set, the ERASE character shall cause the terminal to erase, if possible, the last character in the current line from the display. If there is no character to erase, an implementation may echo an indication that this was the case, or do nothing.

If ECHOK and ICANON are set, the KILL character shall either cause the terminal to erase the line from the display or shall echo the newline character after the KILL character.

If ECHONL and ICANON are set, the newline character shall be echoed even if ECHO is not set.

If ICANON is set, canonical processing shall be enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL, as described in Section 11.1.6 (on page 185).

If ICANON is not set, read requests shall be satisfied directly from the input queue. A read shall not be satisfied until at least MIN bytes have been received or the timeout value TIME expired between bytes. The time value represents tenths of a second. See Section 11.1.7 (on page 186) for more details.

If IEXTEN is set, implementation-defined functions shall be recognized from the input data. It is implementation-defined how IEXTEN being set interacts with ICANON, ISIG, IXON, or IXOFF. If IEXTEN is not set, implementation-defined functions shall not be recognized and the corresponding input characters are processed as described for ICANON, ISIG, IXON, and IXOFF.

If ISIG is set, each input character shall be checked against the special control characters INTR, QUIT, and SUSP. If an input character matches one of these control characters, the function associated with that character shall be performed. If ISIG is not set, no checking shall be done. Thus these special input functions are possible only if ISIG is set.

If NOFLSH is set, the normal flush of the input and output queues associated with the INTR, QUIT, and SUSP characters shall not be done.

If TOSTOP is set, the signal SIGTTOU shall be sent to the process group of a process that tries to write to its controlling terminal if it is not in the foreground process group for that terminal. This signal, by default, stops the members of the process group. Otherwise, the output generated by that process shall be output to the current output stream. Processes that are blocking or ignoring SIGTTOU signals are excepted and allowed to produce output, and the SIGTTOU signal shall not be sent.

The initial local control value after *open*( ) is implementation-defined.

11.2.6 Special Control Characters

The special control character values shall be defined by the array **c_cc**. The subscript name and description for each element in both canonical and non-canonical modes are as follows:

Subscript Usage		
Canonical Mode	**Non-Canonical Mode**	**Description**
VEOF		EOF character
VEOL		EOL character
VERASE		ERASE character
VINTR	VINTR	INTR character
VKILL		KILL character
	VMIN	MIN value
VQUIT	VQUIT	QUIT character
VSUSP	VSUSP	SUSP character
	VTIME	TIME value
VSTART	VSTART	START character
VSTOP	VSTOP	STOP character

The subscript values are unique, except that the VMIN and VTIME subscripts may have the same values as the VEOF and VEOL subscripts, respectively.

Implementations that do not support changing the START and STOP characters may ignore the character values in the **c_cc** array indexed by the VSTART and VSTOP subscripts when *tcsetattr*( ) is called, but shall return the value in use when *tcgetattr*( ) is called.

The initial values of all control characters are implementation-defined.

If the value of one of the changeable special control characters (see Section 11.1.9 (on page 187)) is _POSIX_VDISABLE, that function shall be disabled; that is, no input data is recognized as the disabled special character. If ICANON is not set, the value of _POSIX_VDISABLE has no special meaning for the VMIN and VTIME entries of the **c_cc** array.

Chapter 12

Utility Conventions

12.1 Utility Argument Syntax

This section describes the argument syntax of the standard utilities and introduces terminology used throughout IEEE Std 1003.1-200x for describing the arguments processed by the utilities.

Within IEEE Std 1003.1-200x, a special notation is used for describing the syntax of a utility's arguments. Unless otherwise noted, all utility descriptions use this notation, which is illustrated by this example (see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.9.1, Simple Commands):

```
utility_name[−a][−b][−c option_argument]
    [−d|−e][−foption_argument][operand...]
```

The notation used for the SYNOPSIS sections imposes requirements on the implementors of the standard utilities and provides a simple reference for the application developer or system user.

1. The utility in the example is named *utility_name*. It is followed by *options*, *option-arguments*, and *operands*. The arguments that consist of hyphens and single letters or digits, such as `'a'`, are known as *options* (or, historically, *flags*). Certain options are followed by an *option-argument*, as shown with [−**c** *option_argument*]. The arguments following the last options and option-arguments are named *operands*.

2. Option-arguments are sometimes shown separated from their options by <blank>s, sometimes directly adjacent. This reflects the situation that in some cases an option-argument is included within the same argument string as the option; in most cases it is the next argument. The Utility Syntax Guidelines in Section 12.2 (on page 199) require that the option be a separate argument from its option-argument, but there are some exceptions in IEEE Std 1003.1-200x to ensure continued operation of historical applications:

   a. If the SYNOPSIS of a standard utility shows a space character between an option and option-argument (as with [−**c** *option_argument*] in the example), a conforming application shall use separate arguments for that option and its option-argument.

   b. If a space character is not shown (as with [−**f***option_argument*] in the example), a conforming application shall place an option and its option-argument directly adjacent in the same argument string, without intervening <blank>s.

   c. Notwithstanding the preceding requirements on conforming applications, a conforming system shall permit, but shall not require, an application to specify options and option-arguments as separate arguments whether or not a space character is shown on the synopsis line, except in those cases (marked with the XSI portability warning) where an option-argument is optional and no separation can be used.

   d. A standard utility may also be implemented to operate correctly when the required separation into multiple arguments is violated by a non-conforming application.

   In summary, the following table shows allowable combinations:

	SYNOPSIS Shows:		
	−a *arg*	−b*arg*	−c[*arg*]
Conforming application shall use:	−a *arg*	−b*arg*	N/A
System shall support:	−a *arg*	−b*arg*	−c*arg* or −c
System may support:	−a*arg*	−b *arg*	

3. Options are usually listed in alphabetical order unless this would make the utility description more confusing. There are no implied relationships between the options based upon the order in which they appear, unless otherwise stated in the OPTIONS section, or unless the exception in Guideline 11 of Section 12.2 (on page 199) applies. If an option that does not have option-arguments is repeated, the results are undefined, unless otherwise stated.

4. Frequently, names of parameters that require substitution by actual values are shown with embedded underscores. Alternatively, parameters are shown as follows:

   ```
   <parameter name>
   ```

   The angle brackets are used for the symbolic grouping of a phrase representing a single parameter and conforming applications shall not include them in data submitted to the utility.

5. When a utility has only a few permissible options, they are sometimes shown individually, as in the example. Utilities with many flags generally show all of the individual flags (that do not take option-arguments) grouped, as in:

   ```
   utility_name [−abcDxyz][−p arg][operand]
   ```

   Utilities with very complex arguments may be shown as follows:

   ```
   utility_name [options][operands]
   ```

6. Unless otherwise specified, whenever an operand or option-argument is, or contains, a numeric value:

   - The number is interpreted as a decimal integer.
   - Numerals in the range 0 to 2 147 483 647 are syntactically recognized as numeric values.
   - When the utility description states that it accepts negative numbers as operands or option-arguments, numerals in the range −2 147 483 647 to 2 147 483 647 are syntactically recognized as numeric values.
   - Ranges greater than those listed here are allowed.

   This does not mean that all numbers within the allowable range are necessarily semantically correct. A standard utility that accepts an option-argument or operand that is to be interpreted as a number, and for which a range of values smaller than that shown above is permitted by the IEEE Std 1003.1-200x, describes that smaller range along with the description of the option-argument or operand. If an error is generated, the utility's diagnostic message shall indicate that the value is out of the supported range, not that it is syntactically incorrect.

7. Arguments or option-arguments enclosed in the '[' and ']' notation are optional and can be omitted. Conforming applications shall not include the '[' and ']' symbols in data submitted to the utility.

8. Arguments separated by the '|' vertical bar notation are mutually-exclusive. Conforming applications shall not include the '|' symbol in data submitted to the utility.

Alternatively, mutually-exclusive options and operands may be listed with multiple synopsis lines. For example:

```
utility_name −d[−a][−c option_argument][operand...]
utility_name[−a][−b][operand...]
```

When multiple synopsis lines are given for a utility, it is an indication that the utility has mutually-exclusive arguments. These mutually-exclusive arguments alter the functionality of the utility so that only certain other arguments are valid in combination with one of the mutually-exclusive arguments. Only one of the mutually-exclusive arguments is allowed for invocation of the utility. Unless otherwise stated in an accompanying OPTIONS section, the relationships between arguments depicted in the SYNOPSIS sections are mandatory requirements placed on conforming applications. The use of conflicting mutually-exclusive arguments produces undefined results, unless a utility description specifies otherwise. When an option is shown without the '[' and ']' brackets, it means that option is required for that version of the SYNOPSIS. However, it is not required to be the first argument, as shown in the example above, unless otherwise stated.

9. Ellipses ("...") are used to denote that one or more occurrences of an option or operand are allowed. When an option or an operand followed by ellipses is enclosed in brackets, zero or more options or operands can be specified. The forms:

```
utility_name −f option_argument...[operand...]
utility_name [−g option_argument]...[operand...]
```

indicate that multiple occurrences of the option and its option-argument preceding the ellipses are valid, with semantics as indicated in the OPTIONS section of the utility. (See also Guideline 11 in Section 12.2.) In the first example, each option-argument requires a preceding −**f** and at least one −**f** *option_argument* must be given.

10. When the synopsis line is too long to be printed on a single line in the Shell and Utilities volume of IEEE Std 1003.1-200x, the indented lines following the initial line are continuation lines. An actual use of the command would appear on a single logical line.

12.2 Utility Syntax Guidelines

The following guidelines are established for the naming of utilities and for the specification of options, option-arguments, and operands. The *getopt*( ) function in the System Interfaces volume of IEEE Std 1003.1-200x assists utilities in handling options and operands that conform to these guidelines.

Operands and option-arguments can contain characters not specified in the portable character set.

The guidelines are intended to provide guidance to the authors of future utilities, such as those written specific to a local system or that are components of a larger application. Some of the standard utilities do not conform to all of these guidelines; in those cases, the OPTIONS sections describe the deviations.

Guideline 1: Utility names should be between two and nine characters, inclusive.

Guideline 2: Utility names should include lowercase letters (the **lower** character classification) and digits only from the portable character set.

Guideline 3: Each option name should be a single alphanumeric character (the **alnum** character classification) from the portable character set. The −**W** (capital-W) option shall be reserved for vendor options.

Multi-digit options should not be allowed.

Guideline 4: All options should be preceded by the '−' delimiter character.

Guideline 5: Options without option-arguments should be accepted when grouped behind one '−' delimiter.

Guideline 6: Each option and option-argument should be a separate argument, except as noted in Section 12.1 (on page 197), item (2).

Guideline 7: Option-arguments should not be optional.

Guideline 8: When multiple option-arguments are specified to follow a single option, they should be presented as a single argument, using commas within that argument or <blank>s within that argument to separate them.

Guideline 9: All options should precede operands on the command line.

Guideline 10: The argument −− should be accepted as a delimiter indicating the end of options. Any following arguments should be treated as operands, even if they begin with the '−' character. The −− argument should not be used as an option or as an operand.

Guideline 11: The order of different options relative to one another should not matter, unless the options are documented as mutually-exclusive and such an option is documented to override any incompatible options preceding it. If an option that has option-arguments is repeated, the option and option-argument combinations should be interpreted in the order specified on the command line.

Guideline 12: The order of operands may matter and position-related interpretations should be determined on a utility-specific basis.

Guideline 13: For utilities that use operands to represent files to be opened for either reading or writing, the '−' operand should be used only to mean standard input (or standard output when it is clear from context that an output file is being specified).

The utilities in the Shell and Utilities volume of IEEE Std 1003.1-200x that claim conformance to these guidelines shall conform completely to these guidelines as if these guidelines contained the term “shall” instead of “should”. On some implementations, the utilities accept usage in violation of these guidelines for backward compatibility as well as accepting the required form.

It is recommended that all future utilities and applications use these guidelines to enhance user portability. The fact that some historical utilities could not be changed (to avoid breaking existing applications) should not deter this future goal.

Chapter 13

Headers

This chapter describes the contents of headers.

Headers contain function prototypes, the definition of symbolic constants, common structures, preprocessor macros, and defined types. Each function in the System Interfaces volume of IEEE Std 1003.1-200x specifies the headers that an application shall include in order to use that function. In most cases, only one header is required. These headers are present on an application development system; they need not be present on the target execution system.

13.1 Format of Entries

The entries in this chapter are based on a common format as follows. The only sections relating to conformance are the SYNOPSIS and DESCRIPTION.

NAME

This section gives the name or names of the entry and briefly states its purpose.

SYNOPSIS

This section summarizes the use of the entry being described.

DESCRIPTION

This section describes the functionality of the header.

APPLICATION USAGE

This section is non-normative.

This section gives warnings and advice to application writers about the entry. In the event of conflict between warnings and advice and a normative part of this volume of IEEE Std 1003.1-200x, the normative material is to be taken as correct.

RATIONALE

This section is non-normative.

This section contains historical information concerning the contents of this volume of IEEE Std 1003.1-200x and why features were included or discarded by the standard developers.

FUTURE DIRECTIONS

This section is non-normative.

This section provides comments which should be used as a guide to current thinking; there is not necessarily a commitment to adopt these future directions.

SEE ALSO

This section is non-normative.

This section gives references to related information.

CHANGE HISTORY

This section is non-normative.

This section shows the derivation of the entry and any significant changes that have been made to it.

NAME

aio.h — asynchronous input and output (**REALTIME**)

SYNOPSIS

AIO

```
#include <aio.h>
```

DESCRIPTION

The **<aio.h>** header shall define the **aiocb** structure which shall include at least the following members:

`int`	`aio_fildes`	File descriptor.
`off_t`	`aio_offset`	File offset.
`volatile void`	`*aio_buf`	Location of buffer.
`size_t`	`aio_nbytes`	Length of transfer.
`int`	`aio_reqprio`	Request priority offset.
`struct sigevent`	`aio_sigevent`	Signal number and value.
`int`	`aio_lio_opcode`	Operation to be performed.

This header shall also include the following constants:

AIO_CANCELED A return value indicating that all requested operations have been canceled.

AIO_NOTCANCELED
A return value indicating that some of the requested operations could not be canceled since they are in progress.

AIO_ALLDONE A return value indicating that none of the requested operations could be canceled since they are already complete.

LIO_WAIT A *lio_listio*( ) synchronization operation indicating that the calling thread is to suspend until the *lio_listio*( ) operation is complete.

LIO_NOWAIT A *lio_listio*( ) synchronization operation indicating that the calling thread is to continue execution while the *lio_listio*( ) operation is being performed, and no notification is given when the operation is complete.

LIO_READ A *lio_listio*( ) element operation option requesting a read.

LIO_WRITE A *lio_listio*( ) element operation option requesting a write.

LIO_NOP A *lio_listio*( ) element operation option indicating that no transfer is requested.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int      aio_cancel(int, struct aiocb *);
int      aio_error(const struct aiocb *);
int      aio_fsync(int, struct aiocb *);
int      aio_read(struct aiocb *);
ssize_t  aio_return(struct aiocb *);
int      aio_suspend(const struct aiocb *const[], int,
             const struct timespec *);
int      aio_write(struct aiocb *);
int      lio_listio(int, struct aiocb *restrict const[restrict], int,
             struct sigevent *restrict);
```

Inclusion of the **<aio.h>** header may make visible symbols defined in the headers **<fcntl.h>**, **<signal.h>**, **<sys/types.h>**, and **<time.h>**.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<fcntl.h>, **<signal.h>**, **<sys/types.h>**, **<time.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *fsync*( ), *lseek*( ), *read*( ), *write*( )

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The **<aio.h>** header is marked as part of the Asynchronous Input and Output option.

The description of the constants is expanded.

The **restrict** keyword is added to the prototype for *lio_listio*( ).

NAME

arpa/inet.h — definitions for internet operations

SYNOPSIS

```
#include <arpa/inet.h>
```

DESCRIPTION

The **in_port_t** and **in_addr_t** types shall be defined as described in **<netinet/in.h>**.

The **in_addr** structure shall be defined as described in **<netinet/in.h>**.

IP6 The INET_ADDRSTRLEN and INET6_ADDRSTRLEN macros shall be defined as described in **<netinet/in.h>**.

The following shall either be declared as functions, defined as macros, or both. If functions are declared, function prototypes shall be provided.

```
uint32_t htonl(uint32_t);
uint16_t htons(uint16_t);
uint32_t ntohl(uint32_t);
uint16_t ntohs(uint16_t);
```

The **uint32_t** and **uint16_t** types shall be defined as described in **<inttypes.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
in_addr_t    inet_addr(const char *);
char        *inet_ntoa(struct in_addr);
const char  *inet_ntop(int, const void *restrict, char *restrict,
                 socklen_t);
int          inet_pton(int, const char *restrict, void *restrict);
```

Inclusion of the **<arpa/inet.h>** header may also make visible all symbols from **<netinet/in.h>** and **<inttypes.h>**.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<netinet/in.h>, **<inttypes.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *htonl*( ), *inet_addr*( )

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The **restrict** keyword is added to the prototypes for *inet_ntop*( ) and *inet_pton*( ).

NAME

assert.h — verify program assertion

SYNOPSIS

```
#include <assert.h>
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-200x defers to the ISO C standard.

The **<assert.h>** header shall define the *assert*( ) macro. It refers to the macro NDEBUG which is not defined in the header. If NDEBUG is defined as a macro name before the inclusion of this header, the *assert*( ) macro shall be defined simply as:

```
#define assert(ignore)((void) 0)
```

Otherwise, the macro behaves as described in *assert*( ).

The *assert*( ) macro shall be redefined according to the current state of NDEBUG each time **<assert.h>** is included.

The *assert*( ) macro shall be implemented as a macro, not as a function. If the macro definition is suppressed in order to access an actual function, the behavior is undefined.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *assert*( )

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The definition of the *assert*() macro is changed for alignment with the ISO/IEC 9899:1999 standard.

NAME

complex.h — complex arithmetic

SYNOPSIS

```
#include <complex.h>
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-200x defers to the ISO C standard.

The **<complex.h>** header shall define the following macros:

complex Expands to **_Complex**.

_Complex_I Expands to a constant expression of type **const float _Complex**, with the value of the imaginary unit (that is, a number such that $i^2=-1$).

imaginary Expands to **_Imaginary**.

_Imaginary_I Expands to a constant expression of type **const float _Imaginary** with the value of the imaginary unit.

I Expands to either _Imaginary_I or _Complex_I. If _Imaginary_I is not defined, I expands to _Complex_I.

The macros imaginary and _Imaginary_I shall be defined if and only if the implementation supports imaginary types.

An application may undefine and then, perhaps, redefine the complex, imaginary, and I macros.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
double               cabs(double complex);
float                cabsf(float complex);
long double          cabsl(long double complex);
double complex       cacos(double complex);
float complex        cacosf(float complex);
double complex       cacosh(double complex);
float complex        cacoshf(float complex);
long double complex  cacoshl(long double complex);
long double complex  cacosl(long double complex);
double               carg(double complex);
float                cargf(float complex);
long double          cargl(long double complex);
double complex       casin(double complex);
float complex        casinf(float complex);
double complex       casinh(double complex);
float complex        casinhf(float complex);
long double complex  casinhl(long double complex);
long double complex  casinl(long double complex);
double complex       catan(double complex);
float complex        catanf(float complex);
double complex       catanh(double complex);
float complex        catanhf(float complex);
long double complex  catanhl(long double complex);
long double complex  catanl(long double complex);
```

```
double complex       ccos(double complex);
float complex        ccosf(float complex);
double complex       ccosh(double complex);
float complex        ccoshf(float complex);
long double complex  ccoshl(long double complex);
long double complex  ccosl(long double complex);
double complex       cexp(double complex);
float complex        cexpf(float complex);
long double complex  cexpl(long double complex);
double               cimag(double complex);
float                cimagf(float complex);
long double          cimagl(long double complex);
double complex       clog(double complex);
float complex        clogf(float complex);
long double complex  clogl(long double complex);
double complex       conj(double complex);
float complex        conjf(float complex);
long double complex  conjl(long double complex);
double complex       cpow(double complex, double complex);
float complex        cpowf(float complex, float complex);
long double complex  cpowl(long double complex, long double complex);
double complex       cproj(double complex);
float complex        cprojf(float complex);
long double complex  cprojl(long double complex);
double               creal(double complex);
float                crealf(float complex);
long double          creall(long double complex);
double complex       csin(double complex);
float complex        csinf(float complex);
double complex       csinh(double complex);
float complex        csinhf(float complex);
long double complex  csinhl(long double complex);
long double complex  csinl(long double complex);
double complex       csqrt(double complex);
float complex        csqrtf(float complex);
long double complex  csqrtl(long double complex);
double complex       ctan(double complex);
float complex        ctanf(float complex);
double complex       ctanh(double complex);
float complex        ctanhf(float complex);
long double complex  ctanhl(long double complex);
long double complex  ctanl(long double complex);
```

APPLICATION USAGE

Values are interpreted as radians, not degrees.

RATIONALE

The choice of *I* instead of *i* for the imaginary unit concedes to the widespread use of the identifier *i* for other purposes. The application can use a different identifier, say *j*, for the imaginary unit by following the inclusion of the **<complex.h>** header with:

```
#undef I
#define j _Imaginary_I
```

An *I* suffix to designate imaginary constants is not required, as multiplication by *I* provides a sufficiently convenient and more generally useful notation for imaginary terms. The corresponding real type for the imaginary unit is **float**, so that use of *I* for algorithmic or notational convenience will not result in widening types.

On systems with imaginary types, the application has the ability to control whether use of the macro I introduces an imaginary type, by explicitly defining I to be _Imaginary_I or _Complex_I. Disallowing imaginary types is useful for some applications intended to run on implementations without support for such types.

The macro _Imaginary_I provides a test for whether imaginary types are supported.

The *cis*( ) function (*cos*(*x*) + *I***sin*(*x*)) was considered but rejected because its implementation is easy and straightforward, even though some implementations could compute sine and cosine more efficiently in tandem.

FUTURE DIRECTIONS

The following function names and the same names suffixed with *f* or *l* are reserved for future use, and may be added to the declarations in the **<complex.h>** header.

cerf( )
cerfc( )
cexp2( )
cexpm1( )
clog10( )
clog1p( )
clog2( )
clgamma( )
ctgamma( )

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *cabs*( ), *cacos*( ), *cacosh*( ), *carg*( ), *casin*( ), *casinh*( ), *catan*( ), *catanh*( ), *ccos*( ), *ccosh*( ), *cexp*( ), *cimag*( ), *clog*( ), *conj*( ), *cpow*( ), *cproj*( ), *creal*( ), *csin*( ), *csinh*( ), *csqrt*( ), *ctan*( ), *ctanh*( )

CHANGE HISTORY

First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

NAME

cpio.h — cpio archive values

SYNOPSIS

XSI

```
#include <cpio.h>
```

DESCRIPTION

Values needed by the *c_mode* field of the *cpio* archive format are described as follows:

Name	Description	Value (Octal)
C_IRUSR	Read by owner.	0000400
C_IWUSR	Write by owner.	0000200
C_IXUSR	Execute by owner.	0000100
C_IRGRP	Read by group.	0000040
C_IWGRP	Write by group.	0000020
C_IXGRP	Execute by group.	0000010
C_IROTH	Read by others.	0000004
C_IWOTH	Write by others.	0000002
C_IXOTH	Execute by others.	0000001
C_ISUID	Set user ID.	0004000
C_ISGID	Set group ID.	0002000
C_ISVTX	On directories, restricted deletion flag.	0001000
C_ISDIR	Directory.	0040000
C_ISFIFO	FIFO.	0010000
C_ISREG	Regular file.	0100000
C_ISBLK	Block special.	0060000
C_ISCHR	Character special.	0020000
C_ISCTG	Reserved.	0110000
C_ISLNK	Symbolic link.	0120000
C_ISSOCK	Socket.	0140000

The header shall define the symbolic constant:

```
MAGIC    "070707"
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Shell and Utilities volume of IEEE Std 1003.1-200x, *pax*

CHANGE HISTORY

First released in Issue 3 of the Headers Interface, Issue 3 specification. Derived from the POSIX.1-1988 standard.

Issue 6

The SEE ALSO is updated to refer to *pax*, since the *cpio* utility is not included in the Shell and Utilities volume of IEEE Std 1003.1-200x.

NAME

ctype.h — character types

SYNOPSIS

```
#include <ctype.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these symbols in this header.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int    isalnum(int);
int    isalpha(int);
int    isascii(int);
int    isblank(int);
int    iscntrl(int);
int    isdigit(int);
int    isgraph(int);
int    islower(int);
int    isprint(int);
int    ispunct(int);
int    isspace(int);
int    isupper(int);
int    isxdigit(int);
int    toascii(int);
int    tolower(int);
int    toupper(int);
```

(XSI: isascii( ), toascii( ))

The following are defined as macros:

XSI

```
int    _toupper(int);
int    _tolower(int);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<locale.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *isalnum*( ), *isalpha*( ), *isascii*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ), *isxdigit*( ), *mblen*( ), *mbstowcs*( ), *mbtowc*( ), *setlocale*( ), *toascii*( ), *tolower*( ), *_tolower*( ), *toupper*( ), *_toupper*( ), *wcstombs*( ), *wctomb*( )

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are now marked.

NAME

dirent.h — format of directory entries

SYNOPSIS

```
#include <dirent.h>
```

DESCRIPTION

The internal format of directories is unspecified.

The **<dirent.h>** header shall define the following type:

DIR A type representing a directory stream.

It shall also define the structure **dirent** which shall include the following members:

XSI	`ino_t`	`d_ino`	File serial number.
	`char`	`d_name[]`	Name of entry.

XSI The type **ino_t** shall be defined as described in **<sys/types.h>**.

The character array *d_name* is of unspecified size, but the number of bytes preceding the terminating null byte shall not exceed {NAME_MAX}.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int            closedir(DIR *);
DIR           *opendir(const char *);
struct dirent *readdir(DIR *);
```

TSF

```
int            readdir_r(DIR *restrict, struct dirent *restrict,
                   struct dirent **restrict);
```

```
void           rewinddir(DIR *);
```

XSI

```
void           seekdir(DIR *, long);
long           telldir(DIR *);
```

APPLICATION USAGE

None.

RATIONALE

Information similar to that in the **<dirent.h>** header is contained in a file **<sys/dir.h>** in 4.2 BSD and 4.3 BSD. The equivalent in these implementations of **struct dirent** from this volume of IEEE Std 1003.1-200x is **struct direct**. The filename was changed because the name **<sys/dir.h>** was also used in earlier implementations to refer to definitions related to the older access method; this produced name conflicts. The name of the structure was changed because this volume of IEEE Std 1003.1-200x does not completely define what is in the structure, so it could be different on some implementations from **struct direct**.

The name of an array of **char** of an unspecified size should not be used as an lvalue. Use of:

```
sizeof(d_name)
```

is incorrect; use:

```
strlen(d_name)
```

instead.

The array of **char** *d_name* is not a fixed size. Implementations may need to declare **struct dirent** with an array size for *d_name* of 1, but the actual number of characters provided matches (or only slightly exceeds) the length of the filename.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *closedir*( ), *opendir*( ), *readdir*( ), *readdir_r*( ), *rewinddir*( ), *seekdir*( ), *telldir*( )

CHANGE HISTORY

First released in Issue 2.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The Open Group Corrigendum U026/7 is applied, correcting the prototype for *readdir_r*( ).

The **restrict** keyword is added to the prototype for *readdir_r*( ).

NAME

dlfcn.h — dynamic linking

SYNOPSIS

XSI
```
#include <dlfcn.h>
```

DESCRIPTION

The **<dlfcn.h>** header shall define at least the following macros for use in the construction of a *dlopen*( ) *mode* argument:

RTLD_LAZY Relocations are performed at an implementation-defined time.

RTLD_NOW Relocations are performed when the object is loaded.

RTLD_GLOBAL All symbols are available for relocation processing of other modules.

RTLD_LOCAL All symbols are not made available for relocation processing by other modules.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided. |

```
int    dlclose(void *);
char  *dlerror(void);
void  *dlopen(const char *, int);
void  *dlsym(void *restrict, const char *restrict);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *dlopen*( ), *dlclose*( ), *dlsym*( ), *dlerror*( )

CHANGE HISTORY

First released in Issue 5.

Issue 6

The **restrict** keyword is added to the prototype for *dlsym*( ).

NAME

errno.h — system error numbers

SYNOPSIS

```
#include <errno.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-200x defers to the ISO C standard.

CX The ISO C standard only requires the symbols [EDOM], [EILSEQ], and [ERANGE] to be defined.

The **<errno.h>** header shall provide a declaration for *errno* and give positive values for the following symbolic constants. Their values shall be unique except as noted below:

[E2BIG]	Argument list too long.
[EACCES]	Permission denied.
[EADDRINUSE]	Address in use.
[EADDRNOTAVAIL]	Address not available.
[EAFNOSUPPORT]	Address family not supported.
[EAGAIN]	Resource unavailable, try again (may be the same value as [EWOULDBLOCK]).
[EALREADY]	Connection already in progress.
[EBADF]	Bad file descriptor.
[EBADMSG]	Bad message.
[EBUSY]	Device or resource busy.
[ECANCELED]	Operation canceled.
[ECHILD]	No child processes.
[ECONNABORTED]	Connection aborted.
[ECONNREFUSED]	Connection refused.
[ECONNRESET]	Connection reset.
[EDEADLK]	Resource deadlock would occur.
[EDESTADDRREQ]	Destination address required.
[EDOM]	Mathematics argument out of domain of function.
[EDQUOT]	Reserved.
[EEXIST]	File exists.
[EFAULT]	Bad address.
[EFBIG]	File too large.
[EHOSTUNREACH]	Host is unreachable.
[EIDRM]	Identifier removed.
[EILSEQ]	Illegal byte sequence.

	[EINPROGRESS]	Operation in progress.
	[EINTR]	Interrupted function.
	[EINVAL]	Invalid argument.
	[EIO]	I/O error.
	[EISCONN]	Socket is connected.
	[EISDIR]	Is a directory.
	[ELOOP]	Too many levels of symbolic links.
	[EMFILE]	Too many open files.
	[EMLINK]	Too many links.
	[EMSGSIZE]	Message too large.
	[EMULTIHOP]	Reserved.
	[ENAMETOOLONG]	Filename too long.
	[ENETDOWN]	Network is down.
	[ENETUNREACH]	Network unreachable.
	[ENFILE]	Too many files open in system.
	[ENOBUFS]	No buffer space available.
XSR	[ENODATA]	No message is available on the STREAM head read queue.
	[ENODEV]	No such device.
	[ENOENT]	No such file or directory.
	[ENOEXEC]	Executable file format error.
	[ENOLCK]	No locks available.
	[ENOLINK]	Reserved.
	[ENOMEM]	Not enough space.
	[ENOMSG]	No message of the desired type.
	[ENOPROTOOPT]	Protocol not available.
	[ENOSPC]	No space left on device.
XSR	[ENOSR]	No STREAM resources.
XSR	[ENOSTR]	Not a STREAM.
	[ENOSYS]	Function not supported.
	[ENOTCONN]	The socket is not connected.
	[ENOTDIR]	Not a directory.
	[ENOTEMPTY]	Directory not empty.
	[ENOTSOCK]	Not a socket.
	[ENOTSUP]	Not supported.

	[ENOTTY]	Inappropriate I/O control operation.
	[ENXIO]	No such device or address.
	[EOPNOTSUPP]	Operation not supported on socket.
	[EOVERFLOW]	Value too large to be stored in data type.
	[EPERM]	Operation not permitted.
	[EPIPE]	Broken pipe.
	[EPROTO]	Protocol error.
	[EPROTONOSUPPORT]	Protocol not supported.
	[EPROTOTYPE]	Protocol wrong type for socket.
	[ERANGE]	Result too large.
	[EROFS]	Read-only file system.
	[ESPIPE]	Invalid seek.
	[ESRCH]	No such process.
	[ESTALE]	Reserved.
XSR	[ETIME]	Stream *ioctl*( ) timeout.
	[ETIMEDOUT]	Connection timed out.
	[ETXTBSY]	Text file busy.
	[EWOULDBLOCK]	Operation would block (may be the same value as [EAGAIN]).
	[EXDEV]	Cross-device link.

APPLICATION USAGE

Additional error numbers may be defined on conforming systems; see the System Interfaces volume of IEEE Std 1003.1-200x.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, Section 2.3, Error Numbers

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Updated for alignment with the POSIX Realtime Extension.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The majority of the error conditions previously marked as extensions are now mandatory, except for the STREAMS-related error conditions.

Values for *errno* are now required to be distinct positive values rather than non-zero values. This change is for alignment with the ISO/IEC 9899:1999 standard.

NAME

fcntl.h — file control options

SYNOPSIS

```
#include <fcntl.h>
```

DESCRIPTION

The **<fcntl.h>** header shall define the following requests and arguments for use by the functions *fcntl*( ) and *open*( ).

Values for *cmd* used by *fcntl*( ) (the following values are unique) are as follows:

F_DUPFD	Duplicate file descriptor.
F_GETFD	Get file descriptor flags.
F_SETFD	Set file descriptor flags.
F_GETFL	Get file status flags and file access modes.
F_SETFL	Set file status flags.
F_GETLK	Get record locking information.
F_SETLK	Set record locking information.
F_SETLKW	Set record locking information; wait if blocked.
F_GETOWN	Get process or process group ID to receive SIGURG signals.
F_SETOWN	Set process or process group ID to receive SIGURG signals.

File descriptor flags used for *fcntl*( ) are as follows:

FD_CLOEXEC	Close the file descriptor upon execution of an *exec* family function.

Values for *l_type* used for record locking with *fcntl*( ) (the following values are unique) are as follows:

F_RDLCK	Shared or read lock.
F_UNLCK	Unlock.
F_WRLCK	Exclusive or write lock.

XSI The values used for *l_whence*, SEEK_SET, SEEK_CUR, and SEEK_END shall be defined as described in **<unistd.h>**.

The following values are file creation flags and are used in the *oflag* value to *open*( ). They shall be bitwise-distinct.

O_CREAT	Create file if it does not exist.
O_EXCL	Exclusive use flag.
O_NOCTTY	Do not assign controlling terminal.
O_TRUNC	Truncate flag.

File status flags used for *open*( ) and *fcntl*( ) are as follows:

	O_APPEND	Set append mode.
SIO	O_DSYNC	Write according to synchronized I/O data integrity completion.
	O_NONBLOCK	Non-blocking mode.

SIO O_RSYNC	Synchronized read I/O operations.
O_SYNC	Write according to synchronized I/O file integrity completion.

Mask for use with file access modes is as follows:

O_ACCMODE	Mask for file access modes.

File access modes used for *open*( ) and *fcntl*( ) are as follows:

O_RDONLY	Open for reading only.
O_RDWR	Open for reading and writing.
O_WRONLY	Open for writing only.

XSI The symbolic names for file modes for use as values of **mode_t** shall be defined as described in **<sys/stat.h>**.

ADV Values for *advice* used by *posix_fadvise*( ) are as follows:

POSIX_FADV_NORMAL
The application has no advice to give on its behavior with respect to the specified data. It is the default characteristic if no advice is given for an open file.

POSIX_FADV_SEQUENTIAL
The application expects to access the specified data sequentially from lower offsets to higher offsets.

POSIX_FADV_RANDOM
The application expects to access the specified data in a random order.

POSIX_FADV_WILLNEED
The application expects to access the specified data in the near future.

POSIX_FADV_DONTNEED
The application expects that it will not access the specified data in the near future.

POSIX_FADV_NOREUSE
The application expects to access the specified data once and then not reuse it thereafter.

The structure **flock** describes a file lock. It shall include the following members:

`short`	`l_type`	Type of lock; F_RDLCK, F_WRLCK, F_UNLCK.
`short`	`l_whence`	Flag for starting offset.
`off_t`	`l_start`	Relative offset in bytes.
`off_t`	`l_len`	Size; if 0 then until EOF.
`pid_t`	`l_pid`	Process ID of the process holding the lock; returned with F_GETLK.

The **mode_t**, **off_t**, and **pid_t** types shall be defined as described in **<sys/types.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int  creat(const char *, mode_t);
int  fcntl(int, int, ...);
int  open(const char *, int, ...);
int  posix_fadvise(int, off_t, size_t, int);
int  posix_fallocate(int, off_t, size_t);
```

XSI Inclusion of the **<fcntl.h>** header may also make visible all symbols from **<sys/stat.h>** and **<unistd.h>**.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/stat.h>, **<sys/types.h>**, **<unistd.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *creat*( ), *exec*( ), *fcntl*( ), *open*( ), *posix_fadvise*( ), *posix_fallocate*( ), *posix_madvise*( )

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with POSIX Realtime Extension.

Issue 6

The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- O_DSYNC and O_RSYNC are marked as part of the Synchronized Input and Output option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The definition of the **mode_t**, **off_t**, and **pid_t** types is mandated.

The F_GETOWN and F_SETOWN values are added for sockets.

The *posix_fadvise*( ), *posix_fallocate*( ), and *posix_madvise*( ) functions are added for alignment with IEEE Std 1003.1d-1999.

IEEE PASC Interpretation 1003.1 #102 is applied moving the prototype for *posix_madvise*( ) to **<sys/mman.h>**. |

NAME

fenv.h — floating-point environment

SYNOPSIS

```
#include <fenv.h>
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-200x defers to the ISO C standard.

The **<fenv.h>** header shall define the following data types through **typedef**:

fenv_t Represents the entire floating-point environment. The floating-point environment refers collectively to any floating-point status flags and control modes supported by the implementation.

fexcept_t Represents the floating-point status flags collectively, including any status the implementation associates with the flags. A floating-point status flag is a system variable whose value is set (but never cleared) when a floating-point exception is raised, which occurs as a side effect of exceptional floating-point arithmetic to provide auxiliary information. A floating-point control mode is a system variable whose value may be set by the user to affect the subsequent behavior of floating-point arithmetic.

The **<fenv.h>** header shall define the following constants if and only if the implementation supports the floating-point exception by means of the floating-point functions *fwclearexcept*( ), *fegetexceptflag*( ), *feraiseexcept*( ), *fesetexceptflag*( ), and *fetestexcept*( ). Each expands to an integer constant expression with values such that bitwise-inclusive ORs of all combinations of the constants result in distinct values.

FE_DIVBYZERO
FE_INEXACT
FE_INVALID
FE_OVERFLOW
FE_UNDERFLOW

The **<fenv.h>** header shall define the following constant, which is simply the bitwise-inclusive OR of all floating-point exception constants defined above:

FE_ALL_EXCEPT

The **<fenv.h>** header shall define the following constants if and only if the implementation supports getting and setting the represented rounding direction by means of the *fegetround*( ) and *fesetround*( ) functions. Each expands to an integer constant expression whose values are distinct non-negative vales.

FE_DOWNWARD
FE_TONEAREST
FE_TOWARDZERO
FE_UPWARD

The **<fenv.h>** header shall define the following constant, which represents the default floating-point environment (that is, the one installed at program startup) and has type pointer to const-qualified **fenv_t**. It can be used as an argument to the functions within the **<fenv.h>** header that manage the floating-point environment.

FE_DFL_ENV

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int  feclearexcept(int);
int  fegetexceptflag(fexcept_t *, int);
int  feraiseexcept(int);
int  fesetexceptflag(const fexcept_t *, int);
int  fetestexcept(int);
int  fegetround(void);
int  fesetround(int);
int  fegetenv(fenv_t *);
int  feholdexcept(fenv_t *);
int  fesetenv(const fenv_t *);
int  feupdateenv(const fenv_t *);
```

The FENV_ACCESS pragma provides a means to inform the implementation when an application might access the floating-point environment to test floating-point status flags or run under non-default floating-point control modes. The pragma shall occur either outside external declarations or preceding all explicit declarations and statements inside a compound statement. When outside external declarations, the pragma takes effect from its occurrence until another FENV_ACCESS pragma is encountered, or until the end of the translation unit. When inside a compound statement, the pragma takes effect from its occurrence until another FENV_ACCESS pragma is encountered (including within a nested compound statement), or until the end of the compound statement; at the end of a compound statement the state for the pragma is restored to its condition just before the compound statement. If this pragma is used in any other context, the behavior is undefined. If part of an application tests floating-point status flags, sets floating-point control modes, or runs under non-default mode settings, but was translated with the state for the FENV_ACCESS pragma off, the behavior is undefined. The default state (on or off) for the pragma is implementation-defined. (When execution passes from a part of the application translated with FENV_ACCESS off to a part translated with FENV_ACCESS on, the state of the floating-point status flags is unspecified and the floating-point control modes have their default settings.)

APPLICATION USAGE

This header is designed to support the floating-point exception status flags and directed-rounding control modes required by the IEC 60559: 1989 standard, and other similar floating-point state information. Also it is designed to facilitate code portability among all systems.

Certain application programming conventions support the intended model of use for the floating-point environment:

- A function call does not alter its caller's floating-point control modes, clear its caller's floating-point status flags, nor depend on the state of its caller's floating-point status flags unless the function is so documented.

- A function call is assumed to require default floating-point control modes, unless its documentation promises otherwise.

- A function call is assumed to have the potential for raising floating-point exceptions, unless its documentation promises otherwise.

With these conventions, an application can safely assume default floating-point control modes (or be unaware of them). The responsibilities associated with accessing the floating-point environment fall on the application that does so explicitly.

Even though the rounding direction macros may expand to constants corresponding to the values of FLT_ROUNDS, they are not required to do so.

For example:

```
#include <fenv.h>
void f(double x)
{
    #pragma STDC FENV_ACCESS ON
    void g(double);
    void h(double);
    /* ... */
    g(x + 1);
    h(x + 1);
    /* ... */
}
```

If the function *g*( ) might depend on status flags set as a side effect of the first *x*+1, or if the second *x*+1 might depend on control modes set as a side effect of the call to function *g*( ), then the application shall contain an appropriately placed invocation as follows:

```
#pragma STDC FENV_ACCESS ON
```

RATIONALE

The fexcept_t Type

fexcept_t does not have to be an integer type. Its values must be obtained by a call to *fegetexceptflag*( ), and cannot be created by logical operations from the exception macros. An implementation might simply implement **fexcept_t** as an **int** and use the representations reflected by the exception macros, but is not required to; other representations might contain extra information about the exceptions. **fexcept_t** might be a **struct** with a member for each exception (that might hold the address of the first or last floating-point instruction that caused that exception). The ISO/IEC 9899:1999 standard makes no claims about the internals of an **fexcept_t**, and so the user cannot inspect it.

Exception and Rounding Macros

Macros corresponding to unsupported modes and rounding directions are not defined by the implementation and must not be defined by the application. An application might use **#ifdef** to test for this. |

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *feclearexcept*( ), *fegetenv*( ), *fegetexceptflag*( ), *fegetround*( ), *feholdexcept*( ), *feraiseexcept*( ), *fesetenv*( ), *fesetexceptflag*( ), *fesetround*( ), *fetestexcept*( ), *feupdateenv*( )

CHANGE HISTORY

First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

The return types for *feclearexcept*( ), *fegetexcepflag*( ), *feraiseexcept*( ), *fesetexceptflag*( ), *fegetenv*( ), *fesetenv*( ), and *feupdateenv*( ) are changed from **void** to **int** for alignment with the ISO/IEC 9899:1999 standard, Defect Report 202.

NAME

float.h — floating types

SYNOPSIS

```
#include <float.h>
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-200x defers to the ISO C standard.

The characteristics of floating types are defined in terms of a model that describes a representation of floating-point numbers and values that provide information about an implementation's floating-point arithmetic.

The following parameters are used to define the model for each floating-point type:

s Sign (±1).

b Base or radix of exponent representation (an integer >1).

e Exponent (an integer between a minimum $e_{\min}$ and a maximum $e_{\max}$).

p Precision (the number of base–*b* digits in the significand).

f_k Non-negative integers less than *b* (the significand digits).

A floating-point number *x* is defined by the following model:

$$x = sb^e \sum_{k=1}^{p} f_k \, b^{-k}, \; e_{\min} \le e \le e_{\max}$$

In addition to normalized floating-point numbers (f_1>0 if $x \neq 0$), floating types may be able to contain other kinds of floating-point numbers, such as subnormal floating-point numbers ($x \neq 0$, $e = e_{\min}$, $f_1 = 0$) and unnormalized floating-point numbers ($x \neq 0$, $e > e_{\min}$, $f_1 = 0$), and values that are not floating-point numbers, such as infinities and NaNs. A *NaN* is an encoding signifying Not-a-Number. A *quiet NaN* propagates through almost every arithmetic operation without raising a floating-point exception; a *signaling NaN* generally raises a floating-point exception when occurring as an arithmetic operand.

The accuracy of the floating-point operations ('+', '−', '*', '/') and of the library functions in **<math.h>** and **<complex.h>** that return floating-point results is implementation-defined. The implementation may state that the accuracy is unknown.

All integer values in the **<float.h>** header, except FLT_ROUNDS, shall be constant expressions suitable for use in **#if** preprocessing directives; all floating values shall be constant expressions. All except DECIMAL_DIG, FLT_EVAL_METHOD, FLT_RADIX, and FLT_ROUNDS have separate names for all three floating-point types. The floating-point model representation is provided for all values except FLT_EVAL_METHOD and FLT_ROUNDS.

The rounding mode for floating-point addition is characterized by the implementation-defined value of FLT_ROUNDS:

−1 Indeterminable.

0 Toward zero.

1 To nearest.

2 Toward positive infinity.

3 Toward negative infinity.

All other values for FLT_ROUNDS characterize implementation-defined rounding behavior.

The values of operations with floating operands and values subject to the usual arithmetic conversions and of floating constants are evaluated to a format whose range and precision may be greater than required by the type. The use of evaluation formats is characterized by the implementation-defined value of FLT_EVAL_METHOD:

−1 Indeterminable.

0 Evaluate all operations and constants just to the range and precision of the type.

1 Evaluate operations and constants of type **float** and **double** to the range and precision of the **double** type, evaluate **long double** operations and constants to the range and precision of the **long double** type.

2 Evaluate all operations and constants to the range and precision of the **long double** type.

All other negative values for FLT_EVAL_METHOD characterize implementation-defined behavior.

The values given in the following list shall be defined as constant expressions with implementation-defined values that are greater or equal in magnitude (absolute value) to those shown, with the same sign.

- Radix of exponent representation, *b*.

  FLT_RADIX 2

- Number of base-FLT_RADIX digits in the floating-point significand, *p*.

  FLT_MANT_DIG

  DBL_MANT_DIG

  LDBL_MANT_DIG

- Number of decimal digits, *n*, such that any floating-point number in the widest supported floating type with $p_{\max}$ radix *b* digits can be rounded to a floating-point number with *n* decimal digits and back again without change to the value.

$$\begin{cases} p_{\max} \log_{10} b & \text{if } b \text{ is a power of } 10 \\ \lceil 1 + p_{\max} \log_{10} b \rceil & \text{otherwise} \end{cases}$$

  DECIMAL_DIG 10

- Number of decimal digits, *q*, such that any floating-point number with *q* decimal digits can be rounded into a floating-point number with *p* radix *b* digits and back again without change to the *q* decimal digits.

$$\begin{cases} p \log_{10} b & \text{if } b \text{ is a power of } 10 \\ \lfloor (p - 1) \log_{10} b \rfloor & \text{otherwise} \end{cases}$$

  FLT_DIG 6

  DBL_DIG 10

LDBL_DIG	10

- Minimum negative integer such that FLT_RADIX raised to that power minus 1 is a normalized floating-point number, $e_{\min}$.

FLT_MIN_EXP

DBL_MIN_EXP

LDBL_MIN_EXP

- Minimum negative integer such that 10 raised to that power is in the range of normalized floating-point numbers.

$$\left\lceil \log_{10} b^{e_{\min}-1} \right\rceil$$

FLT_MIN_10_EXP	−37
DBL_MIN_10_EXP	−37
LDBL_MIN_10_EXP	−37

- Maximum integer such that FLT_RADIX raised to that power minus 1 is a representable finite floating-point number, $e_{\max}$.

FLT_MAX_EXP

DBL_MAX_EXP

LDBL_MAX_EXP

- Maximum integer such that 10 raised to that power is in the range of representable finite floating-point numbers.

$$\left\lfloor \log_{10}((1-b^{-p})\, b^{e_{\max}}) \right\rfloor$$

FLT_MAX_10_EXP	+37
DBL_MAX_10_EXP	+37
LDBL_MAX_10_EXP	+37

The values given in the following list shall be defined as constant expressions with implementation-defined values that are greater than or equal to those shown:

- Maximum representable finite floating-point number.

$$(1-b^{-p})\, b^{e_{\max}}$$

FLT_MAX	1E+37
DBL_MAX	1E+37
LDBL_MAX	1E+37

The values given in the following list shall be defined as constant expressions with implementation-defined (positive) values that are less than or equal to those shown:

- The difference between 1 and the least value greater that 1 that is representable in the given floating-point type, b^{1-p}.

FLT_EPSILON	1E−5
DBL_EPSILON	1E−9

LDBL_EPSILON	1E–9

- Minimum normalized positive floating-point number, $b^{e_{\min}-1}$.

FLT_MIN	1E–37
DBL_MIN	1E–37
LDBL_MIN	1E–37

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<complex.h>, **<math.h>**

CHANGE HISTORY

First released in Issue 4. Derived from the ISO C standard.

Issue 6

The description of the operations with floating-point values is updated for alignment with the ISO/IEC 9899:1999 standard.

NAME

fmtmsg.h — message display structures

SYNOPSIS

XSI
```
#include <fmtmsg.h>
```

DESCRIPTION

The **<fmtmsg.h>** header shall define the following macros, which expand to constant integer expressions:

MM_HARD	Source of the condition is hardware.
MM_SOFT	Source of the condition is software.
MM_FIRM	Source of the condition is firmware.
MM_APPL	Condition detected by application.
MM_UTIL	Condition detected by utility.
MM_OPSYS	Condition detected by operating system.
MM_RECOVER	Recoverable error.
MM_NRECOV	Non-recoverable error.
MM_HALT	Error causing application to halt.
MM_ERROR	Application has encountered a non-fatal fault.
MM_WARNING	Application has detected unusual non-error condition.
MM_INFO	Informative message.
MM_NOSEV	No severity level provided for the message.
MM_PRINT	Display message on standard error.
MM_CONSOLE	Display message on system console.

The table below indicates the null values and identifiers for *fmtmsg*( ) arguments. The **<fmtmsg.h>** header shall define the macros in the **Identifier** column, which expand to constant expressions that expand to expressions of the type indicated in the **Type** column:

Argument	Type	Null-Value	Identifier
label	**char** *	(char*)0	MM_NULLLBL
severity	**int**	0	MM_NULLSEV
class	**long**	**0L**	MM_NULLMC
text	**char** *	(char*)0	MM_NULLTXT
action	**char** *	(char*)0	MM_NULLACT
tag	**char** *	(char*)0	MM_NULLTAG

The **<fmtmsg.h>** header shall also define the following macros for use as return values for *fmtmsg*( ):

MM_OK	The function succeeded.
MM_NOTOK	The function failed completely.
MM_NOMSG	The function was unable to generate a message on standard error, but otherwise succeeded.

MM_NOCON The function was unable to generate a console message, but otherwise succeeded.

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided.

```
int fmtmsg(long, const char *, int,
    const char *, const char *, const char *);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *fmtmsg*( )

CHANGE HISTORY

First released in Issue 4, Version 2.

NAME

fnmatch.h — filename-matching types

SYNOPSIS

```
#include <fnmatch.h>
```

DESCRIPTION

The **<fnmatch.h>** header shall define the following constants:

FNM_NOMATCH	The string does not match the specified pattern.
FNM_PATHNAME	Slash in *string* only matches slash in *pattern*.
FNM_PERIOD	Leading period in *string* must be exactly matched by period in *pattern*.
FNM_NOESCAPE	Disable backslash escaping.
OB XSI FNM_NOSYS	Reserved.

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided.

```
int fnmatch(const char *, const char *, int);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *fnmatch*( ), the Shell and Utilities volume of IEEE Std 1003.1-200x

CHANGE HISTORY

First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 6

The constant FNM_NOSYS is marked obsolescent.

NAME

ftw.h — file tree traversal

SYNOPSIS

XSI

```
#include <ftw.h>
```

DESCRIPTION

The **<ftw.h>** header shall define the **FTW** structure that includes at least the following members:

```
int  base
int  level
```

The **<ftw.h>** header shall define macros for use as values of the third argument to the application-supplied function that is passed as the second argument to *ftw*( ) and *nftw*( ):

FTW_F	File.
FTW_D	Directory.
FTW_DNR	Directory without read permission.
FTW_DP	Directory with subdirectories visited.
FTW_NS	Unknown type; *stat*( ) failed.
FTW_SL	Symbolic link.
FTW_SLN	Symbolic link that names a nonexistent file.

The **<ftw.h>** header shall define macros for use as values of the fourth argument to *nftw*( ):

FTW_PHYS	Physical walk, does not follow symbolic links. Otherwise, *nftw*( ) follows links but does not walk down any path that crosses itself.
FTW_MOUNT	The walk does not cross a mount point.
FTW_DEPTH	All subdirectories are visited before the directory itself.
FTW_CHDIR	The walk changes to each directory before reading it.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided. |

```
int ftw(const char *,
    int (*)(const char *, const struct stat *, int), int);
int nftw(const char *, int (*)
    (const char *, const struct stat *, int, struct FTW*),
    int, int);
```

The **<ftw.h>** header shall define the **stat** structure and the symbolic names for *st_mode* and the file type test macros as described in **<sys/stat.h>**.

Inclusion of the **<ftw.h>** header may also make visible all symbols from **<sys/stat.h>**.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/stat.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *ftw*( ), *nftw*( )

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

A description of FTW_DP is added.

NAME

glob.h — pathname pattern-matching types

SYNOPSIS

```
#include <glob.h>
```

DESCRIPTION

The **<glob.h>** header shall define the structures and symbolic constants used by the *glob*( ) function.

The structure type **glob_t** shall contain at least the following members:

`size_t`	`gl_pathc`	Count of paths matched by *pattern*.
`char`	`**gl_pathv`	Pointer to a list of matched pathnames.
`size_t`	`gl_offs`	Slots to reserve at the beginning of *gl_pathv*.

The following constants shall be provided as values for the *flags* argument:

GLOB_APPEND Append generated pathnames to those previously obtained.

GLOB_DOOFFS Specify how many null pointers to add to the beginning of *pglob->gl_pathv*.

GLOB_ERR Cause *glob*( ) to return on error.

GLOB_MARK Each pathname that is a directory that matches *pattern* has a slash appended.

GLOB_NOCHECK If *pattern* does not match any pathname, then return a list consisting of only *pattern*.

GLOB_NOESCAPE Disable backslash escaping.

GLOB_NOSORT Do not sort the pathnames returned.

The following constants shall be defined as error return values:

GLOB_ABORTED The scan was stopped because GLOB_ERR was set or (**errfunc*)( ) returned non-zero.

GLOB_NOMATCH The pattern does not match any existing pathname, and GLOB_NOCHECK was not set in flags.

GLOB_NOSPACE An attempt to allocate memory failed.

OB XSI GLOB_NOSYS Reserved.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int  glob(const char *restrict, int, int (*restrict)(const char *, int),
          glob_t *restrict);
void globfree (glob_t *);
```

The implementation may define additional macros or constants using names beginning with GLOB_.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *glob*( ), the Shell and Utilities volume of IEEE Std 1003.1-200x

CHANGE HISTORY

First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 6

The **restrict** keyword is added to the prototype for *glob*( ).

The constant GLOB_NOSYS is marked obsolescent.

NAME

grp.h — group structure

SYNOPSIS

```
#include <grp.h>
```

DESCRIPTION

The **<grp.h>** header shall declare the structure **group** which shall include the following members:

```
char    *gr_name  The name of the group.
gid_t    gr_gid   Numerical group ID.
char   **gr_mem   Pointer to a null-terminated array of character
                  pointers to member names.
```

The **gid_t** type shall be defined as described in **<sys/types.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided. |

```
     struct group  *getgrgid(gid_t);
     struct group  *getgrnam(const char *);
TSF  int            getgrgid_r(gid_t, struct group *, char *,
                        size_t, struct group **);
     int            getgrnam_r(const char *, struct group *, char *,
                        size_t , struct group **);
XSI  struct group  *getgrent(void);
     void           endgrent(void);
     void           setgrent(void);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *endgrent*( ), *getgrgid*( ), *getgrnam*( )

CHANGE HISTORY

First released in Issue 1.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The definition of **gid_t** is mandated.
- The *getgrgid_r*( ) and *getgrnam_r*( ) functions are marked as part of the Thread-Safe Functions option.

NAME

iconv.h — codeset conversion facility

SYNOPSIS

XSI

```
#include <iconv.h>
```

DESCRIPTION

The **<iconv.h>** header shall define the following type:

iconv_t Identifies the conversion from one codeset to another.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
iconv_t iconv_open(const char *, const char *);
size_t  iconv(iconv_t, char **restrict, size_t *restrict, char **restrict,
              size_t *restrict);
int     iconv_close(iconv_t);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *iconv*( ), *iconv_close*( ), *iconv_open*( )

CHANGE HISTORY

First released in Issue 4.

Issue 6

The **restrict** keyword is added to the prototype for *iconv*( ).

NAME

inttypes.h — fixed size integer types

SYNOPSIS

```
#include <inttypes.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these symbols in this header.

The **<inttypes.h>** header shall include the **<stdint.h>** header.

The **<inttypes.h>** header shall include a definition of at least the following type:

imaxdiv_t Structure type that is the type of the value returned by the *imaxdiv*( ) function.

The following macros shall be defined. Each expands to a character string literal containing a conversion specifier, possibly modified by a length modifier, suitable for use within the *format* argument of a formatted input/output function when converting the corresponding integer type. These macros have the general form of PRI (character string literals for the *fprintf*( ) and *fwprintf*( ) family of functions) or SCN (character string literals for the *fscanf*( ) and *fwscanf*( ) family of functions), followed by the conversion specifier, followed by a name corresponding to a similar type name in **<stdint.h>**. In these names, *N* represents the width of the type as described in **<stdint.h>**. For example, *PRIdFAST32* can be used in a format string to print the value of an integer of type **int_fast32_t**.

The *fprintf*( ) macros for signed integers are:

PRId*N*	PRIdLEAST*N*	PRIdFAST*N*	PRIdMAX	PRIdPTR
PRIi*N*	PRIiLEAST*N*	PRIiFAST*N*	PRIiMAX	PRIiPTR

The *fprintf*( ) macros for unsigned integers are:

PRIo*N*	PRIoLEAST*N*	PRIoFAST*N*	PRIoMAX	PRIoPTR
PRIu*N*	PRIuLEAST*N*	PRIuFAST*N*	PRIuMAX	PRIuPTR
PRIx*N*	PRIxLEAST*N*	PRIxFAST*N*	PRIxMAX	PRIxPTR
PRIX*N*	PRIXLEAST*N*	PRIXFAST*N*	PRIXMAX	PRIXPTR

The *fscanf*( ) macros for signed integers are:

SCNd*N*	SCNdLEAST*N*	SCNdFAST*N*	SCNdMAX	SCNdPTR
SCNi*N*	SCNiLEAST*N*	SCNiFAST*N*	SCNiMAX	SCNiPTR

The *fscanf*( ) macros for unsigned integers are:

SCNo*N*	SCNoLEAST*N*	SCNoFAST*N*	SCNoMAX	SCNoPTR
SCNu*N*	SCNuLEAST*N*	SCNuFAST*N*	SCNuMAX	SCNuPTR
SCNx*N*	SCNxLEAST*N*	SCNxFAST*N*	SCNxMAX	SCNxPTR

For each type that the implementation provides in **<stdint.h>**, the corresponding *fprintf*( ) macros shall be defined and the corresponding *fscanf*( ) macros shall be defined unless the implementation does not have a suitable fscanf length modifier for the type.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
intmax_t  imaxabs(intmax_t);
imaxdiv_t imaxdiv(intmax_t, intmax_t);
intmax_t  strtoimax(const char *restrict, char **restrict, int);
```

```
uintmax_t strtoumax(const char *restrict, char **restrict, int);
intmax_t  wcstoimax(const wchar_t *restrict, wchar_t **restrict, int);
uintmax_t wcstoumax(const wchar_t *restrict, wchar_t **restrict, int);
```

EXAMPLES

```
#include <inttypes.h>
#include <wchar.h>
int main(void)
{
    uintmax_t i = UINTMAX_MAX; // This type always exists.
    wprintf(L"The largest integer value is %020"
        PRIxMAX "\n", i);
    return 0;
}
```

APPLICATION USAGE

None.

RATIONALE

The ISO/IEC 9899:1990 standard specifies that the language should support four signed and unsigned integer data types—**char**, **short**, **int**, and **long**—but places very little requirement on their size other than that **int** and **short** be at least 16 bits and **long** be at least as long as **int** and not smaller than 32 bits. For 16-bit systems, most implementations assign 8, 16, 16, and 32 bits to **char**, **short**, **int**, and **long**, respectively. For 32-bit systems, the common practice is to assign 8, 16, 32, and 32 bits to these types. This difference in **int** size can create some problems for users who migrate from one system to another which assigns different sizes to integer types, because the ISO C standard integer promotion rule can produce silent changes unexpectedly. The need for defining an extended integer type increased with the introduction of 64-bit systems.

The purpose of **<inttypes.h>** is to provide a set of integer types whose definitions are consistent across machines and independent of operating systems and other implementation idiosyncrasies. It defines, via **typedef**, integer types of various sizes. Implementations are free to **typedef** them as ISO C standard integer types or extensions that they support. Consistent use of this header will greatly increase the portability of a users program across platforms.

FUTURE DIRECTIONS

Macro names beginning with PRI or SCN followed by any lowercase letter or 'X' may be added to the macros defined in the **<inttypes.h>** header.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *imaxdiv*( )

CHANGE HISTORY

First released in Issue 5.

Issue 6

The Open Group Base Resolution bwg97-006 is applied.

This reference page is updated to align with the ISO/IEC 9899:1999 standard.

NAME

iso646.h — alternative spellings

SYNOPSIS

```
#include <iso646.h>
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-200x defers to the ISO C standard.

The **<iso646.h>** header shall define the following eleven macros (on the left) that expand to the corresponding tokens (on the right):

and	&&
and_eq	&=
bitand	&
bitor	|
compl	~
not	!
not_eq	!=
or	||
or_eq	|=
xor	^
xor_eq	^=

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

None.

CHANGE HISTORY

First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

NAME

langinfo.h — language information constants

SYNOPSIS

XSI `#include <langinfo.h>`

DESCRIPTION

The **<langinfo.h>** header contains the constants used to identify items of *langinfo* data (see *nl_langinfo*( )). The type of the constant, **nl_item**, shall be defined as described in **<nl_types.h>**.

The following constants shall be defined. The entries under **Category** indicate in which *setlocale*( ) category each item is defined.

Constant	Category	Meaning
CODESET	*LC_CTYPE*	Codeset name.
D_T_FMT	*LC_TIME*	String for formatting date and time.
D_FMT	*LC_TIME*	Date format string.
T_FMT	*LC_TIME*	Time format string.
T_FMT_AMPM	*LC_TIME*	a.m. or p.m. time format string.
AM_STR	*LC_TIME*	Ante Meridian affix.
PM_STR	*LC_TIME*	Post Meridian affix.
DAY_1	*LC_TIME*	Name of the first day of the week (for example, Sunday).
DAY_2	*LC_TIME*	Name of the second day of the week (for example, Monday).
DAY_3	*LC_TIME*	Name of the third day of the week (for example, Tuesday).
DAY_4	*LC_TIME*	Name of the fourth day of the week (for example, Wednesday).
DAY_5	*LC_TIME*	Name of the fifth day of the week (for example, Thursday).
DAY_6	*LC_TIME*	Name of the sixth day of the week (for example, Friday).
DAY_7	*LC_TIME*	Name of the seventh day of the week (for example, Saturday).
ABDAY_1	*LC_TIME*	Abbreviated name of the first day of the week.
ABDAY_2	*LC_TIME*	Abbreviated name of the second day of the week.
ABDAY_3	*LC_TIME*	Abbreviated name of the third day of the week.
ABDAY_4	*LC_TIME*	Abbreviated name of the fourth day of the week.
ABDAY_5	*LC_TIME*	Abbreviated name of the fifth day of the week.
ABDAY_6	*LC_TIME*	Abbreviated name of the sixth day of the week.
ABDAY_7	*LC_TIME*	Abbreviated name of the seventh day of the week.
MON_1	*LC_TIME*	Name of the first month of the year.
MON_2	*LC_TIME*	Name of the second month.
MON_3	*LC_TIME*	Name of the third month.
MON_4	*LC_TIME*	Name of the fourth month.
MON_5	*LC_TIME*	Name of the fifth month.
MON_6	*LC_TIME*	Name of the sixth month.
MON_7	*LC_TIME*	Name of the seventh month.
MON_8	*LC_TIME*	Name of the eighth month.
MON_9	*LC_TIME*	Name of the ninth month.
MON_10	*LC_TIME*	Name of the tenth month.
MON_11	*LC_TIME*	Name of the eleventh month.
MON_12	*LC_TIME*	Name of the twelfth month.

Constant	Category	Meaning
ABMON_1	*LC_TIME*	Abbreviated name of the first month.
ABMON_2	*LC_TIME*	Abbreviated name of the second month.
ABMON_3	*LC_TIME*	Abbreviated name of the third month.
ABMON_4	*LC_TIME*	Abbreviated name of the fourth month.
ABMON_5	*LC_TIME*	Abbreviated name of the fifth month.
ABMON_6	*LC_TIME*	Abbreviated name of the sixth month.
ABMON_7	*LC_TIME*	Abbreviated name of the seventh month.
ABMON_8	*LC_TIME*	Abbreviated name of the eighth month.
ABMON_9	*LC_TIME*	Abbreviated name of the ninth month.
ABMON_10	*LC_TIME*	Abbreviated name of the tenth month.
ABMON_11	*LC_TIME*	Abbreviated name of the eleventh month.
ABMON_12	*LC_TIME*	Abbreviated name of the twelfth month.
ERA	*LC_TIME*	Era description segments.
ERA_D_FMT	*LC_TIME*	Era date format string.
ERA_D_T_FMT	*LC_TIME*	Era date and time format string.
ERA_T_FMT	*LC_TIME*	Era time format string.
ALT_DIGITS	*LC_TIME*	Alternative symbols for digits.
RADIXCHAR	*LC_NUMERIC*	Radix character.
THOUSEP	*LC_NUMERIC*	Separator for thousands.
YESEXPR	*LC_MESSAGES*	Affirmative response expression.
NOEXPR	*LC_MESSAGES*	Negative response expression.
CRNCYSTR	*LC_MONETARY*	Currency symbol, preceded by '−' if the symbol should appear before the value, '+' if the symbol should appear after the value, or '.' if the symbol should replace the radix character.

If the locale's values for **p_cs_precedes** and **n_cs_precedes** do not match, the value of *nl_langinfo*(CRNCYSTR) is unspecified.

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided.

```
char *nl_langinfo(nl_item);
```

Inclusion of the **<langinfo.h>** header may also make visible all symbols from **<nl_types.h>**.

APPLICATION USAGE

Wherever possible, users are advised to use functions compatible with those in the ISO C standard to access items of *langinfo* data. In particular, the *strftime*( ) function should be used to access date and time information defined in category *LC_TIME*. The *localeconv*( ) function should be used to access information corresponding to RADIXCHAR, THOUSEP, and CRNCYSTR.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *nl_langinfo*( ), *localeconv*( ), *strfmon*( ), *strftime*( ), Chapter 7 (on page 119)

CHANGE HISTORY

First released in Issue 2.

Issue 5

The constants YESSTR and NOSTR are marked LEGACY.

Issue 6

The constants YESSTR and NOSTR are removed.

NAME

libgen.h — definitions for pattern matching functions

SYNOPSIS

XSI

```
#include <libgen.h>
```

DESCRIPTION

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided. |

```
char  *basename(char *);
char  *dirname(char *);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *basename*( ), *dirname*( )

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

The function prototypes for *basename*( ) and *dirname*( ) are changed to indicate that the first argument is of type **char *** rather than **const char ***.

Issue 6

The **__loc1** symbol and the *regcmp*( ) and *regex*( ) functions are removed.

NAME

limits.h — implementation-defined constants

SYNOPSIS

```
#include <limits.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these symbols in this header.

CX Many of the symbols listed here are not defined by the ISO/IEC 9899:1999 standard. Such symbols are not shown as CX shaded.

The **<limits.h>** header shall define various symbolic names. Different categories of names are described below.

The names represent various limits on resources that the implementation imposes on applications.

Implementations may choose any appropriate value for each limit, provided it is not more restrictive than the Minimum Acceptable Values listed below. Symbolic constant names beginning with _POSIX may be found in **<unistd.h>**.

Applications should not assume any particular value for a limit. To achieve maximum portability, an application should not require more resource than the Minimum Acceptable Value quantity. However, an application wishing to avail itself of the full amount of a resource available on an implementation may make use of the value given in **<limits.h>** on that particular implementation, by using the symbolic names listed below. It should be noted, however, that many of the listed limits are not invariant, and at runtime, the value of the limit may differ from those given in this header, for the following reasons:

- The limit is pathname-dependent.
- The limit differs between the compile and runtime machines.

For these reasons, an application may use the *fpathconf*( ), *pathconf*( ), and *sysconf*( ) functions to determine the actual value of a limit at runtime.

The items in the list ending in _MIN give the most negative values that the mathematical types are guaranteed to be capable of representing. Numbers of a more negative value may be supported on some implementations, as indicated by the **<limits.h>** header on the implementation, but applications requiring such numbers are not guaranteed to be portable to all implementations. For positive constants ending in _MIN, this indicates the minimum acceptable value.

The Minimum Acceptable Value symbol '*' indicates that there is no guaranteed value across all conforming implementations.

Runtime Invariant Values (Possibly Indeterminate)

A definition of one of the symbolic names in the following list shall be omitted from **<limits.h>** on specific implementations where the corresponding value is equal to or greater than the stated minimum, but is unspecified.

This indetermination might depend on the amount of available memory space on a specific instance of a specific implementation. The actual value supported by a specific instance shall be provided by the *sysconf*( ) function.

AIO {AIO_LISTIO_MAX}
Maximum number of I/O operations in a single list I/O call supported by the implementation.
Minimum Acceptable Value: {_POSIX_AIO_LISTIO_MAX}

AIO {AIO_MAX}
Maximum number of outstanding asynchronous I/O operations supported by the implementation.
Minimum Acceptable Value: {_POSIX_AIO_MAX}

AIO {AIO_PRIO_DELTA_MAX}
The maximum amount by which a process can decrease its asynchronous I/O priority level from its own scheduling priority.
Minimum Acceptable Value: 0

{ARG_MAX}
Maximum length of argument to the *exec* functions including environment data.
Minimum Acceptable Value: {_POSIX_ARG_MAX}

XSI {ATEXIT_MAX}
Maximum number of functions that may be registered with *atexit*( ).
Minimum Acceptable Value: 32

{CHILD_MAX}
Maximum number of simultaneous processes per real user ID.
Minimum Acceptable Value: {_POSIX_CHILD_MAX}

TMR {DELAYTIMER_MAX}
Maximum number of timer expiration overruns.
Minimum Acceptable Value: {_POSIX_DELAYTIMER_MAX}

{HOST_NAME_MAX}
Maximum length of a host name (not including the terminating null) as returned from the *gethostname*( ) function.
Minimum Acceptable Value: {_POSIX_HOST_NAME_MAX}

XSI {IOV_MAX}
Maximum number of **iovec** structures that one process has available for use with *readv*( ) or *writev*( ).
Minimum Acceptable Value: {_XOPEN_IOV_MAX}

{LOGIN_NAME_MAX}
Maximum length of a login name.
Minimum Acceptable Value: {_POSIX_LOGIN_NAME_MAX}

MSG {MQ_OPEN_MAX}
The maximum number of open message queue descriptors a process may hold.
Minimum Acceptable Value: {_POSIX_MQ_OPEN_MAX}

MSG {MQ_PRIO_MAX}
The maximum number of message priorities supported by the implementation.
Minimum Acceptable Value: {_POSIX_MQ_PRIO_MAX}

{OPEN_MAX}
Maximum number of files that one process can have open at any one time.
Minimum Acceptable Value: {_POSIX_OPEN_MAX}

{PAGESIZE}
Size in bytes of a page.
Minimum Acceptable Value: 1

XSI {PAGE_SIZE}
Equivalent to {PAGESIZE}. If either {PAGESIZE} or {PAGE_SIZE} is defined, the other is defined with the same value.

THR {PTHREAD_DESTRUCTOR_ITERATIONS}
Maximum number of attempts made to destroy a thread's thread-specific data values on thread exit.
Minimum Acceptable Value: {_POSIX_THREAD_DESTRUCTOR_ITERATIONS}

THR {PTHREAD_KEYS_MAX}
Maximum number of data keys that can be created by a process.
Minimum Acceptable Value: {_POSIX_THREAD_KEYS_MAX}

THR {PTHREAD_STACK_MIN}
Minimum size in bytes of thread stack storage.
Minimum Acceptable Value: 0

THR {PTHREAD_THREADS_MAX}
Maximum number of threads that can be created per process.
Minimum Acceptable Value: {_POSIX_THREAD_THREADS_MAX}

{RE_DUP_MAX}
The number of repeated occurrences of a BRE permitted by the *regexec*( ) and *regcomp*( ) functions when using the interval notation {\(*m*,*n*\}; see Section 9.3.6 (on page 170).
Minimum Acceptable Value: {_POSIX2_RE_DUP_MAX}

RTS {RTSIG_MAX}
Maximum number of realtime signals reserved for application use in this implementation.
Minimum Acceptable Value: {_POSIX_RTSIG_MAX}

SEM {SEM_NSEMS_MAX}
Maximum number of semaphores that a process may have.
Minimum Acceptable Value: {_POSIX_SEM_NSEMS_MAX}

SEM {SEM_VALUE_MAX}
The maximum value a semaphore may have.
Minimum Acceptable Value: {_POSIX_SEM_VALUE_MAX}

RTS {SIGQUEUE_MAX}
Maximum number of queued signals that a process may send and have pending at the receiver(s) at any time.
Minimum Acceptable Value: {_POSIX_SIGQUEUE_MAX}

SS|TSP {SS_REPL_MAX}
The maximum number of replenishment operations that may be simultaneously pending for a particular sporadic server scheduler.
Minimum Acceptable Value: {_POSIX_SS_REPL_MAX}

{STREAM_MAX}
The number of streams that one process can have open at one time. If defined, it has the same value as {FOPEN_MAX} (see **<stdio.h>**).
Minimum Acceptable Value: {_POSIX_STREAM_MAX}

{SYMLOOP_MAX}
Maximum number of symbolic links that can be reliably traversed in the resolution of a pathname in the absence of a loop.
Minimum Acceptable Value: {_POSIX_SYMLOOP_MAX}

TMR {TIMER_MAX}
Maximum number of timers per-process supported by the implementation.
Minimum Acceptable Value: {_POSIX_TIMER_MAX}

TRC {TRACE_EVENT_NAME_MAX}
Maximum length of the trace event name.
Minimum Acceptable Value: {_POSIX_TRACE_EVENT_NAME_MAX}

TRC {TRACE_NAME_MAX}
Maximum length of the trace generation version string or of the trace stream name.
Minimum Acceptable Value: {_POSIX_TRACE_NAME_MAX}

TRC {TRACE_SYS_MAX}
Maximum number of trace streams that may simultaneously exist in the system.
Minimum Acceptable Value: {_POSIX_TRACE_SYS_MAX}

TRC {TRACE_USER_EVENT_MAX}
Maximum number of user trace event type identifiers that may simultaneously exist in a traced process, including the predefined user trace event POSIX_TRACE_UNNAMED_USER_EVENT.
Minimum Acceptable Value: {_POSIX_TRACE_USER_EVENT_MAX}

{TTY_NAME_MAX}
Maximum length of terminal device name.
Minimum Acceptable Value: {_POSIX_TTY_NAME_MAX}

{TZNAME_MAX}
Maximum number of bytes supported for the name of a timezone (not of the TZ variable).
Minimum Acceptable Value: {_POSIX_TZNAME_MAX}

Note: The length given by {TZNAME_MAX} does not include the quoting characters mentioned in Section 8.3 (on page 161).

Pathname Variable Values

The values in the following list may be constants within an implementation or may vary from one pathname to another. For example, file systems or directories may have different characteristics.

A definition of one of the values shall be omitted from the **<limits.h>** header on specific implementations where the corresponding value is equal to or greater than the stated minimum, but where the value can vary depending on the file to which it is applied. The actual value supported for a specific pathname shall be provided by the *pathconf*( ) function.

{FILESIZEBITS}
Minimum number of bits needed to represent, as a signed integer value, the maximum size of a regular file allowed in the specified directory.
Minimum Acceptable Value: 32

{LINK_MAX}
Maximum number of links to a single file.
Minimum Acceptable Value: {_POSIX_LINK_MAX}

{MAX_CANON}
Maximum number of bytes in a terminal canonical input line.
Minimum Acceptable Value: {_POSIX_MAX_CANON}

{MAX_INPUT}
Minimum number of bytes for which space is available in a terminal input queue; therefore, the maximum number of bytes a conforming application may require to be typed as input before reading them.
Minimum Acceptable Value: {_POSIX_MAX_INPUT}

{NAME_MAX}
Maximum number of bytes in a filename (not including terminating null).
Minimum Acceptable Value: {_POSIX_NAME_MAX}
XSI Minimum Acceptable Value: {_XOPEN_NAME_MAX}

{PATH_MAX}
Maximum number of bytes in a pathname, including the terminating null character.
Minimum Acceptable Value: {_POSIX_PATH_MAX}
XSI Minimum Acceptable Value: {_XOPEN_PATH_MAX}

{PIPE_BUF}
Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.
Minimum Acceptable Value: {_POSIX_PIPE_BUF}

ADV {POSIX_ALLOC_SIZE_MIN}
Minimum number of bytes of storage actually allocated for any portion of a file.
Minimum Acceptable Value: Not specified.

ADV {POSIX_REC_INCR_XFER_SIZE}
Recommended increment for file transfer sizes between the {POSIX_REC_MIN_XFER_SIZE} and {POSIX_REC_MAX_XFER_SIZE} values.
Minimum Acceptable Value: Not specified.

ADV {POSIX_REC_MAX_XFER_SIZE}
Maximum recommended file transfer size.
Minimum Acceptable Value: Not specified.

ADV {POSIX_REC_MIN_XFER_SIZE}
Minimum recommended file transfer size.
Minimum Acceptable Value: Not specified.

ADV {POSIX_REC_XFER_ALIGN}
Recommended file transfer buffer alignment.
Minimum Acceptable Value: Not specified.

{SYMLINK_MAX}
Maximum number of bytes in a symbolic link.
Minimum Acceptable Value: {_POSIX_SYMLINK_MAX}

Runtime Increasable Values

The magnitude limitations in the following list shall be fixed by specific implementations. An application should assume that the value supplied by **<limits.h>** in a specific implementation is the minimum that pertains whenever the application is run under that implementation. A specific instance of a specific implementation may increase the value relative to that supplied by **<limits.h>** for that implementation. The actual value supported by a specific instance shall be provided by the *sysconf*( ) function.

{BC_BASE_MAX}
Maximum *obase* values allowed by the *bc* utility.
Minimum Acceptable Value: {_POSIX2_BC_BASE_MAX}

{BC_DIM_MAX}
Maximum number of elements permitted in an array by the *bc* utility.
Minimum Acceptable Value: {_POSIX2_BC_DIM_MAX}

{BC_SCALE_MAX}
Maximum *scale* value allowed by the *bc* utility.
Minimum Acceptable Value: {_POSIX2_BC_SCALE_MAX}

{BC_STRING_MAX}
Maximum length of a string constant accepted by the *bc* utility.
Minimum Acceptable Value: {_POSIX2_BC_STRING_MAX}

{CHARCLASS_NAME_MAX}
Maximum number of bytes in a character class name.
Minimum Acceptable Value: {_POSIX2_CHARCLASS_NAME_MAX}

{COLL_WEIGHTS_MAX}
Maximum number of weights that can be assigned to an entry of the *LC_COLLATE* **order** keyword in the locale definition file; see Chapter 7 (on page 119).
Minimum Acceptable Value: {_POSIX2_COLL_WEIGHTS_MAX}

{EXPR_NEST_MAX}
Maximum number of expressions that can be nested within parentheses by the *expr* utility.
Minimum Acceptable Value: {_POSIX2_EXPR_NEST_MAX}

{LINE_MAX}
Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing newline.
Minimum Acceptable Value: {_POSIX2_LINE_MAX}

{NGROUPS_MAX}
Maximum number of simultaneous supplementary group IDs per process.
Minimum Acceptable Value: {_POSIX_NGROUPS_MAX}

{RE_DUP_MAX}
Maximum number of repeated occurrences of a regular expression permitted when using the interval notation \{*m*,*n*\}; see Chapter 9 (on page 165).
Minimum Acceptable Value: {_POSIX2_RE_DUP_MAX}

Maximum Values

TMR The symbolic constants in the following list shall be defined in **<limits.h>** with the values shown. These are symbolic names for the most restrictive value for certain features on an implementation supporting the Timers option. A conforming implementation shall provide values no larger than these values. A conforming application must not require a smaller value for correct operation. |

TMR {_POSIX_CLOCKRES_MIN}
The resolution of the CLOCK_REALTIME clock, in nanoseconds.
Value: 20 000 000

MON If the Monotonic Clock option is supported, the resolution of the CLOCK_MONOTONIC clock, in nanoseconds, is represented by {_POSIX_CLOCKRES_MIN}.

Minimum Values

The symbolic constants in the following list shall be defined in **<limits.h>** with the values shown. These are symbolic names for the most restrictive value for certain features on an implementation conforming to this volume of IEEE Std 1003.1-200x. Related symbolic constants are defined elsewhere in this volume of IEEE Std 1003.1-200x which reflect the actual implementation and which need not be as restrictive. A conforming implementation shall provide values at least this large. A strictly conforming application must not require a larger value for correct operation.

AIO {_POSIX_AIO_LISTIO_MAX}
The number of I/O operations that can be specified in a list I/O call.
Value: 2

AIO {_POSIX_AIO_MAX}
The number of outstanding asynchronous I/O operations.
Value: 1

{_POSIX_ARG_MAX}
Maximum length of argument to the *exec* functions including environment data.
Value: 4 096

{_POSIX_CHILD_MAX}
Maximum number of simultaneous processes per real user ID.
Value: 6

TMR {_POSIX_DELAYTIMER_MAX}
The number of timer expiration overruns.
Value: 32 |

{_POSIX_HOST_NAME_MAX} |
Maximum length of a host name (not including the terminating null) as returned from the *gethostname*( ) function. |
Value: 255 |

{_POSIX_LINK_MAX}
Maximum number of links to a single file.
Value: 8

{_POSIX_LOGIN_NAME_MAX}
The size of the storage required for a login name, in bytes, including the terminating null.
Value: 9

{_POSIX_MAX_CANON}
Maximum number of bytes in a terminal canonical input queue.
Value: 255

{_POSIX_MAX_INPUT}
Maximum number of bytes allowed in a terminal input queue.
Value: 255

MSG {_POSIX_MQ_OPEN_MAX}
The number of message queues that can be open for a single process.
Value: 8

MSG {_POSIX_MQ_PRIO_MAX}
The maximum number of message priorities supported by the implementation.
Value: 32

{_POSIX_NAME_MAX}
Maximum number of bytes in a filename (not including terminating null).
Value: 14

{_POSIX_NGROUPS_MAX}
Maximum number of simultaneous supplementary group IDs per process.
Value: 8

{_POSIX_OPEN_MAX}
Maximum number of files that one process can have open at any one time.
Value: 20

{_POSIX_PATH_MAX}
Maximum number of bytes in a pathname. |
Value: 256

{_POSIX_PIPE_BUF}
Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.
Value: 512

{_POSIX_RE_DUP_MAX}
The number of repeated occurrences of a BRE permitted by the *regexec*( ) and *regcomp*( ) functions when using the interval notation {\(*m*,*n*\}; see Section 9.3.6 (on page 170).
Value: 255

RTS {_POSIX_RTSIG_MAX}
The number of realtime signal numbers reserved for application use.
Value: 8

SEM {_POSIX_SEM_NSEMS_MAX}
The number of semaphores that a process may have.
Value: 256

SEM {_POSIX_SEM_VALUE_MAX}
The maximum value a semaphore may have.
Value: 32 767

RTS {_POSIX_SIGQUEUE_MAX}
The number of queued signals that a process may send and have pending at the receiver(s) at any time.
Value: 32

{_POSIX_SSIZE_MAX}
The value that can be stored in an object of type **ssize_t**.
Value: 32 767

{_POSIX_STREAM_MAX}
The number of streams that one process can have open at one time.
Value: 8

SS|TSP {_POSIX_SS_REPL_MAX}
The number of replenishment operations that may be simultaneously pending for a particular sporadic server scheduler.
Value: 4

{_POSIX_SYMLINK_MAX}
The number of bytes in a symbolic link.
Value: 255

{_POSIX_SYMLOOP_MAX}
The number of symbolic links that can be traversed in the resolution of a pathname in the absence of a loop. |
Value: 8

THR {_POSIX_THREAD_DESTRUCTOR_ITERATIONS}
The number of attempts made to destroy a thread's thread-specific data values on thread exit.
Value: 4

THR {_POSIX_THREAD_KEYS_MAX}
The number of data keys per process.
Value: 128

THR {_POSIX_THREAD_THREADS_MAX}
The number of threads per process.
Value: 64

TMR {_POSIX_TIMER_MAX}
The per process number of timers.
Value: 32

TRC {_POSIX_TRACE_EVENT_NAME_MAX}
The length in bytes of a trace event name.
Value: 30

TRC {_POSIX_TRACE_NAME_MAX}
The length in bytes of a trace generation version string or a trace stream name.
Value: 8

TRC {_POSIX_TRACE_SYS_MAX}
The number of trace streams that may simultaneously exist in the system.
Value: 8

TRC {_POSIX_TRACE_USER_EVENT_MAX}
The number of user trace event type identifiers that may simultaneously exist in a traced process, including the predefined user trace event POSIX_TRACE_UNNAMED_USER_EVENT.
Value: 32

{_POSIX_TTY_NAME_MAX}
The size of the storage required for a terminal device name, in bytes, including the

terminating null.
Value: 9

{_POSIX_TZNAME_MAX}
Maximum number of bytes supported for the name of a timezone (not of the TZ variable).
Value: 6

Note: The length given by {_POSIX_TZNAME_MAX} does not include the quoting characters mentioned in Section 8.3 (on page 161).

{_POSIX2_BC_BASE_MAX}
Maximum *obase* values allowed by the *bc* utility.
Value: 99

{_POSIX2_BC_DIM_MAX}
Maximum number of elements permitted in an array by the *bc* utility.
Value: 2 048

{_POSIX2_BC_SCALE_MAX}
Maximum *scale* value allowed by the *bc* utility.
Value: 99

{_POSIX2_BC_STRING_MAX}
Maximum length of a string constant accepted by the *bc* utility.
Value: 1 000

{_POSIX2_CHARCLASS_NAME_MAX}
Maximum number of bytes in a character class name.
Value: 14

{_POSIX2_COLL_WEIGHTS_MAX}
Maximum number of weights that can be assigned to an entry of the *LC_COLLATE* **order** keyword in the locale definition file; see Chapter 7 (on page 119).
Value: 2

{_POSIX2_EXPR_NEST_MAX}
Maximum number of expressions that can be nested within parentheses by the *expr* utility.
Value: 32

{_POSIX2_LINE_MAX}
Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing newline.
Value: 2 048

{_POSIX2_RE_DUP_MAX]
Maximum number of repeated occurrences of a regular expression permitted when using the interval notation \{*m*,*n*\}; see Chapter 9 (on page 165).
Value: 255

XSI {_XOPEN_IOV_MAX}
Maximum number of **iovec** structures that one process has available for use with *readv*( ) or *writev*( ).
Value: 16

XSI {_XOPEN_NAME_MAX}
Maximum number of bytes in a filename (not including terminating null).
Value: 255

{_XOPEN_PATH_MAX}
Maximum number of bytes in a pathname.
Value: 1 024

Numerical Limits

The values in the following lists shall be defined in **<limits.h>** and are constant expressions suitable for use in **#if** preprocessing directives. Moreover, except for {CHAR_BIT}, {DBL_DIG}, {DBL_MAX}, {FLT_DIG}, {FLT_MAX}, {LONG_BIT}, {WORD_BIT}, and {MB_LEN_MAX}, the symbolic names are defined as expressions of the correct type.

If the value of an object of type **char** is treated as a signed integer when used in an expression, the value of {CHAR_MIN} is the same as that of {SCHAR_MIN} and the value of {CHAR_MAX} is the same as that of {SCHAR_MAX}. Otherwise, the value of {CHAR_MIN} is 0 and the value of {CHAR_MAX} is the same as that of {UCHAR_MAX}.

{CHAR_BIT}
Number of bits in a type **char**.
Value: 8

{CHAR_MAX}
Maximum value of type **char**.
Minimum Acceptable Value: {UCHAR_MAX} or {SCHAR_MAX}

{INT_MAX}
Maximum value of an **int**.
Minimum Acceptable Value: 2 147 483 647

{LONG_BIT}
Number of bits in a **long**.
Minimum Acceptable Value: 32

{LONG_MAX}
Maximum value of a **long**.
Minimum Acceptable Value: +2 147 483 647

{MB_LEN_MAX}
Maximum number of bytes in a character, for any supported locale.
Minimum Acceptable Value: 1

{SCHAR_MAX}
Maximum value of type **signed char**.
Value: +127

{SHRT_MAX}
Maximum value of type **short**.
Minimum Acceptable Value: +32 767

{SSIZE_MAX}
Maximum value of an object of type **ssize_t**.
Minimum Acceptable Value: {_POSIX_SSIZE_MAX}

{UCHAR_MAX}
Maximum value of type **unsigned char**.
Value: 255

{UINT_MAX}
Maximum value of type **unsigned**.
Minimum Acceptable Value: 4 294 967 295

{ULONG_MAX}
Maximum value of type **unsigned long**.
Minimum Acceptable Value: 4 294 967 295

{USHRT_MAX}
Maximum value for a type **unsigned short**.
Minimum Acceptable Value: 65 535

XSI {WORD_BIT}
Number of bits in a word or type **int**.
Minimum Acceptable Value: 16

{CHAR_MIN}
Minimum value of type **char**.
Maximum Acceptable Value: {SCHAR_MIN} or 0

{INT_MIN}
Minimum value of type **int**.
Maximum Acceptable Value: –2 147 483 647

{LONG_MIN}
Minimum value of type **long**.
Maximum Acceptable Value: –2 147 483 647

{SCHAR_MIN}
Minimum value of type **signed char**.
CX Value: –128

{SHRT_MIN}
Minimum value of type **short**.
Maximum Acceptable Value: –32 767

{LLONG_MIN}
Minimum value of type **long long**.
Maximum Acceptable Value: –9223372036854775807

{LLONG_MAX}
Maximum value of type **long long**.
Minimum Acceptable Value: +9223372036854775807

{ULLONG_MAX}
Maximum value of type **unsigned long long**.
Minimum Acceptable Value: 18446744073709551615

Other Invariant Values

XSI The following constants shall be defined on all implementations in **<limits.h>**:

XSI {CHARCLASS_NAME_MAX}
Maximum number of bytes in a character class name.
Minimum Acceptable Value: 14

XSI {NL_ARGMAX}
Maximum value of *digit* in calls to the *printf*( ) and *scanf*( ) functions.
Minimum Acceptable Value: 9

XSI {NL_LANGMAX}
Maximum number of bytes in a *LANG* name.
Minimum Acceptable Value: 14

XSI {NL_MSGMAX}
Maximum message number.
Minimum Acceptable Value: 32 767

XSI {NL_NMAX}
Maximum number of bytes in an N-to-1 collation mapping.
Minimum Acceptable Value: ’*’

XSI {NL_SETMAX}
Maximum set number.
Minimum Acceptable Value: 255

XSI {NL_TEXTMAX}
Maximum number of bytes in a message string.
Minimum Acceptable Value: {_POSIX2_LINE_MAX}

XSI {NZERO}
Default process priority.
Minimum Acceptable Value: 20

APPLICATION USAGE

None.

RATIONALE

A request was made to reduce the value of {_POSIX_LINK_MAX} from the value of 8 specified for it in the POSIX.1-1990 standard to 2. The standard developers decided to deny this request for several reasons.

- They wanted to avoid making any changes to the standard that could break conforming applications, and the requested change could have that effect.
- The use of multiple hard links to a file cannot always be replaced with use of symbolic links. Symbolic links are semantically different from hard links in that they associate a pathname with another pathname rather than a pathname with a file. This has implications for access control, file permanence, and transparency.
- The original standard developers had considered the issue of allowing for implementations that did not in general support hard links, and decided that this would reduce consensus on the standard.

Systems that support historical versions of the development option of the ISO POSIX-2 standard retain the name {_POSIX2_RE_DUP_MAX} as an alias for {_POSIX_RE_DUP_MAX}.

{PATH_MAX}
IEEE PASC Interpretation 1003.1 #15 addressed the inconsistency in the standard with the definition of pathname and the description of {PATH_MAX}, allowing application writers to allocate either {PATH_MAX} or {PATH_MAX}+1 bytes. The inconsistency has been removed by correction to the {PATH_MAX} definition to include the null character. With this change, applications that previously allocated {PATH_MAX} bytes will continue to succeed.

{SYMLINK_MAX}
This symbol refers to space for data that is stored in the file system, as opposed to {PATH_MAX} which is the length of a name that can be passed to a function. In some existing implementations, the filenames pointed to by symbolic links are stored in the inodes of the links, so it is important that {SYMLINK_MAX} not be constrained to be as large as {PATH_MAX}.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *fpathconf*( ), *pathconf*( ), *sysconf*( )

CHANGE HISTORY

First released in Issue 1.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

{FILESIZEBITS} added for the Large File Summit extensions.

The minimum acceptable values for {INT_MAX}, {INT_MIN}, and {UINT_MAX} are changed to make 32-bit values the minimum requirement.

The entry is restructured to improve readability.

Issue 6

The Open Group Corrigendum U033/4 is applied. The wording is made clear for {CHAR_MIN}, {INT_MIN}, {LONG_MIN}, {SCHAR_MIN}, and {SHRT_MIN} that these are maximum acceptable values.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The minimum value for {CHILD_MAX} is 25. This is a FIPS requirement.
- The minimum value for {OPEN_MAX} is 20. This is a FIPS requirement.
- The minimum value for {NGROUPS_MAX} is 8. This is also a FIPS requirement.

Symbolic constants are added for {_POSIX_SYMLINK_MAX}, {_POSIX_SYMLOOP_MAX}, {_POSIX_RE_DUP_MAX}, {RE_DUP_MAX}, {SYMLOOP_MAX}, and {SYMLINK_MAX}.

The following values are added for alignment with IEEE Std 1003.1d-1999:

{_POSIX_SS_REPL_MAX}
{SS_REPL_MAX}
{POSIX_ALLOC_SIZE_MIN}
{POSIX_REC_INCR_XFER_SIZE}
{POSIX_REC_MAX_XFER_SIZE}
{POSIX_REC_MIN_XFER_SIZE}
{POSIX_REC_XFER_ALIGN}

Reference to CLOCK_MONOTONIC is added in the description of {_POSIX_CLOCKRES_MIN} for alignment with IEEE Std 1003.1j-2000.

The constants {LLONG_MIN}, {LLONG_MAX}, and {ULLONG_MAX} are added for alignment with the ISO/IEC 9899:1999 standard.

The following values are added for alignment with IEEE Std 1003.1q-2000:

{_POSIX_TRACE_EVENT_NAME_MAX}
{_POSIX_TRACE_NAME_MAX}
{_POSIX_TRACE_SYS_MAX}
{_POSIX_TRACE_USER_EVENT_MAX}
{TRACE_EVENT_NAME_MAX}
{TRACE_NAME_MAX}
{TRACE_SYS_MAX}
{TRACE_USER_EVENT_MAX}

The new limits {_XOPEN_NAME_MAX} and {_XOPEN_PATH_MAX} are added as minimum values for {PATH_MAX} and {NAME_MAX} limits on XSI-conformant systems.

The legacy symbols {PASS_MAX} and {TMP_MAX} are removed.

The values for the limits {CHAR_BIT}, {CHAR_MAX}, {SCHAR_MAX}, and {UCHAR_MAX} are now required to be 8, +127, 255, and −128, respectively.

NAME

locale.h — category macros

SYNOPSIS

```
#include <locale.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-200x defers to the ISO C standard.

The **<locale.h>** header shall provide a definition for structure **lconv**, which shall include at least the following members. (See the definitions of *LC_MONETARY* in the Section 7.3.3 (on page 138), and Section 7.4.4 (on page 141).)

```
char    *currency_symbol
char    *decimal_point
char     frac_digits
char    *grouping
char    *int_curr_symbol
char     int_frac_digits
char     int_n_cs_precedes
char     int_n_sep_by_space
char     int_n_sign_posn
char     int_p_cs_precedes
char     int_p_sep_by_space
char     int_p_sign_posn
char    *mon_decimal_point
char    *mon_grouping
char    *mon_thousands_sep
char    *negative_sign
char     n_cs_precedes
char     n_sep_by_space
char     n_sign_posn
char    *positive_sign
char     p_cs_precedes
char     p_sep_by_space
char     p_sign_posn
char    *thousands_sep
```

The **<locale.h>** header shall define NULL (as defined in **<stddef.h>**) and at least the following as macros:

LC_ALL
LC_COLLATE
LC_CTYPE
CX *LC_MESSAGES*
LC_MONETARY
LC_NUMERIC
LC_TIME

which shall expand to distinct integer constant expressions, for use as the first argument to the *setlocale*( ) function.

Additional macro definitions, beginning with the characters *LC_* and an uppercase letter, may also be given here.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
struct  lconv *localeconv (void);
char   *setlocale(int, const char *);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *localeconv*( ), *setlocale*( ), Chapter 8 (on page 157)

CHANGE HISTORY

First released in Issue 3.

Entry included for alignment with the ISO C standard.

Issue 6

The **lconv** structure is expanded with new members (**int_n_cs_precedes**, **int_n_sep_by_space**, **int_n_sign_posn**, **int_p_cs_precedes**, **int_p_sep_by_space**, and **int_p_sign_posn**) for alignment with the ISO / IEC 9899: 1999 standard.

Extensions beyond the ISO C standard are now marked.

NAME

math.h — mathematical declarations

SYNOPSIS

```
#include <math.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these symbols in this header.

The **<math.h>** header shall include definitions for at least the following types:

float_t A real-floating type at least as wide as **float**.

double_t A real-floating type at least as wide as **double**, and at least as wide as **float_t**.

If FLT_EVAL_METHOD equals 0, **float_t** and **double_t** shall be **float** and **double**, respectively; if FLT_EVAL_METHOD equals 1, they shall both be **double**; if FLT_EVAL_METHOD equals 2, they shall both be **long double**; for other values of FLT_EVAL_METHOD, they are otherwise implementation-defined.

The **<math.h>** header shall define the following macros, where **real-floating** indicates that the argument shall be an expression of real-floating type:

```
int fpclassify(real-floating x);
int isfinite(real-floating x);
int isinf(real-floating x);
int isnan(real-floating x);
int isnormal(real-floating x);
int signbit(real-floating x);
int isgreater(real-floating x, real-floating y);
int isgreaterequal(real-floating x, real-floating y);
int isless(real-floating x, real-floating y);
int islessequal(real-floating x, real-floating y);
int islessgreater(real-floating x, real-floating y);
int isunordered(real-floating x, real-floating y);
```

The **<math.h>** header shall provide for the following constants. The values are of type **double** and are accurate within the precision of the **double** type.

XSI

M_E	Value of e
M_LOG2E	Value of $\log_2 e$
M_LOG10E	Value of $\log_{10} e$
M_LN2	Value of $\log_e 2$
M_LN10	Value of $\log_e 10$
M_PI	Value of π
M_PI_2	Value of $\pi/2$
M_PI_4	Value of $\pi/4$
M_1_PI	Value of $1/\pi$
M_2_PI	Value of $2/\pi$

M_2_SQRTPI Value of $2/\sqrt{\pi}$

M_SQRT2 Value of $\sqrt{2}$

M_SQRT1_2 Value of $1/\sqrt{2}$

The header shall define the following symbolic constants:

XSI MAXFLOAT Value of maximum non-infinite single-precision floating-point number.

HUGE_VAL A positive **double** expression, not necessarily representable as a **float**. Used as an error value returned by the mathematics library. HUGE_VAL evaluates to +infinity on systems supporting IEEE Std 754-1985.

HUGE_VALF A positive **float** constant expression. Used as an error value returned by the mathematics library. HUGE_VALF evaluates to +infinity on systems supporting IEEE Std 754-1985.

HUGE_VALL A positive **long double** constant expression. Used as an error value returned by the mathematics library. HUGE_VALL evaluates to +infinity on systems supporting IEEE Std 754-1985.

INFINITY A constant expression of type **float** representing positive or unsigned infinity, if available; else a positive constant of type **float** that overflows at translation time.

NAN A constant expression of type **float** representing a quiet NaN. This symbolic constant is only defined if the implementation supports quiet NaNs for the **float** type.

The following macros shall be defined for number classification. They represent the mutually-exclusive kinds of floating-point values. They expand to integer constant expressions with distinct values. Additional implementation-defined floating-point classifications, with macro definitions beginning with FP_ and an uppercase letter, may also be specified by the implementation.

FP_INFINITE
FP_NAN
FP_NORMAL
FP_SUBNORMAL
FP_ZERO

The following optional macros indicate whether the *fma*( ) family of functions are fast compared with direct code:

FP_FAST_FMA
FP_FAST_FMAF
FP_FAST_FMAL

The FP_FAST_FMA macro shall be defined to indicate that the *fma*( ) function generally executes about as fast as, or faster than, a multiply and an add of **double** operands. The other macros have the equivalent meaning for the **float** and **long double** versions.

The following macros shall expand to integer constant expressions whose values are returned by *ilogb*(*x*) if *x* is zero or NaN, respectively. The value of FP_ILOGB0 shall be either {INT_MIN} or −{INT_MAX}. The value of FP_ILOGBNAN shall be either {INT_MAX} or {INT_MIN}.

FP_ILOGB0
FP_ILOGBNAN

The following macros shall expand to the integer constants 1 and 2, respectively;

MATH_ERRNO
MATH_ERREXCEPT

The following macro shall expand to an expression that has type **int** and the value MATH_ERRNO, MATH_ERREXCEPT, or the bitwise-inclusive OR of both:

math_errhandling

The value of math_errhandling is constant for the duration of the program. It is unspecified whether math_errhandling is a macro or an identifier with external linkage. If a macro definition is suppressed or a program defines an identifier with the name math_errhandling , the behavior is undefined. If the expression (math_errhandling & MATH_ERREXCEPT) can be non-zero, the implementation shall define the macros FE_DIVBYZERO, FE_INVALID, and FE_OVERFLOW in **<fenv.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
double      acos(double);
float       acosf(float);
double      acosh(double);
float       acoshf(float);
long double acoshl(long double);
long double acosl(long double);
double      asin(double);
float       asinf(float);
double      asinh(double);
float       asinhf(float);
long double asinhl(long double);
long double asinl(long double);
double      atan(double);
double      atan2(double, double);
float       atan2f(float, float);
long double atan2l(long double, long double);
float       atanf(float);
double      atanh(double);
float       atanhf(float);
long double atanhl(long double);
long double atanl(long double);
double      cbrt(double);
float       cbrtf(float);
long double cbrtl(long double);
double      ceil(double);
float       ceilf(float);
long double ceill(long double);
double      copysign(double, double);
float       copysignf(float, float);
long double copysignl(long double, long double);
double      cos(double);
float       cosf(float);
double      cosh(double);
float       coshf(float);
long double coshl(long double);
```

```
        long double cosl(long double);
        double      erf(double);
        double      erfc(double);
        float       erfcf(float);
        long double erfcl(long double);
        float       erff(float);
        long double erfl(long double);
        double      exp(double);
        double      exp2(double);
        float       exp2f(float);
        long double exp2l(long double);
        float       expf(float);
        long double expl(long double);
        double      expm1(double);
        float       expm1f(float);
        long double expm1l(long double);
        double      fabs(double);
        float       fabsf(float);
        long double fabsl(long double);
        double      fdim(double, double);
        float       fdimf(float, float);
        long double fdiml(long double, long double);
        double      floor(double);
        float       floorf(float);
        long double floorl(long double);
        double      fma(double, double, double);
        float       fmaf(float, float, float);
        long double fmal(long double, long double, long double);
        double      fmax(double, double);
        float       fmaxf(float, float);
        long double fmaxl(long double, long double);
        double      fmin(double, double);
        float       fminf(float, float);
        long double fminl(long double, long double);
        double      fmod(double, double);
        float       fmodf(float, float);
        long double fmodl(long double, long double);
        double      frexp(double, int *);
        float       frexpf(float value, int *);
        long double frexpl(long double value, int *);
        double      hypot(double, double);
        float       hypotf(float, float);
        long double hypotl(long double, long double);
        int         ilogb(double);
        int         ilogbf(float);
        int         ilogbl(long double);
XSI     double      j0(double);
        double      j1(double);
        double      jn(int, double);
        double      ldexp(double, int);
        float       ldexpf(float, int);
        long double ldexpl(long double, int);
```

```
double      lgamma(double);
float       lgammaf(float);
long double lgammal(long double);
long long   llrint(double);
long long   llrintf(float);
long long   llrintl(long double);
long long   llround(double);
long long   llroundf(float);
long long   llroundl(long double);
double      log(double);
double      log10(double);
float       log10f(float);
long double log10l(long double);
double      log1p(double);
float       log1pf(float);
long double log1pl(long double);
double      log2(double);
float       log2f(float);
long double log2l(long double);
double      logb(double);
float       logbf(float);
long double logbl(long double);
float       logf(float);
long double logl(long double);
long        lrint(double);
long        lrintf(float);
long        lrintl(long double);
long        lround(double);
long        lroundf(float);
long        lroundl(long double);
double      modf(double, double *);
float       modff(float, float *);
long double modfl(long double, long double *);
double      nan(const char *);
float       nanf(const char *);
long double nanl(const char *);
double      nearbyint(double);
float       nearbyintf(float);
long double nearbyintl(long double);
double      nextafter(double, double);
float       nextafterf(float, float);
long double nextafterl(long double, long double);
double      nexttoward(double, long double);
float       nexttowardf(float, long double);
long double nexttowardl(long double, long double);
double      pow(double, double);
float       powf(float, float);
long double powl(long double, long double);
double      remainder(double, double);
float       remainderf(float, float);
long double remainderl(long double, long double);
double      remquo(double, double, int *);
```

```
    float       remquof(float, float, int *);
    long double remquol(long double, long double, int *);
    double      rint(double);
    float       rintf(float);
    long double rintl(long double);
    double      round(double);
    float       roundf(float);
    long double roundl(long double);
XSI double      scalb(double, double);
    double      scalbln(double, long);
    float       scalblnf(float, long);
    long double scalblnl(long double, long);
    double      scalbn(double, int);
    float       scalbnf(float, int);
    long double scalbnl(long double, int);
    double      sin(double);
    float       sinf(float);
    double      sinh(double);
    float       sinhf(float);
    long double sinhl(long double);
    long double sinl(long double);
    double      sqrt(double);
    float       sqrtf(float);
    long double sqrtl(long double);
    double      tan(double);
    float       tanf(float);
    double      tanh(double);
    float       tanhf(float);
    long double tanhl(long double);
    long double tanl(long double);
    double      tgamma(double);
    float       tgammaf(float);
    long double tgammal(long double);
    double      trunc(double);
    float       truncf(float);
    long double truncl(long double);
XSI double      y0(double);
    double      y1(double);
    double      yn(int, double);
```

The following external variable shall be defined:

```
XSI extern int signgam;
```

The behavior of each of the functions defined in **<math.h>** is specified in the System Interfaces volume of IEEE Std 1003.1-200x for all representable values of its input arguments, except where stated otherwise. Each function shall execute as if it were a single operation without generating any externally visible exceptional conditions.

APPLICATION USAGE

The FP_CONTRACT pragma can be used to allow (if the state is on) or disallow (if the state is off) the implementation to contract expressions. Each pragma can occur either outside external declarations or preceding all explicit declarations and statements inside a compound statement. When outside external declarations, the pragma takes effect from its occurrence until another FP_CONTRACT pragma is encountered, or until the end of the translation unit. When inside a compound statement, the pragma takes effect from its occurrence until another FP_CONTRACT pragma is encountered (including within a nested compound statement), or until the end of the compound statement; at the end of a compound statement the state for the pragma is restored to its condition just before the compound statement. If this pragma is used in any other context, the behavior is undefined. The default state (on or off) for the pragma is implementation-defined.

RATIONALE

Before the ISO / IEC 9899: 1999 standard, the math library was defined only for the floating type **double**. All the names formed by appending `'f'` or `'l'` to a name in **<math.h>** were reserved to allow for the definition of **float** and **long double** libraries; and the ISO / IEC 9899: 1999 standard provides for all three versions of math functions.

The functions *ecvt*( ), *fcvt*( ), and *gcvt*( ) have been dropped from the ISO C standard since their capability is available through *sprintf*( ). These are provided on XSI-conformant systems supporting the Legacy Option Group.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *acos*( ), *acosh*( ), *asin*( ), *atan*( ), *atan2*( ), *cbrt*( ), *ceil*( ), *cos*( ), *cosh*( ), *erf*( ), *exp*( ), *expm1*( ), *fabs*( ), *floor*( ), *fmod*( ), *frexp*( ), *hypot*( ), *ilogb*( ), *isnan*( ), *j0*( ), *ldexp*( ), *lgamma*( ), *log*( ), *log10*( ), *log1p*( ), *logb*( ), *modf*( ), *nextafter*( ), *pow*( ), *remainder*( ), *rint*( ), *scalb*( ), *sin*( ), *sinh*( ), *sqrt*( ), *tan*( ), *tanh*( ), *y0*( )

CHANGE HISTORY

First released in Issue 1.

Issue 6

This reference page is updated to align with the ISO / IEC 9899: 1999 standard.

NAME

monetary.h — monetary types

SYNOPSIS

XSI

```
#include <monetary.h>
```

DESCRIPTION

The **<monetary.h>** header shall define the following types:

size_t As described in **<stddef.h>**.

ssize_t As described in **<sys/types.h>**.

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided. |

```
ssize_t  strfmon(char *restrict, size_t, const char *restrict, ...);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *strfmon*( )

CHANGE HISTORY

First released in Issue 4.

Issue 6

The **restrict** keyword is added to the prototype for *strfmon*( ).

NAME

mqueue.h — message queues (**REALTIME**)

SYNOPSIS

MSG
```
#include <mqueue.h>
```

DESCRIPTION

The **<mqueue.h>** header shall define the **mqd_t** type, which is used for message queue descriptors. This is not an array type.

The **<mqueue.h>** header shall define the **sigevent** structure (as described in **<signal.h>**) and the **mq_attr** structure, which is used in getting and setting the attributes of a message queue. Attributes are initially set when the message queue is created. An **mq_attr** structure shall have at least the following fields:

`long`	`mq_flags`	Message queue flags.
`long`	`mq_maxmsg`	Maximum number of messages.
`long`	`mq_msgsize`	Maximum message size.
`long`	`mq_curmsgs`	Number of messages currently queued.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided. |

```
int      mq_close(mqd_t);
int      mq_getattr(mqd_t, struct mq_attr *);
int      mq_notify(mqd_t, const struct sigevent *);
mqd_t    mq_open(const char *, int, ...);
ssize_t  mq_receive(mqd_t, char *, size_t, unsigned *);
int      mq_send(mqd_t, const char *, size_t, unsigned );
int      mq_setattr(mqd_t, const struct mq_attr *restrict,
             struct mq_attr *restrict);
```

TMO
```
ssize_t  mq_timedreceive(mqd_t, char *restrict, size_t,
             unsigned *restrict, const struct timespec *restrict);
int      mq_timedsend(mqd_t, const char *, size_t, unsigned ,
             const struct timespec *);
```

```
int      mq_unlink(const char *);
```

Inclusion of the **<mqueue.h>** header may make visible symbols defined in the headers **<fcntl.h>**, **<signal.h>**, **<sys/types.h>**, and **<time.h>**.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<fcntl.h>, **<signal.h>**, **<sys/types.h>**, **<time.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *mq_close*( ), *mq_getattr*( ), *mq_notify*( ), *mq_open*( ), *mq_receive*( ), *mq_send*( ), *mq_setattr*( ), *mq_timedreceive*( ), *mq_timedsend*( ), *mq_unlink*( )

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The **<mqueue.h>** header is marked as part of the Message Passing option.

The *mq_timedreceive*( ) and *mq_timedsend*( ) functions are added for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the prototypes for *mq_setattr*( ) and *mq_timedreceive*( ).

NAME

ndbm.h — definitions for ndbm database operations

SYNOPSIS

XSI
```
#include <ndbm.h>
```

DESCRIPTION

The **<ndbm.h>** header shall define the **datum** type as a structure that includes at least the following members:

```
void    *dptr   A pointer to the application's data.
size_t  dsize   The size of the object pointed to by dptr.
```

The **size_t** type shall be defined as described in **<stddef.h>**.

The **<ndbm.h>** header shall define the **DBM** type.

The following constants shall be defined as possible values for the *store_mode* argument to *dbm_store*( ):

DBM_INSERT	Insertion of new entries only.
DBM_REPLACE	Allow replacing existing entries.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided. |

```
int     dbm_clearerr(DBM *);
void    dbm_close(DBM *);
int     dbm_delete(DBM *, datum);
int     dbm_error(DBM *);
datum   dbm_fetch(DBM *, datum);
datum   dbm_firstkey(DBM *);
datum   dbm_nextkey(DBM *);
DBM    *dbm_open(const char *, int, mode_t);
int     dbm_store(DBM *, datum, datum, int);
```

The **mode_t** type shall be defined through **typedef** as described in **<sys/types.h>**.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *dbm_clearerr*( )

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

References to the definitions of **size_t** and **mode_t** are added to the DESCRIPTION.

NAME

net/if.h — sockets local interfaces

SYNOPSIS

```
#include <net/if.h>
```

DESCRIPTION

The **<net/if.h>** header shall define the **if_nameindex** structure that includes at least the following members:

```
unsigned  if_index   Numeric index of the interface.
char     *if_name    Null-terminated name of the interface.
```

The **<net/if.h>** header shall define the following macro for the length of a buffer containing an interface name (including the terminating NULL character):

IF_NAMESIZE   Interface name length.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
unsigned              if_nametoindex(const char *);
char                 *if_indextoname(unsigned, char *);
struct if_nameindex  *if_nameindex(void);
void                  if_freenameindex(struct if_nameindex *);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *if_freenameindex*( ), *if_indextoname*( ), *if_nameindex*( ), *if_nametoindex*( )

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

NAME

netdb.h — definitions for network database operations

SYNOPSIS

```
#include <netdb.h>
```

DESCRIPTION

The **<netdb.h>** header may define the **in_port_t** type and the **in_addr_t** type as described in **<netinet/in.h>**.

The **<netdb.h>** header shall define the **hostent** structure that includes at least the following members:

`char`	`*h_name`	Official name of the host.
`char`	`**h_aliases`	A pointer to an array of pointers to alternative host names, terminated by a null pointer.
`int`	`h_addrtype`	Address type.
`int`	`h_length`	The length, in bytes, of the address.
`char`	`**h_addr_list`	A pointer to an array of pointers to network addresses (in network byte order) for the host, terminated by a null pointer.

The **<netdb.h>** header shall define the **netent** structure that includes at least the following members:

`char`	`*n_name`	Official, fully-qualified (including the domain) name of the host.
`char`	`**n_aliases`	A pointer to an array of pointers to alternative network names, terminated by a null pointer.
`int`	`n_addrtype`	The address type of the network.
`uint32_t`	`n_net`	The network number, in host byte order.

The **uint32_t** type shall be defined as described in **<inttypes.h>**.

The **<netdb.h>** header shall define the **protoent** structure that includes at least the following members:

`char`	`*p_name`	Official name of the protocol.
`char`	`**p_aliases`	A pointer to an array of pointers to alternative protocol names, terminated by a null pointer.
`int`	`p_proto`	The protocol number.

The **<netdb.h>** header shall define the **servent** structure that includes at least the following members:

`char`	`*s_name`	Official name of the service.
`char`	`**s_aliases`	A pointer to an array of pointers to alternative service names, terminated by a null pointer.
`int`	`s_port`	The port number at which the service resides, in network byte order.
`char`	`*s_proto`	The name of the protocol to use when contacting the service.

The **<netdb.h>** header shall define the IPPORT_RESERVED macro with the value of the highest reserved Internet port number.

OB When the **<netdb.h>** header is included, *h_errno* shall be available as a modifiable *l*-value of type **int**. It is unspecified whether *h_errno* is a macro or an identifier declared with external linkage.

The **<netdb.h>** header shall define the following macros for use as error values for *gethostbyaddr*( ) and *gethostbyname*( ):

HOST_NOT_FOUND
NO_DATA
NO_RECOVERY
TRY_AGAIN

Address Information Structure

The **<netdb.h>** header shall define the **addrinfo** structure that includes at least the following members:

`int`	`ai_flags`	Input flags.
`int`	`ai_family`	Address family of socket.
`int`	`ai_socktype`	Socket type.
`int`	`ai_protocol`	Protocol of socket.
`socklen_t`	`ai_addrlen`	Length of socket address.
`struct sockaddr`	`*ai_addr`	Socket address of socket.
`char`	`*ai_canonname`	Canonical name of service location.
`struct addrinfo`	`*ai_next`	Pointer to next in list.

The **<netdb.h>** header shall define the following macros that evaluate to bitwise-distinct integer constants for use in the *flags* field of the **addrinfo** structure:

AI_PASSIVE Socket address is intended for *bind*( ).

AI_CANONNAME
Request for canonical name.

AI_NUMERICHOST
Return numeric host address as name.

AI_NUMERICSERV
Inhibit service name resolution.

AI_V4MAPPED
If no IPv6 addresses are found, query for IPv4 addresses and return them to the caller as IPv4-mapped IPv6 addresses.

AI_ALL Query for both IPv4 and IPv6 addresses.

AI_ADDRCONFIG
Query for IPv4 addresses only when an IPv4 address is configured; query for IPv6 addresses only when an IPv6 address is configured.

The **<netdb.h>** header shall define the following macros that evaluate to bitwise-distinct integer constants for use in the *flags* argument to *getnameinfo*( ):

NI_NOFQDN Only the nodename portion of the FQDN is returned for local hosts.

NI_NUMERICHOST
The numeric form of the node's address is returned instead of its name.

NI_NAMEREQD Return an error if the node's name cannot be located in the database.

NI_NUMERICSERV
The numeric form of the service address is returned instead of its name.

NI_DGRAM Indicates that the service is a datagram service (SOCK_DGRAM).

Address Information Errors

The **<netdb.h>** header shall define the following macros for use as error values for *getaddrinfo*( ) and *getnameinfo*( ):

EAI_AGAIN The name could not be resolved at this time. Future attempts may succeed.

EAI_BADFLAGS The flags had an invalid value.

EAI_FAIL A non-recoverable error occurred.

EAI_FAMILY The address family was not recognized or the address length was invalid for the specified family.

EAI_MEMORY There was a memory allocation failure.

EAI_NONAME The name does not resolve for the supplied parameters.

NI_NAMEREQD is set and the host's name cannot be located, or both *nodename* and *servname* were null.

EAI_SERVICE The service passed was not recognized for the specified socket type.

EAI_SOCKTYPE The intended socket type was not recognized.

EAI_SYSTEM A system error occurred. The error code can be found in *errno*.

EAI_OVERFLOW An argument buffer overflowed.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
void              endhostent(void);
void              endnetent(void);
void              endprotoent(void);
void              endservent(void);
void              freeaddrinfo(struct addrinfo *);
const char       *gai_strerror(int);
int               getaddrinfo(const char *restrict, const char *restrict,
                      const struct addrinfo *restrict,
                      struct addrinfo **restrict);
struct hostent   *gethostbyaddr(const void *, socklen_t, int);
struct hostent   *gethostbyname(const char *);
struct hostent   *gethostent(void);
int               getnameinfo(const struct sockaddr *restrict, socklen_t,
                      char *restrict, socklen_t, char *restrict,
                      socklen_t, unsigned);
struct netent    *getnetbyaddr(uint32_t, int);
struct netent    *getnetbyname(const char *);
struct netent    *getnetent(void);
struct protoent  *getprotobyname(const char *);
struct protoent  *getprotobynumber(int);
struct protoent  *getprotoent(void);
```

```
        struct servent   *getservbyname(const char *, const char *);
        struct servent   *getservbyport(int, const char *);
        struct servent   *getservent(void);
        void              sethostent(int);
        void              setnetent(int);
        void              setprotoent(int);
        void              setservent(int);
```

The type **socklen_t** shall be defined through **typedef** as described in **<sys/socket.h>**.

Inclusion of the **<netdb.h>** header may also make visible all symbols from **<netinet/in.h>**, **<sys/socket.h>**, and **<inttypes.h>**.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<netinet/in.h>, **<inttypes.h>**, **<sys/socket.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *bind*( ), *endhostent*( ), *endnetent*( ), *endprotoent*( ), *endservent*( ), *getaddrinfo*( ), *getnameinfo*( )

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The Open Group Base Resolution bwg2001-009 is applied, which changes the return type for *gai_strerror*( ) from **char *** to **const char ***. This is for coordination with the IPnG Working Group.

NAME

netinet/in.h — Internet address family

SYNOPSIS

```
#include <netinet/in.h>
```

DESCRIPTION

The **<netinet/in.h>** header shall define the following types:

in_port_t Equivalent to the type **uint16_t** as defined in **<inttypes.h>**.

in_addr_t Equivalent to the type **uint32_t** as defined in **<inttypes.h>**.

The **sa_family_t** type shall be defined as described in **<sys/socket.h>**.

The **uint8_t** and **uint32_t** type shall be defined as described in **<inttypes.h>**. Inclusion of the **<netinet/in.h>** header may also make visible all symbols from **<inttypes.h>** and **<sys/socket.h>**.

The **<netinet/in.h>** header shall define the **in_addr** structure that includes at least the following member:

```
in_addr_t  s_addr
```

The **<netinet/in.h>** header shall define the **sockaddr_in** structure that includes at least the following members (all in network byte order):

`sa_family_t`	`sin_family`	AF_INET.
`in_port_t`	`sin_port`	Port number.
`struct in_addr`	`sin_addr`	IP address.

The **sockaddr_in** structure is used to store addresses for the Internet address family. Values of this type shall be cast by applications to **struct sockaddr** for use with socket functions.

IP6 The **<netinet/in.h>** header shall define the **in6_addr** structure that contains at least the following member:

```
uint8_t s6_addr[16]
```

This array is used to contain a 128-bit IPv6 address, stored in network byte order.

The **<netinet/in.h>** header shall define the **sockaddr_in6** structure that includes at least the following members (all in network byte order):

`sa_family_t`	`sin6_family`	AF_INET6.
`in_port_t`	`sin6_port`	Port number.
`uint32_t`	`sin6_flowinfo`	IPv6 traffic class and flow information.
`struct in6_addr`	`sin6_addr`	IPv6 address.
`uint32_t`	`sin6_scope_id`	Set of interfaces for a scope.

The **sockaddr_in6** structure shall be set to zero by an application prior to using it, since implementations are free to have additional, implementation-defined fields in **sockaddr_in6**.

The *sin6_scope_id* field is a 32-bit integer that identifies a set of interfaces as appropriate for the scope of the address carried in the *sin6_addr* field. For a link scope *sin6_addr*, *sin6_scope_id* would be an interface index. For a site scope *sin6_addr*, *sin6_scope_id* would be a site identifier. The mapping of *sin6_scope_id* to an interface or set of interfaces is implementation-defined.

The **<netinet/in.h>** header shall declare the following external variable:

```
struct in6_addr in6addr_any
```

This variable is initialized by the system to contain the wildcard IPv6 address. The **<netinet/in.h>** header also defines the IN6ADDR_ANY_INIT macro. This macro must be constant at compile time and can be used to initialize a variable of type **struct in6_addr** to the IPv6 wildcard address.

The **<netinet/in.h>** header shall declare the following external variable:

```
struct in6_addr in6addr_loopback
```

This variable is initialized by the system to contain the loopback IPv6 address. The **<netinet/in.h>** header also defines the IN6ADDR_LOOPBACK_INIT macro. This macro must be constant at compile time and can be used to initialize a variable of type **struct in6_addr** to the IPv6 loopback address.

The **<netinet/in.h>** header shall define the **ipv6_mreq** structure that includes at least the following members:

```
struct in6_addr  ipv6mr_multiaddr   IPv6 multicast address.
unsigned         ipv6mr_interface   Interface index.
```

The **<netinet/in.h>** header shall define the following macros for use as values of the *level* argument of *getsockopt*( ) and *setsockopt*( ):

	IPPROTO_IP	Internet protocol.
IP6	IPPROTO_IPV6	Internet Protocol Version 6.
	IPPROTO_ICMP	Control message protocol.
RS	IPPROTO_RAW	Raw IP Packets Protocol.
	IPPROTO_TCP	Transmission control protocol.
	IPPROTO_UDP	User datagram protocol.

The **<netinet/in.h>** header shall define the following macros for use as destination addresses for *connect*( ), *sendmsg*( ), and *sendto*( ):

INADDR_ANY	IPv4 local host address.
INADDR_BROADCAST	IPv4 broadcast address.

The **<netinet/in.h>** header shall define the following macro to help applications declare buffers of the proper size to store IPv4 addresses in string form:

INET_ADDRSTRLEN	16. Length of the string form for IP.

The *htonl*( ), *htons*( ), *ntohl*( ), and *ntohs*( ) functions shall be available as defined in **<arpa/inet.h>**. Inclusion of the **<netinet/in.h>** header may also make visible all symbols from **<arpa/inet.h>**.

IP6 The **<netinet/in.h>** header shall define the following macro to help applications declare buffers of the proper size to store IPv6 addresses in string form:

INET6_ADDRSTRLEN	46. Length of the string form for IPv6.

The **<netinet/in.h>** header shall define the following macros, with distinct integer values, for use in the *option_name* argument in the *getsockopt*( ) or *setsockopt*( ) functions at protocol level IPPROTO_IPV6:

IPV6_JOIN_GROUP	Join a multicast group.
IPV6_LEAVE_GROUP	Quit a multicast group.
IPV6_MULTICAST_HOPS	Multicast hop limit.

IPV6_MULTICAST_IF Interface to use for outgoing multicast packets.

IPV6_MULTICAST_LOOP
Multicast packets are delivered back to the local application.

IPV6_UNICAST_HOPS Unicast hop limit.

IPV6_V6ONLY Restrict AF_INET6 socket to IPv6 communications only.

The **<netinet/in.h>** header shall define the following macros that test for special IPv6 addresses. Each macro is of type **int** and takes a single argument of type **const struct in6_addr ***:

IN6_IS_ADDR_UNSPECIFIED
Unspecified address.

IN6_IS_ADDR_LOOPBACK
Loopback address.

IN6_IS_ADDR_MULTICAST
Multicast address.

IN6_IS_ADDR_LINKLOCAL
Unicast link-local address.

IN6_IS_ADDR_SITELOCAL
Unicast site-local address.

IN6_IS_ADDR_V4MAPPED
IPv4 mapped address.

IN6_IS_ADDR_V4COMPAT
IPv4-compatible address.

IN6_IS_ADDR_MC_NODELOCAL
Multicast node-local address.

IN6_IS_ADDR_MC_LINKLOCAL
Multicast link-local address.

IN6_IS_ADDR_MC_SITELOCAL
Multicast site-local address.

IN6_IS_ADDR_MC_ORGLOCAL
Multicast organization-local address.

IN6_IS_ADDR_MC_GLOBAL
Multicast global address.

IN6_IS_ADDR_LINKLOCAL and IN6_IS_ADDR_SITELOCAL return true only for the two local-use IPv6 unicast addresses. They do not return true for multicast addresses of either link-local or site-local scope.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 4.8 (on page 97), **<arpa/inet.h>**, **<inttypes.h>**, **<sys/socket.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *connect*( ), *getsockopt*( ), *htonl*( ), *htons*( ), *ntohl*( ), *ntohs*( ), *sendmsg*( ), *sendto*( ), *setsockopt*( )

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The *sin_zero* member was removed from the **sockaddr_in** structure as per The Open Group Base Resolution bwg2001-004.

NAME

netinet / tcp.h — definitions for the Internet Transmission Control Protocol (TCP)

SYNOPSIS

```
#include <netinet/tcp.h>
```

DESCRIPTION

The **<netinet/tcp.h>** header shall define the following macro for use as a socket option at the IPPROTO_TCP level:

TCP_NODELAY Avoid coalescing of small segments.

The macro shall be defined in the header. The implementation need not allow the value of the option to be set via *setsockopt*( ) or retrieved via *getsockopt*( ).

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/socket.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *getsockopt*( ), *setsockopt*( )

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

NAME

nl_types.h — data types

SYNOPSIS

XSI

```
#include <nl_types.h>
```

DESCRIPTION

The **<nl_types.h>** header shall contain definitions of at least the following types:

nl_catd Used by the message catalog functions *catopen*( ), *catgets*( ), and *catclose*( ) to identify a catalog descriptor.

nl_item Used by *nl_langinfo*( ) to identify items of *langinfo* data. Values of objects of type **nl_item** are defined in **<langinfo.h>**.

The **<nl_types.h>** header shall contain definitions of at least the following constants:

NL_SETD Used by *gencat* when no $*set* directive is specified in a message text source file; see the Internationalization Guide. This constant can be passed as the value of *set_id* on subsequent calls to *catgets*( ) (that is, to retrieve messages from the default message set). The value of NL_SETD is implementation-defined.

NL_CAT_LOCALE Value that must be passed as the *oflag* argument to *catopen*( ) to ensure that message catalog selection depends on the *LC_MESSAGES* locale category, rather than directly on the *LANG* environment variable.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided. |

```
int      catclose(nl_catd);
char    *catgets(nl_catd, int, int, const char *);
nl_catd  catopen(const char *, int);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<langinfo.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *catclose*( ), *catgets*( ), *catopen*( ), *nl_langinfo*( ), the Shell and Utilities volume of IEEE Std 1003.1-200x, *gencat*

CHANGE HISTORY

First released in Issue 2.

NAME

poll.h — definitions for the poll( ) function

SYNOPSIS

XSI

```
#include <poll.h>
```

DESCRIPTION

The **<poll.h>** header shall define the **pollfd** structure that includes at least the following members:

`int`	`fd`	The following descriptor being polled.
`short`	`events`	The input event flags (see below).
`short`	`revents`	The output event flags (see below).

The **<poll.h>** header shall define the following type through **typedef**:

nfds_t	An unsigned integer type used for the number of file descriptors.

The implementation shall support one or more programming environments in which the width of **nfds_t** is no greater than the width of type **long**. The names of these programming environments can be obtained using the *confstr*( ) function or the *getconf* utility.

The following symbolic constants shall be defined, zero or more of which may be OR'ed together to form the *events* or *revents* members in the **pollfd** structure:

POLLIN	Data other than high-priority data may be read without blocking.
POLLRDNORM	Normal data may be read without blocking.
POLLRDBAND	Priority data may be read without blocking.
POLLPRI	High priority data may be read without blocking.
POLLOUT	Normal data may be written without blocking.
POLLWRNORM	Equivalent to POLLOUT.
POLLWRBAND	Priority data may be written.
POLLERR	An error has occurred (*revents* only).
POLLHUP	Device has been disconnected (*revents* only).
POLLNVAL	Invalid *fd* member (*revents* only).

The significance and semantics of normal, priority, and high-priority data are file and device-specific.

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided.

```
int    poll(struct pollfd[], nfds_t, int);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *confstr*( ), *poll*( ), the Shell and Utilities volume of IEEE Std 1003.1-200x, *getconf*

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 6

The description of the symbolic constants is updated to match the *poll*( ) function.

Text related to STREAMS has been moved to the *poll*( ) reference page.

A note is added to the DESCRIPTION regarding the significance and semantics of normal, priority, and high-priority data.

NAME

pthread.h — threads

SYNOPSIS

THR

```
#include <pthread.h>
```

DESCRIPTION

The **<pthread.h>** header shall define the following symbols:

BAR PTHREAD_BARRIER_SERIAL_THREAD
PTHREAD_CANCEL_ASYNCHRONOUS
PTHREAD_CANCEL_ENABLE
PTHREAD_CANCEL_DEFERRED
PTHREAD_CANCEL_DISABLE
PTHREAD_CANCELED
PTHREAD_COND_INITIALIZER
PTHREAD_CREATE_DETACHED
PTHREAD_CREATE_JOINABLE
PTHREAD_EXPLICIT_SCHED
PTHREAD_INHERIT_SCHED
XSI PTHREAD_MUTEX_DEFAULT
PTHREAD_MUTEX_ERRORCHECK
PTHREAD_MUTEX_INITIALIZER
XSI PTHREAD_MUTEX_NORMAL
PTHREAD_MUTEX_RECURSIVE
PTHREAD_ONCE_INIT
TPP|TPI PTHREAD_PRIO_INHERIT
PTHREAD_PRIO_NONE
PTHREAD_PRIO_PROTECT
PTHREAD_PROCESS_SHARED
PTHREAD_PROCESS_PRIVATE
TPS PTHREAD_SCOPE_PROCESS
PTHREAD_SCOPE_SYSTEM

The following types shall be defined as described in **<sys/types.h>**:

pthread_attr_t
BAR **pthread_barrier_t**
pthread_barrierattr_t
pthread_cond_t
pthread_condattr_t
pthread_key_t
pthread_mutex_t
pthread_mutexattr_t
pthread_once_t
pthread_rwlock_t
pthread_rwlockattr_t
SPI **pthread_spinlock_t**
pthread_t

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided. |

```
      int   pthread_atfork(void (*)(void), void (*)(void),
                void(*)(void));
      int   pthread_attr_destroy(pthread_attr_t *);
      int   pthread_attr_getdetachstate(const pthread_attr_t *, int *);
XSI   int   pthread_attr_getguardsize(const pthread_attr_t *restrict,
                size_t *restrict);
TPS   int   pthread_attr_getinheritsched(const pthread_attr_t *restrict,
                int *restrict);
      int   pthread_attr_getschedparam(const pthread_attr_t *restrict,
                struct sched_param *restrict);
TPS   int   pthread_attr_getschedpolicy(const pthread_attr_t *restrict,
                int *restrict);
TPS   int   pthread_attr_getscope(const pthread_attr_t *restrict,
                int *restrict);
XSI   int   pthread_attr_getstack(const pthread_attr_t *restrict,
                void **restrict, size_t *restrict);
TSA   int   pthread_attr_getstackaddr(const pthread_attr_t *restrict,
                void **restrict);
      int   pthread_attr_getstacksize(const pthread_attr_t *restrict,
                size_t *restrict);
      int   pthread_attr_init(pthread_attr_t *);
      int   pthread_attr_setdetachstate(pthread_attr_t *, int);
XSI   int   pthread_attr_setguardsize(pthread_attr_t *, size_t);
TPS   int   pthread_attr_setinheritsched(pthread_attr_t *, int);
      int   pthread_attr_setschedparam(pthread_attr_t *restrict,
                const struct sched_param *restrict);
TPS   int   pthread_attr_setschedpolicy(pthread_attr_t *, int);
      int   pthread_attr_setscope(pthread_attr_t *, int);
XSI   int   pthread_attr_setstack(pthread_attr_t *, void *, size_t);
TSA   int   pthread_attr_setstackaddr(pthread_attr_t *, void *);
      int   pthread_attr_setstacksize(pthread_attr_t *, size_t);
BAR   int   pthread_barrier_destroy(pthread_barrier_t *);
      int   pthread_barrier_init(pthread_barrier_t *restrict,
                const pthread_barrierattr_t *restrict, unsigned);
      int   pthread_barrier_wait(pthread_barrier_t *);
      int   pthread_barrierattr_destroy(pthread_barrierattr_t *);
      int   pthread_barrierattr_getpshared( \                            |
                const pthread_barrierattr_t *restrict, int *restrict);   |
      int   pthread_barrierattr_init(pthread_barrierattr_t *);           |
      int   pthread_barrierattr_setpshared(pthread_barrierattr_t *, int);
      int   pthread_cancel(pthread_t);
      void  pthread_cleanup_push(void (*)(void *), void *);
      void  pthread_cleanup_pop(int);
      int   pthread_cond_broadcast(pthread_cond_t *);
      int   pthread_cond_destroy(pthread_cond_t *);
      int   pthread_cond_init(pthread_cond_t *restrict,
                const pthread_condattr_t *restrict);
      int   pthread_cond_signal(pthread_cond_t *);
      int   pthread_cond_timedwait(pthread_cond_t *restrict,
                pthread_mutex_t *restrict, const struct timespec *restrict);
      int   pthread_cond_wait(pthread_cond_t *restrict,
                pthread_mutex_t *restrict);
```

```
         int   pthread_condattr_destroy(pthread_condattr_t *);
CS       int   pthread_condattr_getclock(const pthread_condattr_t *restrict,
                   clockid_t *restrict);
         int   pthread_condattr_getpshared(const pthread_condattr_t *restrict,
                   int *restrict);
         int   pthread_condattr_init(pthread_condattr_t *);
CS       int   pthread_condattr_setclock(pthread_condattr_t *, clockid_t);
         int   pthread_condattr_setpshared(pthread_condattr_t *, int);
         int   pthread_create(pthread_t *restrict, const pthread_attr_t *restrict,
                   void *(*)(void *), void *restrict);
         int   pthread_detach(pthread_t);
         int   pthread_equal(pthread_t, pthread_t);
         void  pthread_exit(void *);
XSI      int   pthread_getconcurrency(void);
TCT      int   pthread_getcpuclockid(pthread_t, clockid_t *);
TPS      int   pthread_getschedparam(pthread_t, int *restrict,
                   struct sched_param *restrict);
         void *pthread_getspecific(pthread_key_t);
         int   pthread_join(pthread_t, void **);
         int   pthread_key_create(pthread_key_t *, void (*)(void *));
         int   pthread_key_delete(pthread_key_t);
         int   pthread_mutex_destroy(pthread_mutex_t *);
TPP      int   pthread_mutex_getprioceiling(const pthread_mutex_t *restrict,
                   int *restrict);
         int   pthread_mutex_init(pthread_mutex_t *restrict,
                   const pthread_mutexattr_t *restrict);
         int   pthread_mutex_lock(pthread_mutex_t *);
TPP      int   pthread_mutex_setprioceiling(pthread_mutex_t *restrict, int,
                   int *restrict);
TMO      int   pthread_mutex_timedlock(pthread_mutex_t *,
                   const struct timespec *);
         int   pthread_mutex_trylock(pthread_mutex_t *);
         int   pthread_mutex_unlock(pthread_mutex_t *);
         int   pthread_mutexattr_destroy(pthread_mutexattr_t *);
TPP|TPI  int   pthread_mutexattr_getprioceiling( \                                |
                   const pthread_mutexattr_t *restrict, int *restrict);           |
         int   pthread_mutexattr_getprotocol(const pthread_mutexattr_t *restrict, |
                   int *restrict);
         int   pthread_mutexattr_getpshared(const pthread_mutexattr_t *restrict,
                   int *restrict);
XSI      int   pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict,
                   int *restrict);
         int   pthread_mutexattr_init(pthread_mutexattr_t *);
TPP|TPI  int   pthread_mutexattr_setprioceiling(pthread_mutexattr_t *, int);
         int   pthread_mutexattr_setprotocol(pthread_mutexattr_t *, int);
         int   pthread_mutexattr_setpshared(pthread_mutexattr_t *, int);
XSI      int   pthread_mutexattr_settype(pthread_mutexattr_t *, int);
         int   pthread_once(pthread_once_t *, void (*)(void));
         int   pthread_rwlock_destroy(pthread_rwlock_t *);
         int   pthread_rwlock_init(pthread_rwlock_t *restrict,
                   const pthread_rwlockattr_t *restrict);
         int   pthread_rwlock_rdlock(pthread_rwlock_t *);
```

```
          int    pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict,
                     const struct timespec *restrict);
          int    pthread_rwlock_timedwrlock(pthread_rwlock_t *restrict,
                     const struct timespec *restrict);
          int    pthread_rwlock_tryrdlock(pthread_rwlock_t *);
          int    pthread_rwlock_trywrlock(pthread_rwlock_t *);
          int    pthread_rwlock_unlock(pthread_rwlock_t *);
          int    pthread_rwlock_wrlock(pthread_rwlock_t *);
          int    pthread_rwlockattr_destroy(pthread_rwlockattr_t *);
          int    pthread_rwlockattr_getpshared(const pthread_rwlockattr_t *restrict,
                     int *restrict);
          int    pthread_rwlockattr_init(pthread_rwlockattr_t *);
          int    pthread_rwlockattr_setpshared(pthread_rwlockattr_t *, int);
          pthread_t
                 pthread_self(void);
          int    pthread_setcancelstate(int, int *);
          int    pthread_setcanceltype(int, int *);
XSI       int    pthread_setconcurrency(int);
TPS       int    pthread_setschedparam(pthread_t, int,
                     const struct sched_param *);
THR TPS   int    pthread_setschedprio(pthread_t, int);
          int    pthread_setspecific(pthread_key_t, const void *);
SPI       int    pthread_spin_destroy(pthread_spinlock_t *);
          int    pthread_spin_init(pthread_spinlock_t *, int);
          int    pthread_spin_lock(pthread_spinlock_t *);
          int    pthread_spin_trylock(pthread_spinlock_t *);
          int    pthread_spin_unlock(pthread_spinlock_t *);
          void   pthread_testcancel(void);
```

Inclusion of the **<pthread.h>** header shall make symbols defined in the headers **<sched.h>** and **<time.h>** visible.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sched.h>, **<time.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *pthread_attr_getguardsize*( ), *pthread_attr_init*( ), *pthread_attr_setscope*( ), *pthread_barrier_destroy*( ), *pthread_barrier_init*( ), *pthread_barrier_wait*( ), *pthread_barrierattr_destroy*( ), *pthread_barrierattr_getpshared*( ), *pthread_barrierattr_init*( ), *pthread_barrierattr_setpshared*( ), *pthread_cancel*( ), *pthread_cleanup_pop*( ), *pthread_cond_init*( ), *pthread_cond_signal*( ), *pthread_cond_wait*( ), *pthread_condattr_getclock*( ), *pthread_condattr_init*( ), *pthread_condattr_setclock*( ), *pthread_create*( ), *pthread_detach*( ), *pthread_equal*( ), *pthread_exit*( ), *pthread_getconcurrency*( ), *pthread_getcpuclockid*( ), *pthread_getschedparam*( ), *pthread_join*( ), *pthread_key_create*( ), *pthread_key_delete*( ), *pthread_mutex_init*( ), *pthread_mutex_lock*( ), *pthread_mutex_setprioceiling*( ), *pthread_mutex_timedlock*( ), *pthread_mutexattr_init*( ), *pthread_mutexattr_gettype*( ), *pthread_mutexattr_setprotocol*( ), *pthread_once*( ), *pthread_rwlock_destroy*( ), *pthread_rwlock_init*( ), *pthread_rwlock_rdlock*( ), *pthread_rwlock_timedrdlock*( ), *pthread_rwlock_timedwrlock*( ), *pthread_rwlock_tryrdlock*( ), *pthread_rwlock_trywrlock*( ), *pthread_rwlock_unlock*( ), *pthread_rwlock_wrlock*( ), *pthread_rwlockattr_destroy*( ), *pthread_rwlockattr_getpshared*( ), *pthread_rwlockattr_init*( ), *pthread_rwlockattr_setpshared*( ), *pthread_self*( ), *pthread_setcancelstate*( ), *pthread_setspecific*( ), *pthread_spin_destroy*( ), *pthread_spin_init*( ), *pthread_spin_lock*( ), *pthread_spin_trylock*( ), *pthread_spin_unlock*( )

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The RTT margin markers are now broken out into their POSIX options.

The Open Group Corrigendum U021/9 is applied, correcting the prototype for the *pthread_cond_wait*( ) function.

The Open Group Corrigendum U026/2 is applied correcting the prototype for the *pthread_setschedparam*( ) function so that its second argument is of type **int**.

The *pthread_getcpuclockid*( ) and *pthread_mutex_timedlock*( ) functions are added for alignment with IEEE Std 1003.1d-1999.

The following functions are added for alignment with IEEE Std 1003.1j-2000: *pthread_barrier_destroy*( ), *pthread_barrier_init*( ), *pthread_barrier_wait*( ), *pthread_barrierattr_destroy*( ), *pthread_barrierattr_getpshared*( ), *pthread_barrierattr_init*( ), *pthread_barrierattr_setpshared*( ), *pthread_condattr_getclock*( ), *pthread_condattr_setclock*( ), *pthread_rwlock_timedrdlock*( ), *pthread_rwlock_timedwrlock*( ), *pthread_spin_destroy*( ), *pthread_spin_init*( ), *pthread_spin_lock*( ), *pthread_spin_trylock*( ), and *pthread_spin_unlock*( ).

PTHREAD_RWLOCK_INITIALIZER is deleted for alignment with IEEE Std 1003.1j-2000.

Functions previously marked as part of the Read-Write Locks option are now moved to the Threads option.

The **restrict** keyword is added to the prototypes for *pthread_attr_getguardsize*( ), *pthread_attr_getinheritsched*( ), *pthread_attr_getschedparam*( ), *pthread_attr_getschedpolicy*( ), *pthread_attr_getscope*( ), *pthread_attr_getstackaddr*( ), *pthread_attr_getstacksize*( ), *pthread_attr_setschedparam*( ), *pthread_barrier_init*( ), *pthread_barrierattr_getpshared*( ), *pthread_cond_init*( ), *pthread_cond_signal*( ), *pthread_cond_timedwait*( ), *pthread_cond_wait*( ), *pthread_condattr_getclock*( ), *pthread_condattr_getpshared*( ), *pthread_create*( ), *pthread_getschedparam*( ), *pthread_mutex_getprioceiling*( ), *pthread_mutex_init*( ), *pthread_mutex_setprioceiling*( ), *pthread_mutexattr_getprioceiling*( ), *pthread_mutexattr_getprotocol*( ), *pthread_mutexattr_getpshared*( ), *pthread_mutexattr_gettype*( ), *pthread_rwlock_init*( ), *pthread_rwlock_timedrdlock*( ), *pthread_rwlock_timedwrlock*( ), *pthread_rwlockattr_getpshared*( ), and *pthread_sigmask*( ).

IEEE PASC Interpretation 1003.1 #86 is applied, allowing the symbols from **<sched.h>** and **<time.h>** to be made visible when **<pthread.h>** is included. Previously this was an XSI extension.

IEEE PASC Interpretation 1003.1c #42 is applied, removing the requirement for prototypes for the *pthread_kill*( ) and *pthread_sigmask*( ) functions. These are required to be in the **<signal.h>** header. They are allowed here through the name space rules.

IEEE PASC Interpretation 1003.1 #96 is applied, adding the *pthread_setschedprio*( ) function.

NAME

pwd.h — password structure

SYNOPSIS

```
#include <pwd.h>
```

DESCRIPTION

The **<pwd.h>** header shall provide a definition for **struct passwd**, which shall include at least the following members:

`char`	`*pw_name`	User's login name.
`uid_t`	`pw_uid`	Numerical user ID.
`gid_t`	`pw_gid`	Numerical group ID.
`char`	`*pw_dir`	Initial working directory.
`char`	`*pw_shell`	Program to use as shell.

The **gid_t** and **uid_t** types shall be defined as described in **<sys/types.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
struct passwd *getpwnam(const char *);
struct passwd *getpwuid(uid_t);
```

TSF

```
int            getpwnam_r(const char *, struct passwd *, char *,
                   size_t, struct passwd **);
int            getpwuid_r(uid_t, struct passwd *, char *,
                   size_t, struct passwd **);
```

XSI

```
void           endpwent(void);
struct passwd *getpwent(void);
void           setpwent(void);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *endpwent*( ), *getpwnam*( ), *getpwuid*( )

CHANGE HISTORY

First released in Issue 1.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The **gid_t** and **uid_t** types are mandated.
- The *getpwnam_r*( ) and *getpwuid_r*( ) functions are marked as part of the _POSIX_THREAD_SAFE_FUNCTIONS option.

NAME

regex.h — regular expression matching types

SYNOPSIS

```
#include <regex.h>
```

DESCRIPTION

The **<regex.h>** header shall define the structures and symbolic constants used by the *regcomp*( ), *regexec*( ), *regerror*( ), and *regfree*( ) functions.

The structure type **regex_t** shall contain at least the following member:

`size_t`	`re_nsub`	Number of parenthesized subexpressions.

The type **size_t** shall be defined as described in **<sys/types.h>**.

The type **regoff_t** shall be defined as a signed integer type that can hold the largest value that can be stored in either a type **off_t** or type **ssize_t**. The structure type **regmatch_t** shall contain at least the following members:

`regoff_t`	`rm_so`	Byte offset from start of string to start of substring.
`regoff_t`	`rm_eo`	Byte offset from start of string of the first character after the end of substring.

Values for the *cflags* parameter to the *regcomp*( ) function:

REG_EXTENDED	Use Extended Regular Expressions.
REG_ICASE	Ignore case in match.
REG_NOSUB	Report only success or fail in *regexec*( ).
REG_NEWLINE	Change the handling of newline.

Values for the *eflags* parameter to the *regexec*( ) function:

REG_NOTBOL	The circumflex character (`'^'`), when taken as a special character, does not match the beginning of *string*.
REG_NOTEOL	The dollar sign (`'$'`), when taken as a special character, does not match the end of *string*.

The following constants shall be defined as error return values:

REG_NOMATCH	*regexec*( ) failed to match.
REG_BADPAT	Invalid regular expression.
REG_ECOLLATE	Invalid collating element referenced.
REG_ECTYPE	Invalid character class type referenced.
REG_EESCAPE	Trailing `'\'` in pattern.
REG_ESUBREG	Number in *digit* invalid or in error.
REG_EBRACK	`"[]"` imbalance.
REG_EPAREN	`"\(\)"` or `"()"` imbalance.
REG_EBRACE	`"\{\}"` imbalance.
REG_BADBR	Content of `"\{\}"` invalid: not a number, number too large, more than two numbers, first larger than second.

	REG_ERANGE	Invalid endpoint in range expression.
	REG_ESPACE	Out of memory.
	REG_BADRPT	`'?'`, `'*'`, or `'+'` not preceded by valid regular expression.
OB	REG_ENOSYS	Reserved.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int    regcomp(regex_t *restrict, const char *restrict, int);
size_t regerror(int, const regex_t *restrict, char *restrict, size_t);
int    regexec(const regex_t *restrict, const char *restrict, size_t,
           regmatch_t[restrict], int);
void   regfree(regex_t *);
```

The implementation may define additional macros or constants using names beginning with REG_.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *regcomp*( ), the Shell and Utilities volume of IEEE Std 1003.1-200x

CHANGE HISTORY

First released in Issue 4.

Originally derived from the ISO POSIX-2 standard.

Issue 6

The REG_ENOSYS constant is marked obsolescent.

The **restrict** keyword is added to the prototypes for *regcomp*( ), *regerror*( ), and *regexec*( ).

A statement is added that the **size_t** type is defined as described in **<sys/types.h>**.

NAME

sched.h — execution scheduling (**REALTIME**)

SYNOPSIS

PS

```
#include <sched.h>
```

DESCRIPTION

The **<sched.h>** header shall define the **sched_param** structure, which contains the scheduling parameters required for implementation of each supported scheduling policy. This structure shall contain at least the following member:

`int`	`sched_priority`	Process execution scheduling priority.

SS|TSP In addition, if _POSIX_SPORADIC_SERVER or _POSIX_THREAD_SPORADIC_SERVER is defined, the **sched_param** structure defined in **<sched.h>** shall contain the following members in addition to those specified above:

`int`	`sched_ss_low_priority`	Low scheduling priority for sporadic server.
`struct timespec`	`sched_ss_repl_period`	Replenishment period for sporadic server.
`struct timespec`	`sched_ss_init_budget`	Initial budget for sporadic server.
`int`	`sched_ss_max_repl`	Maximum pending replenishments for sporadic server.

Each process is controlled by an associated scheduling policy and priority. Associated with each policy is a priority range. Each policy definition specifies the minimum priority range for that policy. The priority ranges for each policy may overlap the priority ranges of other policies.

Four scheduling policies are defined; others may be defined by the implementation. The four standard policies are indicated by the values of the following symbolic constants:

SCHED_FIFO	First in-first out (FIFO) scheduling policy.
SCHED_RR	Round robin scheduling policy.
SCHED_SPORADIC	Sporadic server scheduling policy. (SS|TSP)
SCHED_OTHER	Another scheduling policy.

The values of these constants are distinct.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided. |

```
int    sched_get_priority_max(int);
int    sched_get_priority_min(int);
int    sched_getparam(pid_t, struct sched_param *);
int    sched_getscheduler(pid_t);
int    sched_rr_get_interval(pid_t, struct timespec *);
int    sched_setparam(pid_t, const struct sched_param *);
int    sched_setscheduler(pid_t, int, const struct sched_param *);
int    sched_yield(void);
```

Inclusion of the **<sched.h>** header makes symbols defined in the header **<time.h>** visible.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<time.h>

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The **<sched.h>** header is marked as part of the Process Scheduling option.

Sporadic server members are added to the **sched_param** structure, and the SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

IEEE PASC Interpretation 1003.1 #108 is applied, correcting the **sched_param** structure whose members *sched_ss_repl_period* and *sched_ss_init_budget* members should be type **struct timespec** and not **timespec**.

NAME

search.h — search tables

SYNOPSIS

XSI

```
#include <search.h>
```

DESCRIPTION

The **<search.h>** header shall define the **ENTRY** type for structure **entry** which shall include the following members:

```
char    *key
void    *data
```

and shall define **ACTION** and **VISIT** as enumeration data types through type definitions as follows:

```
enum { FIND, ENTER } ACTION;
enum { preorder, postorder, endorder, leaf } VISIT;
```

The **size_t** type shall be defined as described in **<sys/types.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int    hcreate(size_t);
void   hdestroy(void);
ENTRY *hsearch(ENTRY, ACTION);
void   insque(void *, void *);
void  *lfind(const void *, const void *, size_t *,
          size_t, int (*)(const void *, const void *));
void  *lsearch(const void *, void *, size_t *,
          size_t, int (*)(const void *, const void *));
void   remque(void *);
void  *tdelete(const void *restrict, void **restrict,
          int(*)(const void *, const void *));
void  *tfind(const void *, void *const *,
          int(*)(const void *, const void *));
void  *tsearch(const void *, void **,
          int(*)(const void *, const void *));
void   twalk(const void *,
          void (*)(const void *, VISIT, int ));
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *hcreate*( ), *insque*( ), *lsearch*( ), *remque*( ), *tsearch*( )

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The Open Group Corrigendum U021/6 is applied updating the prototypes for *tdelete*( ) and *tsearch*( ).

The **restrict** keyword is added to the prototype for *tdelete*( ).

NAME

semaphore.h — semaphores (**REALTIME**)

SYNOPSIS

SEM
```
#include <semaphore.h>
```

DESCRIPTION

The **<semaphore.h>** header shall define the **sem_t** type, used in performing semaphore operations. The semaphore may be implemented using a file descriptor, in which case applications are able to open up at least a total of {OPEN_MAX} files and semaphores. The symbol SEM_FAILED shall be defined (see *sem_open*( )).

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided. |

```
int    sem_close(sem_t *);
int    sem_destroy(sem_t *);
int    sem_getvalue(sem_t *restrict, int *restrict);
int    sem_init(sem_t *, int, unsigned);
sem_t *sem_open(const char *, int, ...);
int    sem_post(sem_t *);
int    sem_timedwait(sem_t *restrict, const struct timespec *restrict);
int    sem_trywait(sem_t *);
int    sem_unlink(const char *);
int    sem_wait(sem_t *);
```
TMO

Inclusion of the **<semaphore.h>** header may make visible symbols defined in the headers **<fcntl.h>** and **<sys/types.h>**.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<fcntl.h>, **<sys/types.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *sem_destroy*( ), *sem_getvalue*( ), *sem_init*( ), *sem_open*( ), *sem_post*( ), *sem_timedwait*( ), *sem_trywait*( ), *sem_unlink*( ), *sem_wait*( )

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The **<semaphore.h>** header is marked as part of the Semaphores option.

The Open Group Corrigendum U021/3 is applied, adding a description of SEM_FAILED.

The *sem_timedwait*( ) function is added for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the prototypes for *sem_getvalue*( ) and *sem_timedwait*( ).

NAME

setjmp.h — stack environment declarations

SYNOPSIS

```
#include <setjmp.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these symbols in this header.

CX The **<setjmp.h>** header shall define the array types **jmp_buf** and **sigjmp_buf**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
    void   longjmp(jmp_buf, int);
CX  void   siglongjmp(sigjmp_buf, int);
XSI void  _longjmp(jmp_buf, int);
```

The following may be declared as a function, or defined as a macro, or both. Function prototypes shall be provided.

```
    int    setjmp(jmp_buf);
CX  int    sigsetjmp(sigjmp_buf, int);
XSI int   _setjmp(jmp_buf);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *longjmp*( ), *_longjmp*( ), *setjmp*( ), *siglongjmp*( ), *sigsetjmp*( )

CHANGE HISTORY

First released in Issue 1.

Issue 6

Extensions beyond the ISO C standard are now marked.

NAME

signal.h — signals

SYNOPSIS

```
#include <signal.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these symbols in this header.

The **<signal.h>** header shall define the following symbolic constants, each of which expands to a distinct constant expression of the type:

```
void (*)(int)
```

whose value matches no declarable function.

SIG_DFL Request for default signal handling.

SIG_ERR Return value from *signal*( ) in case of error.

CX SIG_HOLD Request that signal be held.

SIG_IGN Request that signal be ignored.

The following data types shall be defined through **typedef**:

sig_atomic_t Possibly volatile-qualified integer type of an object that can be accessed as an atomic entity, even in the presence of asynchronous interrupts.

CX **sigset_t** Integer or structure type of an object used to represent sets of signals.

CX **pid_t** As described in **<sys/types.h>**.

RTS The **<signal.h>** header shall define the **sigevent** structure, which has at least the following members:

```
int                     sigev_notify             Notification type.
int                     sigev_signo              Signal number.
union sigval            sigev_value              Signal value.
void(*)(union sigval)   sigev_notify_function    Notification function.
(pthread_attr_t *)      sigev_notify_attributes  Notification attributes.
```

The following values of *sigev_notify* shall be defined:

SIGEV_NONE No asynchronous notification is delivered when the event of interest occurs.

SIGEV_SIGNAL A queued signal, with an application-defined value, is generated when the event of interest occurs.

SIGEV_THREAD A notification function is called to perform notification.

The **sigval** union shall be defined as:

```
int     sival_int      Integer signal value.
void   *sival_ptr      Pointer signal value.
```

This header shall also declare the macros SIGRTMIN and SIGRTMAX, which evaluate to integer expressions, and specify a range of signal numbers that are reserved for application use and for which the realtime signal behavior specified in this volume of IEEE Std 1003.1-200x is supported.

The signal numbers in this range do not overlap any of the signals specified in the following table.

The range SIGRTMIN through SIGRTMAX inclusive shall include at least {RTSIG_MAX} signal numbers.

It is implementation-defined whether realtime signal behavior is supported for other signals.

This header also declares the constants that are used to refer to the signals that occur in the system. Signals defined here begin with the letters SIG. Each of the signals have distinct positive integer values. The value 0 is reserved for use as the null signal (see *kill*( )). Additional implementation-defined signals may occur in the system.

CX The ISO C standard only requires the signal names SIGABRT, SIGFPE, SIGILL, SIGINT, SIGSEGV, and SIGTERM to be defined.

The following signals shall be supported on all implementations (default actions are explained below the table):

	Signal	Default Action	Description
	SIGABRT	A	Process abort signal.
	SIGALRM	T	Alarm clock.
	SIGBUS	A	Access to an undefined portion of a memory object.
	SIGCHLD	I	Child process terminated, stopped,
XSI			or continued.
	SIGCONT	C	Continue executing, if stopped.
	SIGFPE	A	Erroneous arithmetic operation.
	SIGHUP	T	Hangup.
	SIGILL	A	Illegal instruction.
	SIGINT	T	Terminal interrupt signal.
	SIGKILL	T	Kill (cannot be caught or ignored).
	SIGPIPE	T	Write on a pipe with no one to read it.
	SIGQUIT	A	Terminal quit signal.
	SIGSEGV	A	Invalid memory reference.
	SIGSTOP	S	Stop executing (cannot be caught or ignored).
	SIGTERM	T	Termination signal.
	SIGTSTP	S	Terminal stop signal.
	SIGTTIN	S	Background process attempting read.
	SIGTTOU	S	Background process attempting write.
	SIGUSR1	T	User-defined signal 1.
	SIGUSR2	T	User-defined signal 2.
XSI	SIGPOLL	T	Pollable event.
	SIGPROF	T	Profiling timer expired.
	SIGSYS	A	Bad system call.
	SIGTRAP	A	Trace / breakpoint trap.
	SIGURG	I	High bandwidth data is available at a socket.
XSI	SIGVTALRM	T	Virtual timer expired.
	SIGXCPU	A	CPU time limit exceeded.
	SIGXFSZ	A	File size limit exceeded.

The default actions are as follows:

T Abnormal termination of the process. The process is terminated with all the consequences of *_exit*( ) except that the status made available to *wait*( ) and *waitpid*( ) indicates abnormal termination by the specified signal.

A Abnormal termination of the process.

XSI Additionally, implementation-defined abnormal termination actions, such as creation of a core file, may occur.

I Ignore the signal.

S Stop the process.

C Continue the process, if it is stopped; otherwise, ignore the signal.

CX The header shall provide a declaration of **struct sigaction**, including at least the following members:

`void (*sa_handler)(int)`	What to do on receipt of signal.
`sigset_t sa_mask`	Set of signals to be blocked during execution of the signal handling function.
`int      sa_flags`	Special flags.
`void (*)(int, siginfo_t *, void *) sa_sigaction`	Pointer to signal handler function or one of the macros SIG_IGN or SIG_DFL.

XSI The storage occupied by *sa_handler* and *sa_sigaction* may overlap, and a portable program must not use both simultaneously.

The following shall be declared as constants:

CX XSI	SA_NOCLDSTOP	Do not generate SIGCHLD when children stop or stopped children continue.
CX	SIG_BLOCK	The resulting set is the union of the current set and the signal set pointed to by the argument *set*.
CX	SIG_UNBLOCK	The resulting set is the intersection of the current set and the complement of the signal set pointed to by the argument *set*.
CX	SIG_SETMASK	The resulting set is the signal set pointed to by the argument *set*.
XSI	SA_ONSTACK	Causes signal delivery to occur on an alternate stack.
XSI	SA_RESETHAND	Causes signal dispositions to be set to SIG_DFL on entry to signal handlers.
XSI	SA_RESTART	Causes certain functions to become restartable.
XSI	SA_SIGINFO	Causes extra information to be passed to signal handlers at the time of receipt of a signal.
XSI	SA_NOCLDWAIT	Causes implementations not to create zombie processes on child death.
XSI	SA_NODEFER	Causes signal not to be automatically blocked on entry to signal handler.
XSI	SS_ONSTACK	Process is executing on an alternate signal stack.
XSI	SS_DISABLE	Alternate signal stack is disabled.
XSI	MINSIGSTKSZ	Minimum stack size for a signal handler.
XSI	SIGSTKSZ	Default size in bytes for the alternate signal stack.

XSI The **ucontext_t** structure shall be defined through **typedef** as described in **<ucontext.h>**.

The **mcontext_t** type shall be defined through **typedef** as described in **<ucontext.h>**.

The **<signal.h>** header shall define the **stack_t** type as a structure that includes at least the following members:

`void`	`*ss_sp`	Stack base or pointer.
`size_t`	`ss_size`	Stack size.
`int`	`ss_flags`	Flags.

The **<signal.h>** header shall define the **sigstack** structure that includes at least the following members:

`int`	`ss_onstack`	Non-zero when signal stack is in use.
`void`	`*ss_sp`	Signal stack pointer.

CX The **<signal.h>** header shall define the **siginfo_t** type as a structure that includes at least the following members:

CX	`int`	`si_signo`	Signal number.
XSI	`int`	`si_errno`	If non-zero, an *errno* value associated with this signal, as defined in **<errno.h>.**
CX	`int`	`si_code`	Signal code.
XSI	`pid_t`	`si_pid`	Sending process ID.
	`uid_t`	`si_uid`	Real user ID of sending process.
	`void`	`*si_addr`	Address of faulting instruction.
	`int`	`si_status`	Exit value or signal.
	`long`	`si_band`	Band event for SIGPOLL.
RTS	`union sigval`	`si_value`	Signal value.

The macros specified in the **Code** column of the following table are defined for use as values of XSI *si_code* that are signal-specific or non-signal-specific reasons why the signal was generated.

XSI

Signal	Code	Reason
SIGILL	ILL_ILLOPC	Illegal opcode.
	ILL_ILLOPN	Illegal operand.
	ILL_ILLADR	Illegal addressing mode.
	ILL_ILLTRP	Illegal trap.
	ILL_PRVOPC	Privileged opcode.
	ILL_PRVREG	Privileged register.
	ILL_COPROC	Coprocessor error.
	ILL_BADSTK	Internal stack error.
SIGFPE	FPE_INTDIV	Integer divide by zero.
	FPE_INTOVF	Integer overflow.
	FPE_FLTDIV	Floating-point divide by zero.
	FPE_FLTOVF	Floating-point overflow.
	FPE_FLTUND	Floating-point underflow.
	FPE_FLTRES	Floating-point inexact result.
	FPE_FLTINV	Invalid floating-point operation.
	FPE_FLTSUB	Subscript out of range.
SIGSEGV	SEGV_MAPERR	Address not mapped to object.
	SEGV_ACCERR	Invalid permissions for mapped object.
SIGBUS	BUS_ADRALN	Invalid address alignment.
	BUS_ADRERR	Non-existent physical address.
	BUS_OBJERR	Object specific hardware error.
SIGTRAP	TRAP_BRKPT	Process breakpoint.
	TRAP_TRACE	Process trace trap.
SIGCHLD	CLD_EXITED	Child has exited.
	CLD_KILLED	Child has terminated abnormally and did not create a core file.
	CLD_DUMPED	Child has terminated abnormally and created a core file.
	CLD_TRAPPED	Traced child has trapped.
	CLD_STOPPED	Child has stopped.
	CLD_CONTINUED	Stopped child has continued.
SIGPOLL	POLL_IN	Data input available.
	POLL_OUT	Output buffers available.
	POLL_MSG	Input message available.
	POLL_ERR	I/O error.
	POLL_PRI	High priority input available.
	POLL_HUP	Device disconnected.
Any	SI_USER	Signal sent by *kill*( ).
	SI_QUEUE	Signal sent by the *sigqueue*( ).
	SI_TIMER	Signal generated by expiration of a timer set by *timer_settime*( ).
	SI_ASYNCIO	Signal generated by completion of an asynchronous I/O request.
	SI_MESGQ	Signal generated by arrival of a message on an empty message queue.

XSI Implementations may support additional *si_code* values not included in this list, may generate values included in this list under circumstances other than those described in this list, and may contain extensions or limitations that prevent some values from being generated. Implementations do not generate a different value from the ones described in this list for circumstances described in this list.

In addition, the following signal-specific information shall be available:

Signal	Member	Value
SIGILL SIGFPE	**void * si_addr**	Address of faulting instruction.
SIGSEGV SIGBUS	**void * si_addr**	Address of faulting memory reference.
SIGCHLD	**pid_t si_pid** **int si_status** **uid_t si_uid**	Child process ID. Exit value or signal. Real user ID of the process that sent the signal.
SIGPOLL	**long si_band**	Band event for POLL_IN, POLL_OUT, or POLL_MSG.

For some implementations, the value of *si_addr* may be inaccurate.

The following shall be declared as functions and may also be defined as macros:

```
XSI   void (*bsd_signal(int, void (*)(int)))(int);
CX    int    kill(pid_t, int);
XSI   int    killpg(pid_t, int);
THR   int    pthread_kill(pthread_t, int);
      int    pthread_sigmask(int, const sigset_t *, sigset_t *);
      int    raise(int);
CX    int    sigaction(int, const struct sigaction *restrict,
                 struct sigaction *restrict);
      int    sigaddset(sigset_t *, int);
XSI   int    sigaltstack(const stack_t *restrict, stack_t *restrict);
CX    int    sigdelset(sigset_t *, int);
      int    sigemptyset(sigset_t *);
      int    sigfillset(sigset_t *);
XSI   int    sighold(int);
      int    sigignore(int);
      int    siginterrupt(int, int);
CX    int    sigismember(const sigset_t *, int);
      void (*signal(int, void (*)(int)))(int);
XSI   int    sigpause(int);
CX    int    sigpending(sigset_t *);
      int    sigprocmask(int, const sigset_t *restrict, sigset_t *restrict);
RTS   int    sigqueue(pid_t, int, const union sigval);
XSI   int    sigrelse(int);
      void (*sigset(int, void (*)(int)))(int);
CX    int    sigsuspend(const sigset_t *);
RTS   int    sigtimedwait(const sigset_t *restrict, siginfo_t *restrict,
                 const struct timespec *restrict);
CX    int    sigwait(const sigset_t *restrict, int *restrict);
RTS   int    sigwaitinfo(const sigset_t *restrict, siginfo_t *restrict);
```

CX Inclusion of the **<signal.h>** header may make visible all symbols from the **<time.h>** header.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<errno.h>, **<stropts.h>**, **<sys/types.h>**, **<time.h>**, **<ucontext.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *alarm*( ), *bsd_signal*( ), *ioctl*( ), *kill*( ), *killpg*( ), *raise*( ), *sigaction*( ), *sigaddset*( ), *sigaltstack*( ), *sigdelset*( ), *sigemptyset*( ), *sigfillset*( ), *siginterrupt*( ), *sigismember*( ), *signal*( ), *sigpending*( ), *sigprocmask*( ), *sigqueue*( ), *sigsuspend*( ), *sigwaitinfo*( ), *wait*( ), *waitid*( )

CHANGE HISTORY

First released in Issue 1.

Issue 5

The DESCRIPTION is updated for alignment with POSIX Realtime Extension and the POSIX Threads Extension.

The default action for SIGURG is changed for i to iii. The function prototype for *sigmask*( ) is removed.

Issue 6

The Open Group Corrigendum U035/2 is applied. In the DESCRIPTION, the wording for abnormal termination is clarified.

The Open Group Corrigendum U028/8 is applied, correcting the prototype for the *sigset*( ) function.

The Open Group Corrigendum U026/3 is applied, correcting the type of the *sigev_notify_function* function member of the **sigevent** structure.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP, SIGTTIN, and SIGTTOU signals are now mandated. This is also a FIPS requirement.
- The **pid_t** definition is mandated.

The RT markings are now changed to RTS to denote that the semantics are part of the Realtime Signals Extension option.

The **restrict** keyword is added to the prototypes for *sigaction*( ), *sigaltstack*( ), *sigprocmask*( ), *sigtimedwait*( ), *sigwait*( ), and *sigwaitinfo*( ).

IEEE PASC Interpretation 1003.1 #85 is applied, adding the statement that symbols from **<time.h>** may be made visible when **<signal.h>** is included. Extensions beyond the ISO C standard are now marked.

NAME

spawn.h — spawn (**ADVANCED REALTIME**)

SYNOPSIS

```
#include <spawn.h>
```

DESCRIPTION

The **<spawn.h>** header shall define the **posix_spawnattr_t** and **posix_spawn_file_actions_t** types used in performing spawn operations.

The **<spawn.h>** header shall define the flags that may be set in a **posix_spawnattr_t** object using the *posix_spawnattr_setflags*( ) function:

POSIX_SPAWN_RESETIDS
POSIX_SPAWN_SETPGROUP
POSIX_SPAWN_SETSCHEDPARAM
POSIX_SPAWN_SETSCHEDULER
POSIX_SPAWN_SETSIGDEF
POSIX_SPAWN_SETSIGMASK

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int   posix_spawn(pid_t *restrict, const char *restrict,
          const posix_spawn_file_actions_t *,
          const posix_spawnattr_t *restrict, char *const [restrict],
          char *const [restrict]);
int   posix_spawn_file_actions_addclose(posix_spawn_file_actions_t *,
          int);
int   posix_spawn_file_actions_adddup2(posix_spawn_file_actions_t *,
          int, int);
int   posix_spawn_file_actions_addopen(posix_spawn_file_actions_t *restrict,
          int, const char *restrict, int, mode_t);
int   posix_spawn_file_actions_destroy(posix_spawn_file_actions_t *);
int   posix_spawn_file_actions_init(posix_spawn_file_actions_t *);
int   posix_spawnattr_destroy(posix_spawnattr_t *);
int   posix_spawnattr_getsigdefault(const posix_spawnattr_t *restrict,
          sigset_t *restrict);
int   posix_spawnattr_getflags(const posix_spawnattr_t *restrict,
          short *restrict);
int   posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict,
          pid_t *restrict);
int   posix_spawnattr_getschedparam(const posix_spawnattr_t *restrict,
          struct sched_param *restrict);
int   posix_spawnattr_getschedpolicy(const posix_spawnattr_t *restrict,
          int *restrict);
int   posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict,
          sigset_t *restrict);
int   posix_spawnattr_init(posix_spawnattr_t *);
int   posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict,
          const sigset_t *restrict);
int   posix_spawnattr_setflags(posix_spawnattr_t *, short);
int   posix_spawnattr_setpgroup(posix_spawnattr_t *, pid_t);
int   posix_spawnattr_setschedparam(posix_spawnattr_t *restrict,
          const struct sched_param *restrict);
int   posix_spawnattr_setschedpolicy(posix_spawnattr_t *, int);
int   posix_spawnattr_setsigmask(posix_spawnattr_t *restrict,
          const sigset_t *restrict);
int   posix_spawnp(pid_t *restrict, const char *restrict,
          const posix_spawn_file_actions_t *,
          const posix_spawnattr_t *restrict,
          char *const [restrict], char *const [restrict]);
```

Inclusion of the **<spawn.h>** header may make visible symbols defined in the **<sched.h>**, **<signal.h>**, and **<sys/types.h>** headers.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sched.h>, **<semaphore.h>**, **<signal.h>**, **<sys/types.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *posix_spawnattr_destroy*( ), *posix_spawnattr_getsigdefault*( ), *posix_spawnattr_getflags*( ), *posix_spawnattr_getpgroup*( ), *posix_spawnattr_getschedparam*( ), *posix_spawnattr_getschedpolicy*( ), *posix_spawnattr_getsigmask*( ), *posix_spawnattr_init*( ), *posix_spawnattr_setsigdefault*( ), *posix_spawnattr_setflags*( ), *posix_spawnattr_setpgroup*( ), *posix_spawnattr_setschedparam*( ), *posix_spawnattr_setschedpolicy*( ), *posix_spawnattr_setsigmask*( ), *posix_spawn*( ), *posix_spawn_file_actions_addclose*( ), *posix_spawn_file_actions_adddup2*( ), *posix_spawn_file_actions_addopen*( ), *posix_spawn_file_actions_destroy*( ), *posix_spawn_file_actions_init*( ), *posix_spawnp*( )

CHANGE HISTORY

First released in Issue 6. Included for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the prototypes for *posix_spawn*( ), *posix_spawn_file_actions_addopen*( ), *posix_spawnattr_getsigdefault*( ), *posix_spawnattr_getflags*( ), *posix_spawnattr_getpgroup*( ), *posix_spawnattr_getschedparam*( ), *posix_spawnattr_getschedpolicy*( ), *posix_spawnattr_getsigmask*( ), *posix_spawnattr_setsigdefault*( ), *posix_spawnattr_setschedparam*( ), *posix_spawnattr_setsigmask*( ), and *posix_spawnp*( ).

NAME

stdarg.h — handle variable argument list

SYNOPSIS

```
#include <stdarg.h>

void va_start(va_list ap, argN);
void va_copy(va_list dest, va_list src);
type va_arg(va_list ap, type);
void va_end(va_list ap);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-200x defers to the ISO C standard.

The **<stdarg.h>** header shall contain a set of macros which allows portable functions that accept variable argument lists to be written. Functions that have variable argument lists (such as *printf*( )) but do not use these macros, are inherently non-portable, as different systems use different argument-passing conventions.

The type **va_list** shall be defined for variables used to traverse the list.

The *va_start*( ) macro is invoked to initialize *ap* to the beginning of the list before any calls to *va_arg*( ).

The *va_copy*( ) macro initializes as a copy of *src*, as if the *va_start*( ) macro had been applied to *dest* followed by the same sequence of uses of the *va_arg*( ) macro as had previously been used to reach the present state of *src*. Neither the *va_copy*( ) nor *va_start*( ) macro shall be invoked to reinitialize *dest* without an intervening invocation of the *va_end*( ) macro for the same *dest*.

The object *ap* may be passed as an argument to another function; if that function invokes the *va_arg*( ) macro with parameter *ap*, the value of *ap* in the calling function is unspecified and shall be passed to the *va_end*( ) macro prior to any further reference to *ap*. The parameter *argN* is the identifier of the rightmost parameter in the variable parameter list in the function definition (the one just before the ...). If the parameter *argN* is declared with the **register** storage class, with a function type or array type, or with a type that is not compatible with the type that results after application of the default argument promotions, the behavior is undefined.

The *va_arg*( ) macro shall return the next argument in the list pointed to by *ap*. Each invocation of *va_arg*( ) modifies *ap* so that the values of successive arguments are returned in turn. The *type* parameter is the type the argument is expected to be. This is the type name specified such that the type of a pointer to an object that has the specified type can be obtained simply by suffixing a `'*'` to type. Different types can be mixed, but it is up to the routine to know what type of argument is expected.

The *va_end*( ) macro is used to clean up; it invalidates *ap* for use (unless *va_start*( ) or *va_copy*( ) is invoked again).

Each invocation of the *va_start*( ) and *va_copy*( ) macros shall be matched by a corresponding invocation of the *va_end*( ) macro in the same function.

Multiple traversals, each bracketed by *va_start*( ) ... *va_end*( ), are possible.

EXAMPLES

This example is a possible implementation of *execl*( ):

```
#include <stdarg.h>
```

```
#define  MAXARGS     31

/*
 * execl is called by
 * execl(file, arg1, arg2, ..., (char *)(0));
 */
int execl(const char *file, const char *args, ...)
{
    va_list ap;
    char *array[MAXARGS];
    int argno = 0;
        va_start(ap, args);
    while (args != 0) {
        array[argno++] = args;
        args = va_arg(ap, const char *);
}
va_end(ap);
return execv(file, array);
}
```

APPLICATION USAGE

It is up to the calling routine to communicate to the called routine how many arguments there are, since it is not always possible for the called routine to determine this in any other way. For example, *execl*( ) is passed a null pointer to signal the end of the list. The *printf*( ) function can tell how many arguments are there by the *format* argument.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *exec*( ), *printf*( )

CHANGE HISTORY

First released in Issue 4. Derived from the ANSI C standard.

NAME

stdbool.h — boolean type and values

SYNOPSIS

```
#include <stdbool.h>
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-200x defers to the ISO C standard.

The **<stdbool.h>** header shall define the following macros:

bool Expands to **_Bool**.

true Expands to the integer constant 1.

false Expands to the integer constant 0.

__bool_true_false_are_defined
Expands to the integer constant 1.

An application may undefine and then possibly redefine the macros bool, true, and false.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

The ability to undefine and redefine the macros bool, true, and false is an obsolescent feature and may be withdrawn in the future.

SEE ALSO

None.

CHANGE HISTORY

First released in Issue 6. Included for alignment with the ISO/IEC 9899: 1999 standard.

NAME

stddef.h — standard type definitions

SYNOPSIS

```
#include <stddef.h>
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-200x defers to the ISO C standard.

The **<stddef.h>** header shall define the following macros:

NULL Null pointer constant.

offsetof(*type*, *member-designator*)
Integer constant expression of type **size_t**, the value of which is the offset in bytes to the structure member (*member-designator*), from the beginning of its structure (*type*).

The **<stddef.h>** header shall define the following types:

ptrdiff_t Signed integer type of the result of subtracting two pointers.

wchar_t Integer type whose range of values can represent distinct wide-character codes for all members of the largest character set specified among the locales supported by the compilation environment: the null character has the code value 0 and each member of the portable character set has a code value equal to its value when used as the lone character in an integer character constant.

size_t Unsigned integer type of the result of the *sizeof* operator.

The implementation shall support one or more programming environments in which the widths of **ptrdiff_t**, **size_t**, and **wchar_t** are no greater than the width of type **long**. The names of these programming environments can be obtained using the *confstr*( ) function or the *getconf* utility.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<wchar.h>, **<sys/types.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *confstr*( ), the Shell and Utilities volume of IEEE Std 1003.1-200x, *getconf*

CHANGE HISTORY

First released in Issue 4. Derived from the ANSI C standard.

NAME

stdint.h — integer types

SYNOPSIS

```
#include <stdint.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these symbols in this header.

The **<stdint.h>** header shall declare sets of integer types having specified widths, and shall define corresponding sets of macros. It shall also define macros that specify limits of integer types corresponding to types defined in other standard headers.

Note: The ‘‘width’’ of an integer type is the number of bits used to store its value in a pure binary system; the actual type may use more bits than that (for example, a 28-bit type could be stored in 32 bits of actual storage). An N-bit signed type has values in the range -2^{N-1} or $1-2^{N-1}$ to $2^{N-1}-1$, while an N-bit unsigned type has values in the range 0 to $2^{N}-1$.

Types are defined in the following categories:

- Integer types having certain exact widths
- Integer types having at least certain specified widths
- Fastest integer types having at least certain specified widths
- Integer types wide enough to hold pointers to objects
- Integer types having greatest width

(Some of these types may denote the same type.)

Corresponding macros specify limits of the declared types and construct suitable constants.

For each type described herein that the implementation provides, the **<stdint.h>** header shall declare that **typedef** name and define the associated macros. Conversely, for each type described herein that the implementation does not provide, the **<stdint.h>** header shall not declare that **typedef** name, nor shall it define the associated macros. An implementation shall provide those types described as required, but need not provide any of the others (described as optional).

Integer Types

When **typedef** names differing only in the absence or presence of the initial *u* are defined, they shall denote corresponding signed and unsigned types as described in the ISO/IEC 9899:1999 standard, Section 6.2.5; an implementation providing one of these corresponding types shall also provide the other.

In the following descriptions, the symbol N represents an unsigned decimal integer with no leading zeros (for example, 8 or 24, but not 04 or 048).

- Exact-width integer types

  The **typedef** name **int*N*_t** designates a signed integer type with width N, no padding bits, and a two's-complement representation. Thus, **int8_t** denotes a signed integer type with a width of exactly 8 bits.

  The **typedef** name **uint*N*_t** designates an unsigned integer type with width N. Thus, **uint24_t** denotes an unsigned integer type with a width of exactly 24 bits.

CX The following types are required:

int8_t
int16_t
int32_t
uint8_t
uint16_t
uint32_t

If an implementation provides integer types with width 64 that meet these requirements, then the following types are required:

int64_t
uint64_t

CX In particular, this will be the case if any of the following are true:

— The implementation supports the _POSIX_V6_ILP32_OFFBIG programming environment and the application is being built in the _POSIX_V6_ILP32_OFFBIG programming environment (see the Shell and Utilities volume of IEEE Std 1003.1-200x, *c99*, Programming Environments).

— The implementation supports the _POSIX_V6_LP64_OFF64 programming environment and the application is being built in the _POSIX_V6_LP64_OFF64 programming environment.

— The implementation supports the _POSIX_V6_LPBIG_OFFBIG programming environment and the application is being built in the _POSIX_V6_LPBIG_OFFBIG programming environment.

All other types are of this form optional.

- Minimum-width integer types

The **typedef** name **int_least***N***_t** designates a signed integer type with a width of at least *N*, such that no signed integer type with lesser size has at least the specified width. Thus, **int_least32_t** denotes a signed integer type with a width of at least 32 bits.

The **typedef** name **uint_least***N***_t** designates an unsigned integer type with a width of at least *N*, such that no unsigned integer type with lesser size has at least the specified width. Thus, **uint_least16_t** denotes an unsigned integer type with a width of at least 16 bits.

The following types are required:

int_least8_t
int_least16_t
int_least32_t
int_least64_t
uint_least8_t
uint_least16_t
uint_least32_t
uint_least64_t

All other types of this form are optional.

- Fastest minimum-width integer types

Each of the following types designates an integer type that is usually fastest to operate with among all integer types that have at least the specified width.

The designated type is not guaranteed to be fastest for all purposes; if the implementation has no clear grounds for choosing one type over another, it will simply pick some integer type satisfying the signedness and width requirements.

The **typedef** name **int_fast***N***_t** designates the fastest signed integer type with a width of at least *N*. The **typedef** name **uint_fast***N***_t** designates the fastest unsigned integer type with a width of at least *N*.

The following types are required:

int_fast8_t
int_fast16_t
int_fast32_t
int_fast64_t
uint_fast8_t
uint_fast16_t
uint_fast32_t
uint_fast64_t

All other types of this form are optional.

- Integer types capable of holding object pointers

  The following type designates a signed integer type with the property that any valid pointer to **void** can be converted to this type, then converted back to a pointer to **void**, and the result will compare equal to the original pointer:

  intptr_t

  The following type designates an unsigned integer type with the property that any valid pointer to **void** can be converted to this type, then converted back to a pointer to **void**, and the result will compare equal to the original pointer:

  uintptr_t

  XSI On XSI-conformant systems, the **intptr_t** and **uintptr_t** types are required;otherwise, they are optional.

- Greatest-width integer types

  The following type designates a signed integer type capable of representing any value of any signed integer type:

  intmax_t

  The following type designates an unsigned integer type capable of representing any value of any unsigned integer type:

  uintmax_t

  These types are required.

Note: Applications can test for optional types by using the corresponding limit macro from **Limits of Specified-Width Integer Types** (on page 316).

Limits of Specified-Width Integer Types

The following macros specify the minimum and maximum limits of the types declared in the **<stdint.h>** header. Each macro name corresponds to a similar type name in **Integer Types** (on page 313).

Each instance of any defined macro shall be replaced by a constant expression suitable for use in **#if** preprocessing directives, and this expression shall have the same type as would an expression that is an object of the corresponding type converted according to the integer promotions. Its implementation-defined value shall be equal to or greater in magnitude (absolute value) than the corresponding value given below, with the same sign, except where stated to be exactly the given value.

- Limits of exact-width integer types
  - Minimum values of exact-width signed integer types:

    {INT*N*_MIN} Exactly $-(2^{N-1})$
  - Maximum values of exact-width signed integer types:

    {INT*N*_MAX} Exactly $2^{N-1} - 1$
  - Maximum values of exact-width unsigned integer types:

    {UINT*N*_MAX} Exactly $2^{N} - 1$
- Limits of minimum-width integer types
  - Minimum values of minimum-width signed integer types:

    {INT_LEAST*N*_MIN} $-(2^{N-1} - 1)$
  - Maximum values of minimum-width signed integer types:

    {INT_LEAST*N*_MAX} $2^{N-1} - 1$
  - Maximum values of minimum-width unsigned integer types:

    {UINT_LEAST*N*_MAX} $2^{N} - 1$
- Limits of fastest minimum-width integer types
  - Minimum values of fastest minimum-width signed integer types:

    {INT_FAST*N*_MIN} $-(2^{N-1} - 1)$
  - Maximum values of fastest minimum-width signed integer types:

    {INT_FAST*N*_MAX} $2^{N-1} - 1$
  - Maximum values of fastest minimum-width unsigned integer types:

    {UINT_FAST*N*_MAX} $2^{N} - 1$
- Limits of integer types capable of holding object pointers
  - Minimum value of pointer-holding signed integer type:

    {INTPTR_MIN} $-(2^{15} - 1)$
  - Maximum value of pointer-holding signed integer type:

    {INTPTR_MAX} $2^{15} - 1$
  - Maximum value of pointer-holding unsigned integer type:

{UINTPTR_MAX}	$2^{16} - 1$

- Limits of greatest-width integer types
  - Minimum value of greatest-width signed integer type:

    {INTMAX_MIN} $-(2^{63} - 1)$
  - Maximum value of greatest-width signed integer type:

    {INTMAX_MAX} $2^{63} - 1$
  - Maximum value of greatest-width unsigned integer type:

    {UINTMAX_MAX} $2^{64} - 1$

Limits of Other Integer Types

The following macros specify the minimum and maximum limits of integer types corresponding to types defined in other standard headers.

Each instance of these macros shall be replaced by a constant expression suitable for use in **#if** preprocessing directives, and this expression shall have the same type as would an expression that is an object of the corresponding type converted according to the integer promotions. Its implementation-defined value shall be equal to or greater in magnitude (absolute value) than the corresponding value given below, with the same sign.

- Limits of **ptrdiff_t**:

{PTRDIFF_MIN}	−65535
{PTRDIFF_MAX}	+65535

- Limits of **sig_atomic_t**:

{SIG_ATOMIC_MIN}	See below.
{SIG_ATOMIC_MAX}	See below.

- Limit of **size_t**:

{SIZE_MAX}	65535

- Limits of **wchar_t**:

{WCHAR_MIN}	See below.
{WCHAR_MAX}	See below.

- Limits of **wint_t**:

{WINT_MIN}	See below.
{WINT_MAX}	See below.

If **sig_atomic_t** (see the **<signal.h>** header) is defined as a signed integer type, the value of {SIG_ATOMIC_MIN} shall be no greater than −127 and the value of {SIG_ATOMIC_MAX} shall be no less than 127; otherwise, **sig_atomic_t** shall be defined as an unsigned integer type, and the value of {SIG_ATOMIC_MIN} shall be 0 and the value of {SIG_ATOMIC_MAX} shall be no less than 255.

If **wchar_t** (see the **<stddef.h>** header) is defined as a signed integer type, the value of {WCHAR_MIN} shall be no greater than −127 and the value of {WCHAR_MAX} shall be no less than 127; otherwise, **wchar_t** shall be defined as an unsigned integer type, and the value of {WCHAR_MIN} shall be 0 and the value of {WCHAR_MAX} shall be no less than 255.

If **wint_t** (see the **<wchar.h>** header) is defined as a signed integer type, the value of {WINT_MIN} shall be no greater than −32767 and the value of {WINT_MAX} shall be no less than 32767; otherwise, **wint_t** shall be defined as an unsigned integer type, and the value of {WINT_MIN} shall be 0 and the value of {WINT_MAX} shall be no less than 65535.

Macros for Integer Constant Expressions

The following macros expand to integer constant expressions suitable for initializing objects that have integer types corresponding to types defined in the **<stdint.h>** header. Each macro name corresponds to a similar type name listed under *Minimum-width integer types* and *Greatest-width integer types*.

Each invocation of one of these macros shall expand to an integer constant expression suitable for use in **#if** preprocessing directives. The type of the expression shall have the same type as would an expression that is an object of the corresponding type converted according to the integer promotions. The value of the expression shall be that of the argument.

The argument in any instance of these macros shall be a decimal, octal, or hexadecimal constant with a value that does not exceed the limits for the corresponding type.

- Macros for minimum-width integer constant expressions

  The macro *INTN_C*(*value*) shall expand to an integer constant expression corresponding to the type **int_least***N***_t**. The macro *UINTN_C*(*value*) shall expand to an integer constant expression corresponding to the type **uint_least***N***_t**. For example, if **uint_least64_t** is a name for the type **unsigned long long**, then *UINT64_C*(0x123) might expand to the integer constant 0x123ULL.

- Macros for greatest-width integer constant expressions

  The following macro expands to an integer constant expression having the value specified by its argument and the type **intmax_t**:

  INTMAX_C(*value*)

  The following macro expands to an integer constant expression having the value specified by its argument and the type **uintmax_t**:

  UINTMAX_C(*value*)

APPLICATION USAGE

None.

RATIONALE

The **<stdint.h>** header is a subset of the **<inttypes.h>** header more suitable for use in freestanding environments, which might not support the formatted I/O functions. In some environments, if the formatted conversion support is not wanted, using this header instead of the **<inttypes.h>** header avoids defining such a large number of macros.

As a consequence of adding **int8_t** the following are true:

- A byte is exactly 8 bits.

- {CHAR_BIT} has the value 8, {SCHAR_MAX} has the value 127, {SCHAR_MIN} has the value −127 or −128, and {UCHAR_MAX} has the value 255.

FUTURE DIRECTIONS

typedef names beginning with **int** or **uint** and ending with _t may be added to the types defined in the **<stdint.h>** header. Macro names beginning with INT or UINT and ending with _MAX, _MIN, or _C may be added to the macros defined in the **<stdint.h>** header.

SEE ALSO

<signal.h>, **<stddef.h>**, **<wchar.h>**, **<inttypes.h>**

CHANGE HISTORY

First released in Issue 6. Included for alignment with the ISO/IEC 9899: 1999 standard.

ISO/IEC 9899: 1999 standard, Technical Corrigendum No. 1 is incorporated.

NAME

stdio.h — standard buffered input/output

SYNOPSIS

```
#include <stdio.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these symbols in this header.

The **<stdio.h>** header shall define the following macros as positive integer constant expressions:

	BUFSIZ	Size of **<stdio.h>** buffers.
	_IOFBF	Input/output fully buffered.
	_IOLBF	Input/output line buffered.
	_IONBF	Input/output unbuffered.
CX	L_ctermid	Maximum size of character array to hold *ctermid*( ) output.
	L_tmpnam	Maximum size of character array to hold *tmpnam*( ) output.
	SEEK_CUR	Seek relative to current position.
	SEEK_END	Seek relative to end-of-file.
	SEEK_SET	Seek relative to start-of-file.

The following macros shall be defined as positive integer constant expressions which denote implementation limits:

	{FILENAME_MAX}	Maximum size in bytes of the longest filename string that the implementation guarantees can be opened.
	{FOPEN_MAX}	Number of streams which the implementation guarantees can be open simultaneously. The value is at least eight.
XSI	{TMP_MAX}	Minimum number of unique filenames generated by *tmpnam*( ). Maximum number of times an application can call *tmpnam*( ) reliably. The value of {TMP_MAX} is at least 25. On XSI-conformant systems, the value of {TMP_MAX} is at least 10,000.

The following macro name shall be defined as a negative integer constant expression:

	EOF	End-of-file return value.

The following macro name shall be defined as a null pointer constant:

	NULL	Null pointer.

The following macro name shall be defined as a string constant:

XSI	P_tmpdir	Default directory prefix for *tempnam*( ).

The following shall be defined as expressions of type ‘‘pointer to **FILE**’’ that point to the **FILE** objects associated, respectively, with the standard error, input, and output streams:

stderr	Standard error output stream.
stdin	Standard input stream.

stdout Standard output stream.

The following data types shall be defined through **typedef**:

FILE A structure containing information about a file.

fpos_t A non-array type containing all information needed to specify uniquely every position within a file.

XSI **va_list** As described in **<stdarg.h>**.

size_t As described in **<stddef.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
     void     clearerr(FILE *);
CX   char    *ctermid(char *);
     int      fclose(FILE *);
CX   FILE    *fdopen(int, const char *);
     int      feof(FILE *);
     int      ferror(FILE *);
     int      fflush(FILE *);
     int      fgetc(FILE *);
     int      fgetpos(FILE *restrict, fpos_t *restrict);
     char    *fgets(char *restrict, int, FILE *restrict);
CX   int      fileno(FILE *);
TSF  void     flockfile(FILE *);
     FILE    *fopen(const char *restrict, const char *restrict);
     int      fprintf(FILE *restrict, const char *restrict, ...);
     int      fputc(int, FILE *);
     int      fputs(const char *restrict, FILE *restrict);
     size_t   fread(void *restrict, size_t, size_t, FILE *restrict);
     FILE    *freopen(const char *restrict, const char *restrict,
                  FILE *restrict);
     int      fscanf(FILE *restrict, const char *restrict, ...);
     int      fseek(FILE *, long, int);
CX   int      fseeko(FILE *, off_t, int);
     int      fsetpos(FILE *, const fpos_t *);
     long     ftell(FILE *);
CX   off_t    ftello(FILE *);
TSF  int      ftrylockfile(FILE *);
     void     funlockfile(FILE *);
     size_t   fwrite(const void *restrict, size_t, size_t, FILE *restrict);
     int      getc(FILE *);
     int      getchar(void);
TSF  int      getc_unlocked(FILE *);
     int      getchar_unlocked(void);
     char    *gets(char *);
CX   int      pclose(FILE *);
     void     perror(const char *);
CX   FILE    *popen(const char *, const char *);
     int      printf(const char *restrict, ...);
     int      putc(int, FILE *);
     int      putchar(int);
TSF
```

```
        int      putc_unlocked(int, FILE *);
        int      putchar_unlocked(int);
        int      puts(const char *);
        int      remove(const char *);
        int      rename(const char *, const char *);
        void     rewind(FILE *);
        int      scanf(const char *restrict, ...);
        void     setbuf(FILE *restrict, char *restrict);
        int      setvbuf(FILE *restrict, char *restrict, int, size_t);
        int      snprintf(char *restrict, size_t, const char *restrict, ...);
        int      sprintf(char *restrict, const char *restrict, ...);
        int      sscanf(const char *restrict, const char *restrict, int ...);
XSI     char    *tempnam(const char *, const char *);
        FILE    *tmpfile(void);
        char    *tmpnam(char *);
        int      ungetc(int, FILE *);
        int      vfprintf(FILE *restrict, const char *restrict, va_list);
        int      vfscanf(FILE *restrict, const char *restrict, va_list);
        int      vprintf(const char *restrict, va_list);
        int      vscanf(const char *restrict, va_list);
        int      vsnprintf(char *restrict, size_t, const char *restrict, va_list;
        int      vsprintf(char *restrict, const char *restrict, va_list);
        int      vsscanf(const char *restrict, const char *restrict, va_list arg);
```

XSI Inclusion of the **<stdio.h>** header may also make visible all symbols from **<stddef.h>**.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *clearerr*( ), *ctermid*( ), *fclose*( ), *fdopen*( ), *fgetc*( ), *fgetpos*( ), *ferror*( ), *feof*( ), *fflush*( ), *fgets*( ), *fileno*( ), *flockfile*( ), *fopen*( ), *fputc*( ), *fputs*( ), *fread*( ), *freopen*( ), *fseek*( ), *fsetpos*( ), *ftell*( ), *fwrite*( ), *getc*( ), *getc_unlocked*( ), *getwchar*( ), *getchar*( ), *getopt*( ), *gets*( ), *pclose*( ), *perror*( ), *popen*( ), *printf*( ), *putc*( ), *putchar*( ), *puts*( ), *putwchar*( ), *remove*( ), *rename*( ), *rewind*( ), *scanf*( ), *setbuf*( ), *setvbuf*( ), *sscanf*( ), *stdin*, *system*( ), *tempnam*( ), *tmpfile*( ), *tmpnam*( ), *ungetc*( ), *vfscanf*( ), *vscanf*( ), *vprintf*( ), *vsscanf*( )

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Large File System extensions are added.

The constant L_cuserid and the external variables *optarg*, *opterr*, *optind*, and *optopt* are marked as extensions and LEGACY.

The *cuserid*( ) and *getopt*( ) functions are marked LEGACY.

Issue 6

The constant L_cuserid and the external variables *optarg*, *opterr*, *optind*, and *optopt* are removed as they were previously marked LEGACY.

The *cuserid*( ), *getopt*( ), and *getw*( ) functions are removed as they were previously marked LEGACY. |

Several functions are marked as part of the _POSIX_THREAD_SAFE_FUNCTIONS option.

This reference page is updated to align with the ISO/IEC 9899:1999 standard. Note that the description of the **fpos_t** type is now explicitly updated to exclude array types.

Extensions beyond the ISO C standard are now marked. |

NAME

stdlib.h — standard library definitions

SYNOPSIS

```
#include <stdlib.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these symbols in this header.

The **<stdlib.h>** header shall define the following macros:

EXIT_FAILURE Unsuccessful termination for *exit*( ); evaluates to a non-zero value.

EXIT_SUCCESS Successful termination for *exit*( ); evaluates to 0.

NULL Null pointer.

{RAND_MAX} Maximum value returned by *rand*( ); at least 32,767.

{MB_CUR_MAX} Integer expression whose value is the maximum number of bytes in a character specified by the current locale.

The following data types shall be defined through **typedef**:

div_t Structure type returned by the *div*( ) function.

ldiv_t Structure type returned by the *ldiv*( ) function.

lldiv_t Structure type returned by the *lldiv*( ) function.

size_t As described in **<stddef.h>**.

wchar_t As described in **<stddef.h>**.

In addition, the following symbolic names and macros shall be defined as in **<sys/wait.h>**, for use in decoding the return value from *system*( ):

XSI WNOHANG
WUNTRACED
WEXITSTATUS
WIFEXITED
WIFSIGNALED
WIFSTOPPED
WSTOPSIG
WTERMSIG

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
void          _Exit(int);
long          a64l(const char *);
void          abort(void);
int           abs(int);
int           atexit(void (*)(void));
double        atof(const char *);
int           atoi(const char *);
long          atol(const char *);
```

XSI (a64l)

```
     long long     atoll(const char *);
     void         *bsearch(const void *, const void *, size_t, size_t,
                       int (*)(const void *, const void *));
     void         *calloc(size_t, size_t);
     div_t         div(int, int);
XSI  double        drand48(void);
     char         *ecvt(double, int, int *restrict, int *restrict); (LEGACY)
     double        erand48(unsigned short[3]);
     void          exit(int);
XSI  char         *fcvt(double, int, int *restrict, int *restrict); (LEGACY)
     void          free(void *);
XSI  char         *gcvt(double, int, char *); (LEGACY)
     char         *getenv(const char *);
XSI  int           getsubopt(char **, char *const *, char **);
     int           grantpt(int);
     char         *initstate(unsigned, char *, size_t);
     long          jrand48(unsigned short[3]);
     char         *l64a(long);
     long          labs(long);
XSI  void          lcong48(unsigned short[7]);
     ldiv_t        ldiv(long, long);
     long long     llabs(long long);
     lldiv_t       lldiv(long long, long long);
XSI  long          lrand48(void);
     void         *malloc(size_t);
     int           mblen(const char *, size_t);
     size_t        mbstowcs(wchar_t *restrict, const char *restrict, size_t);
     int           mbtowc(wchar_t *restrict, const char *restrict, size_t);
XSI  char         *mktemp(char *); (LEGACY)
     int           mkstemp(char *);
     long          mrand48(void);
     long          nrand48(unsigned short[3]);
ADV  int           posix_memalign(void **, size_t, size_t);
XSI  int           posix_openpt(int);
     char         *ptsname(int);
     int           putenv(char *);
     void          qsort(void *, size_t, size_t, int (*)(const void *,
                       const void *));
     int           rand(void);
TSF  int           rand_r(unsigned *);
XSI  long          random(void);
     void         *realloc(void *, size_t);
XSI  char         *realpath(const char *restrict, char *restrict);
     unsigned short seed48(unsigned short[3]);
CX   int           setenv(const char *, const char *, int);
XSI  void          setkey(const char *);
     char         *setstate(const char *);
     void          srand(unsigned);
XSI  void          srand48(long);
     void          srandom(unsigned);
     double        strtod(const char *restrict, char **restrict);
     float         strtof(const char *restrict, char **restrict);
```

```
        long           strtol(const char *restrict, char **restrict, int);
        long double    strtold(const char *restrict, char **restrict);
        long long      strtoll(const char *restrict, char **restrict, int);
        unsigned long  strtoul(const char *restrict, char **restrict, int);
        long long      strtoull(const char *restrict, char **restrict, int);
        int            system(const char *);
XSI     int            unlockpt(int);
CX      int            unsetenv(const char *);
        size_t         wcstombs(char *restrict, const wchar_t *restrict, size_t);
        int            wctomb(char *, wchar_t);
```

XSI Inclusion of the **<stdlib.h>** header may also make visible all symbols from **<stddef.h>**, **<limits.h>**, **<math.h>**, and **<sys/wait.h>**.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *_Exit*( ), *a64l*( ), *abort*( ), *abs*( ), *atexit*( ), *atof*( ), *atoi*( ), *atol*( ), *atoll*( ), *bsearch*( ), *calloc*( ), *div*( ), *drand48*( ), *erand48*( ), *exit*( ), *free*( ), *getenv*( ), *getsubopt*( ), *grantpt*( ), *initstate*( ), *jrand48*( ), *l64a*( ), *labs*( ), *lcong48*( ), *ldiv*( ), *llabs*( ), *lldiv*( ), *lrand48*( ), *malloc*( ), *mblen*( ), *mbstowcs*( ), *mbtowc*( ), *mkstemp*( ), *mrand48*( ), *nrand48*( ), *posix_memalign*( ), *ptsname*( ), *putenv*( ), *qsort*( ), *rand*( ), *realloc*( ), *realpath*( ), *setstate*( ), *srand*( ), *srand48*( ), *srandom*( ), *strtod*( ), *strtof*( ), *strtol*( ), *strtold*( ), *strtoll*( ), *strtoul*( ), *strtoull*( ), *unlockpt*( ), *wcstombs*( ), *wctomb*( )

CHANGE HISTORY

First released in Issue 3.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

The *ttyslot*( ) and *valloc*( ) functions are marked LEGACY.

The type of the third argument to *initstate*( ) is changed from **int** to **size_t**. The type of the return value from *setstate*( ) is changed from **char** to **char ***, and the type of the first argument is changed from **char *** to **const char ***.

Issue 6

The Open Group Corrigendum U021/1 is applied, correcting the prototype for *realpath*( ) to be consistent with the reference page.

The Open Group Corrigendum U028/13 is applied, correcting the prototype for *putenv*( ) to be consistent with the reference page.

The *rand_r*( ) function is marked as part of the _POSIX_THREAD_SAFE_FUNCTIONS option.

Function prototypes for *setenv*( ) and *unsetenv*( ) are added.

The *posix_memalign*( ) function is added for alignment with IEEE Std 1003.1d-1999.

This reference page is updated to align with the ISO/IEC 9899:1999 standard.

The *ecvt*( ), *fcvt*( ), *gcvt*( ), and *mktemp*( ) functions are marked LEGACY.

The *ttyslot*( ) and *valloc*( ) functions are removed as they were previously marked LEGACY.

Extensions beyond the ISO C standard are now marked.

NAME

string.h — string operations

SYNOPSIS

```
#include <string.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these symbols in this header.

The **<string.h>** header shall define the following:

NULL	Null pointer constant.
size_t	As described in **<stddef.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
void    *memccpy(void *restrict, const void *restrict, int, size_t);
void    *memchr(const void *, int, size_t);
int      memcmp(const void *, const void *, size_t);
void    *memcpy(void *restrict, const void *restrict, size_t);
void    *memmove(void *, const void *, size_t);
void    *memset(void *, int, size_t);
char    *strcat(char *restrict, const char *restrict);
char    *strchr(const char *, int);
int      strcmp(const char *, const char *);
int      strcoll(const char *, const char *);
char    *strcpy(char *restrict, const char *restrict);
size_t   strcspn(const char *, const char *);
char    *strdup(const char *);
char    *strerror(int);
size_t   strlen(const char *);
char    *strncat(char *restrict, const char *restrict, size_t);
int      strncmp(const char *, const char *, size_t);
char    *strncpy(char *restrict, const char *restrict, size_t);
char    *strpbrk(const char *, const char *);
char    *strrchr(const char *, int);
size_t   strspn(const char *, const char *);
char    *strstr(const char *, const char *);
char    *strtok(char *restrict, const char *restrict);
char    *strtok_r(char *, const char *, char **);
size_t   strxfrm(char *restrict, const char *restrict, size_t);
```

XSI Inclusion of the **<string.h>** header may also make visible all symbols from **<stddef.h>**.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *memccpy*( ), *memchr*( ), *memcmp*( ), *memcpy*( ), *memmove*( ), *memset*( ), *strcat*( ), *strchr*( ), *strcmp*( ), *strcoll*( ), *strcpy*( ), *strcspn*( ), *strdup*( ), *strerror*( ), *strlen*( ), *strncat*( ), *strncmp*( ), *strncpy*( ), *strpbrk*( ), *strrchr*( ), *strspn*( ), *strstr*( ), *strtok*( ), *strxfrm*( )

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The *strtok_r*( ) function is marked as part of the _POSIX_THREAD_SAFE_FUNCTIONS option.

This reference page is updated to align with the ISO / IEC 9899: 1999 standard.

NAME

strings.h — string operations

SYNOPSIS

XSI

```
#include <strings.h>
```

DESCRIPTION

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int    bcmp(const void *, const void *, size_t); (LEGACY)
void   bcopy(const void *, void *, size_t); (LEGACY)
void   bzero(void *, size_t); (LEGACY)
int    ffs(int);
char  *index(const char *, int); (LEGACY)
char  *rindex(const char *, int); (LEGACY)
int    strcasecmp(const char *, const char *);
int    strncasecmp(const char *, const char *, size_t);
```

The **size_t** type shall be defined through **typedef** as described in **<stddef.h>**.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<stddef.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *ffs*( ), *strcasecmp*( ), *strncasecmp*( )

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 6

The Open Group Corrigendum U021/2 is applied, correcting the prototype for *index*( ) to be consistent with the reference page.

The *bcmp*( ), *bcopy*( ), *bzero*( ), *index*( ), and *rindex*( ) functions are marked LEGACY.

NAME

stropts.h — STREAMS interface (**STREAMS**)

SYNOPSIS

XSR

```
#include <stropts.h>
```

DESCRIPTION

The **<stropts.h>** header shall define the **bandinfo** structure that includes at least the following members:

`unsigned char`	`bi_pri`	Priority band.
`int`	`bi_flag`	Flushing type.

The **<stropts.h>** header shall define the **strpeek** structure that includes at least the following members:

`struct strbuf`	`ctlbuf`	The control portion of the message.
`struct strbuf`	`databuf`	The data portion of the message.
`t_uscalar_t`	`flags`	RS_HIPRI or 0.

The **<stropts.h>** header shall define the **strbuf** structure that includes at least the following members:

`int`	`maxlen`	Maximum buffer length.
`int`	`len`	Length of data.
`char`	`*buf`	Pointer to buffer.

The **<stropts.h>** header shall define the **strfdinsert** structure that includes at least the following members:

`struct strbuf`	`ctlbuf`	The control portion of the message.
`struct strbuf`	`databuf`	The data portion of the message.
`t_uscalar_t`	`flags`	RS_HIPRI or 0.
`int`	`fildes`	File descriptor of the other STREAM.
`int`	`offset`	Relative location of the stored value.

The **<stropts.h>** header shall define the **strioctl** structure that includes at least the following members:

`int`	`ic_cmd`	*ioctl*( ) command.
`int`	`ic_timout`	Timeout for response.
`int`	`ic_len`	Length of data.
`char`	`*ic_dp`	Pointer to buffer.

The **<stropts.h>** header shall define the **strrecvfd** structure that includes at least the following members:

`int`	`fda`	Received file descriptor.
`uid_t`	`uid`	UID of sender.
`gid_t`	`gid`	GID of sender.

The **uid_t** and **gid_t** types shall be defined through **typedef** as described in **<sys/types.h>**.

The **<stropts.h>** header shall define the **t_scalar_t** and **t_uscalar_t** types respectively as signed and unsigned opaque types of equal length of at least 32 bits.

The **<stropts.h>** header shall define the **str_list** structure that includes at least the following members:

`int`	`sl_nmods`	Number of STREAMS module names.
`struct str_mlist`	`*sl_modlist`	STREAMS module names.

The **<stropts.h>** header shall define the **str_mlist** structure that includes at least the following member:

`char`	`l_name[FMNAMESZ+1]`	A STREAMS module name.

At least the following macros shall be defined for use as the *request* argument to *ioctl*( ):

I_PUSH	Push a STREAMS module.
I_POP	Pop a STREAMS module.
I_LOOK	Get the top module name.
I_FLUSH	Flush a STREAM.
I_FLUSHBAND	Flush one band of a STREAM.
I_SETSIG	Ask for notification signals.
I_GETSIG	Retrieve current notification signals.
I_FIND	Look for a STREAMS module.
I_PEEK	Peek at the top message on a STREAM.
I_SRDOPT	Set the read mode.
I_GRDOPT	Get the read mode.
I_NREAD	Size the top message.
I_FDINSERT	Send implementation-defined information about another STREAM.
I_STR	Send a STREAMS *ioctl*( ).
I_SWROPT	Set the write mode.
I_GWROPT	Get the write mode.
I_SENDFD	Pass a file descriptor through a STREAMS pipe.
I_RECVFD	Get a file descriptor sent via I_SENDFD.
I_LIST	Get all the module names on a STREAM.
I_ATMARK	Is the top message ‘‘marked’’?
I_CKBAND	See if any messages exist in a band.
I_GETBAND	Get the band of the top message on a STREAM.
I_CANPUT	Is a band writable?
I_SETCLTIME	Set close time delay.
I_GETCLTIME	Get close time delay.
I_LINK	Connect two STREAMs.
I_UNLINK	Disconnect two STREAMs.
I_PLINK	Persistently connect two STREAMs.
I_PUNLINK	Dismantle a persistent STREAMS link.

At least the following macros shall be defined for use with I_LOOK:

FMNAMESZ The minimum size in bytes of the buffer referred to by the *arg* argument.

At least the following macros shall be defined for use with I_FLUSH:

FLUSHR Flush read queues.

FLUSHW Flush write queues.

FLUSHRW Flush read and write queues.

At least the following macros shall be defined for use with I_SETSIG:

S_RDNORM A normal (priority band set to 0) message has arrived at the head of a STREAM head read queue.

S_RDBAND A message with a non-zero priority band has arrived at the head of a STREAM head read queue.

S_INPUT A message, other than a high-priority message, has arrived at the head of a STREAM head read queue.

S_HIPRI A high-priority message is present on a STREAM head read queue.

S_OUTPUT The write queue for normal data (priority band 0) just below the STREAM head is no longer full. This notifies the process that there is room on the queue for sending (or writing) normal data downstream.

S_WRNORM Equivalent to S_OUTPUT.

S_WRBAND The write queue for a non-zero priority band just below the STREAM head is no longer full.

S_MSG A STREAMS signal message that contains the SIGPOLL signal reaches the front of the STREAM head read queue.

S_ERROR Notification of an error condition reaches the STREAM head.

S_HANGUP Notification of a hangup reaches the STREAM head.

S_BANDURG When used in conjunction with S_RDBAND, SIGURG is generated instead of SIGPOLL when a priority message reaches the front of the STREAM head read queue.

At least the following macros shall be defined for use with I_PEEK:

RS_HIPRI Only look for high-priority messages.

At least the following macros shall be defined for use with I_SRDOPT:

RNORM Byte-STREAM mode, the default.

RMSGD Message-discard mode.

RMSGN Message-nondiscard mode.

RPROTNORM Fail *read*( ) with [EBADMSG] if a message containing a control part is at the front of the STREAM head read queue.

RPROTDAT Deliver the control part of a message as data when a process issues a *read*( ).

RPROTDIS Discard the control part of a message, delivering any data part, when a process issues a *read*( ).

At least the following macros shall be defined for use with I_SWOPT:

SNDZERO	Send a zero-length message downstream when a *write*( ) of 0 bytes occurs.

At least the following macros shall be defined for use with I_ATMARK:

ANYMARK	Check if the message is marked.
LASTMARK	Check if the message is the last one marked on the queue.

At least the following macros shall be defined for use with I_UNLINK:

MUXID_ALL	Unlink all STREAMs linked to the STREAM associated with *fildes*.

The following macros shall be defined for *getmsg*( ), *getpmsg*( ), *putmsg*( ), and *putpmsg*( ):

MSG_ANY	Receive any message.
MSG_BAND	Receive message from specified band.
MSG_HIPRI	Send/receive high-priority message.
MORECTL	More control information is left in message.
MOREDATA	More data is left in message.

The **<stropts.h>** header may make visible all of the symbols from **<unistd.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int    isastream(int);
int    getmsg(int, struct strbuf *restrict, struct strbuf *restrict,
           int *restrict);
int    getpmsg(int, struct strbuf *restrict, struct strbuf *restrict,
           int *restrict, int *restrict);
int    ioctl(int, int, ... );
int    putmsg(int, const struct strbuf *, const struct strbuf *, int);
int    putpmsg(int, const struct strbuf *, const struct strbuf *, int,
           int);
int    fattach(int, const char *);
int    fdetach(const char *);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *close*( ), *fcntl*( ), *getmsg*( ), *ioctl*( ), *open*( ), *pipe*( ), *read*( ), *poll*( ), *putmsg*( ), *signal*( ), *write*( )

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

The *flags* member of the **strpeek** and **strfdinsert** structures are changed from **type long** to **t_uscalar_t**.

Issue 6

This header is marked as part of the XSI STREAMS Option Group.

The **restrict** keyword is added to the prototypes for *getmsg*( ) and *getpmsg*( ).

NAME

sys/ipc.h — XSI interprocess communication access structure

SYNOPSIS

XSI

```
#include <sys/ipc.h>
```

DESCRIPTION

The **<sys/ipc.h>** header is used by three mechanisms for XSI interprocess communication (IPC): messages, semaphores, and shared memory. All use a common structure type, **ipc_perm** to pass information used in determining permission to perform an IPC operation.

The **ipc_perm** structure shall contain the following members:

`uid_t`	`uid`	Owner's user ID.
`gid_t`	`gid`	Owner's group ID.
`uid_t`	`cuid`	Creator's user ID.
`gid_t`	`cgid`	Creator's group ID.
`mode_t`	`mode`	Read/write permission.

The **uid_t**, **gid_t**, **mode_t**, and **key_t** types shall be defined as described in **<sys/types.h>**.

Definitions shall be provided for the following constants:

Mode bits:

IPC_CREAT	Create entry if key does not exist.
IPC_EXCL	Fail if key exists.
IPC_NOWAIT	Error if request must wait.

Keys:

IPC_PRIVATE	Private key.

Control commands:

IPC_RMID	Remove identifier.
IPC_SET	Set options.
IPC_STAT	Get options.

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided. |

```
key_t  ftok(const char *, int);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *ftok*( )

CHANGE HISTORY

First released in Issue 2. Derived from System V Release 2.0.

NAME

sys/mman.h — memory management declarations

SYNOPSIS

```
#include <sys/mman.h>
```

DESCRIPTION

The **<sys/mman.h>** header shall be supported if the implementation supports at least one of the following options:

- MF The Memory Mapped Files option
- SHM The Shared Memory Objects option
- ML The Process Memory Locking option
- MPR The Memory Protection option
- TYM The Typed Memory Objects option
- SIO The Synchronized Input and Output option
- ADV The Advisory Information option
- TYM The Typed Memory Objects option

MC2 If one or more of the Advisory Information, Memory Mapped Files, or Shared Memory Objects options are supported, the following protection options shall be defined:

MC2	PROT_READ	Page can be read.
MC2	PROT_WRITE	Page can be written.
MC2	PROT_EXEC	Page can be executed.
MC2	PROT_NONE	Page cannot be accessed.

The following *flag* options shall be defined:

MF|SHM	MAP_SHARED	Share changes.
MF|SHM	MAP_PRIVATE	Changes are private.
MF|SHM	MAP_FIXED	Interpret *addr* exactly.

The following flags shall be defined for *msync*( ):

MF|SIO	MS_ASYNC	Perform asynchronous writes.
MF|SIO	MS_SYNC	Perform synchronous writes.
MF|SIO	MS_INVALIDATE	Invalidate mappings.

ML The following symbolic constants shall be defined for the *mlockall*( ) function:

ML	MCL_CURRENT	Lock currently mapped pages.
ML	MCL_FUTURE	Lock pages that become mapped.

MF|SHM The symbolic constant MAP_FAILED shall be defined to indicate a failure from the *mmap*( ) function.

MC1 If the Advisory Information and either the Memory Mapped Files or Shared Memory Objects options are supported, values for *advice* used by *posix_madvise*( ) shall be defined as follows:

POSIX_MADV_NORMAL
The application has no advice to give on its behavior with respect to the specified range. It

is the default characteristic if no advice is given for a range of memory.

POSIX_MADV_SEQUENTIAL
The application expects to access the specified range sequentially from lower addresses to higher addresses.

POSIX_MADV_RANDOM
The application expects to access the specified range in a random order.

POSIX_MADV_WILLNEED
The application expects to access the specified range in the near future.

POSIX_MADV_DONTNEED
The application expects that it will not access the specified range in the near future.

TYM The following flags shall be defined for *posix_typed_mem_open*( ):

POSIX_TYPED_MEM_ALLOCATE
Allocate on *mmap*( ).

POSIX_TYPED_MEM_ALLOCATE_CONTIG
Allocate contiguously on *mmap*( ).

POSIX_TYPED_MEM_MAP_ALLOCATABLE Map on *mmap*( ), without affecting allocatability.

The **mode_t**, **off_t**, and **size_t** types shall be defined as described in **<sys/types.h>**.

TYM The **<sys/mman.h>** header shall define the structure **posix_typed_mem_info**, which includes at least the following member:

```
size_t  posix_tmi_length   Maximum length which may be allocated
                           from a typed memory object.
```

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided. |

```
ML       int    mlock(const void *, size_t);
         int    mlockall(int);
MF|SHM   void  *mmap(void *, size_t, int, int, int, off_t);
MPR      int    mprotect(void *, size_t, int);
MF|SIO   int    msync(void *, size_t, int);
ML       int    munlock(const void *, size_t);
         int    munlockall(void);
MF|SHM   int    munmap(void *, size_t);
ADV      int    posix_madvise(void *, size_t, int);
TYM      int    posix_mem_offset(const void *restrict, size_t, off_t *restrict,
                    size_t *restrict, int *restrict);
         int    posix_typed_mem_get_info(int, struct posix_typed_mem_info *);
         int    posix_typed_mem_open(const char *, int, int);
SHM      int    shm_open(const char *, int, mode_t);
         int    shm_unlink(const char *);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *mlock*( ), *mlockall*( ), *mmap*( ), *mprotect*( ), *msync*( ), *munlock*( ), *munlockall*( ), *munmap*( ), *posix_mem_offset*( ), *posix_typed_mem_get_info*( ), *posix_typed_mem_open*( ), *shm_open*( ), *shm_unlink*( )

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Updated for alignment with the POSIX Realtime Extension.

Issue 6

The **<sys/mman.h>** header is marked as dependent on support for either the _POSIX_MAPPED_FILES, _POSIX_MEMLOCK, or _POSIX_SHARED_MEMORY options.

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- The TYM margin code is added to the list of margin codes for the **<sys/mman.h>** header line, as well as for other lines.
- The POSIX_TYPED_MEM_ALLOCATE, POSIX_TYPED_MEM_ALLOCATE_CONTIG, and POSIX_TYPED_MEM_MAP_ALLOCATABLE flags are added.
- The **posix_tmi_length** structure is added.
- The *posix_mem_offset*( ), *posix_typed_mem_get_info*( ), and *posix_typed_mem_open*( ) functions are added.

The **restrict** keyword is added to the prototype for *posix_mem_offset*( ).

IEEE PASC Interpretation 1003.1 #102 is applied adding the prototype for *posix_madvise*( ).

NAME

sys/msg.h — XSI message queue structures

SYNOPSIS

XSI
```
#include <sys/msg.h>
```

DESCRIPTION

The **<sys/msg.h>** header shall define the following constant and members of the structure **msqid_ds**.

The following data types shall be defined through **typedef**:

msgqnum_t	Used for the number of messages in the message queue.
msglen_t	Used for the number of bytes allowed in a message queue.

These types shall be unsigned integer types that are able to store values at least as large as a type **unsigned short**.

Message operation flag:

MSG_NOERROR	No error if big message.

The **msqid_ds** structure shall contain the following members:

`struct ipc_perm`	`msg_perm`	Operation permission structure.
`msgqnum_t`	`msg_qnum`	Number of messages currently on queue.
`msglen_t`	`msg_qbytes`	Maximum number of bytes allowed on queue.
`pid_t`	`msg_lspid`	Process ID of last *msgsnd*( ).
`pid_t`	`msg_lrpid`	Process ID of last *msgrcv*( ).
`time_t`	`msg_stime`	Time of last *msgsnd*( ).
`time_t`	`msg_rtime`	Time of last *msgrcv*( ).
`time_t`	`msg_ctime`	Time of last change.

The **pid_t**, **time_t**, **key_t**, **size_t**, and **ssize_t** types shall be defined as described in **<sys/types.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided. |

```
int       msgctl(int, int, struct msqid_ds *);
int       msgget(key_t, int);
ssize_t   msgrcv(int, void *, size_t, long, int);
int       msgsnd(int, const void *, size_t, int);
```

In addition, all of the symbols from **<sys/ipc.h>** shall be defined when this header is included.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>, *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( )

CHANGE HISTORY

First released in Issue 2. Derived from System V Release 2.0.

NAME

sys/resource.h — definitions for XSI resource operations

SYNOPSIS

XSI
```
#include <sys/resource.h>
```

DESCRIPTION

The **<sys/resource.h>** header shall define the following symbolic constants as possible values of the *which* argument of *getpriority*( ) and *setpriority*( ):

PRIO_PROCESS	Identifies the *who* argument as a process ID.
PRIO_PGRP	Identifies the *who* argument as a process group ID.
PRIO_USER	Identifies the *who* argument as a user ID.

The following type shall be defined through **typedef**:

rlim_t	Unsigned integer type used for limit values.

The following symbolic constants shall be defined:

RLIM_INFINITY	A value of **rlim_t** indicating no limit.
RLIM_SAVED_MAX	A value of type **rlim_t** indicating an unrepresentable saved hard limit.
RLIM_SAVED_CUR	A value of type **rlim_t** indicating an unrepresentable saved soft limit.

On implementations where all resource limits are representable in an object of type **rlim_t**, RLIM_SAVED_MAX and RLIM_SAVED_CUR need not be distinct from RLIM_INFINITY.

The following symbolic constants shall be defined as possible values of the *who* parameter of *getrusage*( ):

RUSAGE_SELF	Returns information about the current process.
RUSAGE_CHILDREN	Returns information about children of the current process.

The **<sys/resource.h>** header shall define the **rlimit** structure that includes at least the following members:

`rlim_t rlim_cur`	The current (soft) limit.
`rlim_t rlim_max`	The hard limit.

The **<sys/resource.h>** header shall define the **rusage** structure that includes at least the following members:

`struct timeval ru_utime`	User time used.
`struct timeval ru_stime`	System time used.

The **timeval** structure shall be defined as described in **<sys/time.h>**.

The following symbolic constants shall be defined as possible values for the *resource* argument of *getrlimit*( ) and *setrlimit*( ):

RLIMIT_CORE	Limit on size of core dump file.
RLIMIT_CPU	Limit on CPU time per process.
RLIMIT_DATA	Limit on data segment size.
RLIMIT_FSIZE	Limit on file size.

RLIMIT_NOFILE	Limit on number of open files.
RLIMIT_STACK	Limit on stack size.
RLIMIT_AS	Limit on address space size.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int  getpriority(int, id_t);
int  getrlimit(int, struct rlimit *);
int  getrusage(int, struct rusage *);
int  setpriority(int, id_t, int);
int  setrlimit(int, const struct rlimit *);
```

The **id_t** type shall be defined through **typedef** as described in **<sys/types.h>**.

Inclusion of the **<sys/resource.h>** header may also make visible all symbols from **<sys/time.h>**.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/time.h>, **<sys/types.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *getpriority*( ), *getrusage*( ), *getrlimit*( )

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Large File System extensions are added.

NAME

sys/select.h — select types

SYNOPSIS

```
#include <sys/select.h>
```

DESCRIPTION

The **<sys/select.h>** header shall define the **timeval** structure that includes at least the following members:

`time_t`	`tv_sec`	Seconds.
`suseconds_t`	`tv_usec`	Microseconds.

The **time_t** and **suseconds_t** types shall be defined as described in **<sys/types.h>**.

The **sigset_t** type shall be defined as described in **<signal.h>**.

The **timespec** structure shall be defined as described in **<time.h>**.

The **<sys/select.h>** header shall define the **fd_set** type as a structure.

Each of the following may be declared as a function, or defined as a macro, or both:

void *FD_CLR*(int *fd*, fd_set **fdset*)
Clears the bit for the file descriptor *fd* in the file descriptor set *fdset*.

int *FD_ISSET*(int *fd*, fd_set **fdset*)
Returns a non-zero value if the bit for the file descriptor *fd* is set in the file descriptor set by *fdset*, and 0 otherwise.

void *FD_SET*(int *fd*, fd_set **fdset*)
Sets the bit for the file descriptor *fd* in the file descriptor set *fdset*.

void *FD_ZERO*(fd_set **fdset*)
Initializes the file descriptor set *fdset* to have zero bits for all file descriptors.

If implemented as macros, these may evaluate their arguments more than once, so applications should ensure that the arguments they supply are never expressions with side effects.

The following shall be defined as a macro:

FD_SETSIZE
Maximum number of file descriptors in an **fd_set** structure.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int  pselect(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
         const struct timespec *restrict, const sigset_t *restrict);
int  select(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
         struct timeval *restrict);
```

Inclusion of the **<sys/select.h>** header may make visible all symbols from the headers **<signal.h>**, **<sys/time.h>**, and **<time.h>**.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<signal.h>, **<sys/time.h>**, **<sys/types.h>**, **<time.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *pselect*( ), *select*( )

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1g-2000. |

The requirement for the **fd_set** structure to have a member *fds_bits* has been removed as per The Open Group Base Resolution bwg2001-005. |

NAME

sys/sem.h — XSI semaphore facility

SYNOPSIS

XSI

```
#include <sys/sem.h>
```

DESCRIPTION

The **<sys/sem.h>** header shall define the following constants and structures.

Semaphore operation flags:

SEM_UNDO	Set up adjust on exit entry.

Command definitions for the *semctl*( ) function shall be provided as follows:

GETNCNT	Get *semncnt*.
GETPID	Get *sempid*.
GETVAL	Get *semval*.
GETALL	Get all cases of *semval*.
GETZCNT	Get *semzcnt*.
SETVAL	Set *semval*.
SETALL	Set all cases of *semval*.

The **semid_ds** structure shall contain the following members:

`struct ipc_perm`	`sem_perm`	Operation permission structure.
`unsigned short`	`sem_nsems`	Number of semaphores in set.
`time_t`	`sem_otime`	Last *semop*( ) time.
`time_t`	`sem_ctime`	Last time changed by *semctl*( ).

The **pid_t**, **time_t**, **key_t**, and **size_t** types shall be defined as described in **<sys/types.h>**.

A semaphore shall be represented by an anonymous structure containing the following members:

`unsigned short`	`semval`	Semaphore value.
`pid_t`	`sempid`	Process ID of last operation.
`unsigned short`	`semncnt`	Number of processes waiting for *semval* to become greater than current value.
`unsigned short`	`semzcnt`	Number of processes waiting for *semval* to become 0.

The **sembuf** structure shall contain the following members:

`unsigned short`	`sem_num`	Semaphore number.
`short`	`sem_op`	Semaphore operation.
`short`	`sem_flg`	Operation flags.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int   semctl(int, int, int, ...);
int   semget(key_t, int, int);
int   semop(int, struct sembuf *, size_t);
```

In addition, all of the symbols from **<sys/ipc.h>** shall be defined when this header is included.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>, *semctl*( ), *semget*( ), *semop*( )

CHANGE HISTORY

First released in Issue 2. Derived from System V Release 2.0.

NAME

sys/shm.h — XSI shared memory facility

SYNOPSIS

XSI

```
#include <sys/shm.h>
```

DESCRIPTION

The **<sys/shm.h>** header shall define the following symbolic constants:

SHM_RDONLY	Attach read-only (else read-write).
SHM_RND	Round attach address to SHMLBA.

The **<sys/shm.h>** header shall define the following symbolic value:

SHMLBA	Segment low boundary address multiple.

The following data types shall be defined through **typedef**:

shmatt_t	Unsigned integer used for the number of current attaches that must be able to store values at least as large as a type **unsigned short**.

The **shmid_ds** structure shall contain the following members:

`struct ipc_perm`	`shm_perm`	Operation permission structure.
`size_t`	`shm_segsz`	Size of segment in bytes.
`pid_t`	`shm_lpid`	Process ID of last shared memory operation.
`pid_t`	`shm_cpid`	Process ID of creator.
`shmatt_t`	`shm_nattch`	Number of current attaches.
`time_t`	`shm_atime`	Time of last *shmat*( ).
`time_t`	`shm_dtime`	Time of last *shmdt*( ).
`time_t`	`shm_ctime`	Time of last change by *shmctl*( ).

The **pid_t**, **time_t**, **key_t**, and **size_t** types shall be defined as described in **<sys/types.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
void *shmat(int, const void *, int);
int   shmctl(int, int, struct shmid_ds *);
int   shmdt(const void *);
int   shmget(key_t, size_t, int);
```

In addition, all of the symbols from **<sys/ipc.h>** shall be defined when this header is included.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *shmat*( ), *shmctl*( ), *shmdt*( ), *shmget*( )

CHANGE HISTORY

First released in Issue 2. Derived from System V Release 2.0.

Issue 5

The type of *shm_segsz* is changed from **int** to **size_t**.

NAME

sys/socket.h — main sockets header

SYNOPSIS

```
#include <sys/socket.h>
```

DESCRIPTION

The **<sys/socket.h>** header shall define the type **socklen_t**, which is an integer type of width of at least 32 bits; see APPLICATION USAGE.

The **<sys/socket.h>** header shall define the unsigned integer type **sa_family_t**.

The **<sys/socket.h>** header shall define the **sockaddr** structure that includes at least the following members:

`sa_family_t`	`sa_family`	Address family.
`char`	`sa_data[]`	Socket address (variable-length data).

The **sockaddr** structure is used to define a socket address which is used in the *bind*( ), *connect*( ), *getpeername*( ), *getsockname*( ), *recvfrom*( ), and *sendto*( ) functions.

The **<sys/socket.h>** header shall define the **sockaddr_storage** structure. This structure shall be:

- Large enough to accommodate all supported protocol-specific address structures
- Aligned at an appropriate boundary so that pointers to it can be cast as pointers to protocol-specific address structures and used to access the fields of those structures without alignment problems

The **sockaddr_storage** structure shall contain at least the following members:

```
sa_family_t   ss_family
```

When a **sockaddr_storage** structure is cast as a **sockaddr** structure, the *ss_family* field of the **sockaddr_storage** structure shall map onto the *sa_family* field of the **sockaddr** structure. When a **sockaddr_storage** structure is cast as a protocol-specific address structure, the *ss_family* field shall map onto a field of that structure that is of type **sa_family_t** and that identifies the protocol's address family.

The **<sys/socket.h>** header shall define the **msghdr** structure that includes at least the following members:

`void`	`*msg_name`	Optional address.
`socklen_t`	`msg_namelen`	Size of address.
`struct iovec`	`*msg_iov`	Scatter/gather array.
`int`	`msg_iovlen`	Members in msg_iov.
`void`	`*msg_control`	Ancillary data; see below.
`socklen_t`	`msg_controllen`	Ancillary data buffer len.
`int`	`msg_flags`	Flags on received message.

The **msghdr** structure is used to minimize the number of directly supplied parameters to the *recvmsg*( ) and *sendmsg*( ) functions. This structure is used as a *value*-*result* parameter in the *recvmsg*( ) function and *value* only for the *sendmsg*( ) function.

The **iovec** structure shall be defined as described in **<sys/uio.h>**.

The **<sys/socket.h>** header shall define the **cmsghdr** structure that includes at least the following members:

`socklen_t`	`cmsg_len`	Data byte count, including the cmsghdr.
`int`	`cmsg_level`	Originating protocol.

`int` `cmsg_type` Protocol-specific type.

The **cmsghdr** structure is used for storage of ancillary data object information.

Ancillary data consists of a sequence of pairs, each consisting of a **cmsghdr** structure followed by a data array. The data array contains the ancillary data message, and the **cmsghdr** structure contains descriptive information that allows an application to correctly parse the data.

The values for *cmsg_level* shall be legal values for the *level* argument to the *getsockopt*( ) and *setsockopt*( ) functions. The system documentation shall specify the *cmsg_type* definitions for the supported protocols.

Ancillary data is also possible at the socket level. The **<sys/socket.h>** header defines the following macro for use as the *cmsg_type* value when *cmsg_level* is SOL_SOCKET:

SCM_RIGHTS Indicates that the data array contains the access rights to be sent or received.

The **<sys/socket.h>** header defines the following macros to gain access to the data arrays in the ancillary data associated with a message header:

CMSG_DATA(*cmsg*)
If the argument is a pointer to a **cmsghdr** structure, this macro shall return an unsigned character pointer to the data array associated with the **cmsghdr** structure.

CMSG_NXTHDR(*mhdr,cmsg*)
If the first argument is a pointer to a **msghdr** structure and the second argument is a pointer to a **cmsghdr** structure in the ancillary data pointed to by the *msg_control* field of that **msghdr** structure, this macro shall return a pointer to the next **cmsghdr** structure, or a null pointer if this structure is the last **cmsghdr** in the ancillary data.

CMSG_FIRSTHDR(*mhdr*)
If the argument is a pointer to a **msghdr** structure, this macro shall return a pointer to the first **cmsghdr** structure in the ancillary data associated with this **msghdr** structure, or a null pointer if there is no ancillary data associated with the **msghdr** structure.

The **<sys/socket.h>** header shall define the **linger** structure that includes at least the following members:

`int` `l_onoff` Indicates whether linger option is enabled.
`int` `l_linger` Linger time, in seconds.

The **<sys/socket.h>** header shall define the following macros, with distinct integer values:

SOCK_DGRAM Datagram socket.

RS SOCK_RAW Raw Protocol Interface.

SOCK_SEQPACKET Sequenced-packet socket.

SOCK_STREAM Byte-stream socket.

The **<sys/socket.h>** header shall define the following macro for use as the *level* argument of *setsockopt*( ) and *getsockopt*( ).

SOL_SOCKET Options to be accessed at socket level, not protocol level.

The **<sys/socket.h>** header shall define the following macros, with distinct integer values, for use as the *option_name* argument in *getsockopt*( ) or *setsockopt*( ) calls:

SO_ACCEPTCONN Socket is accepting connections.

SO_BROADCAST	Transmission of broadcast messages is supported.
SO_DEBUG	Debugging information is being recorded.
SO_DONTROUTE	Bypass normal routing.
SO_ERROR	Socket error status.
SO_KEEPALIVE	Connections are kept alive with periodic messages.
SO_LINGER	Socket lingers on close.
SO_OOBINLINE	Out-of-band data is transmitted in line.
SO_RCVBUF	Receive buffer size.
SO_RCVLOWAT	Receive ‘‘low water mark’’.
SO_RCVTIMEO	Receive timeout.
SO_REUSEADDR	Reuse of local addresses is supported.
SO_SNDBUF	Send buffer size.
SO_SNDLOWAT	Send ‘‘low water mark’’.
SO_SNDTIMEO	Send timeout.
SO_TYPE	Socket type.

The **<sys/socket.h>** header shall define the following macro as the maximum *backlog* queue length which may be specified by the *backlog* field of the *listen*( ) function:

SOMAXCONN	The maximum *backlog* queue length.

The **<sys/socket.h>** header shall define the following macros, with distinct integer values, for use as the valid values for the *msg_flags* field in the **msghdr** structure, or the *flags* parameter in *recvfrom*( ), *recvmsg*( ), *sendmsg*( ), or *sendto*( ) calls:

MSG_CTRUNC	Control data truncated.
MSG_DONTROUTE	Send without using routing tables.
MSG_EOR	Terminates a record (if supported by the protocol).
MSG_OOB	Out-of-band data.
MSG_PEEK	Leave received data in queue.
MSG_TRUNC	Normal data truncated.
MSG_WAITALL	Attempt to fill the read buffer.

The **<sys/socket.h>** header shall define the following macros, with distinct integer values:

	AF_INET	Internet domain sockets for use with IPv4 addresses.
IP6	AF_INET6	Internet domain sockets for use with IPv6 addresses.
	AF_UNIX	UNIX domain sockets.
	AF_UNSPEC	Unspecified .

The **<sys/socket.h>** header shall define the following macros, with distinct integer values:

SHUT_RD	Disables further receive operations.

SHUT_RDWR	Disables further send and receive operations.
SHUT_WR	Disables further send operations.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int     accept(int, struct sockaddr *restrict, socklen_t *restrict);
int     bind(int, const struct sockaddr *, socklen_t);
int     connect(int, const struct sockaddr *, socklen_t);
int     getpeername(int, struct sockaddr *restrict, socklen_t *restrict);
int     getsockname(int, struct sockaddr *restrict, socklen_t *restrict);
int     getsockopt(int, int, int, void *restrict, socklen_t *restrict);
int     listen(int, int);
ssize_t recv(int, void *, size_t, int);
ssize_t recvfrom(int, void *restrict, size_t, int,
        struct sockaddr *restrict, socklen_t *restrict);
ssize_t recvmsg(int, struct msghdr *, int);
ssize_t send(int, const void *, size_t, int);
ssize_t sendmsg(int, const struct msghdr *, int);
ssize_t sendto(int, const void *, size_t, int, const struct sockaddr *,
        socklen_t);
int     setsockopt(int, int, int, const void *, socklen_t);
int     shutdown(int, int);
int     socket(int, int, int);
int     socketpair(int, int, int, int[2]);
```

Inclusion of **<sys/socket.h>** may also make visible all symbols from **<sys/uio.h>**.

APPLICATION USAGE

To forestall portability problems, it is recommended that applications not use values larger than $2^{31}-1$ for the **socklen_t** type.

The **sockaddr_storage** structure solves the problem of declaring storage for automatic variables which is both large enough and aligned enough for storing the socket address data structure of any family. For example, code with a file descriptor and without the context of the address family can pass a pointer to a variable of this type, where a pointer to a socket address structure is expected in calls such as *getpeername*( ), and determine the address family by accessing the received content after the call.

The example below illustrates a data structure which aligns on a 64-bit boundary. An implementation-defined field *_ss_align* following *_ss_pad1* is used to force a 64-bit alignment which covers proper alignment good enough for needs of at least **sockaddr_in6** (IPv6) and **sockaddr_in** (IPv4) address data structures. The size of padding field *_ss_pad1* depends on the chosen alignment boundary. The size of padding field *_ss_pad2* depends on the value of overall size chosen for the total size of the structure. This size and alignment are represented in the above example by implementation-defined (not required) constants _SS_MAXSIZE (chosen value 128) and _SS_ALIGNMENT (with chosen value 8). Constants _SS_PAD1SIZE (derived value 6) and _SS_PAD2SIZE (derived value 112) are also for illustration and not required. The implementation-defined definitions and structure field names above start with an underscore to denote implementation private name space. Portable code is not expected to access or reference those fields or constants.

```
/*
 *  Desired design of maximum size and alignment.
 */
```

```
#define _SS_MAXSIZE 128
    /* Implementation-defined maximum size. */
#define _SS_ALIGNSIZE (sizeof(int64_t))
    /* Implementation-defined desired alignment. */

/*
 *  Definitions used for sockaddr_storage structure paddings design.
 */
#define _SS_PAD1SIZE (_SS_ALIGNSIZE − sizeof(sa_family_t))
#define _SS_PAD2SIZE (_SS_MAXSIZE − (sizeof(sa_family_t)+
                      _SS_PAD1SIZE + _SS_ALIGNSIZE))
struct sockaddr_storage {
    sa_family_t  ss_family;  /* Address family. */
/*
 * Following fields are implementation-defined. */
 */
    char _ss_pad1[_SS_PAD1SIZE];
        /* 6-byte pad; this is to make implementation-defined
           pad up to alignment field that follows explicit in
           the data structure. */
    int64_t _ss_align;  /* Field to force desired structure
                           storage alignment. */
    char _ss_pad2[_SS_PAD2SIZE];
        /* 112-byte pad to achieve desired size,
           _SS_MAXSIZE value minus size of ss_family
           __ss_pad1, __ss_align fields is 112. */
};
```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/uio.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *accept*( ), *bind*( ), *connect*( ), *getpeername*( ), *getsockname*( ), *getsockopt*( ), *listen*( ), *recv*( ), *recvfrom*( ), *recvmsg*( ), *send*( ), *sendmsg*( ), *sendto*( ), *setsockopt*( ), *shutdown*( ), *socket*( ), *socketpair*( )

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The **restrict** keyword is added to the prototypes for *accept*( ), *getpeername*( ), *getsockname*( ), *getsockopt*( ), and *recvfrom*( ).

NAME

sys/stat.h — data returned by the stat( ) function

SYNOPSIS

```
#include <sys/stat.h>
```

DESCRIPTION

The **<sys/stat.h>** header shall define the structure of the data returned by the functions *fstat*( ), *lstat*( ), and *stat*( ).

The **stat** structure shall contain at least the following members:

	`dev_t`	`st_dev`	ID of device containing file.
	`ino_t`	`st_ino`	File serial number.
	`mode_t`	`st_mode`	Mode of file (see below).
	`nlink_t`	`st_nlink`	Number of hard links to the file.
	`uid_t`	`st_uid`	User ID of file.
	`gid_t`	`st_gid`	Group ID of file.
XSI	`dev_t`	`st_rdev`	Device ID (if file is character or block special).
	`off_t`	`st_size`	For regular files, the file size in bytes. For symbolic links, the length in bytes of the pathname contained in the symbolic link.
SHM			For a shared memory object, the length in bytes.
TYM			For a typed memory object, the length in bytes.
			For other file types, the use of this field is unspecified
	`time_t`	`st_atime`	Time of last access.
	`time_t`	`st_mtime`	Time of last data modification.
	`time_t`	`st_ctime`	Time of last status change.
XSI	`blksize_t`	`st_blksize`	A file system-specific preferred I/O block size for this object. In some file system types, this may vary from file to file.
	`blkcnt_t`	`st_blocks`	Number of blocks allocated for this object.

File serial number and device ID taken together uniquely identify the file within the system. The **blkcnt_t**, **blksize_t**, **dev_t**, **ino_t**, **mode_t**, **nlink_t**, **uid_t**, **gid_t**, **off_t**, and **time_t** types shall be defined as described in **<sys/types.h>**. Times shall be given in seconds since the Epoch.

Unless otherwise specified, the structure members *st_mode*, *st_ino*, *st_dev*, *st_uid*, *st_gid*, *st_atime*, *st_ctime*, and *st_mtime* shall have meaningful values for all file types defined in IEEE Std 1003.1-200x.

For symbolic links, the *st_mode* member shall contain meaningful information, which can be used with the file type macros described below, that take a *mode* argument. The *st_size* member shall contain the length, in bytes, of the pathname contained in the symbolic link. File mode bits and the contents of the remaining members of the **stat** structure are unspecified. The value returned in the *st_size* field shall be the length of the contents of the symbolic link, and shall not count a trailing null if one is present.

The following symbolic names for the values of type *mode_t* shall also be defined.

File type:

XSI S_IFMT Type of file.

S_IFBLK Block special.

S_IFCHR Character special.
S_IFIFO FIFO special.
S_IFREG Regular.
S_IFDIR Directory.
S_IFLNK Symbolic link.
S_IFSOCK Socket.

File mode bits:

S_IRWXU Read, write, execute/search by owner.
S_IRUSR Read permission, owner.
S_IWUSR Write permission, owner.
S_IXUSR Execute/search permission, owner.

S_IRWXG Read, write, execute/search by group.
S_IRGRP Read permission, group.
S_IWGRP Write permission, group.
S_IXGRP Execute/search permission, group.

S_IRWXO Read, write, execute/search by others.
S_IROTH Read permission, others.
S_IWOTH Write permission, others.
S_IXOTH Execute/search permission, others.

S_ISUID Set-user-ID on execution.

S_ISGID Set-group-ID on execution.

XSI S_ISVTX On directories, restricted deletion flag.

XSI The bits defined by S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, S_IWGRP, S_IXGRP, S_IROTH, S_IWOTH, S_IXOTH, S_ISUID, S_ISGID, and S_ISVTX shall be unique.

S_IRWXU is the bitwise-inclusive OR of S_IRUSR, S_IWUSR, and S_IXUSR.

S_IRWXG is the bitwise-inclusive OR of S_IRGRP, S_IWGRP, and S_IXGRP.

S_IRWXO is the bitwise-inclusive OR of S_IROTH, S_IWOTH, and S_IXOTH.

Implementations may OR other implementation-defined bits into S_IRWXU, S_IRWXG, and S_IRWXO, but they shall not overlap any of the other bits defined in this volume of IEEE Std 1003.1-200x. The *file permission bits* are defined to be those corresponding to the bitwise-inclusive OR of S_IRWXU, S_IRWXG, and S_IRWXO.

The following macros shall be provided to test whether a file is of the specified type. The value *m* supplied to the macros is the value of *st_mode* from a **stat** structure. The macro shall evaluate to a non-zero value if the test is true; 0 if the test is false.

S_ISBLK(*m*) Test for a block special file.

S_ISCHR(*m*) Test for a character special file.

S_ISDIR(*m*) Test for a directory.

S_ISFIFO(*m*) Test for a pipe or FIFO special file.

S_ISREG(*m*) Test for a regular file.

S_ISLNK(*m*) Test for a symbolic link.

S_ISSOCK(*m*) Test for a socket.

The implementation may implement message queues, semaphores, or shared memory objects as distinct file types. The following macros shall be provided to test whether a file is of the specified type. The value of the *buf* argument supplied to the macros is a pointer to a **stat** structure. The macro shall evaluate to a non-zero value if the specified object is implemented as a distinct file type and the specified file type is contained in the **stat** structure referenced by *buf*. Otherwise, the macro shall evaluate to zero.

S_TYPEISMQ(*buf*) Test for a message queue.

S_TYPEISSEM(*buf*) Test for a semaphore.

S_TYPEISSHM(*buf*) Test for a shared memory object.

TYM The implementation may implement typed memory objects as distinct file types, and the following macro shall test whether a file is of the specified type. The value of the *buf* argument supplied to the macros is a pointer to a **stat** structure. The macro shall evaluate to a non-zero value if the specified object is implemented as a distinct file type and the specified file type is contained in the **stat** structure referenced by *buf*. Otherwise, the macro shall evaluate to zero.

S_TYPEISTMO(*buf*) Test macro for a typed memory object.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int    chmod(const char *, mode_t);
int    fchmod(int, mode_t);
int    fstat(int, struct stat *);
int    lstat(const char *restrict, struct stat *restrict);
int    mkdir(const char *, mode_t);
int    mkfifo(const char *, mode_t);
int    mknod(const char *, mode_t, dev_t);
int    stat(const char *restrict, struct stat *restrict);
mode_t umask(mode_t);
```

APPLICATION USAGE

Use of the macros is recommended for determining the type of a file.

RATIONALE

A conforming C-language application must include **<sys/stat.h>** for functions that have arguments or return values of type **mode_t**, so that symbolic values for that type can be used. An alternative would be to require that these constants are also defined by including **<sys/types.h>**.

The S_ISUID and S_ISGID bits may be cleared on any write, not just on *open*( ), as some historical implementations do it.

System calls that update the time entry fields in the **stat** structure must be documented by the implementors. POSIX-conforming systems should not update the time entry fields for functions listed in the System Interfaces volume of IEEE Std 1003.1-200x unless the standard requires that they do, except in the case of documented extensions to the standard.

Note that *st_dev* must be unique within a Local Area Network (LAN) in a ‘‘system’’ made up of multiple computers’ file systems connected by a LAN.

Networked implementations of a POSIX-conforming system must guarantee that all files visible within the file tree (including parts of the tree that may be remotely mounted from other machines on the network) on each individual processor are uniquely identified by the combination of the *st_ino* and *st_dev* fields.

FUTURE DIRECTIONS

No new S_IFMT symbolic names for the file type values of **mode_t** will be defined by IEEE Std 1003.1-200x; if new file types are required, they will only be testable through *S_ISxx*( ) macros instead.

SEE ALSO

<sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *chmod*( ), *fchmod*( ), *fstat*( ), *lstat*( ), *mkdir*( ), *mkfifo*( ), *mknod*( ), *stat*( ), *umask*( )

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with POSIX Realtime Extension.

The type of *st_blksize* is changed from **long** to **blksize_t**; the type of *st_blocks* is changed from **long** to **blkcnt_t**.

Issue 6

The S_TYPEISMQ( ), S_TYPEISSEM( ), and S_TYPEISSHM( ) macros are unconditionally mandated.

The Open Group Corrigendum U035/4 is applied. In the DESCRIPTION, the types **blksize_t** and **blkcnt_t** have been described.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The **dev_t**, **ino_t**, **mode_t**, **nlink_t**, **uid_t**, **gid_t**, **off_t**, and **time_t** types are mandated.

S_IFSOCK and S_ISSOCK are added for sockets.

The description of **stat** structure members is changed to reflect contents when file type is a symbolic link.

The test macro S_TYPEISTMO is added for alignment with IEEE Std 1003.1j-2000.

The **restrict** keyword is added to the prototypes for *lstat*( ) and *stat*( ).

The *lstat*( ) function is now mandatory.

NAME

sys/statvfs.h — VFS File System information structure

SYNOPSIS

XSI

```
#include <sys/statvfs.h>
```

DESCRIPTION

The **<sys/statvfs.h>** header shall define the **statvfs** structure that includes at least the following members:

`unsigned long`	`f_bsize`	File system block size.
`unsigned long`	`f_frsize`	Fundamental file system block size.
`fsblkcnt_t`	`f_blocks`	Total number of blocks on file system in units of *f_frsize*.
`fsblkcnt_t`	`f_bfree`	Total number of free blocks.
`fsblkcnt_t`	`f_bavail`	Number of free blocks available to non-privileged process.
`fsfilcnt_t`	`f_files`	Total number of file serial numbers.
`fsfilcnt_t`	`f_ffree`	Total number of free file serial numbers.
`fsfilcnt_t`	`f_favail`	Number of file serial numbers available to non-privileged process.
`unsigned long`	`f_fsid`	File system ID.
`unsigned long`	`f_flag`	Bit mask of *f_flag* values.
`unsigned long`	`f_namemax`	Maximum filename length.

The **fsblkcnt_t** and **fsfilcnt_t** types shall be defined as described in **<sys/types.h>**.

The following flags for the *f_flag* member shall be defined:

`ST_RDONLY`	Read-only file system.
`ST_NOSUID`	Does not support setuid/setgid semantics.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int statvfs(const char *restrict, struct statvfs *restrict);
int fstatvfs(int, struct statvfs *);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *fstatvfs*( ), *statvfs*( )

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

The type of *f_blocks*, *f_bfree*, and *f_bavail* is changed from **unsigned long** to **fsblkcnt_t**; the type of *f_files*, *f_ffree*, and *f_favail* is changed from **unsigned long** to **fsfilcnt_t**.

Issue 6

The Open Group Corrigendum U035/5 is applied. In the DESCRIPTION, the types **fsblkcnt_t** and **fsfilcnt_t** have been described.

The **restrict** keyword is added to the prototype for *statvfs*( ).

NAME

sys/time.h — time types

SYNOPSIS

XSI

```
#include <sys/time.h>
```

DESCRIPTION

The **<sys/time.h>** header shall define the **timeval** structure that includes at least the following members:

`time_t`	`tv_sec`	Seconds.
`suseconds_t`	`tv_usec`	Microseconds.

The **<sys/time.h>** header shall define the **itimerval** structure that includes at least the following members:

`struct timeval`	`it_interval`	Timer interval.
`struct timeval`	`it_value`	Current value.

The **time_t** and **suseconds_t** types shall be defined as described in **<sys/types.h>**.

The **fd_set** type shall be defined as described in **<sys/select.h>**.

The **<sys/time.h>** header shall define the following values for the *which* argument of *getitimer*( ) and *setitimer*( ):

ITIMER_REAL	Decrements in real time.
ITIMER_VIRTUAL	Decrements in process virtual time.
ITIMER_PROF	Decrements both in process virtual time and when the system is running on behalf of the process.

The following shall be defined as described in **<sys/select.h>**:

FD_CLR( )
FD_ISSET( )
FD_SET( )
FD_ZERO( )
FD_SETSIZE( )

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int   getitimer(int, struct itimerval *);
int   gettimeofday(struct timeval *restrict, void *restrict);
int   select(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
          struct timeval *restrict);
int   setitimer(int, const struct itimerval *restrict,
          struct itimerval *restrict);
int   utimes(const char *, const struct timeval [2]); (LEGACY)
```

Inclusion of the **<sys/time.h>** header may make visible all symbols from the **<sys/select.h>** header.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *getitimer*( ), *gettimeofday*( ), *select*( ), *setitimer*( )

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

The type of *tv_usec* is changed from **long** to **suseconds_t**.

Issue 6

The **restrict** keyword is added to the prototypes for *gettimeofday*( ), *select*( ), and *setitimer*( ).

The note is added that inclusion of this header may also make symbols visible from **<sys/socket.h>**.

The *utimes*( ) function is marked LEGACY.

NAME

sys/timeb.h — additional definitions for date and time

SYNOPSIS

XSI
```
#include <sys/timeb.h>
```

DESCRIPTION

The **<sys/timeb.h>** header shall define the **timeb** structure that includes at least the following members:

`time_t`	`time`	The seconds portion of the current time.
`unsigned short`	`millitm`	The milliseconds portion of the current time.
`short`	`timezone`	The local timezone in minutes west of Greenwich.
`short`	`dstflag`	TRUE if Daylight Savings Time is in effect.

The **time_t** type shall be defined as described in **<sys/types.h>**.

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided.

```
int   ftime(struct timeb *); (LEGACY)
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>, **<time.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 6

The *ftime*( ) function is marked LEGACY.

NAME

sys / times.h — file access and modification times structure

SYNOPSIS

```
#include <sys/times.h>
```

DESCRIPTION

The **<sys/times.h>** header shall define the structure **tms**, which is returned by *times*( ) and includes at least the following members:

`clock_t`	`tms_utime`	User CPU time.
`clock_t`	`tms_stime`	System CPU time.
`clock_t`	`tms_cutime`	User CPU time of terminated child processes.
`clock_t`	`tms_cstime`	System CPU time of terminated child processes.

The **clock_t** type shall be defined as described in **<sys/types.h>**.

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided.

```
clock_t times(struct tms *);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *times*( )

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

NAME

sys/types.h — data types

SYNOPSIS

```
#include <sys/types.h>
```

DESCRIPTION

The **<sys/types.h>** header shall include definitions for at least the following types:

	blkcnt_t	Used for file block counts.
	blksize_t	Used for block sizes.
XSI	**clock_t**	Used for system times in clock ticks or CLOCKS_PER_SEC; see **<time.h>**.
TMR	**clockid_t**	Used for clock ID type in the clock and timer functions.
	dev_t	Used for device IDs.
XSI	**fsblkcnt_t**	Used for file system block counts.
XSI	**fsfilcnt_t**	Used for file system file counts.
	gid_t	Used for group IDs.
XSI	**id_t**	Used as a general identifier; can be used to contain at least a **pid_t**, **uid_t**, or **gid_t**.
	ino_t	Used for file serial numbers.
XSI	**key_t**	Used for XSI interprocess communication.
	mode_t	Used for some file attributes.
	nlink_t	Used for link counts.
	off_t	Used for file sizes.
	pid_t	Used for process IDs and process group IDs.
THR	**pthread_attr_t**	Used to identify a thread attribute object.
BAR	**pthread_barrier_t**	Used to identify a barrier.
BAR	**pthread_barrierattr_t**	Used to define a barrier attributes object.
THR	**pthread_cond_t**	Used for condition variables.
THR	**pthread_condattr_t**	Used to identify a condition attribute object.
THR	**pthread_key_t**	Used for thread-specific data keys.
THR	**pthread_mutex_t**	Used for mutexes.
THR	**pthread_mutexattr_t**	Used to identify a mutex attribute object.
THR	**pthread_once_t**	Used for dynamic package initialization.
THR	**pthread_rwlock_t**	Used for read-write locks.
THR	**pthread_rwlockattr_t**	Used for read-write lock attributes.
SPI	**pthread_spinlock_t**	Used to identify a spin lock.
THR	**pthread_t**	Used to identify a thread.

	size_t	Used for sizes of objects.
	ssize_t	Used for a count of bytes or an error indication.
XSI	**suseconds_t**	Used for time in microseconds
	time_t	Used for time in seconds.
TMR	**timer_t**	Used for timer ID returned by *timer_create*( ).
	uid_t	Used for user IDs.
XSI	**useconds_t**	Used for time in microseconds.

All of the types shall be defined as arithmetic types of an appropriate length, with the following exceptions:

XSI **key_t**
THR **pthread_attr_t**
BAR **pthread_barrier_t**
pthread_barrierattr_t
THR **pthread_cond_t**
pthread_condattr_t
pthread_key_t
pthread_mutex_t
pthread_mutexattr_t
pthread_once_t
pthread_rwlock_t
pthread_rwlockattr_t
SPI **pthread_spinlock_t**
TRC **trace_attr_t**
trace_event_id_t
TRC TEF **trace_event_set_t**
TRC **trace_id_t**

Additionally:

- **mode_t** shall be an integer type.
- **nlink_t**, **uid_t**, **gid_t**, and **id_t** shall be integer types.
- **blkcnt_t** and **off_t** shall be signed integer types.
- XSI **fsblkcnt_t**, **fsfilcnt_t**, and **ino_t** shall be defined as unsigned integer types.
- **size_t** shall be an unsigned integer type.
- **blksize_t**, **pid_t**, and **ssize_t** shall be signed integer types.
- **time_t** and **clock_t** shall be integer or real-floating types.

XSI The type **ssize_t** shall be capable of storing values at least in the range [−1, {SSIZE_MAX}]. The type **useconds_t** shall be an unsigned integer type capable of storing values at least in the range [0, 1 000 000]. The type **suseconds_t** shall be a signed integer type capable of storing values at least in the range [−1, 1 000 000].

The implementation shall support one or more programming environments in which the widths of **blksize_t**, **pid_t**, **size_t**, **ssize_t**, **suseconds_t**, and **useconds_t** are no greater than the width of type **long**. The names of these programming environments can be obtained using the *confstr*( ) function or the *getconf* utility.

There are no defined comparison or assignment operators for the following types:

THR **pthread_attr_t**
BAR **pthread_barrier_t**
pthread_barrierattr_t
THR **pthread_cond_t**
pthread_condattr_t
pthread_mutex_t
pthread_mutexattr_t
pthread_rwlock_t
pthread_rwlockattr_t
SPI **pthread_spinlock_t**
TRC **trace_attr_t**

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<time.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *confstr*( ), the Shell and Utilities volume of IEEE Std 1003.1-200x, *getconf*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The **clockid_t** and **timer_t** types are defined for alignment with the POSIX Realtime Extension.

The types **blkcnt_t**, **blksize_t**, **fsblkcnt_t**, **fsfilcnt_t**, and **suseconds_t** are added.

Large File System extensions are added.

Updated for alignment with the POSIX Threads Extension.

Issue 6

The **pthread_barrier_t**, **pthread_barrierattr_t**, and **pthread_spinlock_t** types are added for alignment with IEEE Std 1003.1j-2000.

The margin code is changed from XSI to THR for the **pthread_rwlock_t** and **pthread_rwlockattr_t** types as Read-Write Locks have been absorbed into the POSIX Threads option. The threads types are now marked THR.

NAME

sys/uio.h — definitions for vector I/O operations

SYNOPSIS

XSI

```
#include <sys/uio.h>
```

DESCRIPTION

The **<sys/uio.h>** header shall define the **iovec** structure that includes at least the following members:

```
void    *iov_base   Base address of a memory region for input or output.
size_t   iov_len    The size of the memory pointed to by iov_base.
```

The **<sys/uio.h>** header uses the **iovec** structure for scatter/gather I/O.

The **ssize_t** and **size_t** types shall be defined as described in **<sys/types.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
ssize_t readv(int, const struct iovec *, int);
ssize_t writev(int, const struct iovec *, int);
```

APPLICATION USAGE

The implementation can put a limit on the number of scatter/gather elements which can be processed in one call. The symbol {IOV_MAX} defined in **<limits.h>** should always be used to learn about the limits instead of assuming a fixed value.

RATIONALE

Traditionally, the maximum number of scatter/gather elements the system can process in one call were described by the symbolic value {UIO_MAXIOV}. In IEEE Std 1003.1-200x this value was replaced by the constant {IOV_MAX} which can be found in **<limits.h>**.

FUTURE DIRECTIONS

None.

SEE ALSO

<limits.h>, **<sys/types.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *read*( ), *write*( )

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 6

Text referring to scatter/gather I/O is added to the DESCRIPTION.

NAME

sys/un.h — definitions for UNIX domain sockets

SYNOPSIS

```
#include <sys/un.h>
```

DESCRIPTION

The **<sys/un.h>** header shall define the **sockaddr_un** structure that includes at least the following members:

`sa_family_t`	`sun_family`	Address family.
`char`	`sun_path[]`	Socket pathname.

The **sockaddr_un** structure is used to store addresses for UNIX domain sockets. Values of this type shall be cast by applications to **struct sockaddr** for use with socket functions.

The **sa_family_t** type shall be defined as described in **<sys/socket.h>**.

APPLICATION USAGE

The size of *sun_path* has intentionally been left undefined. This is because different implementations use different sizes. For example, BSD4.3 uses a size of 108, and BSD4.4 uses a size of 104. Since most implementations originate from BSD versions, the size is typically in the range 92 to 108.

Applications should not assume a particular length for *sun_path* or assume that it can hold _POSIX_PATH_MAX characters (255).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/socket.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *bind*( ), *socket*( ), *socketpair*( )

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

NAME

sys/utsname.h — system name structure

SYNOPSIS

```
#include <sys/utsname.h>
```

DESCRIPTION

The **<sys/utsname.h>** header shall define the structure **utsname** which shall include at least the following members:

`char`	`sysname[]`	Name of this implementation of the operating system.
`char`	`nodename[]`	Name of this node within an implementation-defined communications network.
`char`	`release[]`	Current release level of this implementation.
`char`	`version[]`	Current version level of this release.
`char`	`machine[]`	Name of the hardware type on which the system is running.

The character arrays are of unspecified size, but the data stored in them shall be terminated by a null byte.

The following shall be declared as a function and may also be defined as a macro:

```
int uname(struct utsname *);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *uname*( )

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

NAME

sys/wait.h — declarations for waiting

SYNOPSIS

```
#include <sys/wait.h>
```

DESCRIPTION

The **<sys/wait.h>** header shall define the following symbolic constants for use with *waitpid*( ):

WNOHANG	Do not hang if no status is available; return immediately.
WUNTRACED	Report status of stopped child process.

The **<sys/wait.h>** header shall define the following macros for analysis of process status values:

WEXITSTATUS	Return exit status.
WIFCONTINUED	True if child has been continued
WIFEXITED	True if child exited normally.
WIFSIGNALED	True if child exited due to uncaught signal.
WIFSTOPPED	True if child is currently stopped.
WSTOPSIG	Return signal number that caused process to stop.
WTERMSIG	Return signal number that caused process to terminate.

The following symbolic constants shall be defined as possible values for the *options* argument to *waitid*( ):

WEXITED	Wait for processes that have exited.
WSTOPPED	Status is returned for any child that has stopped upon receipt of a signal.
WCONTINUED	Status is returned for any child that was stopped and has been continued.
WNOHANG	Return immediately if there are no children to wait for.
WNOWAIT	Keep the process whose status is returned in *infop* in a waitable state.

The type **idtype_t** shall be defined as an enumeration type whose possible values shall include at least the following:

P_ALL
P_PID
P_PGID

The **id_t** and **pid_t** types shall be defined as described in **<sys/types.h>**.

The **siginfo_t** type shall be defined as described in **<signal.h>**.

The **rusage** structure shall be defined as described in **<sys/resource.h>**.

Inclusion of the **<sys/wait.h>** header may also make visible all symbols from **<signal.h>** and **<sys/resource.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
pid_t  wait(int *);
int    waitid(idtype_t, id_t, siginfo_t *, int);
pid_t  waitpid(pid_t, int *, int);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<signal.h>, **<sys/resource.h>**, **<sys/types.h>**, **<sys/wait.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *wait*( ), *waitid*( )

CHANGE HISTORY

First released in Issue 3.

Entry included for alignment with the POSIX.1-1988 standard.

Issue 6

The *wait3*( ) function is removed.

NAME

syslog — definitions for system error logging

SYNOPSIS

XSI

```
#include <syslog.h>
```

DESCRIPTION

The **<syslog.h>** header shall define the following symbolic constants, zero or more of which may be OR'ed together to form the *logopt* option of *openlog*( ):

LOG_PID	Log the process ID with each message.
LOG_CONS	Log to the system console on error.
LOG_NDELAY	Connect to syslog daemon immediately.
LOG_ODELAY	Delay open until *syslog*( ) is called.
LOG_NOWAIT	Do not wait for child processes.

The following symbolic constants shall be defined as possible values of the *facility* argument to *openlog*( ):

LOG_KERN	Reserved for message generated by the system.
LOG_USER	Message generated by a process.
LOG_MAIL	Reserved for message generated by mail system.
LOG_NEWS	Reserved for message generated by news system.
LOG_UUCP	Reserved for message generated by UUCP system.
LOG_DAEMON	Reserved for message generated by system daemon.
LOG_AUTH	Reserved for message generated by authorization daemon.
LOG_CRON	Reserved for message generated by the clock daemon.
LOG_LPR	Reserved for message generated by printer system.
LOG_LOCAL0	Reserved for local use.
LOG_LOCAL1	Reserved for local use.
LOG_LOCAL2	Reserved for local use.
LOG_LOCAL3	Reserved for local use.
LOG_LOCAL4	Reserved for local use.
LOG_LOCAL5	Reserved for local use.
LOG_LOCAL6	Reserved for local use.
LOG_LOCAL7	Reserved for local use.

The following shall be declared as macros for constructing the *maskpri* argument to *setlogmask*( ). The following macros expand to an expression of type **int** when the argument *pri* is an expression of type **int**:

LOG_MASK(*pri*)	A mask for priority *pri*.

The following constants shall be defined as possible values for the *priority* argument of *syslog*( ):

LOG_EMERG	A panic condition was reported to all processes.
LOG_ALERT	A condition that should be corrected immediately.
LOG_CRIT	A critical condition.
LOG_ERR	An error message.
LOG_WARNING	A warning message.
LOG_NOTICE	A condition requiring special handling.
LOG_INFO	A general information message.
LOG_DEBUG	A message useful for debugging programs.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
void  closelog(void);
void  openlog(const char *, int, int);
int   setlogmask(int);
void  syslog(int, const char *, ...);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *closelog*( )

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved to X/Open UNIX to BASE.

NAME

tar.h — extended tar definitions

SYNOPSIS

```
#include <tar.h>
```

DESCRIPTION

The **<tar.h>** header shall define header block definitions as follows.

General definitions:

Name	Description	Value
TMAGIC	`"ustar"`	ustar plus null byte.
TMAGLEN	`6`	Length of the above.
TVERSION	`"00"`	00 without a null byte.
TVERSLEN	`2`	Length of the above.

Typeflag field definitions:

Name	Description	Value
REGTYPE	`'0'`	Regular file.
AREGTYPE	`'\0'`	Regular file.
LNKTYPE	`'1'`	Link.
SYMTYPE	`'2'`	Symbolic link.
CHRTYPE	`'3'`	Character special.
BLKTYPE	`'4'`	Block special.
DIRTYPE	`'5'`	Directory.
FIFOTYPE	`'6'`	FIFO special.
CONTTYPE	`'7'`	Reserved.

Mode field bit definitions (octal):

Name	Description	Value
TSUID	04000	Set UID on execution.
TSGID	02000	Set GID on execution.
TSVTX	01000	On directories, restricted deletion flag.
TUREAD	00400	Read by owner.
TUWRITE	00200	Write by owner special.
TUEXEC	00100	Execute/search by owner.
TGREAD	00040	Read by group.
TGWRITE	00020	Write by group.
TGEXEC	00010	Execute/search by group.
TOREAD	00004	Read by other.
TOWRITE	00002	Write by other.
TOEXEC	00001	Execute/search by other.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Shell and Utilities volume of IEEE Std 1003.1-200x, *pax*

CHANGE HISTORY

First released in Issue 3. Derived from the entry in the POSIX.1-1988 standard.

Issue 6

The SEE ALSO section now refers to *pax* since the Shell and Utilities volume of IEEE Std 1003.1-200x no longer contains the *tar* utility.

NAME

termios.h — define values for termios

SYNOPSIS

```
#include <termios.h>
```

DESCRIPTION

The **<termios.h>** header contains the definitions used by the terminal I/O interfaces (see Chapter 11 (on page 183) for the structures and names defined).

The termios Structure

The following data types shall be defined through **typedef**:

cc_t Used for terminal special characters.

speed_t Used for terminal baud rates.

tcflag_t Used for terminal modes.

The above types shall be all unsigned integer types.

The implementation shall support one or more programming environments in which the widths of **cc_t**, **speed_t**, and **tcflag_t** are no greater than the width of type **long**. The names of these programming environments can be obtained using the *confstr*( ) function or the *getconf* utility.

The **termios** structure shall be defined, and shall include at least the following members:

```
tcflag_t  c_iflag     Input modes.
tcflag_t  c_oflag     Output modes.
tcflag_t  c_cflag     Control modes.
tcflag_t  c_lflag     Local modes.
cc_t      c_cc[NCCS]  Control characters.
```

A definition shall be provided for:

NCCS Size of the array *c_cc* for control characters.

The following subscript names for the array *c_cc* shall be defined:

Subscript Usage		
Canonical Mode	**Non-Canonical Mode**	**Description**
VEOF		EOF character.
VEOL		EOL character.
VERASE		ERASE character.
VINTR	VINTR	INTR character.
VKILL		KILL character.
	VMIN	MIN value.
VQUIT	VQUIT	QUIT character.
VSTART	VSTART	START character.
VSTOP	VSTOP	STOP character.
VSUSP	VSUSP	SUSP character.
	VTIME	TIME value.

The subscript values shall be unique, except that the VMIN and VTIME subscripts may have the same values as the VEOF and VEOL subscripts, respectively.

The following flags shall be provided.

Input Modes

The *c_iflag* field describes the basic terminal input control:

	BRKINT	Signal interrupt on break.
	ICRNL	Map CR to NL on input.
	IGNBRK	Ignore break condition.
	IGNCR	Ignore CR.
	IGNPAR	Ignore characters with parity errors.
	INLCR	Map NL to CR on input.
	INPCK	Enable input parity check.
	ISTRIP	Strip character.
XSI	IXANY	Enable any character to restart output.
	IXOFF	Enable start/stop input control.
	IXON	Enable start/stop output control.
	PARMRK	Mark parity errors.

Output Modes

The *c_oflag* field specifies the system treatment of output:

	OPOST	Post-process output.	
XSI	ONLCR	Map NL to CR-NL on output.	
	OCRNL	Map CR to NL on output.	
	ONOCR	No CR output at column 0.	
	ONLRET	NL performs CR function.	
	OFILL	Use fill characters for delay.	
	NLDLY	Select newline delays:	
		NL0	<newline> type 0.
		NL1	<newline> type 1.
	CRDLY	Select carriage-return delays:	
		CR0	Carriage-return delay type 0.
		CR1	Carriage-return delay type 1.
		CR2	Carriage-return delay type 2.
		CR3	Carriage-return delay type 3.
	TABDLY	Select horizontal-tab delays:	
		TAB0	Horizontal-tab delay type 0.
		TAB1	Horizontal-tab delay type 1.
		TAB2	Horizontal-tab delay type 2.

	TAB3	Expand tabs to spaces.
BSDLY	Select backspace delays:	
	BS0	Backspace-delay type 0.
	BS1	Backspace-delay type 1.
VTDLY	Select vertical-tab delays:	
	VT0	Vertical-tab delay type 0.
	VT1	Vertical-tab delay type 1.
FFDLY	Select form-feed delays:	
	FF0	Form-feed delay type 0.
	FF1	Form-feed delay type 1.

Baud Rate Selection

The input and output baud rates are stored in the **termios** structure. These are the valid values for objects of type **speed_t**. The following values shall be defined, but not all baud rates need be supported by the underlying hardware.

B0	Hang up
B50	50 baud
B75	75 baud
B110	110 baud
B134	134.5 baud
B150	150 baud
B200	200 baud
B300	300 baud
B600	600 baud
B1200	1200 baud
B1800	1800 baud
B2400	2400 baud
B4800	4800 baud
B9600	9600 baud
B19200	19200 baud
B38400	38400 baud

Control Modes

The *c_cflag* field describes the hardware control of the terminal; not all values specified are required to be supported by the underlying hardware:

CSIZE	Character size:	
	CS5	5 bits
	CS6	6 bits
	CS7	7 bits
	CS8	8 bits
CSTOPB	Send two stop bits, else one.	
CREAD	Enable receiver.	
PARENB	Parity enable.	
PARODD	Odd parity, else even.	
HUPCL	Hang up on last close.	
CLOCAL	Ignore modem status lines.	

The implementation shall support the functionality associated with the symbols CS7, CS8, CSTOPB, PARODD, and PARENB.

Local Modes

The *c_lflag* field of the argument structure is used to control various terminal functions:

ECHO	Enable echo.
ECHOE	Echo erase character as error-correcting backspace.
ECHOK	Echo KILL.
ECHONL	Echo NL.
ICANON	Canonical input (erase and kill processing).
IEXTEN	Enable extended input character processing.
ISIG	Enable signals.
NOFLSH	Disable flush after interrupt or quit.
TOSTOP	Send SIGTTOU for background output.

Attribute Selection

The following symbolic constants for use with *tcsetattr*( ) are defined:

TCSANOW	Change attributes immediately.
TCSADRAIN	Change attributes when output has drained.
TCSAFLUSH	Change attributes when output has drained; also flush pending input.

Line Control

The following symbolic constants for use with *tcflush*( ) shall be defined:

TCIFLUSH Flush pending input. Flush untransmitted output.

TCIOFLUSH Flush both pending input and untransmitted output.

TCOFLUSH Flush untransmitted output.

The following symbolic constants for use with *tcflow*( ) shall be defined:

TCIOFF Transmit a STOP character, intended to suspend input data.

TCION Transmit a START character, intended to restart input data.

TCOOFF Suspend output.

TCOON Restart output.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
speed_t cfgetispeed(const struct termios *);
speed_t cfgetospeed(const struct termios *);
int     cfsetispeed(struct termios *, speed_t);
int     cfsetospeed(struct termios *, speed_t);
int     tcdrain(int);
int     tcflow(int, int);
int     tcflush(int, int);
int     tcgetattr(int, struct termios *);
pid_t   tcgetsid(int);
int     tcsendbreak(int, int);
int     tcsetattr(int, int, const struct termios *);
```

APPLICATION USAGE

The following names are reserved for XSI-conformant systems to use as an extension to the above; therefore strictly conforming applications shall not use them:

CBAUD
DEFECHO
ECHOCTL
ECHOKE
ECHOPRT
EXTA
EXTB
FLUSHO
LOBLK
PENDIN
SWTCH
VDISCARD
VDSUSP
VLNEXT
VREPRINT
VSTATUS
VWERASE

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *cfgetispeed*( ), *cfgetospeed*( ), *cfsetispeed*( ), *cfsetospeed*( ), *getconf*( ), *tcdrain*( ), *tcflow*( ), *tcflush*( ), *tcgetattr*( ), *tcgetsid*( ), *tcsendbreak*( ), *tcsetattr*( ), the Shell and Utilities volume of IEEE Std 1003.1-200x, *getconf*, Chapter 11 (on page 183)

CHANGE HISTORY

First released in Issue 3.

Entry included for alignment with the ISO POSIX-1 standard.

Issue 6

The LEGACY symbols IUCLC, ULCUC, and XCASE are removed.

FIPS 151-2 requirements for the symbols CS7, CS8, CSTOPB, PARODD, and PARENB are reaffirmed.

NAME

tgmath.h — type-generic macros

SYNOPSIS

```
#include <tgmath.h>
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of IEEE Std 1003.1-200x defers to the ISO C standard.

The **<tgmath.h>** header shall include the headers **<math.h>** and **<complex.h>** and shall define several type-generic macros.

Of the functions contained within the **<math.h>** and **<complex.h>** headers without an *f* (**float**) or *l* (**long double**) suffix, several have one or more parameters whose corresponding real type is **double**. For each such function, except *modf*( ), there shall be a corresponding type-generic macro. The parameters whose corresponding real type is **double** in the function synopsis are generic parameters. Use of the macro invokes a function whose corresponding real type and type domain are determined by the arguments for the generic parameters.

Use of the macro invokes a function whose generic parameters have the corresponding real type determined as follows:

- First, if any argument for generic parameters has type **long double**, the type determined is **long double**.
- Otherwise, if any argument for generic parameters has type **double** or is of integer type, the type determined is **double**.
- Otherwise, the type determined is **float**.

For each unsuffixed function in the **<math.h>** header for which there is a function in the **<complex.h>** header with the same name except for a *c* prefix, the corresponding type-generic macro (for both functions) has the same name as the function in the **<math.h>** header. The corresponding type-generic macro for *fabs*( ) and *cabs*( ) is *fabs*( ).

<math.h> **Function**	**<complex.h>** **Function**	**Type-Generic** **Macro**
acos( )	*cacos*( )	*acos*( )
asin( )	*casin*( )	*asin*( )
atan( )	*catan*( )	*atan*( )
acosh( )	*cacosh*( )	*acosh*( )
asinh( )	*casinh*( )	*asinh*( )
atanh( )	*catanh*( )	*atanh*( )
cos( )	*ccos*( )	*cos*( )
sin( )	*csin*( )	*sin*( )
tan( )	*ctan*( )	*tan*( )
cosh( )	*ccosh*( )	*cosh*( )
sinh( )	*csinh*( )	*sinh*( )
tanh( )	*ctanh*( )	*tanh*( )
exp( )	*cexp*( )	*exp*( )
log( )	*clog*( )	*log*( )
pow( )	*cpow*( )	*pow*( )
sqrt( )	*csqrt*( )	*sqrt*( )
fabs( )	*cabs*( )	*fabs*( )

If at least one argument for a generic parameter is complex, then use of the macro invokes a complex function; otherwise, use of the macro invokes a real function.

For each unsuffixed function in the **<math.h>** header without a *c*-prefixed counterpart in the **<complex.h>** header, the corresponding type-generic macro has the same name as the function. These type-generic macros are:

atan2( )
cbrt( )
ceil( )
copysign( )
erf( )
erfc( )
exp2( )
expm1( )
fdim( )
floor( )
fma( )
fmax( )
fmin( )
fmod( )
frexp( )
hypot( )
ilogb( )
ldexp( )
lgamma( )
llrint( )
llround( )
log10( )
log1p( )
log2( )
logb( )
lrint( )
lround( )
nearbyint( )
nextafter( )
nexttoward( )
remainder( )
remquo( )
rint( )
round( )
scalbn( )
scalbln( )
tgamma( )
trunc( )

If all arguments for generic parameters are real, then use of the macro invokes a real function; otherwise, use of the macro results in undefined behavior.

For each unsuffixed function in the **<complex.h>** header that is not a *c*-prefixed counterpart to a function in the **<math.h>** header, the corresponding type-generic macro has the same name as the function. These type-generic macros are:

carg( )
cimag( )
conj( )
cproj( )
creal( )

Use of the macro with any real or complex argument invokes a complex function.

APPLICATION USAGE

With the declarations:

```
#include <tgmath.h>
int n;
float f;
double d;
long double ld;
float complex fc;
double complex dc;
long double complex ldc;
```

functions invoked by use of type-generic macros are shown in the following table:

Macro	Use Invokes
exp(*n*)	*exp*(*n*), the function
acosh(*f*)	*acoshf*(*f*)
sin(*d*)	*sin*(*d*), the function
atan(*ld*)	*atanl*(*ld*)

Macro	Use Invokes
log(*fc*)	*clogf*(*fc*)
sqrt(*dc*)	*csqrt*(*dc*)
pow(*ldc*,*f*)	*cpowl*(*ldc*, *f*)
remainder(*n*,*n*)	*remainder*(*n*, *n*), the function
nextafter(*d*,*f*)	*nextafter*(*d*, *f*), the function
nexttoward(*f*,*ld*)	*nexttowardf*(*f*, *ld*)
copysign(*n*,*ld*)	*copysignl*(*n*, *ld*)
ceil(*fc*)	Undefined behavior
rint(*dc*)	Undefined behavior
fmax(*ldc*,*ld*)	Undefined behavior
carg(*n*)	*carg*(*n*), the function
cproj(*f*)	*cprojf*(*f*)
creal(*d*)	*creal*(*d*), the function
cimag(*ld*)	*cimagl*(*ld*)
cabs(*fc*)	*cabsf*(*fc*)
carg(*dc*)	*carg*(*dc*), the function
cproj(*ldc*)	*cprojl*(*ldc*)

RATIONALE

Type-generic macros allow calling a function whose type is determined by the argument type, as is the case for C operators such as '+' and '*'. For example, with a type-generic *cos*( ) macro, the expression *cos*((**float**)*x*) will have type **float**. This feature enables writing more portably efficient code and alleviates need for awkward casting and suffixing in the process of porting or adjusting precision. Generic math functions are a widely appreciated feature of Fortran.

The only arguments that affect the type resolution are the arguments corresponding to the parameters that have type **double** in the synopsis. Hence the type of a type-generic call to *nexttoward*( ), whose second parameter is **long double** in the synopsis, is determined solely by the type of the first argument.

The term ‘‘type-generic’’ was chosen over the proposed alternatives of intrinsic and overloading. The term is more specific than intrinsic, which already is widely used with a more general meaning, and reflects a closer match to Fortran's generic functions than to C++ overloading.

The macros are placed in their own header in order not to silently break old programs that include the **<math.h>** header; for example, with:

```
printf ("%e", sin(x))
```

modf(**double**, **double** *) is excluded because no way was seen to make it safe without complicating the type resolution.

The implementation might, as an extension, endow appropriate ones of the macros that IEEE Std 1003.1-200x specifies only for real arguments with the ability to invoke the complex functions.

IEEE Std 1003.1-200x does not prescribe any particular implementation mechanism for generic macros. It could be implemented simply with built-in macros. The generic macro for *sqrt*( ), for example, could be implemented with:

```
#undef sqrt
#define sqrt(x) __BUILTIN_GENERIC_sqrt(x)
```

Generic macros are designed for a useful level of consistency with C++ overloaded math functions.

The great majority of existing C programs are expected to be unaffected when the **<tgmath.h>** header is included instead of the **<math.h>** or **<complex.h>** headers. Generic macros are similar to the ISO/IEC 9899:1999 standard library masking macros, though the semantic types of return values differ.

The ability to overload on integer as well as floating types would have been useful for some functions; for example, *copysign*( ). Overloading with different numbers of arguments would have allowed reusing names; for example, *remainder*( ) for *remquo*( ). However, these facilities would have complicated the specification; and their natural consistent use, such as for a floating *abs*( ) or a two-argument *atan*( ), would have introduced further inconsistencies with the ISO/IEC 9899:1999 standard for insufficient benefit.

The ISO C standard in no way limits the implementation's options for efficiency, including inlining library functions.

FUTURE DIRECTIONS

None.

SEE ALSO

<math.h>, **<complex.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *cabs*( ), *fabs*( ), *modf*( )

CHANGE HISTORY

First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

NAME

time.h — time types

SYNOPSIS

```
#include <time.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these symbols in this header.

The **<time.h>** header shall declare the structure **tm**, which shall include at least the following members:

`int`	`tm_sec`	Seconds [0,60].
`int`	`tm_min`	Minutes [0,59].
`int`	`tm_hour`	Hour [0,23].
`int`	`tm_mday`	Day of month [1,31].
`int`	`tm_mon`	Month of year [0,11].
`int`	`tm_year`	Years since 1900.
`int`	`tm_wday`	Day of week [0,6] (Sunday =0).
`int`	`tm_yday`	Day of year [0,365].
`int`	`tm_isdst`	Daylight savings flag.

The value of *tm_isdst* shall be positive if Daylight Saving Time is in effect, 0 if Daylight Saving Time is not in effect, and negative if the information is not available.

The **<time.h>** header shall define the following symbolic names:

NULL Null pointer constant.

CLOCKS_PER_SEC A number used to convert the value returned by the *clock*( ) function into seconds.

TMR | CPT CLOCK_PROCESS_CPUTIME_ID
The identifier of the CPU-time clock associated with the process making a *clock*( ) or *timer**( ) function call.

TMR | TCT CLOCK_THREAD_CPUTIME_ID
The identifier of the CPU-time clock associated with the thread making a *clock*( ) or *timer**( ) function call.

TMR The **<time.h>** header shall declare the structure **timespec**, which has at least the following members:

`time_t`	`tv_sec`	Seconds.
`long`	`tv_nsec`	Nanoseconds.

The **<time.h>** header shall also declare the **itimerspec** structure, which has at least the following members:

`struct timespec`	`it_interval`	Timer period.
`struct timespec`	`it_value`	Timer expiration.

The following manifest constants shall be defined:

CLOCK_REALTIME The identifier of the system-wide realtime clock.

TIMER_ABSTIME Flag indicating time is absolute with respect to the clock associated with a timer.

CLOCK_MONOTONIC
The identifier for the system-wide monotonic clock, which is defined as a clock whose value cannot be set via *clock_settime*() and which cannot have backward clock jumps. The maximum possible clock jump shall be implementation-defined.

The **clock_t**, **size_t**, **time_t**, **clockid_t**, and **timer_t** types shall be defined as described in **<sys/types.h>**.

Although the value of CLOCKS_PER_SEC is required to be 1 million on all XSI-conformant systems, it may be variable on other systems, and it should not be assumed that CLOCKS_PER_SEC is a compile-time constant.

The **<time.h>** header shall provide a declaration for *getdate_err*.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
char      *asctime(const struct tm *);
char      *asctime_r(const struct tm *restrict, char *restrict);
clock_t    clock(void);
int        clock_getcpuclockid(pid_t, clockid_t *);
int        clock_getres(clockid_t, struct timespec *);
int        clock_gettime(clockid_t, struct timespec *);
int        clock_nanosleep(clockid_t, int, const struct timespec *,
               struct timespec *);
int        clock_settime(clockid_t, const struct timespec *);
char      *ctime(const time_t *);
char      *ctime_r(const time_t *, char *);
double     difftime(time_t, time_t);
struct tm *getdate(const char *);
struct tm *gmtime(const time_t *);
struct tm *gmtime_r(const time_t *restrict, struct tm *restrict);
struct tm *localtime(const time_t *);
struct tm *localtime_r(const time_t *restrict, struct tm *restrict);
time_t     mktime(struct tm *);
int        nanosleep(const struct timespec *, struct timespec *);
size_t     strftime(char *restrict, size_t, const char *restrict,
           const struct tm *restrict);
char      *strptime(const char *restrict, const char *restrict,
               struct tm *restrict);
time_t     time(time_t *);
int        timer_create(clockid_t, struct sigevent *restrict,
               timer_t *restrict);
int        timer_delete(timer_t);
int        timer_gettime(timer_t, struct itimerspec *);
int        timer_getoverrun(timer_t);
int        timer_settime(timer_t, int, const struct itimerspec *restrict,
               struct itimerspec *restrict);
void       tzset(void);

```

The following shall be declared as variables:

```
extern int    daylight;
extern long   timezone;
```

CX
```
extern char  *tzname[];
```

CX Inclusion of the **<time.h>** header may make visible all symbols from the **<signal.h>** header.

APPLICATION USAGE

The range [0,60] for *tm_sec* allows for the occasional leap second.

tm_year is a signed value; therefore, years before 1900 may be represented.

To obtain the number of clock ticks per second returned by the *times*( ) function, applications should call *sysconf*(_SC_CLK_TCK).

RATIONALE

The range [0,60] seconds allows for positive or negative leap seconds. The formal definition of UTC does not permit double leap seconds, so all mention of double leap seconds has been removed, and the range shortened from the former [0,61] seconds seen in previous versions of POSIX.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *asctime*( ), *clock*( ), *clock_getcpuclockid*( ), *clock_getres*( ), *clock_nanosleep*( ), *ctime*( ), *difftime*( ), *getdate*( ), *gmtime*( ), *localtime*( ), *mktime*( ), *nanosleep*( ), *strftime*( ), *strptime*( ), *sysconf*( ), *time*( ), *timer_create*( ), *timer_delete*( ), *timer_getoverrun*( ), *tzname*, *tzset*( ), *utime*( )

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Issue 6

The Open Group Corrigendum U035/6 is applied. In the DESCRIPTION, the types **clockid_t** and **timer_t** have been described.

The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- The POSIX timer-related functions are now marked as part of the Timers option.

The symbolic name CLK_TCK is removed. Application usage is added describing how its equivalent functionality can be obtained using *sysconf*( ).

The *clock_getcpuclockid*( ) function and manifest constants CLOCK_PROCESS_CPUTIME_ID and CLOCK_THREAD_CPUTIME_ID are added for alignment with IEEE Std 1003.1d-1999.

The manifest constant CLOCK_MONOTONIC and the *clock_nanosleep*( ) function are added for alignment with IEEE Std 1003.1j-2000.

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The range for seconds is changed from [0,61] to [0,60].
- The **restrict** keyword is added to the prototypes for *asctime_r*( ), *gmtime_r*( ), *localtime_r*( ), *strftime*( ), *strptime*( ), *timer_create*( ), and *timer_settime*( ).

IEEE PASC Interpretation 1003.1 #84 is applied adding the statement that symbols from the **<signal.h>** header may be made visible when the **<time.h>** header is included.

Extensions beyond the ISO C standard are now marked.

NAME

trace.h — tracing

SYNOPSIS

```
TRC    #include <trace.h>
```

DESCRIPTION

The **<trace.h>** header shall define the **posix_trace_event_info** structure that includes at least the following members:

```
       trace_event_id_t  posix_event_id
       pid_t             posix_pid
       void             *posix_prog_address
       int               posix_truncation_status
       struct timespec   posix_timestamp
THR    pthread_t         posix_thread_id
```

The **<trace.h>** header shall define the **posix_trace_status_info** structure that includes at least the following members:

```
       int      posix_stream_status
       int      posix_stream_full_status
       int      posix_stream_overrun_status
TRL    int      posix_stream_flush_status
       int      posix_stream_flush_error
       int      posix_log_overrun_status
       int      posix_log_full_status
```

The **<trace.h>** header shall define the following symbols:

POSIX_TRACE_RUNNING
POSIX_TRACE_SUSPENDED
POSIX_TRACE_FULL
POSIX_TRACE_NOT_FULL
POSIX_TRACE_NO_OVERRUN
POSIX_TRACE_OVERRUN
TRL POSIX_TRACE_FLUSHING
POSIX_TRACE_NOT_FLUSHING
POSIX_TRACE_NOT_TRUNCATED
POSIX_TRACE_TRUNCATED_READ
POSIX_TRACE_TRUNCATED_RECORD
TRL POSIX_TRACE_FLUSH
POSIX_TRACE_LOOP
POSIX_TRACE_UNTIL_FULL
TRI POSIX_TRACE_CLOSE_FOR_CHILD
POSIX_TRACE_INHERITED
TRL POSIX_TRACE_APPEND
POSIX_TRACE_LOOP
POSIX_TRACE_UNTIL_FULL
TEF POSIX_TRACE_FILTER
TRL POSIX_TRACE_FLUSH_START
POSIX_TRACE_FLUSH_STOP
POSIX_TRACE_OVERFLOW

POSIX_TRACE_RESUME
POSIX_TRACE_START
POSIX_TRACE_STOP
POSIX_TRACE_UNNAMED_USER_EVENT

The following types shall be defined as described in **<sys/types.h>**:

trace_attr_t
trace_id_t
trace_event_id_t
TEF **trace_event_set_t**

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
     int  posix_trace_attr_destroy(trace_attr_t *);
     int  posix_trace_attr_getclockres(const trace_attr_t *,
              struct timespec *);
     int  posix_trace_attr_getcreatetime(const trace_attr_t *,
              struct timespec *);
     int  posix_trace_attr_getgenversion(const trace_attr_t *, char *);
TRI  int  posix_trace_attr_getinherited(const trace_attr_t *restrict,
              int *restrict);
TRL  int  posix_trace_attr_getlogfullpolicy(const trace_attr_t *restrict,
              int *restrict);
     int  posix_trace_attr_getlogsize(const trace_attr_t *restrict,
              size_t *restrict);
     int  posix_trace_attr_getmaxdatasize(const trace_attr_t *restrict,
              size_t *restrict);
     int  posix_trace_attr_getmaxsystemeventsize(const trace_attr_t *restrict,
              size_t *restrict);
     int  posix_trace_attr_getmaxusereventsize(const trace_attr_t *restrict,
              size_t, size_t *restrict);
     int  posix_trace_attr_getname(const trace_attr_t *, char *);
     int  posix_trace_attr_getstreamfullpolicy(const trace_attr_t *restrict,
              int *restrict);
     int  posix_trace_attr_getstreamsize(const trace_attr_t *restrict,
              size_t *restrict);
     int  posix_trace_attr_init(trace_attr_t *);
TRI  int  posix_trace_attr_setinherited(trace_attr_t *, int);
TRL  int  posix_trace_attr_setlogfullpolicy(trace_attr_t *, int);
     int  posix_trace_attr_setlogsize(trace_attr_t *, size_t);
     int  posix_trace_attr_setmaxdatasize(trace_attr_t *, size_t);
     int  posix_trace_attr_setname(trace_attr_t *, const char *);
     int  posix_trace_attr_setstreamsize(trace_attr_t *, size_t);
     int  posix_trace_attr_setstreamfullpolicy(trace_attr_t *, int);
     int  posix_trace_clear(trace_id_t);
TRL  int  posix_trace_close(trace_id_t);
     int  posix_trace_create(pid_t, const trace_attr_t *restrict,
              trace_id_t *restrict);
TRL  int  posix_trace_create_withlog(pid_t, const trace_attr_t *restrict,
              int, trace_id_t *restrict);
     void posix_trace_event(trace_event_id_t, const void *restrict, size_t);
```

```
        int  posix_trace_eventid_equal(trace_id_t, trace_event_id_t,
                  trace_event_id_t);
        int  posix_trace_eventid_get_name(trace_id_t, trace_event_id_t, char *);
        int  posix_trace_eventid_open(const char *restrict,
                  trace_event_id_t *restrict);
        int  posix_trace_eventtypelist_getnext_id(trace_id_t,
                  trace_event_id_t *restrict, int *restrict);
        int  posix_trace_eventtypelist_rewind(trace_id_t);
TEF     int  posix_trace_eventset_add(trace_event_id_t, trace_event_set_t *);
        int  posix_trace_eventset_del(trace_event_id_t, trace_event_set_t *);
        int  posix_trace_eventset_empty(trace_event_set_t *);
        int  posix_trace_eventset_fill(trace_event_set_t *, int);
        int  posix_trace_eventset_ismember(trace_event_id_t,
                  const trace_event_set_t *restrict, int *restrict);
        int  posix_trace_flush(trace_id_t);
        int  posix_trace_get_attr(trace_id_t, trace_attr_t *);
TEF     int  posix_trace_get_filter(trace_id_t, trace_event_set_t *);
        int  posix_trace_get_status(trace_id_t,
                  struct posix_trace_status_info *);
        int  posix_trace_getnext_event(trace_id_t,
                  struct posix_trace_event_info *restrict , void *restrict,
                  size_t, size_t *restrict, int *restrict);
TRL     int  posix_trace_open(int, trace_id_t *);
        int  posix_trace_rewind(trace_id_t);
TEF     int  posix_trace_set_filter(trace_id_t, const trace_event_set_t *, int);
        int  posix_trace_shutdown(trace_id_t);
        int  posix_trace_start(trace_id_t);
        int  posix_trace_stop(trace_id_t);
TMO     int  posix_trace_timedgetnext_event(trace_id_t,
                  struct posix_trace_event_info *restrict, void *restrict,
                  size_t, size_t *restrict, int *restrict,
                  const struct timespec *restrict);
TEF     int  posix_trace_trid_eventid_open(trace_id_t, const char *restrict,
                  trace_event_id_t *restrict);
        int  posix_trace_trygetnext_event(trace_id_t,
                  struct posix_trace_event_info *restrict, void *restrict, size_t,
                  size_t *restrict, int *restrict);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.11, Tracing, the System Interfaces volume of IEEE Std 1003.1-200x, *posix_trace_attr_destroy*( ), *posix_trace_attr_getclockres*( ), *posix_trace_attr_getcreatetime*( ), *posix_trace_attr_getgenversion*( ), *posix_trace_attr_getinherited*( ), *posix_trace_attr_getlogfullpolicy*( ), *posix_trace_attr_getlogsize*( ), *posix_trace_attr_getmaxdatasize*( ), *posix_trace_attr_getmaxsystemeventsize*( ), *posix_trace_attr_getmaxusereventsize*( ), *posix_trace_attr_getname*( ), *posix_trace_attr_getstreamfullpolicy*( ), *posix_trace_attr_getstreamsize*( ), *posix_trace_attr_init*( ), *posix_trace_attr_setinherited*( ), *posix_trace_attr_setlogfullpolicy*( ), *posix_trace_attr_setlogsize*( ), *posix_trace_attr_setmaxdatasize*( ), *posix_trace_attr_setname*( ), *posix_trace_attr_setstreamsize*( ), *posix_trace_attr_setstreamfullpolicy*( ), *posix_trace_clear*( ), *posix_trace_close*( ), *posix_trace_create*( ), *posix_trace_create_withlog*( ), *posix_trace_event*( ), *posix_trace_eventid_equal*( ), *posix_trace_eventid_get_name*( ), *posix_trace_eventid_open*( ), *posix_trace_eventtypelist_getnext_id*( ), *posix_trace_eventtypelist_rewind*( ), *posix_trace_eventset_add*( ), *posix_trace_eventset_del*( ), *posix_trace_eventset_empty*( ), *posix_trace_eventset_fill*( ), *posix_trace_eventset_ismember*( ), *posix_trace_flush*( ), *posix_trace_get_attr*( ), *posix_trace_get_filter*( ), *posix_trace_get_status*( ), *posix_trace_getnext_event*( ), *posix_trace_open*( ), *posix_trace_rewind*( ), *posix_trace_set_filter*( ), *posix_trace_shutdown*( ), *posix_trace_start*( ), *posix_trace_stop*( ), *posix_trace_timedgetnext_event*( ), *posix_trace_trid_eventid_open*( ), *posix_trace_trygetnext_event*( )

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

NAME

ucontext.h — user context

SYNOPSIS

XSI

```
#include <ucontext.h>
```

DESCRIPTION

The **<ucontext.h>** header shall define the **mcontext_t** type through **typedef**.

The **<ucontext.h>** header shall define the **ucontext_t** type as a structure that shall include at least the following members:

`ucontext_t`	`*uc_link`	Pointer to the context that is resumed when this context returns.
`sigset_t`	`uc_sigmask`	The set of signals that are blocked when this context is active.
`stack_t`	`uc_stack`	The stack used by this context.
`mcontext_t`	`uc_mcontext`	A machine-specific representation of the saved context.

The types **sigset_t** and **stack_t** shall be defined as in **<signal.h>**.

The following shall be declared as functions and may also be defined as macros, Function prototypes shall be provided.

```
int  getcontext(ucontext_t *);
int  setcontext(const ucontext_t *);
void makecontext(ucontext_t *, void (*)(void), int, ...);
int  swapcontext(ucontext_t *restrict, const ucontext_t *restrict);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<signal.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *getcontext*( ), *makecontext*( ), *sigaction*( ), *sigprocmask*( ), *sigaltstack*( )

CHANGE HISTORY

First released in Issue 4, Version 2.

NAME

ulimit.h — ulimit commands

SYNOPSIS

XSI

```
#include <ulimit.h>
```

DESCRIPTION

The **<ulimit.h>** header shall define the symbolic constants used by the *ulimit*( ) function.

Symbolic constants:

UL_GETFSIZE	Get maximum file size.
UL_SETFSIZE	Set maximum file size.

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided. |

```
long ulimit(int, ...);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *ulimit*( )

CHANGE HISTORY

First released in Issue 3.

NAME

unistd.h — standard symbolic constants and types

SYNOPSIS

```
#include <unistd.h>
```

DESCRIPTION

The **<unistd.h>** header defines miscellaneous symbolic constants and types, and declares miscellaneous functions. The actual value of the constants are unspecified except as shown. The contents of this header are shown below.

Version Test Macros

The following symbolic constants shall be defined:

_POSIX_VERSION

Integer value indicating version of IEEE Std 1003.1 (C-language binding) to which the implementation conforms. For implementations conforming to IEEE Std 1003.1-200x, the value shall be 200xxxL.

_POSIX2_VERSION

Integer value indicating version of the Shell and Utilities volume of IEEE Std 1003.1 to which the implementation conforms. For implementations conforming to IEEE Std 1003.1-200x, the value shall be 200xxxL.

The following symbolic constant shall be defined only if the implementation supports the XSI option; see Section 2.1.4 (on page 19).

XSI _XOPEN_VERSION

Integer value indicating version of the X/Open Portability Guide to which the implementation conforms. The value shall be 600.

Constants for Options and Option Groups

The following symbolic constants, if defined in **<unistd.h>**, shall have a value of –1, 0, or greater, unless otherwise specified below. If these are undefined, the *fpathconf*( ), *pathconf*( ), or *sysconf*( ) functions can be used to determine whether the option is provided for a particular invocation of the application.

If a symbolic constant is defined with the value –1, the option is not supported. Headers, data types, and function interfaces required only for the option need not be supplied. An application that attempts to use anything associated only with the option is considered to be requiring an extension.

If a symbolic constant is defined with a value greater than zero, the option shall always be supported when the application is executed. All headers, data types, and functions shall be present and shall operate as specified.

If a symbolic constant is defined with the value zero, all headers, data types, and functions shall be present. The application can check at runtime to see whether the option is supported by calling *fpathconf*( ), *pathconf*( ), or *sysconf*( ) with the indicated *name* parameter.

Unless explicitly specified otherwise, the behavior of functions associated with an unsupported option is unspecified, and an application that uses such functions without first checking *fpathconf*( ), *pathconf*( ), or *sysconf*( ) is considered to be requiring an extension.

For conformance requirements, refer to Chapter 2 (on page 15).

ADV _POSIX_ADVISORY_INFO
The implementation supports the Advisory Information option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

AIO _POSIX_ASYNCHRONOUS_IO
The implementation supports the Asynchronous Input and Output option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

BAR _POSIX_BARRIERS
The implementation supports the Barriers option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

_POSIX_CHOWN_RESTRICTED
The use of *chown*( ) and *fchown*( ) is restricted to a process with appropriate privileges, and to changing the group ID of a file only to the effective group ID of the process or to one of its supplementary group IDs. This symbol shall always be set to a value other than –1.

CS _POSIX_CLOCK_SELECTION
The implementation supports the Clock Selection option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

CPT _POSIX_CPUTIME
The implementation supports the Process CPU-Time Clocks option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

FSC _POSIX_FSYNC
The implementation supports the File Synchronization option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

_POSIX_JOB_CONTROL
The implementation supports job control. This symbol shall always be set to a value greater than zero.

MF _POSIX_MAPPED_FILES
The implementation supports the Memory Mapped Files option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

ML _POSIX_MEMLOCK
The implementation supports the Process Memory Locking option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

MLR _POSIX_MEMLOCK_RANGE
The implementation supports the Range Memory Locking option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

MPR _POSIX_MEMORY_PROTECTION
The implementation supports the Memory Protection option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

MSG _POSIX_MESSAGE_PASSING
The implementation supports the Message Passing option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

MON _POSIX_MONOTONIC_CLOCK
The implementation supports the Monotonic Clock option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

_POSIX_NO_TRUNC
Pathname components longer than {NAME_MAX} generate an error. This symbol shall

always be set to a value other than −1.

PIO _POSIX_PRIORITIZED_IO

The implementation supports the Prioritized Input and Output option. If this symbol has a value other than −1 or 0, it shall have the value 200xxxL.

PS _POSIX_PRIORITY_SCHEDULING

The implementation supports the Process Scheduling option. If this symbol has a value other than −1 or 0, it shall have the value 200xxxL.

RS _POSIX_RAW_SOCKETS

The implementation supports the Raw Sockets option. If this symbol has a value other than −1 or 0, it shall have the value 200xxxL.

THR _POSIX_READER_WRITER_LOCKS

The implementation supports the Read-Write Locks option. This is always set to a value greater than zero if the Threads option is supported. If this symbol has a value other than −1 or 0, it shall have the value 200xxxL.

RTS _POSIX_REALTIME_SIGNALS

The implementation supports the Realtime Signals Extension option. If this symbol has a value other than −1 or 0, it shall have the value 200xxxL.

_POSIX_REGEXP

The implementation supports the Regular Expression Handling option. This symbol shall always be set to a value greater than zero.

_POSIX_SAVED_IDS

Each process has a saved set-user-ID and a saved set-group-ID. This symbol shall always be set to a value greater than zero.

SEM _POSIX_SEMAPHORES

The implementation supports the Semaphores option. If this symbol has a value other than −1 or 0, it shall have the value 200xxxL.

SHM _POSIX_SHARED_MEMORY_OBJECTS

The implementation supports the Shared Memory Objects option. If this symbol has a value other than −1 or 0, it shall have the value 200xxxL.

_POSIX_SHELL

The implementation supports the POSIX shell. This symbol shall always be set to a value greater than zero.

SPN _POSIX_SPAWN

The implementation supports the Spawn option. If this symbol has a value other than −1 or 0, it shall have the value 200xxxL.

SPI _POSIX_SPIN_LOCKS

The implementation supports the Spin Locks option. If this symbol has a value other than −1 or 0, it shall have the value 200xxxL.

SS _POSIX_SPORADIC_SERVER

The implementation supports the Process Sporadic Server option. If this symbol has a value other than −1 or 0, it shall have the value 200xxxL.

SIO _POSIX_SYNCHRONIZED_IO

The implementation supports the Synchronized Input and Output option. If this symbol has a value other than −1 or 0, it shall have the value 200xxxL.

TSA _POSIX_THREAD_ATTR_STACKADDR

The implementation supports the Thread Stack Address Attribute option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

TSS _POSIX_THREAD_ATTR_STACKSIZE

The implementation supports the Thread Stack Address Size option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

TCT _POSIX_THREAD_CPUTIME

The implementation supports the Thread CPU-Time Clocks option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

TPI _POSIX_THREAD_PRIO_INHERIT

The implementation supports the Thread Priority Inheritance option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

TPP _POSIX_THREAD_PRIO_PROTECT

The implementation supports the Thread Priority Protection option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

TPS _POSIX_THREAD_PRIORITY_SCHEDULING

The implementation supports the Thread Execution Scheduling option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

TSH _POSIX_THREAD_PROCESS_SHARED

The implementation supports the Thread Process-Shared Synchronization option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

TSF _POSIX_THREAD_SAFE_FUNCTIONS

The implementation supports the Thread-Safe Functions option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

TSP _POSIX_THREAD_SPORADIC_SERVER

The implementation supports the Thread Sporadic Server option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

THR _POSIX_THREADS

The implementation supports the Threads option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

TMO _POSIX_TIMEOUTS

The implementation supports the Timeouts option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

TMR _POSIX_TIMERS

The implementation supports the Timers option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

TRC _POSIX_TRACE

The implementation supports the Trace option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

TEF _POSIX_TRACE_EVENT_FILTER

The implementation supports the Trace Event Filter option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

TRI _POSIX_TRACE_INHERIT

The implementation supports the Trace Inherit option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

TRL _POSIX_TRACE_LOG

The implementation supports the Trace Log option. If this symbol has a value other than −1 or 0, it shall have the value 200xxxL.

TYM _POSIX_TYPED_MEMORY_OBJECTS

The implementation supports the Typed Memory Objects option. If this symbol has a value other than −1 or 0, it shall have the value 200xxxL.

_POSIX_VDISABLE

This symbol shall be defined to be the value of a character that shall disable terminal special character handling as described in **<termios.h>**. This symbol shall always be set to a value other than −1.

_POSIX2_C_BIND

The implementation supports the C-Language Binding option. This symbol shall always have the value 200xxxL.

CD _POSIX2_C_DEV

The implementation supports the C-Language Development Utilities option. If this symbol has a value other than −1 or 0, it shall have the value 200xxxL.

_POSIX2_CHAR_TERM

The implementation supports at least one terminal type.

FD _POSIX2_FORT_DEV

The implementation supports the FORTRAN Development Utilities option. If this symbol has a value other than −1 or 0, it shall have the value 200xxxL.

FR _POSIX2_FORT_RUN

The implementation supports the FORTRAN Runtime Utilities option. If this symbol has a value other than −1 or 0, it shall have the value 200xxxL.

_POSIX2_LOCALEDEF

The implementation supports the creation of locales by the *localedef* utility. If this symbol has a value other than −1 or 0, it shall have the value 200xxxL.

BE _POSIX2_PBS

The implementation supports the Batch Environment Services and Utilities option. If this symbol has a value other than −1 or 0, it shall have the value 200xxxL.

BE _POSIX2_PBS_ACCOUNTING

The implementation supports the Batch Accounting option. If this symbol has a value other than −1 or 0, it shall have the value 200xxxL.

BE _POSIX2_PBS_CHECKPOINT

The implementation supports the Batch Checkpoint/Restart option. If this symbol has a value other than −1 or 0, it shall have the value 200xxxL.

BE _POSIX2_PBS_LOCATE

The implementation supports the Locate Batch Job Request option. If this symbol has a value other than −1 or 0, it shall have the value 200xxxL.

BE _POSIX2_PBS_MESSAGE

The implementation supports the Batch Job Message Request option. If this symbol has a value other than −1 or 0, it shall have the value 200xxxL.

BE _POSIX2_PBS_TRACK

The implementation supports the Track Batch Job Request option. If this symbol has a value other than −1 or 0, it shall have the value 200xxxL.

SD _POSIX2_SW_DEV
The implementation supports the Software Development Utilities option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

UP _POSIX2_UPE
The implementation supports the User Portability Utilities option. If this symbol has a value other than –1 or 0, it shall have the value 200xxxL.

_V6_ILP32_OFF32
The implementation provides a C-language compilation environment with 32-bit **int**, **long**, **pointer**, and **off_t** types.

_V6_ILP32_OFFBIG
The implementation provides a C-language compilation environment with 32-bit **int**, **long**, and **pointer** types and an **off_t** type using at least 64 bits.

_V6_LP64_OFF64
The implementation provides a C-language compilation environment with 32-bit **int** and 64-bit **long**, **pointer**, and **off_t** types.

_V6_LPBIG_OFFBIG
The implementation provides a C-language compilation environment with an **int** type using at least 32 bits and **long**, **pointer**, and **off_t** types using at least 64 bits.

XSI _XBS5_ILP32_OFF32 (**LEGACY**)
The implementation provides a C-language compilation environment with 32-bit **int**, **long**, **pointer**, and **off_t** types.

XSI _XBS5_ILP32_OFFBIG (**LEGACY**)
The implementation provides a C-language compilation environment with 32-bit **int**, **long**, and **pointer** types and an **off_t** type using at least 64 bits.

XSI _XBS5_LP64_OFF64 (**LEGACY**)
The implementation provides a C-language compilation environment with 32-bit **int** and 64-bit **long**, **pointer**, and **off_t** types.

XSI _XBS5_LPBIG_OFFBIG (**LEGACY**)
The implementation provides a C-language compilation environment with an **int** type using at least 32 bits and **long**, **pointer**, and **off_t** types using at least 64 bits.

XSI _XOPEN_CRYPT
The implementation supports the X/Open Encryption Option Group.

_XOPEN_ENH_I18N
The implementation supports the Issue 4, Version 2 Enhanced Internationalization Option Group. This symbol shall always be set to a value other than –1.

_XOPEN_LEGACY
The implementation supports the Legacy Option Group.

_XOPEN_REALTIME
The implementation supports the X/Open Realtime Option Group.

_XOPEN_REALTIME_THREADS
The implementation supports the X/Open Realtime Threads Option Group.

_XOPEN_SHM
The implementation supports the Issue 4, Version 2 Shared Memory Option Group. This symbol shall always be set to a value other than –1.

_XOPEN_STREAMS
The implementation supports the XSI STREAMS Option Group.

XSI _XOPEN_UNIX
The implementation supports the XSI extension.

Execution-Time Symbolic Constants

If any of the following constants are not defined in the **<unistd.h>** header, the value shall vary depending on the file to which it is applied.

If any of the following constants are defined to have value −1 in the **<unistd.h>** header, the implementation shall not provide the option on any file; if any are defined to have a value other than −1 in the **<unistd.h>** header, the implementation shall provide the option on all applicable files.

All of the following constants, whether defined in **<unistd.h>** or not, may be queried with respect to a specific file using the *pathconf*( ) or *fpathconf*( ) functions:

_POSIX_ASYNC_IO
Asynchronous input or output operations may be performed for the associated file.

_POSIX_PRIO_IO
Prioritized input or output operations may be performed for the associated file.

_POSIX_SYNC_IO
Synchronized input or output operations may be performed for the associated file.

Constants for Functions

The following symbolic constant shall be defined:

NULL Null pointer

The following symbolic constants shall be defined for the *access*( ) function:

F_OK	Test for existence of file.
R_OK	Test for read permission.
W_OK	Test for write permission.
X_OK	Test for execute (search) permission.

The constants F_OK, R_OK, W_OK, and X_OK and the expressions *R_OK*|*W_OK*, *R_OK*|*X_OK*, and *R_OK*|*W_OK*|*X_OK* shall all have distinct values.

The following symbolic constants shall be defined for the *confstr*( ) function:

_CS_POSIX_PATH
This is the value for the *PATH* environment variable that finds all standard utilities.

_CS_POSIX_V6_ILP32_OFF32_CFLAGS
If *sysconf*(_SC_V6_ILP32_OFF32) returns −1, the meaning of this value is unspecified. Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to build an application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

_CS_POSIX_V6_ILP32_OFF32_LDFLAGS
If *sysconf*(_SC_V6_ILP32_OFF32) returns −1, the meaning of this value is unspecified. Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build an application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

_CS_POSIX_V6_ILP32_OFF32_LIBS

If *sysconf*(_SC_V6_ILP32_OFF32) returns −1, the meaning of this value is unspecified. Otherwise, this value is the set of libraries to be given to the *cc* and *c99* utilities to build an application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

_CS_POSIX_V6_ILP32_OFF32_LINTFLAGS

If *sysconf*(_SC_V6_ILP32_OFF32) returns −1, the meaning of this value is unspecified. Otherwise, this value is the set of options to be given to the *lint* utility to check application source using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

_CS_POSIX_V6_ILP32_OFFBIG_CFLAGS

If *sysconf*(_SC_V6_ILP32_OFFBIG) returns −1, the meaning of this value is unspecified. Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to build an application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an **off_t** type using at least 64 bits.

_CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS

If *sysconf*(_SC_V6_ILP32_OFFBIG) returns −1, the meaning of this value is unspecified. Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build an application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an **off_t** type using at least 64 bits.

_CS_POSIX_V6_ILP32_OFFBIG_LIBS

If *sysconf*(_SC_V6_ILP32_OFFBIG) returns −1, the meaning of this value is unspecified. Otherwise, this value is the set of libraries to be given to the *cc* and *c99* utilities to build an application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an **off_t** type using at least 64 bits.

_CS_POSIX_V6_ILP32_OFFBIG_LINTFLAGS

If *sysconf*(_SC_V6_ILP32_OFFBIG) returns −1, the meaning of this value is unspecified. Otherwise, this value is the set of options to be given to the *lint* utility to check an application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an **off_t** type using at least 64 bits.

_CS_POSIX_V6_LP64_OFF64_CFLAGS

If *sysconf*(_SC_V6_LP64_OFF64) returns −1, the meaning of this value is unspecified. Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to build an application using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t** types.

_CS_POSIX_V6_LP64_OFF64_LDFLAGS

If *sysconf*(_SC_V6_LP64_OFF64) returns −1, the meaning of this value is unspecified. Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build an application using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t** types.

_CS_POSIX_V6_LP64_OFF64_LIBS

If *sysconf*(_SC_V6_LP64_OFF64) returns −1, the meaning of this value is unspecified. Otherwise, this value is the set of libraries to be given to the *cc* and *c99* utilities to build an application using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t** types.

_CS_POSIX_V6_LP64_OFF64_LINTFLAGS

If *sysconf*(_SC_V6_LP64_OFF64) returns −1, the meaning of this value is unspecified. Otherwise, this value is the set of options to be given to the *lint* utility to check application source using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t** types.

_CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS

If *sysconf*(_SC_V6_LPBIG_OFFBIG) returns −1, the meaning of this value is unspecified.

Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to build an application using a programming model with an **int** type using at least 32 bits and **long**, **pointer**, and **off_t** types using at least 64 bits.

_CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS

If *sysconf*(_SC_V6_LPBIG_OFFBIG) returns −1, the meaning of this value is unspecified. Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build an application using a programming model with an **int** type using at least 32 bits and **long**, **pointer**, and **off_t** types using at least 64 bits.

_CS_POSIX_V6_LPBIG_OFFBIG_LIBS

If *sysconf*(_SC_V6_LPBIG_OFFBIG) returns −1, the meaning of this value is unspecified. Otherwise, this value is the set of libraries to be given to the *cc* and *c99* utilities to build an application using a programming model with an **int** type using at least 32 bits and **long**, **pointer**, and **off_t** types using at least 64 bits.

_CS_POSIX_V6_LPBIG_OFFBIG_LINTFLAGS

If *sysconf*(_SC_V6_LPBIG_OFFBIG) returns −1, the meaning of this value is unspecified. Otherwise, this value is the set of options to be given to the *lint* utility to check application source using a programming model with an **int** type using at least 32 bits and **long**, **pointer**, and **off_t** types using at least 64 bits.

_CS_V6_WIDTH_RESTRICTED_ENVS

This value is a <newline>-separated list of names of programming environments supported by the implementation in which the widths of the **blksize_t**, **cc_t**, **mode_t**, **nfds_t**, **pid_t**, **ptrdiff_t**, **size_t**, **speed_t**, **ssize_t**, **suseconds_t**, **tcflag_t**, **useconds_t**, **wchar_t**, and **wint_t** types are no greater than the width of type **long**.

XSI _CS_XBS5_ILP32_OFF32_CFLAGS (**LEGACY**)

If *sysconf*(_SC_XBS5_ILP32_OFF32) returns −1, the meaning of this value is unspecified. Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to build an application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

XSI _CS_XBS5_ILP32_OFF32_LDFLAGS (**LEGACY**)

If *sysconf*(_SC_XBS5_ILP32_OFF32) returns −1, the meaning of this value is unspecified. Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build an application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

XSI _CS_XBS5_ILP32_OFF32_LIBS (**LEGACY**)

If *sysconf*(_SC_XBS5_ILP32_OFF32) returns −1, the meaning of this value is unspecified. Otherwise, this value is the set of libraries to be given to the *cc* and *c99* utilities to build an application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

XSI _CS_XBS5_ILP32_OFF32_LINTFLAGS (**LEGACY**)

If *sysconf*(_SC_XBS5_ILP32_OFF32) returns −1, the meaning of this value is unspecified. Otherwise, this value is the set of options to be given to the *lint* utility to check application source using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

XSI _CS_XBS5_ILP32_OFFBIG_CFLAGS (**LEGACY**)

If *sysconf*(_SC_XBS5_ILP32_OFFBIG) returns −1, the meaning of this value is unspecified. Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to build an application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an **off_t** type using at least 64 bits.

XSI _CS_XBS5_ILP32_OFFBIG_LDFLAGS (**LEGACY**)

If *sysconf*(_SC_XBS5_ILP32_OFFBIG) returns −1, the meaning of this value is unspecified.

Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build an application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an **off_t** type using at least 64 bits.

XSI _CS_XBS5_ILP32_OFFBIG_LIBS (**LEGACY**)

If *sysconf*(_SC_XBS5_ILP32_OFFBIG) returns –1, the meaning of this value is unspecified. Otherwise, this value is the set of libraries to be given to the *cc* and *c99* utilities to build an application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an **off_t** type using at least 64 bits.

XSI _CS_XBS5_ILP32_OFFBIG_LINTFLAGS (**LEGACY**)

If *sysconf*(_SC_XBS5_ILP32_OFFBIG) returns –1, the meaning of this value is unspecified. Otherwise, this value is the set of options to be given to the *lint* utility to check an application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an **off_t** type using at least 64 bits.

XSI _CS_XBS5_LP64_OFF64_CFLAGS (**LEGACY**)

If *sysconf*(_SC_XBS5_LP64_OFF64) returns –1, the meaning of this value is unspecified. Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to build an application using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t** types.

XSI _CS_XBS5_LP64_OFF64_LDFLAGS (**LEGACY**)

If *sysconf*(_SC_XBS5_LP64_OFF64) returns –1, the meaning of this value is unspecified. Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build an application using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t** types.

XSI _CS_XBS5_LP64_OFF64_LIBS (**LEGACY**)

If *sysconf*(_SC_XBS5_LP64_OFF64) returns –1, the meaning of this value is unspecified. Otherwise, this value is the set of libraries to be given to the *cc* and *c99* utilities to build an application using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t** types.

XSI _CS_XBS5_LP64_OFF64_LINTFLAGS (**LEGACY**)

If *sysconf*(_SC_XBS5_LP64_OFF64) returns –1, the meaning of this value is unspecified. Otherwise, this value is the set of options to be given to the *lint* utility to check application source using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t** types.

XSI _CS_XBS5_LPBIG_OFFBIG_CFLAGS (**LEGACY**)

If *sysconf*(_SC_XBS5_LPBIG_OFFBIG) returns –1, the meaning of this value is unspecified. Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to build an application using a programming model with an **int** type using at least 32 bits and **long**, **pointer**, and **off_t** types using at least 64 bits.

XSI _CS_XBS5_LPBIG_OFFBIG_LDFLAGS (**LEGACY**)

If *sysconf*(_SC_XBS5_LPBIG_OFFBIG) returns –1, the meaning of this value is unspecified. Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build an application using a programming model with an **int** type using at least 32 bits and **long**, **pointer**, and **off_t** types using at least 64 bits.

XSI _CS_XBS5_LPBIG_OFFBIG_LIBS (**LEGACY**)

If *sysconf*(_SC_XBS5_LPBIG_OFFBIG) returns –1, the meaning of this value is unspecified. Otherwise, this value is the set of libraries to be given to the *cc* and *c99* utilities to build an application using a programming model with an **int** type using at least 32 bits and **long**, **pointer**, and **off_t** types using at least 64 bits.

XSI _CS_XBS5_LPBIG_OFFBIG_LINTFLAGS (**LEGACY**)

If *sysconf*(_SC_XBS5_LPBIG_OFFBIG) returns –1, the meaning of this value is unspecified.

Otherwise, this value is the set of options to be given to the *lint* utility to check application source using a programming model with an **int** type using at least 32 bits and **long**, **pointer**, and **off_t** types using at least 64 bits.

The following symbolic constants shall be defined for the *lseek*( ) and *fcntl*( ) functions and shall have distinct values:

SEEK_CUR Set file offset to current plus *offset*.

SEEK_END Set file offset to EOF plus *offset*.

SEEK_SET Set file offset to *offset*.

The following symbolic constants shall be defined as possible values for the *function* argument to the *lockf*( ) function:

F_LOCK Lock a section for exclusive use.

F_TEST Test section for locks by other processes.

F_TLOCK Test and lock a section for exclusive use.

F_ULOCK Unlock locked sections.

The following symbolic constants shall be defined for *pathconf*( ):

ADV _PC_ALLOC_SIZE_MIN
AIO _PC_ASYNC_IO
_PC_CHOWN_RESTRICTED
_PC_FILESIZEBITS
_PC_LINK_MAX
_PC_MAX_CANON
_PC_MAX_INPUT
_PC_NAME_MAX
_PC_NO_TRUNC
_PC_PATH_MAX
_PC_PIPE_BUF
_PC_PRIO_IO
ADV _PC_REC_INCR_XFER_SIZE
_PC_REC_MAX_XFER_SIZE
_PC_REC_MIN_XFER_SIZE
_PC_REC_XFER_ALIGN
_PC_SYNC_IO
_PC_VDISABLE

The following symbolic constants shall be defined for *sysconf*( ):

_SC_2_C_BIND
_SC_2_C_DEV
_SC_2_C_VERSION
_SC_2_CHAR_TIME
_SC_2_FORT_DEV
_SC_2_FORT_RUN
_SC_2_LOCALEDEF
_SC_2_PBS
_SC_2_PBS_ACCOUNTING
_SC_2_PBS_CHECKPOINT
_SC_2_PBS_LOCATE
_SC_2_PBS_MESSAGE

```
    _SC_2_PBS_TRACK
    _SC_2_SW_DEV
    _SC_2_UPE
    _SC_2_VERSION
    _SC_ADVISORY_INFO
    _SC_ARG_MAX
    _SC_AIO_LISTIO_MAX
    _SC_AIO_MAX
    _SC_AIO_PRIO_DELTA_MAX
    _SC_ASYNCHRONOUS_IO
XSI _SC_ATEXIT_MAX
BAR _SC_BARRIERS
    _SC_BASE
    _SC_BC_BASE_MAX
    _SC_BC_DIM_MAX
    _SC_BC_SCALE_MAX
    _SC_BC_STRING_MAX
    _SC_C_LANG_SUPPORT
    _SC_C_LANG_SUPPORT_R
    _SC_CHILD_MAX
    _SC_CLK_TCK
CS  _SC_CLOCK_SELECTION
    _SC_COLL_WEIGHTS_MAX
    _SC_CPUTIME
    _SC_DELAYTIMER_MAX
    _SC_DEVICE_IO
    _SC_DEVICE_SPECIFIC
    _SC_DEVICE_SPECIFIC_R
    _SC_EXPR_NEST_MAX
    _SC_FD_MGMT
    _SC_FIFO
    _SC_FILE_ATTRIBUTES
    _SC_FILE_LOCKING
    _SC_FILE_SYSTEM
    _SC_FSYNC
    _SC_GETGR_R_SIZE_MAX
    _SC_GETPW_R_SIZE_MAX
    _SC_HOST_NAME_MAX
XSI _SC_IOV_MAX
    _SC_JOB_CONTROL
    _SC_LINE_MAX
    _SC_LOGIN_NAME_MAX
    _SC_MAPPED_FILES
    _SC_MEMLOCK
    _SC_MEMLOCK_RANGE
    _SC_MEMORY_PROTECTION
    _SC_MESSAGE_PASSING
MON _SC_MONOTONIC_CLOCK
    _SC_MQ_OPEN_MAX
    _SC_MQ_PRIO_MAX
    _SC_MULTIPLE_PROCESS
    _SC_NETWORKING
```

_SC_NGROUPS_MAX
_SC_OPEN_MAX
XSI _SC_PAGE_SIZE
_SC_PAGESIZE
_SC_PIPE
_SC_PRIORITIZED_IO
_SC_PRIORITY_SCHEDULING
_SC_RE_DUP_MAX
THR _SC_READER_WRITER_LOCKS
_SC_REALTIME_SIGNALS
_SC_REGEXP
_SC_RTSIG_MAX
_SC_SAVED_IDS
_SC_SEMAPHORES
_SC_SEM_NSEMS_MAX
_SC_SEM_VALUE_MAX
_SC_SHARED_MEMORY_OBJECTS
_SC_SHELL
_SC_SIGNALS
_SC_SIGQUEUE_MAX
_SC_SINGLE_PROCESS
_SC_SPAWN
SPI _SC_SPIN_LOCKS
_SC_SPORADIC_SERVER
_SC_STREAM_MAX
_SC_SYNCHRONIZED_IO
_SC_SYSTEM_DATABASE
_SC_SYSTEM_DATABASE_R
_SC_THREAD_ATTR_STACKADDR
_SC_THREAD_ATTR_STACKSIZE
_SC_THREAD_CPUTIME
_SC_THREAD_DESTRUCTOR_ITERATIONS
_SC_THREAD_KEYS_MAX
_SC_THREAD_PRIO_INHERIT
_SC_THREAD_PRIO_PROTECT
_SC_THREAD_PRIORITY_SCHEDULING
_SC_THREAD_PROCESS_SHARED
_SC_THREAD_SAFE_FUNCTIONS
_SC_THREAD_SPORADIC_SERVER
_SC_THREAD_STACK_MIN
_SC_THREAD_THREADS_MAX
_SC_TIMEOUTS
_SC_THREADS
_SC_TIMER_MAX
_SC_TIMERS
TRC _SC_TRACE
TEF _SC_TRACE_EVENT_FILTER
TRI _SC_TRACE_INHERIT
TRL _SC_TRACE_LOG
_SC_TTY_NAME_MAX
TYM _SC_TYPED_MEMORY_OBJECTS
_SC_TZNAME_MAX

_SC_USER_GROUPS
_SC_USER_GROUPS_R
_SC_V6_ILP32_OFF32
_SC_V6_ILP32_OFFBIG
_SC_V6_LP64_OFF64
_SC_V6_LPBIG_OFFBIG
_SC_VERSION
XSI _SC_XBS5_ILP32_OFF32 (**LEGACY**)
_SC_XBS5_ILP32_OFFBIG (**LEGACY**)
_SC_XBS5_LP64_OFF64 (**LEGACY**)
_SC_XBS5_LPBIG_OFFBIG (**LEGACY**)
_SC_XOPEN_CRYPT
_SC_XOPEN_ENH_I18N
_SC_XOPEN_LEGACY
_SC_XOPEN_REALTIME
_SC_XOPEN_REALTIME_THREADS
_SC_XOPEN_SHM
_SC_XOPEN_STREAMS
_SC_XOPEN_UNIX
_SC_XOPEN_VERSION
_SC_XOPEN_XCU_VERSION

The two constants _SC_PAGESIZE and _SC_PAGE_SIZE may be defined to have the same value.

The following symbolic constants shall be defined for file streams:

STDERR_FILENO	File number of *stderr*; 2.
STDIN_FILENO	File number of *stdin*; 0.
STDOUT_FILENO	File number of *stdout*; 1.

Type Definitions

The **size_t**, **ssize_t**, **uid_t**, **gid_t**, **off_t**, and **pid_t** types shall be defined as described in **<sys/types.h>**.

The **useconds_t** type shall be defined as described in **<sys/types.h>**.

The **intptr_t** type shall be defined as described in **<inttypes.h>**.

Declarations

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int         access(const char *, int);
unsigned    alarm(unsigned);
int         chdir(const char *);
int         chown(const char *, uid_t, gid_t);
int         close(int);
size_t      confstr(int, char *, size_t);
char       *crypt(const char *, const char *);
char       *ctermid(char *);
int         dup(int);
```

(XSI: crypt, ctermid)

```
     int          dup2(int, int);
XSI  void         encrypt(char[64], int);
     int          execl(const char *, const char *, ...);
     int          execle(const char *, const char *, ...);
     int          execlp(const char *, const char *, ...);
     int          execv(const char *, char *const []);
     int          execve(const char *, char *const [], char *const []);
     int          execvp(const char *, char *const []);
     void        _exit(int);
     int          fchown(int, uid_t, gid_t);
XSI  int          fchdir(int);
SIO  int          fdatasync(int);
     pid_t        fork(void);
     long         fpathconf(int, int);
     int          fsync(int);
     int          ftruncate(int, off_t);
     char        *getcwd(char *, size_t);
     gid_t        getegid(void);
     uid_t        geteuid(void);
     gid_t        getgid(void);
     int          getgroups(int, gid_t []);
XSI  long         gethostid(void);
     int          gethostname(char *, size_t);
     char        *getlogin(void);
     int          getlogin_r(char *, size_t);
     int          getopt(int, char * const [], const char *);
XSI  pid_t        getpgid(pid_t);
     pid_t        getpgrp(void);
     pid_t        getpid(void);
     pid_t        getppid(void);
XSI  pid_t        getsid(pid_t);
     uid_t        getuid(void);
XSI  char        *getwd(char *); (LEGACY)
     int          isatty(int);
XSI  int          lchown(const char *, uid_t, gid_t);
     int          link(const char *, const char *);
XSI  int          lockf(int, int, off_t);
     off_t        lseek(int, off_t, int);
XSI  int          nice(int);
     long         pathconf(const char *, int);
     int          pause(void);
     int          pipe(int [2]);
XSI  ssize_t      pread(int, void *, size_t, off_t);
     ssize_t      pwrite(int, const void *, size_t, off_t);
     ssize_t      read(int, void *, size_t);
     ssize_t      readlink(const char *restrict, char *restrict, size_t);
     int          rmdir(const char *);
     int          setegid(gid_t);
     int          seteuid(uid_t);
     int          setgid(gid_t);
```

```
    int            setpgid(pid_t, pid_t);
XSI pid_t          setpgrp(void);
    int            setregid(gid_t, gid_t);
    int            setreuid(uid_t, uid_t);
    pid_t          setsid(void);
    int            setuid(uid_t);
    unsigned       sleep(unsigned);
XSI void           swab(const void *restrict, void *restrict, ssize_t);
    int            symlink(const char *, const char *);
    void           sync(void);
    long           sysconf(int);
    pid_t          tcgetpgrp(int);
    int            tcsetpgrp(int, pid_t);
XSI int            truncate(const char *, off_t);
    char          *ttyname(int);
    int            ttyname_r(int, char *, size_t);
XSI useconds_t     ualarm(useconds_t, useconds_t);
    int            unlink(const char *);
XSI int            usleep(useconds_t);
    pid_t          vfork(void);
    ssize_t        write(int, const void *, size_t);
```

Implementations may also include the *pthread_atfork*( ) prototype as defined in **<pthread.h>** (on page 286).

The following external variables shall be declared:

```
extern char   *optarg;
extern int    optind, opterr, optopt;
```

APPLICATION USAGE

IEEE Std 1003.1-200x only describes the behavior of systems that claim conformance to it. However, application developers who want to write applications that adapt to other versions of IEEE Std 1003.1 (or to systems that do not conform to any POSIX standard) may find it useful to code them so as to conditionally compile different code depending on the value of _POSIX_VERSION, for example.

```
#if _POSIX_VERSION == 200xxxL
/* Use the newer function that copes with large files. */
off_t pos=ftello(fp);
#else
/* Either this is an old version of POSIX, or _POSIX_VERSION is
   not even defined, so use the traditional function. */
long pos=ftell(fp);
#endif
```

Earlier versions of IEEE Std 1003.1 and of the Single UNIX Specification can be identified by the following macros:

POSIX.1-1988 standard
    _POSIX_VERSION==198808L

POSIX.1-1990 standard
    _POSIX_VERSION==199009L

ISO POSIX-1: 1996 standard
    _POSIX_VERSION==199506L

Single UNIX Specification, Version 1
_XOPEN_UNIX and _XOPEN_VERSION= =4

Single UNIX Specification, Version 2
_XOPEN_UNIX and _XOPEN_VERSION= =500

IEEE Std 1003.1-200x does not make any attempt to define application binary interaction with the underlying operating system. However, application developers may find it useful to query _SC_VERSION at runtime via *sysconf*( ) to determine whether the current version of the operating system supports the necessary functionality as in the following program fragment:

```
if (sysconf(_SC_VERSION) < 200xxxL) {
    fprintf(stderr, "POSIX.1-200x system required, terminating \n");
    exit(1);
}
```

RATIONALE

As IEEE Std 1003.1-200x evolved, certain options became sufficiently standardized that it was concluded that simply requiring one of the option choices was simpler than retaining the option. However, for backwards compatibility, the option flags (with required constant values) are retained.

Version Test Macros

The standard developers considered altering the definition of _POSIX_VERSION and removing _SC_VERSION from the specification of *sysconf*( ) since the utility to an application was deemed by some to be minimal, and since the implementation of the functionality is potentially problematic. However, they recognized that support for existing application binaries is a concern to manufacturers, application developers, and the users of implementations conforming to IEEE Std 1003.1-200x.

While the example using _SC_VERSION in the APPLICATION USAGE section does not provide the greatest degree of imaginable utility to the application developer or user, it is arguably better than a core dump or some other equally obscure result. (It is also possible for implementations to encode and recognize application binaries compiled in various POSIX.1-conforming environments, and modify the semantics of the underlying system to conform to the expectations of the application.) For the reasons outlined in the preceding paragraphs and in the APPLICATION USAGE section, the standard developers elected to retain the _POSIX_VERSION and _SC_VERSION functionality.

Compile-Time Symbolic Constants for System-Wide Options

IEEE Std 1003.1-200x now includes support in certain areas for the newly adopted policy governing options and stubs.

This policy provides flexibility for implementations in how they support options. It also specifies how conforming applications can adapt to different implementations that support different sets of options. It allows the following:

1. If an implementation has no interest in supporting an option, it does not have to provide anything associated with that option beyond the announcement that it does not support it.

2. An implementation can support a partial or incompatible version of an option (as a non-standard extension) as long as it does not claim to support the option.

3. An application can determine whether the option is supported. A strictly conforming application must check this announcement mechanism before first using anything associated with the option.

There is an important implication of this policy. IEEE Std 1003.1-200x cannot dictate the behavior of interfaces associated with an option when the implementation does not claim to support the option. In particular, it cannot require that a function associated with an unsupported option will fail if it does not perform as specified. However, this policy does not prevent a standard from requiring certain functions to always be present, but that they shall always fail on some implementations. The *setpgid*( ) function in the POSIX.1-1990 standard, for example, is considered appropriate.

The POSIX standards include various options, and the C language binding support for an option implies that the implementation must supply data types and function interfaces. An application must be able to discover whether the implementation supports each option.

Any application must consider the following three cases for each option:

1. Option never supported.

   The implementation advertises at compile time that the option will never be supported. In this case, it is not necessary for the implementation to supply any of the data types or function interfaces that are provided only as part of the option. The implementation might provide data types and functions that are similar to those defined by IEEE Std 1003.1-200x, but there is no guarantee for any particular behavior.

2. Option always supported.

   The implementation advertises at compile time that the option will always be supported. In this case, all data types and function interfaces shall be available and shall operate as specified.

3. Option might or might not be supported.

   Some implementations might not provide a mechanism to specify support of options at compile time. In addition, the implementation might be unable or unwilling to specify support or non-support at compile time. In either case, any application that might use the option at runtime must be able to compile and execute. The implementation must provide, at compile time, all data types and function interfaces that are necessary to allow this. In this situation, there must be a mechanism that allows the application to query, at runtime, whether the option is supported. If the application attempts to use the option when it is not supported, the result is unspecified unless explicitly specified otherwise in IEEE Std 1003.1-200x.

FUTURE DIRECTIONS

None.

SEE ALSO

<inttypes.h>, **<limits.h>**, **<sys/socket.h>**, **<sys/types.h>**, **<termios.h>**, **<wctype.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *access*( ), *alarm*( ), *chdir*( ), *chown*( ), *close*( ), *crypt*( ), *ctermid*( ), *dup*( ), *encrypt*( ), *environ*, *exec*( ), *exit*( ), *fchdir*( ), *fchown*( ), *fcntl*( ), *fork*( ), *fpathconf*( ), *fsync*( ), *ftruncate*( ), *getcwd*( ), *getegid*( ), *geteuid*( ), *getgid*( ), *getgroups*( ), *gethostid*( ), *gethostname*( ), *getlogin*( ), *getpgid*( ), *getpgrp*( ), *getpid*( ), *getppid*( ), *getsid*( ), *getuid*( ), *isatty*( ), *lchown*( ), *link*( ), *lockf*( ), *lseek*( ), *nice*( ), *pathconf*( ), *pause*( ), *pipe*( ), *read*( ), *readlink*( ), *rmdir*( ), *setgid*( ), *setpgid*( ), *setpgrp*( ), *setregid*( ), *setreuid*( ), *setsid*( ), *setuid*( ), *sleep*( ), *swab*( ), *symlink*( ), *sync*( ), *sysconf*( ), *tcgetpgrp*( ), *tcsetpgrp*( ), *truncate*( ), *ttyname*( ), *ualarm*( ), *unlink*( ), *usleep*( ), *vfork*( ), *write*( )

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

The symbolic constants _XOPEN_REALTIME and _XOPEN_REALTIME_THREADS are added. _POSIX2_C_BIND, _XOPEN_ENH_I18N, and _XOPEN_SHM must now be set to a value other than −1 by a conforming implementation.

Large File System extensions are added.

The type of the argument to *sbrk*( ) is changed from **int** to **intptr_t**.

XBS constants are added to the list of constants for Options and Option Groups, to the list of constants for the *confstr*( ) function, and to the list of constants to the *sysconf*( ) function. These are all marked EX.

Issue 6

_POSIX2_C_VERSION is removed.

The Open Group Corrigendum U026/4 is applied, adding the prototype for *fdatasync*( ).

The Open Group Corrigendum U026/1 is applied, adding the symbols _SC_XOPEN_LEGACY, _SC_XOPEN_REALTIME, and _SC_XOPEN_REALTIME_THREADS.

The symbols _XOPEN_STREAMS and _SC_XOPEN_STREAMS are added to support the XSI STREAMS Option Group.

Text in the DESCRIPTION relating to conformance requirements is moved elsewhere in IEEE Std 1003.1-200x.

The legacy symbol _SC_PASS_MAX is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The _CS_POSIX_* and _CS_XBS5_* constants are added for the *confstr*( ) function.
- The _SC_XBS5_* constants are added for the *sysconf*( ) function.
- The symbolic constants F_ULOCK, F_LOCK, F_TLOCK, and F_TEST are added.
- The **uid_t**, **gid_t**, **off_t**, **pid_t**, and **useconds_t** types are mandated.

The *gethostname*( ) prototype is added for sockets.

New section added for System Wide Options.

Function prototypes for *setegid*( ) and *seteuid*( ) are added.

Option symbolic constants are added for _POSIX_ADVISORY_INFO, _POSIX_CPUTIME, _POSIX_SPAWN, _POSIX_SPORADIC_SERVER, _POSIX_THREAD_CPUTIME, _POSIX_THREAD_SPORADIC_SERVER, and _POSIX_TIMEOUTS, and *pathconf*( ) variables are added for _PC_ALLOC_SIZE_MIN, _PC_REC_INCR_XFER_SIZE, _PC_REC_MAX_XFER_SIZE, _PC_REC_MIN_XFER_SIZE, and _PC_REC_XFER_ALIGN for alignment with IEEE Std 1003.1d-1999.

The following are added for alignment with IEEE Std 1003.1j-2000:

- Option symbolic constants _POSIX_BARRIERS, _POSIX_CLOCK_SELECTION, _POSIX_MONOTONIC_CLOCK, _POSIX_READER_WRITER_LOCKS, _POSIX_SPIN_LOCKS, and _POSIX_TYPED_MEMORY_OBJECTS

- *sysconf*( ) variables _SC_BARRIERS, _SC_CLOCK_SELECTION, _SC_MONOTONIC_CLOCK, _SC_READER_WRITER_LOCKS, _SC_SPIN_LOCKS, and _SC_TYPED_MEMORY_OBJECTS

The _SC_XBS5 macros associated with the ISO / IEC 9899: 1990 standard are marked LEGACY, and new equivalent _SC_V6 macros associated with the ISO / IEC 9899: 1999 standard are introduced.

The *getwd*( ) function is marked LEGACY.

The **restrict** keyword is added to the prototypes for *readlink*( ) and *swab*( ).

Constants for options are now harmonized, so when supported they take the year of approval of IEEE Std 1003.1-200x as the value.

The following are added for alignment with IEEE Std 1003.1q-2000:

- Optional symbolic constants _POSIX_TRACE, _POSIX_TRACE_EVENT_FILTER, _POSIX_TRACE_LOG, and _POSIX_TRACE_INHERIT
- The *sysconf*( ) symbolic constants _SC_TRACE, _SC_TRACE_EVENT_FILTER, _SC_TRACE_LOG, and _SC_TRACE_INHERIT.

The *brk*( ) and *sbrk*( ) legacy functions are removed.

The Open Group Base Resolution bwg2001-006 is applied, which reworks the XSI versioning information.

The Open Group Base Resolution bwg2001-008 is applied, changing the *namelen* parameter for *gethostname*( ) from **socklen_t** to **size_t**.

NAME

utime.h — access and modification times structure

SYNOPSIS

```
#include <utime.h>
```

DESCRIPTION

The **<utime.h>** header shall declare the structure **utimbuf**, which shall include the following members:

```
time_t    actime     Access time.
time_t    modtime    Modification time.
```

The times shall be measured in seconds since the Epoch.

The type **time_t** shall be defined as described in **<sys/types.h>**.

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided. |

```
int utime(const char *, const struct utimbuf *);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *utime*( )

CHANGE HISTORY

First released in Issue 3.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The **time_t** type is defined.

NAME

utmpx.h — user accounting database definitions

SYNOPSIS

XSI

```
#include <utmpx.h>
```

DESCRIPTION

The **<utmpx.h>** header shall define the **utmpx** structure that shall include at least the following members:

`char`	`ut_user[]`	User login name.
`char`	`ut_id[]`	Unspecified initialization process identifier.
`char`	`ut_line[]`	Device name.
`pid_t`	`ut_pid`	Process ID.
`short`	`ut_type`	Type of entry.
`struct timeval`	`ut_tv`	Time entry was made.

The **pid_t** type shall be defined through **typedef** as described in **<sys/types.h>**.

The **timeval** structure shall be defined as described in **<sys/time.h>**.

Inclusion of the **<utmpx.h>** header may also make visible all symbols from **<sys/time.h>**.

The following symbolic constants shall be defined as possible values for the *ut_type* member of the **utmpx** structure:

EMPTY	No valid user accounting information.
BOOT_TIME	Identifies time of system boot.
OLD_TIME	Identifies time when system clock changed.
NEW_TIME	Identifies time after system clock changed.
USER_PROCESS	Identifies a process.
INIT_PROCESS	Identifies a process spawned by the init process.
LOGIN_PROCESS	Identifies the session leader of a logged in user.
DEAD_PROCESS	Identifies a session leader who has exited.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
void          endutxent(void);
struct utmpx *getutxent(void);
struct utmpx *getutxid(const struct utmpx *);
struct utmpx *getutxline(const struct utmpx *);
struct utmpx *pututxline(const struct utmpx *);
void          setutxent(void);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<sys/time.h>, **<sys/types.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *endutxent*( )

CHANGE HISTORY

First released in Issue 4, Version 2.

NAME

wchar.h — wide-character handling

SYNOPSIS

```
#include <wchar.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these symbols in this header.

The **<wchar.h>** header shall define the following types:

wchar_t As described in **<stddef.h>**.

wint_t An integer type capable of storing any valid value of **wchar_t** or WEOF.

XSI **wctype_t** A scalar type of a data object that can hold values which represent locale-specific character classification.

mbstate_t An object type other than an array type that can hold the conversion state information necessary to convert between sequences of (possibly multi-byte) characters and wide characters. XSI If a codeset is being used such that an **mbstate_t** needs to preserve more than 2 levels of reserved state, the results are unspecified.

XSI **FILE** As described in **<stdio.h>**.

size_t As described in **<stddef.h>**.

XSI **va_list** As described in **<stdarg.h>**.

The implementation shall support one or more programming environments in which the width of **wint_t** is no greater than the width of type **long**. The names of these programming environments can be obtained using the *confstr*( ) function or the *getconf* utility.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
wint_t         btowc(int);
wint_t         fgetwc(FILE *);
wchar_t       *fgetws(wchar_t *restrict, int, FILE *restrict);
wint_t         fputwc(wchar_t, FILE *);
int            fputws(const wchar_t *restrict, FILE *restrict);
int            fwide(FILE *, int);
int            fwprintf(FILE *restrict, const wchar_t *restrict, ...);
int            fwscanf(FILE *restrict, const wchar_t *restrict, ...);
wint_t         getwc(FILE *);
wint_t         getwchar(void);
int            iswalnum(wint_t);
int            iswalpha(wint_t);
int            iswcntrl(wint_t);
int            iswctype(wint_t, wctype_t);
int            iswdigit(wint_t);
int            iswgraph(wint_t);
int            iswlower(wint_t);
int            iswprint(wint_t);
int            iswpunct(wint_t);
```

```
    int           iswspace(wint_t);
    int           iswupper(wint_t);
    int           iswxdigit(wint_t);
    size_t        mbrlen(const char *restrict, size_t, mbstate_t *restrict);
    size_t        mbrtowc(wchar_t *restrict, const char *restrict, size_t,
                      mbstate_t *restrict);
    int           mbsinit(const mbstate_t *);
    size_t        mbsrtowcs(wchar_t *restrict, const char **restrict, size_t,
                      mbstate_t *restrict);
    wint_t        putwc(wchar_t, FILE *);
    wint_t        putwchar(wchar_t);
    int           swprintf(wchar_t *restrict, size_t,
                      const wchar_t *restrict, ...);
    int           swscanf(const wchar_t *restrict,
                      const wchar_t *restrict, ...);
XSI wint_t        towlower(wint_t);
    wint_t        towupper(wint_t);
    wint_t        ungetwc(wint_t, FILE *);
    int           vfwprintf(FILE *restrict, const wchar_t *restrict, va_list);
    int           vfwscanf(FILE *restrict, const wchar_t *restrict, va_list);
    int           vwprintf(const wchar_t *restrict, va_list);
    int           vswprintf(wchar_t *restrict, size_t,
                      const wchar_t *restrict, va_list);
    int           vswscanf(const wchar_t *restrict, const wchar_t *restrict,
                      va_list);
    int           vwscanf(const wchar_t *restrict, va_list);
    size_t        wcrtomb(char *restrict, wchar_t, mbstate_t *restrict);
    wchar_t      *wcscat(wchar_t *restrict, const wchar_t *restrict);
    wchar_t      *wcschr(const wchar_t *, wchar_t);
    int           wcscmp(const wchar_t *, const wchar_t *);
    int           wcscoll(const wchar_t *, const wchar_t *);
    wchar_t      *wcscpy(wchar_t *restrict, const wchar_t *restrict);
    size_t        wcscspn(const wchar_t *, const wchar_t *);
    size_t        wcsftime(wchar_t *restrict, size_t,
                      const wchar_t *restrict, const struct tm *restrict);
    size_t        wcslen(const wchar_t *);
    wchar_t      *wcsncat(wchar_t *restrict, const wchar_t *restrict, size_t);
    int           wcsncmp(const wchar_t *, const wchar_t *, size_t);
    wchar_t      *wcsncpy(wchar_t *restrict, const wchar_t *restrict, size_t);
    wchar_t      *wcspbrk(const wchar_t *, const wchar_t *);
    wchar_t      *wcsrchr(const wchar_t *, wchar_t);
    size_t        wcsrtombs(char *restrict, const wchar_t **restrict,
                      size_t, mbstate_t *restrict);
    size_t        wcsspn(const wchar_t *, const wchar_t *);
    wchar_t      *wcsstr(const wchar_t *restrict, const wchar_t *restrict);
    double        wcstod(const wchar_t *restrict, wchar_t **restrict);
    float         wcstof(const wchar_t *restrict, wchar_t **restrict);
    wchar_t      *wcstok(wchar_t *restrict, const wchar_t *restrict,
                      wchar_t **restrict);
    long          wcstol(const wchar_t *restrict, wchar_t **restrict, int);
    long double   wcstold(const wchar_t *restrict, wchar_t **restrict);
    long long     wcstoll(const wchar_t *restrict, wchar_t **restrict, int);
    unsigned long wcstoul(const wchar_t *restrict, wchar_t **restrict, int);
    unsigned long long
                  wcstoull(const wchar_t *restrict, wchar_t **restrict, int);
XSI wchar_t      *wcswcs(const wchar_t *, const wchar_t *);
    int           wcswidth(const wchar_t *, size_t);
    size_t        wcsxfrm(wchar_t *restrict, const wchar_t *restrict, size_t);
    int           wctob(wint_t);
XSI wctype_t      wctype(const char *);
    int           wcwidth(wchar_t);
    wchar_t      *wmemchr(const wchar_t *, wchar_t, size_t);
    int           wmemcmp(const wchar_t *, const wchar_t *, size_t);
    wchar_t      *wmemcpy(wchar_t *restrict, const wchar_t *restrict, size_t);
    wchar_t      *wmemmove(wchar_t *, const wchar_t *, size_t);
    wchar_t      *wmemset(wchar_t *, wchar_t, size_t);
    int           wprintf(const wchar_t *restrict, ...);
    int           wscanf(const wchar_t *restrict, ...);
```

The **<wchar.h>** header shall define the following macros:

WCHAR_MAX The maximum value representable by an object of type **wchar_t**.

WCHAR_MIN The minimum value representable by an object of type **wchar_t**.

WEOF Constant expression of type **wint_t** that is returned by several WP functions to indicate end-of-file.

NULL As described in **<stddef.h>**.

The tag **tm** shall be declared as naming an incomplete structure type, the contents of which are described in the header **<time.h>**.

CX Inclusion of the **<wchar.h>** header may make visible all symbols from the headers **<ctype.h>**, **<stdio.h>**, **<stdarg.h>**, **<stdlib.h>**, **<string.h>**, **<stddef.h>**, and **<time.h>**.

APPLICATION USAGE

None.

RATIONALE

In the ISO C standard, the symbols referenced as XSI extensions are in **<wctype.h>**. Their presence here is thus an extension.

FUTURE DIRECTIONS

None.

SEE ALSO

<ctype.h>, **<stdarg.h>**, **<stddef.h>**, **<stdio.h>**, **<stdlib.h>**, **<string.h>**, **<time.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *btowc*( ), *confstr*( ), *fgetwc*( ), *fgetws*( ), *fputwc*( ), *fputws*( ), *fwide*( ), *fwprintf*( ), *fwscanf*( ), *getwc*( ), *getwchar*( ), *iswalnum*( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *iswctype*( ), *mbsinit*( ), *mbrlen*( ), *mbrtowc*( ), *mbsrtowcs*( ), *putwc*( ), *putwchar*( ), *swprintf*( ), *swscanf*( ), *towlower*( ), *towupper*( ), *ungetwc*( ), *vfwprintf*( ), *vfwscanf*( ), *vswprintf*( ), *vswscanf*( ), *vwscanf*( ), *wcrtomb*( ), *wcsrtombs*( ), *wcscat*( ), *wcschr*( ), *wcscmp*( ), *wcscoll*( ), *wcscpy*( ), *wcscspn*( ), *wcsftime*( ), *wcslen*( ), *wcsncat*( ), *wcsncmp*( ), *wcsncpy*( ), *wcspbrk*( ), *wcsrchr*( ), *wcsspn*( ), *wcsstr*( ), *wcstod*( ), *wcstof*( ), *wcstok*( ), *wcstol*( ), *wcstold*( ), *wcstoll*( ), *wcstoul*( ), *wcstoull*( ), *wcswcs*( ), *wcswidth*( ), *wcsxfrm*( ), *wctob*( ), *wctype*( ), *wcwidth*( ), *wmemchr*( ), *wmemcmp*( ), *wmemcpy*( ), *wmemmove*( ), *wmemset*( ), *wprintf*( ), *wscanf*( ), the Shell and Utilities volume of IEEE Std 1003.1-200x, *getconf*

CHANGE HISTORY

First released in Issue 4.

Issue 5

Aligned with the ISO / IEC 9899: 1990 / Amendment 1: 1995 (E).

Issue 6

The Open Group Corrigendum U021/10 is applied. The prototypes for *wcswidth*( ) and *wcwidth*( ) are marked as extensions.

The Open Group Corrigendum U028/5 is applied, correcting the prototype for the *mbsinit*( ) function.

The following changes are made for alignment with the ISO / IEC 9899: 1999 standard:

- Various function prototypes are updated to add the **restrict** keyword.
- The functions *vfwscanf*( ), *vswscanf*( ), *wcstof*( ), *wcstold*( ), *wcstoll*( ), and *wcstoull*( ) are added.

The type **wctype_t**, the *isw**( ), *to**( ), and *wctype*( ) functions are marked as XSI extensions.

NAME

wctype.h — wide-character classification and mapping utilities

SYNOPSIS

```
#include <wctype.h>
```

DESCRIPTION

CX Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these symbols in this header.

The **<wctype.h>** header shall define the following types:

wint_t As described in **<wchar.h>**.

wctrans_t A scalar type that can hold values which represent locale-specific character mappings.

wctype_t As described in **<wchar.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int       iswalnum(wint_t);
int       iswalpha(wint_t);
int       iswblank(wint_t);
int       iswcntrl(wint_t);
int       iswdigit(wint_t);
int       iswgraph(wint_t);
int       iswlower(wint_t);
int       iswprint(wint_t);
int       iswpunct(wint_t);
int       iswspace(wint_t);
int       iswupper(wint_t);
int       iswxdigit(wint_t);
int       iswctype(wint_t, wctype_t);
wint_t    towctrans(wint_t, wctrans_t);
wint_t    towlower(wint_t);
wint_t    towupper(wint_t);
wctrans_t wctrans(const char *);
wctype_t  wctype(const char *);
```

The **<wctype.h>** header shall define the following macro name:

WEOF Constant expression of type **wint_t** that is returned by several MSE functions to indicate end-of-file.

For all functions described in this header that accept an argument of type **wint_t**, the value is representable as a **wchar_t** or equals the value of WEOF. If this argument has any other value, the behavior is undefined.

The behavior of these functions shall be affected by the *LC_CTYPE* category of the current locale.

CX Inclusion of the **<wctype.h>** header may make visible all symbols from the headers **<ctype.h>**, **<stdio.h>**, **<stdarg.h>**, **<stdlib.h>**, **<string.h>**, **<stddef.h>**, **<time.h>**, and **<wchar.h>**.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<locale.h>, **<wchar.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *iswalnum*( ), *iswalpha*( ), *iswblank*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), *towctrans*( ), *towlower*( ), *towupper*( ), *wctrans*( ), *wctype*( )

CHANGE HISTORY

First released in Issue 5. Derived from the ISO / IEC 9899: 1990 / Amendment 1: 1995 (E).

Issue 6

The *iswblank*( ) function is added for alignment with the ISO / IEC 9899: 1999 standard.

NAME

wordexp.h — word-expansion types

SYNOPSIS

```
#include <wordexp.h>
```

DESCRIPTION

The **<wordexp.h>** header shall define the structures and symbolic constants used by the *wordexp*( ) and *wordfree*( ) functions.

The structure type **wordexp_t** shall contain at least the following members:

`size_t`	`we_wordc`	Count of words matched by *words.*
`char`	`**we_wordv`	Pointer to list of expanded words.
`size_t`	`we_offs`	Slots to reserve at the beginning of *we_wordv*.

The *flags* argument to the *wordexp*( ) function shall be the bitwise-inclusive OR of the following flags:

WRDE_APPEND Append words to those previously generated.

WRDE_DOOFFS Number of null pointers to prepend to *we_wordv*.

WRDE_NOCMD Fail if command substitution is requested.

WRDE_REUSE The *pwordexp* argument was passed to a previous successful call to *wordexp*( ), and has not been passed to *wordfree*( ). The result is the same as if the application had called *wordfree*( ) and then called *wordexp*( ) without WRDE_REUSE.

WRDE_SHOWERR Do not redirect *stderr* to **/dev/null**.

WRDE_UNDEF Report error on an attempt to expand an undefined shell variable.

The following constants shall be defined as error return values:

WRDE_BADCHAR One of the unquoted characters—<newline>, `'|'`, `'&'`, `';'`, `'<'`, `'>'`, `'('`, `')'`, `'{'`, `'}'`—appears in *words* in an inappropriate context.

WRDE_BADVAL Reference to undefined shell variable when WRDE_UNDEF is set in *flags*.

WRDE_CMDSUB Command substitution requested when WRDE_NOCMD was set in flags.

WRDE_NOSPACE Attempt to allocate memory failed.

OB XSI WRDE_NOSYS Reserved.

WRDE_SYNTAX Shell syntax error, such as unbalanced parentheses or unterminated string.

The **<wordexp.h>** header shall define the following type:

XSI **size_t** As described in **<stddef.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int  wordexp(const char *restrict, wordexp_t *restrict, int);
void wordfree(wordexp_t *);
```

The implementation may define additional macros or constants using names beginning with WRDE_.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<stddef.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *wordexp*( ), the Shell and Utilities volume of IEEE Std 1003.1-200x

CHANGE HISTORY

First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 6

The **restrict** keyword is added to the prototype for *wordexp*( ).

The WRDE_NOSYS constant is marked obsolescent.