This is an extract from:

A Source Book from The Open Group

The Authorized Guide to the Single UNIX Specification, Version 3

The Open Group

A Source Book from The Open Group

The Authorized Guide to the Single UNIX Specification, Version 3

Published in the U.K. by The Open Group, January 2005.

Any comments relating to the material contained in this document may be submitted to:

The Open Group
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:

OGSpecs@opengroup.org

Chapter 1

The Single UNIX Specification

Many names have been applied to the work that has culminated in the Single UNIX Specification and the accompanying UNIX certification program. It began as the *Common API Specification*, became *Spec 1170*, and is now in its latest iteration the *Single UNIX Specification, Version 3*, published in a number of The Open Group's Technical Standards, the core of which are also IEEE Std 1003.1-2001.

This introductory chapter describes the Single UNIX Specification, giving an overview of its evolution to become the Single UNIX Specification, Version 3.

1.1 Introduction

This document introduces the Single UNIX Specification, Version 3 providing a brief history of the work that led to its creation, information on what's new in Version 3, migration information on the changes between Version 2 and Version 3, what is covered in each part of the specification, and how to use the documents to advantage to develop portable applications.

1.2 Background

Today, the cost of developing applications software is considerably greater than the cost of the hardware on which the applications run. Preserving this investment in an organization's critical applications, while freeing the organization to mix and match platforms as hardware costs fall, has led to the idea of ''open systems''. One of the cornerstones of open systems is portability—the ability to move an application's source code to another platform and rebuild the application without changing the source.

Software portability problems do not just affect an organization directly through its own applications. Hardware vendors often find their market limited by the applications that run on their platforms. Independent software vendors (ISVs) often limit themselves in the number of platforms on which they sell their products, due to the costs of maintaining multiple ports of the application for multiple platforms. Consumers are caught, often choosing hardware platforms based on the applications they need to purchase, but without the ability to know what new applications will be available on these platforms in the future.

The Single UNIX Specification, provides a single, open consensus specification to support development of portable applications. There is also a mark, or brand, that is used to identify those products that conform to the Single UNIX Specification. Both the specification and the trademark are managed and held in trust for the industry by The Open Group.

1.3 The Value of Standards

The UNIX system's increasing popularity spawned the development of a number of variations of the UNIX operating system in the 1980s, and the existence of these caused a mid-life crisis. Standardization had progressed slowly and methodically in domains such as telecommunications and third-generation languages; yet no one had addressed standards at the operating system level. For suppliers, the thought of a uniform operating environment was disconcerting. Consumer lock-in was woven tightly into the fabric of the industry. Individual consumers, particularly those with UNIX system experience, envisioned standardized environments, but had no way to pull the market in their direction.

However, for one category of consumer—governments—the standardization of the UNIX system was both desirable and within reach. Governments have clout and are the largest consumers of information technology products and services in the world. Driven by the need to improve commonality, both U.S. and European governments endorsed a shift to the UNIX system. The Institute of Electrical and Electronic Engineers POSIX family of standards, along with standards from ISO, ANSI, and others, led the way. Consortia such as the X/Open Company (merged with the Open Software Foundation in 1995 to form The Open Group) hammered out draft standards to accelerate the process.

In 1994, the definitive specification of what constitutes a UNIX system was finalized through X/Open Company's consensus process. The Single UNIX Specification was born, not from a theoretical, ivory tower approach, but by analyzing the applications that were in use in businesses across the world.

With the active support of government and commercial buyers alike, vendors began to converge on products that implement the Single UNIX Specification, and now all major vendors have products labeled UNIX 95, UNIX 98, or UNIX 03 which indicates that the vendor guarantees that the product conforms to the Single UNIX Specification.

Vendors continue to add value to the UNIX system, particularly in areas of new technology; however, that value will always be built upon a single, consensus standard. Meanwhile, the functionality of the UNIX system was established and the mid-life crisis was resolved. Suppliers today provide UNIX systems that are built upon the single, consensus standard.

1.4 The Single UNIX Specification

The Single UNIX Specification is a set of open, consensus specifications that define the requirements for a conformant UNIX system. The standardized programming environment provides a broad-based functional set of interfaces to support the porting of existing UNIX applications and the development of new applications. The environment also supports a rich set of tools for application development.

The Single UNIX Specification came into being when in 1994 Novell (who had acquired the UNIX systems business of AT&T/USL) decided to get out of that business. Rather than sell the business as a single entity, Novell transferred the rights to the UNIX trademark and the specification (that subsequently became the Single UNIX Specification) to The Open Group (at the time X/Open Company). Subsequently, it sold the source code and the product implementation (UNIXWARE) to SCO. The Open Group also owns the trademark UNIXWARE, transferred to them from SCO more recently.

Today, The Open Group's UNIX trademark may be applied to any operating system product that is guaranteed to meet the Single UNIX Specification. The Single UNIX Specification is designed to give software developers a single set of APIs to be supported by every UNIX system, and by making the specification freely available on the Web (*www.unix.org*), thousands of developers

are able to access this resource.

The most significant consequence of the Single UNIX Specification initiative is that it shifts the focus of attention away from incompatible UNIX system product implementations on to compliance with a single, agreed set of APIs. If an operating system meets the specification, and commonly available applications can run on it, then it can be reliably viewed as open.

1.5 Benefits for Application Developers

A single standard for the UNIX operating system means:

- Improved portability
- Faster development through the increased number of standard interfaces
- More innovation is possible, due to the reduced time spent porting applications

1.6 Benefits for Users

The Single UNIX Specification will evolve and develop in response to market needs protecting users' investment in existing systems and applications. The availability of the UNIX system from multiple suppliers gives users freedom of choice rather than being locked in to a single supplier. And the wide range of applications—built on the UNIX system's strengths of scalability, availability, and reliability—ensures that mission-critical business needs can be met.

1.7 The Common API Specification

The Common API Specification project started when several vendors (Sun Microsystems, IBM, Hewlett-Packard, Novell/USL, and OSF) joined together to provide a single unified specification of the UNIX system services. By implementing a single common definition of the UNIX system services, third-party independent software vendors (ISVs) would be able to more easily deliver strategic applications on all of these vendors' platforms at once.

The focus of this initiative was to deliver the core application interfaces used by current application programs. The economic driver that initiated the project was to ease the porting of existing successful applications. While the work was led by a central group of vendors, it received widespread support within the industry.

A two-pronged approach was used to develop the Common API Specification. First, a set of formal industry specifications was chosen to form the overall base for the work. This would provide stability and vendor neutrality, and lay a well charted course for future application development, taking advantage of the careful work that had gone into developing these specifications. It would also preserve the portability of existing applications already developed to these core models.

The X/Open XPG4 Base was chosen as the stable functional base from which to start. XPG4 Base supports the POSIX.1 system interface and the ANSI C/ISO C standard at its core. It also provides a rich set of 174 commands and utilities.

Many UNIX systems already conformed to this specification. To this base was added the traditional UNIX System V Interface Definition (SVID) Edition 3, Level 1 calls, and the OSF Application Environment Specification (AES) Full Use interface definitions. These represented the stable central core of the latter two specifications.

The second part of the approach was to incorporate interfaces that are acknowledged common practice but have not yet been incorporated into any formal specification or standard. The intent was to ensure that existing applications running on UNIX systems would port with relative ease to a platform supporting the Common API Specification. A survey of real-world applications was used to determine what additional interfaces would be required in the specification.

Fifty successful application packages were selected to be analyzed using the following criteria:

- Ranked in International Data Corporation's *Survey of Leading UNIX Applications*, 1992
- The application's domain of applicability was checked to ensure that no single application type (for example, databases) was overly represented
- The application had to be available for analysis either as source code, or as a shared or dynamic linked library

From the group of fifty, the top ten were selected carefully, ensuring that no more than two representative application packages in a particular problem space were chosen. The ten chosen applications were:

AutoCAD	FrameMaker	Island Write/Paint	SAS (4GL)	Teamwork
Cadence	Informix	Lotus 1-2-3	Sybase	WordPerfect

APIs used by the applications that were not part of the base specifications were analyzed:

- If an API was used by any of the top ten applications, it was considered for inclusion.
- If an API was not used by one of the top ten, but was used by any three of the remaining 40 applications, it was considered for inclusion.
- While the investigation of these 50 applications was representative of large complex applications, it was still not considered a broad enough survey, so an additional 3,500 modules were scanned. If an API was used at least seven times in modules that came from at least two platforms (to screen out vendor-specific libraries), then the interface was considered for inclusion.

The goal was to ensure that APIs in common use were included, even if they were not in the formal specifications that made up the base. Making the Common API Specification a superset of existing base specifications ensured that any existing applications should work unmodified. The sponsors of the work considered pruning the Common API Specification of interfaces from the base specifications that were not found to be in common use in the application survey.

This idea was rejected for two reasons.

While some of the interfaces in the base specifications were not yet considered common practice, their inclusion in the overall specification meant there existed clear signposts for future applications development work. Also, it was recognized that the applications chosen for the survey were still only a representative sample, and that many other applications not surveyed may use these interfaces.

When the survey was complete, there were 130 interfaces that did not already appear in the base specification. These interfaces seemed to be predominantly Berkeley BSD calls that had never been covered in XPG4 Base, the SVID, or the OSF AES, but did represent common practice in UNIX system applications developed originally on BSD-derived platforms. Such things as sockets and the 4.3BSD memory management calls were commonly used in many applications.

The resulting Common API Specification was impressive in its coverage. The top ten applications surveyed were completely covered. Of the remaining 40 application packages, the next 25 were within 5% of complete coverage. The software vendors involved all acknowledged

that it would be fairly straightforward for them to modify the 5% of applications to conform fully to the specification.

There were 1170 interfaces in the complete specification when the work was done (926 programming interfaces, 70 headers, 174 commands and utilities), and Spec 1170 was born.

Because of the breadth and origins of the specification, duplication of functionality existed. There were similar interfaces for doing the same thing in such areas as memory management (*bcopy*( ) *versus memmove*( )) and creating temporary filenames (*tmpnam*( ) *versus mktemp*( )). This duplication was allowed since it would increase the number of existing applications that would be portable in the new model. At the same time, certain functions were identified as the recommended practice for future development. There are cases where the duplicated functionality cannot coexist in the same application (for example, conflicting signals models), and it is important to ensure that the application is correctly configured if the expected behavior is to be observed.

In December 1993, Spec 1170 was delivered to X/Open for fasttrack processing into a proper industry supported specification. This work progressed through 1994 and culminated in publication of the Single UNIX Specification in October 1994. There were then more than 1170 interfaces in the specification as the review process shaped the document.

1.8 The Single UNIX Specification, Version 2

The Single UNIX Specification, Version 2 was published in February 1997. It incorporated standards for threads, realtime, removal of any architectural dependencies to permit 64-bit processing (which emerged from the Aspen Group), large file support, enhanced multibyte support, and addressed Year 2000 issues.

The five CAE specifications that make up the Single UNIX Specification, Version 2 are as follows:

- System Interface Definitions, Issue 5 (XBD5)
- System Interfaces and Headers, Issue 5 (XSH5)
- Commands and Utilities, Issue 5 (XCU5)
- Networking Services, Issue 5 (XNS5)
- X/Open Curses, Issue 4, Version 2

1.9 The Single UNIX Specification, Version 3

Work commenced in late 1998 on the Single UNIX Specification, Version 3. The core of the Single UNIX Specification, Version 3, collectively known as the *Base Specifications*, was developed, and is maintained, by a joint working group of members of the IEEE Portable Applications Standards Committee, members of The Open Group, and members of ISO/IEC Joint Technical Committee 1. This joint working group is known as the Austin Group.[1]

---

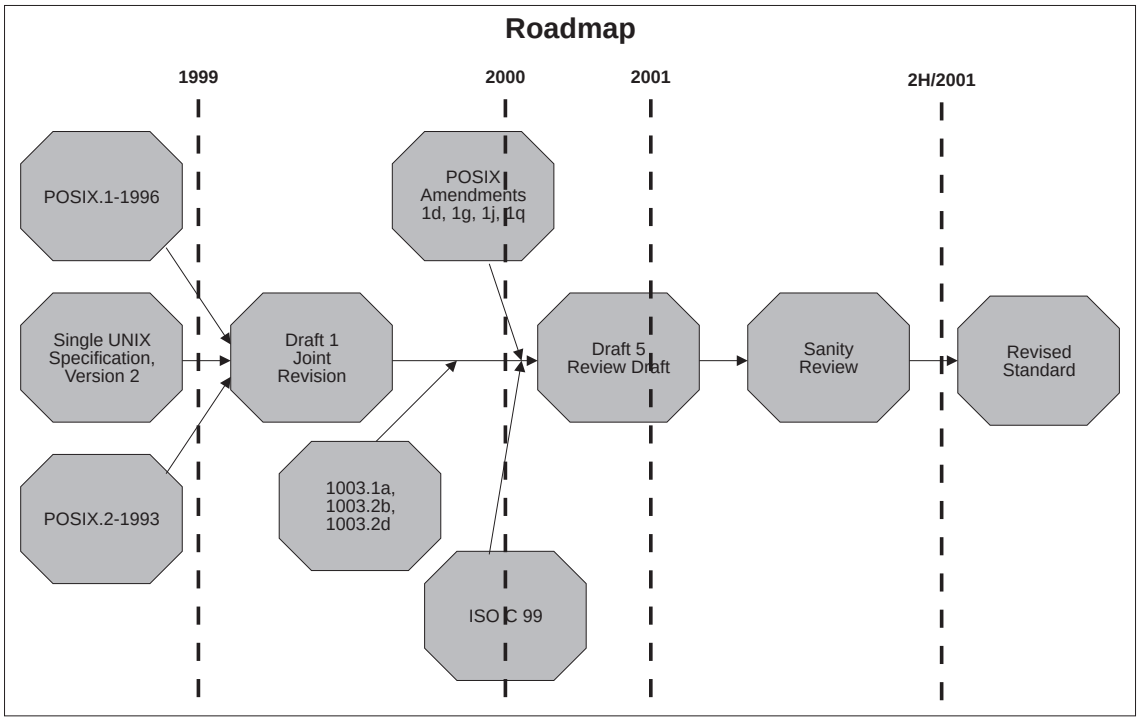
1. The Austin Group is named after the location of the inaugural meeting held at the IBM facility in Austin, Texas in September 1998.

The Austin Group arose out of discussions amongst the parties which started in early 1998, leading to an initial meeting and formation of the group in September 1998. This represented a seachange in attitude regarding development of two related specifications by three development organizations, which to date had been developed separately, often with the same standards developers involved.

The purpose of the Austin Group has been to revise, combine, and update the following standards: ISO/IEC 9945-1, ISO/IEC 9945-2, IEEE Std 1003.1, IEEE Std 1003.2, and the Base Specifications of The Open Group Single UNIX Specification.

After two initial meetings, an agreement was signed in July 1999 between The Open Group and the Institute of Electrical and Electronics Engineers (IEEE), Inc. to formalize the project, with the first draft of the revised specifications being made available at the same time. Under this agreement, The Open Group and IEEE agreed to share joint copyright of the resulting work. The Open Group has provided the chair and secretariat for the Austin Group.

The following figure shows the development roadmap from base documents through to approved standard:



This unique development has combined both the industry-led efforts and the formal standardization activities into a single initiative, and included a wide spectrum of participants including commercial, academia, government, and the open source communities.

The approach to specification development was one of ‘‘write once, adopt everywhere’’, with the resulting set of specifications being approved as IEEE Std 1003.1-2001 (POSIX), The Open Group Base Specifications, Issue 6, and ISO/IEC 9945. This set of specifications forms the core of the Single UNIX Specification, Version 3.

The Base Specifications, Issue 6 consist of the following Technical Standards:

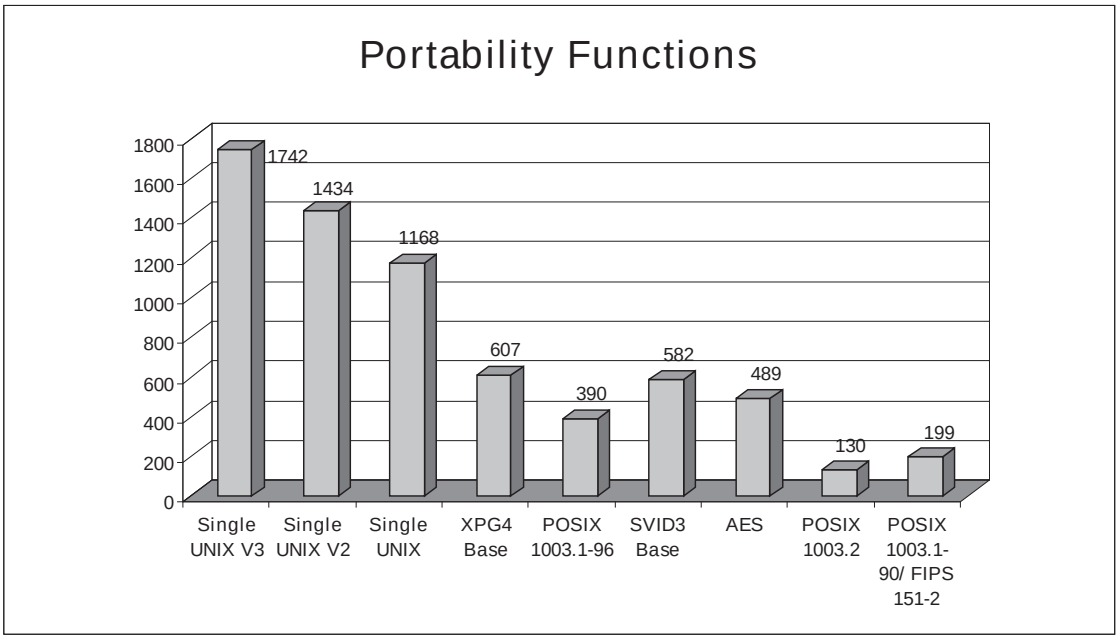
- Base Definitions, Issue 6 (XBD)
- Shell and Utilities, Issue 6 (XCU)
- System Interfaces, Issue 6 (XSH)
- Rationale (Informative)

The revision of the Base Specifications has tried to minimize the number of changes required to implementations which conform to the earlier versions of the approved standards to bring them into conformance with the current standard. Specifically, the scope of this work excluded doing any ‘‘new’’ work, but rather collecting into a single document what had been spread across a number of documents, and presenting it in what had been proven in practice to be a more effective way. Some changes to prior conforming implementations were unavoidable, primarily as a consequence of resolving conflicts found in prior revisions, or which became apparent when bringing the various pieces together.

However, since the revision now references the 1999 version of the ISO C standard, there are a number of unavoidable changes that have been made which will affect applications portability.

In addition to the Base Specifications, the Single UNIX Specification, Version 3 includes the X/Open Curses, Issue 4, Version 2 specification. Updates to X/Open Curses, Issue 4, Version 2 have been limited to production of a Corrigendum to allow it to exist in a Base Specifications, Issue 6 environment.

The following figure shows the interface counts for the different versions of the Single UNIX Specification and related documents.



1.10 IEEE Std 1003.1

The core of the Single UNIX Specification, Version 3 is also IEEE Std 1003.1-2001. IEEE Std 1003.1-2001 is a major revision and incorporates IEEE Std 1003.1-1990 (POSIX.1) and its subsequent amendments, and IEEE Std 1003.2-1992 (POSIX.2) and its subsequent amendments, combined with the core volumes of the Single UNIX Specification, Version 2. It is technically identical to The Open Group, Base Specifications, Issue 6; they are one and the same documents, the front cover having both designations. The final draft achieved 98% approval by the IEEE ballot group and was officially approved by the IEEE-SA Standards Board on December 6th 2001.

1.11 ISO/IEC 9945

The core of the Single UNIX Specification, Version 3 is also ISO/IEC 9945. ISO/IEC 9945 is the third edition of ISO/IEC 9945 and incorporates the former ISO/IEC 9945-1: 1996 (POSIX-1) and ISO/IEC 9945-2: 1993 (POSIX-2) plus subsequent IEEE amendments, combined with the core volumes of the Single UNIX Specification, Version 2. It is technically identical to The Open Group, Base Specifications, Issue 6 and IEEE Std 1003.1-2001; they are one and the same documents, the front cover having all the designations. The 2003 Edition of ISO/IEC 9945 was published on August 18th 2003.

ISO/IEC 9945 consists of the following parts, under the general title Information Technology — Portable Operating System Interface (POSIX):

Part 1: Base Definitions
Part 2: System Interfaces
Part 3: Shell and Utilities
Part 4: Rationale

1.12 The Austin Group

The Austin Group continues as the maintenance body for the Base Specifications. Since the initial publication of the Base Specifications, the Austin Group has produced two Technical Corrigenda making corrections to the document. Technical Corrigendum 1, also known as (identical to ISO/IEC 9945: 2002/Cor 1: 2003), was incorporated into the 2003 Edition of the document. Technical Corrigendum 2, also known as (identical to ISO/IEC 9945: 2003/Cor 1: 2004), is incorporated into the 2004 Edition.

Anyone wishing to participate in the Austin Group should contact the chair with their request. There are no fees for participation or membership. You may participate as an observer or as a contributor. You do not have to attend face-to-face meetings to participate; electronic participation is most welcome. For more information on the Austin Group and how to participate, see *www.opengroup.org/austin*.

1.13 Summary

The Single UNIX Specification offers benefits for all.

For users, products that support the Single UNIX Specification provide a powerful open platform for the development, implementation, and management of mission-critical computer systems. The specification facilitates freedom of choice across the widest possible range of systems, thereby protecting investment in existing software and data.

For software vendors, the Single UNIX Specification offers an opportunity to cut significantly the cost of developing and maintaining their products. It also expands the market opportunity by expanding the market itself, cutting time-to-market, and reducing product development costs.

For system vendors, the Single UNIX Specification unifies a previously fragmented market and offers the chance for all to compete on equal terms, while reducing the cost required to maintain technical incompatibilities.

Chapter 13

System Interfaces Migration

This chapter contains a section for each system interface defined in XSH, Issue 6 (including Technical Corrigendum 1 and Technical Corrigendum 2). Each section contains the SYNOPSIS and identifies syntax and semantic changes made to the interface in Issue 6 (if any), complete with examples where appropriate. Only changes that might affect an application programmer are identified. It should be noted that the following changes made in Issue 6 are not further discussed in this chapter.

ISO/IEC Terminology Changes

In order for the XSH document to also be an ISO/IEC standard, a number of terminology changes in the normative text were made, notably use of the term ‘‘shall’’ instead of the term ‘‘will’’ for requirements, and avoiding the term ‘‘must’’ for application requirements. Although these terminology changes affected most interfaces, no functional change is intended and these changes are not further noted in this chapter.

ISO C Extension Markings

Extensions beyond the ISO C standard have been marked in many interfaces. This explicitly shows where the ISO C standard has been extended within the text. No functional changes have been intended by this change, and this change is not further noted in this chapter.

_longjmp, _setjmp

Non-local goto

XSI
```
#include <setjmp.h>

void _longjmp(jmp_buf env, int val);
int _setjmp(jmp_buf env);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

_tolower

Transliterate uppercase characters to lowercase

XSI
```
#include <ctype.h>

int _tolower(int c);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

_toupper

Transliterate lowercase characters to uppercase

XSI
```
#include <ctype.h>

int _toupper(int c);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

a64l, l64a

Convert between a 32-bit integer and a radix-64 ASCII string

XSI
```
#include <stdlib.h>

long a64l(const char *s);
char *l64a(long value);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

abort

Generate an abnormal process abort

```
#include <stdlib.h>

void abort(void);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 , item XSH/TC1/D6/10 is applied, changing the DESCRIPTION of abnormal termination processing.

The ISO/IEC 9899: 1999 standard requires the *abort*() function to be async-signal-safe. Since IEEE Std 1003.1-2001 defers to the ISO C standard, this required a change to the DESCRIPTION from ‘‘shall include the effect of *fclose*()’’ to ‘‘may include an attempt to effect *fclose*().’’

The revised wording permits some backwards-compatibility and avoids a potential deadlock situation.

The Open Group Base Resolution bwg2002-003 is applied, removing the following XSI shaded paragraph from the DESCRIPTION:

> ‘‘On XSI-conformant systems, in addition the abnormal termination processing shall include the effect of *fclose*() on message catalog descriptors.’’

There were several reasons to remove this paragraph:

- No special processing of open message catalogs needs to be performed prior to abnormal process termination.
- The main reason to specifically mention that *abort*() includes the effect of *fclose*() on open streams is to flush output queued on the stream. Message catalogs in this context are read-only and, therefore, do not need to be flushed.
- The effect of *fclose*() on a message catalog descriptor is unspecified. Message catalog descriptors are allowed, but not required to be implemented using a file descriptor, but there is no mention in IEEE Std 1003.1-2001 of a message catalog descriptor using a standard I/O stream FILE object as would be expected by *fclose*().

abs

Return an integer absolute value

```
#include <stdlib.h>

int abs(int i);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

accept

Accept a new connection on a socket

```
#include <sys/socket.h>

int accept(int socket, struct sockaddr *restrict address,
    socklen_t *restrict address_len);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 The following change is made over the XNS, Issue 5.2 specification:

- The **restrict** keyword is added to the *accept*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

access

Determine accessibility of a file

```
#include <unistd.h>

int access(const char *path, int amode);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.
- The [ETXTBSY] optional error condition is added.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

acos, acosf, acosl

Arc cosine functions

```
#include <math.h>

double acos(double x);
float acosf(float x);
long double acosl(long double x);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *acosf*() and *acosl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

acosh, acoshf, acoshl

Inverse hyperbolic cosine functions

```
#include <math.h>

double acosh(double x);
float acoshf(float x);
long double acoshl(long double x);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The *acosh*() function is no longer marked as an extension.

The *acoshf*() and *acoshl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

aio_cancel

Cancel an asynchronous I/O request (**REALTIME**)

AIO
```
#include <aio.h>

int aio_cancel(int fildes, struct aiocb *aiocbp);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

, item XSH/TC2/D6/10 is applied, removing the words ‘‘to the calling process’’ in the RETURN VALUE section. The term was unnecessary and precluded threads.

aio_error

Retrieve errors status for an asynchronous I/O operation (**REALTIME**)

AIO
```
#include <aio.h>

int aio_error(const struct aiocb *aiocbp);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

aio_fsync

Asynchronous file synchronization (**REALTIME**)

AIO
```
#include <aio.h>

int aio_fsync(int op, struct aiocb *aiocbp);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

, item XSH/TC2/D6/11 is applied, removing the words ‘‘to the calling process’’ in the RETURN VALUE section. The term was unnecessary and precluded threads.

aio_read

Asynchronous read from a file (**REALTIME**)

AIO
```
#include <aio.h>

int aio_read(struct aiocb *aiocbp);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file description. This change is to support large files.
- In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support large files.

, item XSH/TC2/D6/12 is applied, rewording the DESCRIPTION when prioritized I/O is supported to account for threads, and removing the words ‘‘to the calling process’’ in the RETURN VALUE section.

, item XSH/TC2/D6/13 is applied, updating the [EINVAL] error, so that detection of an [EINVAL] error for an invalid value of *aiocbp->aio_reqprio* is only required if the Prioritized Input and Output option is supported.

aio_return

Retrieve return status of an asynchronous I/O operation (**REALTIME**)

AIO
```
#include <aio.h>

ssize_t aio_return(struct aiocb *aiocbp);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

The [EINVAL] error condition is updated as a ‘‘may fail’’. This is for consistency with the DESCRIPTION.

aio_suspend

Wait for an asynchronous I/O request (**REALTIME**)

AIO
```
#include <aio.h>

int aio_suspend(const struct aiocb *const list[], int nent,
    const struct timespec *timeout);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that the CLOCK_MONOTONIC clock, if supported, is used.

aio_write

Asynchronous write to a file (**REALTIME**)

AIO
```
#include <aio.h>

int aio_write(struct aiocb *aiocbp);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs past the offset maximum established in the open file description associated with *aiocbp->aio_fildes*.

- The [EFBIG] error is added as part of the large file support extensions.

, item XSH/TC2/D6/14 is applied, rewording the DESCRIPTION when prioritized I/O is supported to account for threads, and removing the words ‘‘to the calling process’’ in the RETURN VALUE section.

, item XSH/TC2/D6/15 is applied, updating the [EINVAL] error, so that detection of an [EINVAL] error for an invalid value of *aiocbp->aio_reqprio* is only required if the Prioritized Input and Output option is supported.

alarm

Schedule an alarm signal

```
#include <unistd.h>

unsigned alarm(unsigned seconds);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to indicate that interactions with the *setitimer*( ), *ualarm*( ), and *usleep*( ) functions are unspecified.

, item XSH/TC2/D6/16 is applied, replacing ‘‘an implementation-defined maximum value’’ with ‘‘an implementation-specific maximum value’’ in the RATIONALE.

asctime, asctime_r

Convert date and time to a string

```
#include <time.h>

char *asctime(const struct tm *timeptr);
TSF char *asctime_r(const struct tm *restrict tm, char *restrict buf);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *asctime_r*( ) function is marked as part of the Thread-Safe Functions option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

The DESCRIPTION of *asctime_r*( ) is updated to describe the format of the string returned.

The **restrict** keyword is added to the *asctime_r*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC2/D6/17 is applied, adding the CX extension in the RETURN VALUE section requiring that if the *asctime*( ) function is unsuccessful it returns NULL.

asin, asinf, asinl

Arc sine function

```
#include <math.h>

double asin(double x);
float asinf(float x);
long double asinl(long double x);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *asinf*() and *asinl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

asinh, asinhf, asinhl

Inverse hyperbolic sine functions

```
#include <math.h>

double asinh(double x);
float asinhf(float x);
long double asinhl(long double x);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The *asinh*() function is no longer marked as an extension.

The *asinhf*() and *asinhl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

assert

Insert program diagnostics

```
#include <assert.h>

void assert(scalar expression);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The prototype for the *expression* argument to *assert*() is changed from **int** to **scalar** for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION of *assert*() is updated for alignment with the ISO/IEC 9899: 1999 standard.

atan, atanf, atanl

Arc tangent function

```
#include <math.h>

double atan(double x);
float atanf(float x);
long double atanl(long double x);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *atanf*() and *atanl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

atan2, atan2f, atan2l

Arc tangent functions

```
#include <math.h>

double atan2(double y, double x);
float atan2f(float y, float x);
long double atan2l(long double y, long double x);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *atan2f*() and *atan2l*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

atanh, atanhf, atanhl

Inverse hyperbolic tangent functions

```
#include <math.h>

double atanh(double x);
float atanhf(float x);
long double atanhl(long double x);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The *atanh*() function is no longer marked as an extension.

The *atanhf*() and *atanhl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

atexit

Register a function to run at process termination

```
#include <stdlib.h>

int atexit(void (*func)(void));
```

Derivation First released in Issue 4. Derived from ANSI standard X3.159-1989, Programming Language C (ANSI C).

Issue 6 The DESCRIPTION is updated for alignment with the ISO/IEC 9899: 1999 standard.

atof

Convert a string to double-precision number

```
#include <stdlib.h>

double atof(const char *str);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

atoi

Convert a string to an integer

```
#include <stdlib.h>

int atoi(const char *str);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

atol, atoll

Convert a string to a long integer

```
#include <stdlib.h>

long atol(const char *str);
long long atoll(const char *nptr);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *atoll*() function is added for alignment with the ISO/IEC 9899: 1999 standard.

basename

Return the last component of a pathname

XSI
```
#include <libgen.h>

char *basename(char *path);
```

Derivation First released in Issue 4, Version 2.

Issue 6 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety. A portable multi-threaded application may only safely call the function when access to the function is serialized.

, item XSH/TC2/D6/20 is applied, changing the DESCRIPTION to make it clear that the string referenced is the string pointed to by *path*.

bcmp

Memory operations (**LEGACY**)

XSI
```
#include <strings.h>

int bcmp(const void *s1, const void *s2, size_t n);
```

Derivation First released in Issue 4, Version 2.

Issue 6 This function is marked LEGACY and may not be available on all implementations.

The *memcmp*() function is preferred over this function.

For maximum portability, it is recommended to replace the function call to *bcmp*() as follows:

```
#define bcmp(b1,b2,len) memcmp((b1), (b2), (size_t)(len))
```

bcopy

Memory operations (**LEGACY**)

XSI
```
#include <strings.h>

void bcopy(const void *s1, void *s2, size_t n);
```

Derivation First released in Issue 4, Version 2.

Issue 6 This function is marked LEGACY and may not be available on all implementations.

The *memmove*() function is preferred over this function.

The following are approximately equivalent (note the order of the arguments):

```
bcopy(s1,s2,n) ~= memmove(s2,s1,n)
```

For maximum portability, it is recommended to replace the function call to *bcopy*() as follows:

```
#define bcopy(b1,b2,len) (memmove((b2), (b1), (len)), (void) 0)
```

bind

Bind a name to a socket

```
#include <sys/socket.h>

int bind(int socket, const struct sockaddr *address,
    socklen_t address_len);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 No functional changes from the XNS, Issue 5.2 specification.

bsd_signal

Simplified signal facilities

OB XSI

```
#include <signal.h>

void (*bsd_signal(int sig, void (*func)(int)))(int);
```

Derivation First released in Issue 4, Version 2.

Issue 6 This function is marked obsolescent and is recommended not for use in new applications.

This function is a direct replacement for the BSD *signal*( ) function for simple applications that are installing a single-argument signal handler function. If a BSD signal handler function is being installed that expects more than one argument, the application has to be modified to use *sigaction*( ). The *bsd_signal*( ) function differs from *signal*( ) in that the SA_RESTART flag is set and the SA_RESETHAND is clear when *bsd_signal*( ) is used. The state of these flags is not specified for *signal*( ).

It is recommended that new applications use the *sigaction*( ) function.

bsearch

Binary search a sorted table

```
#include <stdlib.h>

void *bsearch(const void *key, const void *base, size_t nel,
    size_t width, int (*compar)(const void *, const void *));
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 , item XSH/TC1/D6/11 is applied, adding to the DESCRIPTION the last sentence of the first non-shaded paragraph, and the following three paragraphs. These changes are for alignment with the ISO C standard.

btowc

Single byte to wide character conversion

```
#include <stdio.h>
#include <wchar.h>

wint_t btowc(int c);
```

Derivation First released in Issue 5. Included for alignment with the ISO/IEC 9899: 1990 standard.

Issue 6 No functional changes are made in this issue.

bzero

Memory operations (**LEGACY**)

XSI
```
#include <strings.h>

void bzero(void *s, size_t n);
```

Derivation First released in Issue 4, Version 2.

Issue 6 This function is marked LEGACY and may not be available on all implementations.

The *memset*() function is preferred over this function.

For maximum portability, it is recommended to replace the function call to *bzero*() as follows:

```
#define bzero(b,len) (memset((b), '\0', (len)), (void) 0)
```

cabs, cabsf, cabsl

Return a complex absolute value

```
#include <complex.h>

double cabs(double complex z);
float cabsf(float complex z);
long double cabsl(long double complex z);
```

These functions compute the complex absolute value (also called norm, modulus, or magnitude) of *z*.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 6 These are new functions included for alignment with the ISO/IEC 9899: 1999 standard.

cacos, cacosf, cacosl

Complex arc cosine functions

```
#include <complex.h>

double complex cacos(double complex z);
float complex cacosf(float complex z);
long double complex cacosl(long double complex z);
```

These functions compute the complex arc cosine of *z*, with branch cuts outside the interval [–1, +1] along the real axis.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 6 These are new functions included for alignment with the ISO/IEC 9899: 1999 standard.

cacosh, cacoshf, cacoshl

Complex arc hyperbolic cosine functions

```
#include <complex.h>

double complex cacosh(double complex z);
float complex cacoshf(float complex z);
long double complex cacoshl(long double complex z);
```

These functions compute the complex arc hyperbolic cosine of *z*, with a branch cut at values less than 1 along the real axis.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 6 These are new functions included for alignment with the ISO/IEC 9899: 1999 standard.

calloc

A memory allocator

```
#include <stdlib.h>

void *calloc(size_t nelem, size_t elsize);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The setting of *errno* and the [ENOMEM] error condition are mandatory if an insufficient memory condition occurs.

carg, cargf, cargl

Complex argument functions

```
#include <complex.h>

double carg(double complex z);
float cargf(float complex z);
long double cargl(long double complex z);
```

These functions compute the argument (also called phase angle) of *z*, with a branch cut along the negative real axis.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 6 These are new functions included for alignment with the ISO/IEC 9899: 1999 standard.

casin, casinf, casinl

Complex arc sine functions

```
#include <complex.h>

double complex casin(double complex z);
float complex casinf(float complex z);
long double complex casinl(long double complex z);
```

These functions compute the complex arc sine of *z*, with branch cuts outside the interval [–1, +1] along the real axis.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 6 These are new functions included for alignment with the ISO/IEC 9899: 1999 standard.

casinh, casinhf, casinhl

Complex arc hyperbolic sine functions

```
#include <complex.h>

double complex casinh(double complex z);
float complex casinhf(float complex z);
long double complex casinhl(long double complex z);
```

These functions compute the complex arc hyperbolic sine of *z*, with branch cuts outside the interval [–*i*, +*i*] along the imaginary axis.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 6 These are new functions included for alignment with the ISO/IEC 9899: 1999 standard.

catan, catanf, catanl

Complex arc tangent functions

```
#include <complex.h>

double complex catan(double complex z);
float complex catanf(float complex z);
long double complex catanl(long double complex z);
```

These functions compute the complex arc tangent of *z*, with branch cuts outside the interval [–*i*, +*i*] along the imaginary axis.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 6 These are new functions included for alignment with the ISO/IEC 9899: 1999 standard.

catanh, catanhf, catanhl

Complex arc hyperbolic tangent functions

```
#include <complex.h>

double complex catanh(double complex z);
float complex catanhf(float complex z);
long double complex catanhl(long double complex z);
```

These functions compute the complex arc hyperbolic tangent of *z*, with branch cuts outside the interval [–1, +1] along the real axis.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 6 These are new functions included for alignment with the ISO/IEC 9899: 1999 standard.

catclose

Close a message catalog descriptor

XSI
```
#include <nl_types.h>
int catclose(nl_catd catd);
```

Derivation First released in Issue 2.

Issue 6 No functional changes are made in this issue.

catgets

Read a program message

XSI
```
#include <nl_types.h>
char *catgets(nl_catd catd, int set_id, int msg_id, const char *s);
```

Derivation First released in Issue 2.

Issue 6 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety. A portable multi-threaded application may only safely call the function when access to the function is serialized.

catopen

Open a message catalog

XSI
```
#include <nl_types.h>
nl_catd catopen(const char *name, int oflag);
```

Derivation First released in Issue 2.

Issue 6 No functional changes are made in this issue.

cbrt, cbrtf, cbrtl

Cube root functions

```
#include <math.h>
double cbrt(double x);
float cbrtf(float x);
long double cbrtl(long double x);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The *cbrt*() function is no longer marked as an extension.

The *cbrtf*() and *cbrtl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

ccos, ccosf, ccosl

Complex cosine functions

```
#include <complex.h>

double complex ccos(double complex z);
float complex ccosf(float complex z);
long double complex ccosl(long double complex z);
```

These functions compute the complex cosine of *z*.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

ccosh, ccoshf, ccoshl

Complex hyperbolic cosine functions

```
#include <complex.h>

double complex ccosh(double complex z);
float complex ccoshf(float complex z);
long double complex ccoshl(long double complex z);
```

These functions compute the complex hyperbolic cosine of *z*.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

ceil, ceilf, ceill

Ceiling value function

```
#include <math.h>

double ceil(double x);
float ceilf(float x);
long double ceill(long double x);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *ceilf*() and *ceill*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

cexp, cexpf, cexpl

Complex exponential functions

```
#include <complex.h>

double complex cexp(double complex z);
float complex cexpf(float complex z);
long double complex cexpl(long double complex z);
```

These functions compute the complex exponent of *z*, defined as e^z.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

cfgetispeed

Get input baud rate

```
#include <termios.h>

speed_t cfgetispeed(const struct termios *termios_p);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 No functional changes are made in this issue.

cfgetospeed

Get output baud rate

```
#include <termios.h>

speed_t cfgetospeed(const struct termios *termios_p);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 No functional changes are made in this issue.

cfsetispeed

Set input baud rate

```
#include <termios.h>

int cfsetispeed(struct termios *termios_p, speed_t speed);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The optional setting of *errno* and the [EINVAL] error conditions are added.

cfsetospeed

Set output baud rate

```
#include <termios.h>

int cfsetospeed(struct termios *termios_p, speed_t speed);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The optional setting of *errno* and the [EINVAL] error conditions are added.

chdir

Change working directory

```
#include <unistd.h>

int chdir(const char *path);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

chmod

Change mode of a file

```
#include <sys/stat.h>

int chmod(const char *path, mode_t mode);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EINVAL] and [EINTR] optional error conditions are added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

chown

Change owner and group of a file

```
#include <unistd.h>

int chown(const char *path, uid_t owner, gid_t group);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following changes are made for alignment with ISO/IEC 9945-1: 1996 (POSIX-1):

- The wording describing the optional dependency on _POSIX_CHOWN_RESTRICTED is restored.
- The [EPERM] error is restored as an error dependent on _POSIX_CHOWN_RESTRICTED. This is since its operand is a pathname and applications should be aware that the error may not occur for that pathname if the file system does not support _POSIX_CHOWN_RESTRICTED.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The value for *owner* of (**uid_t**)–1 allows the use of –1 by the owner of a file to change the group ID only. A corresponding change is made for group.
- The [ELOOP] mandatory error condition is added.
- The [EIO] and [EINTR] optional error conditions are added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- Clarification is added that the S_ISUID and S_ISGID bits do not need to be cleared when the process has appropriate privileges.
- The [ELOOP] optional error condition is added.

cimag, cimagf, cimagl

Complex imaginary functions

```
#include <complex.h>

double cimag(double complex z);
float cimagf(float complex z);
long double cimagl(long double complex z);
```

These functions compute the imaginary part of *z*.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 6 These are new functions included for alignment with the ISO/IEC 9899: 1999 standard.

clearerr

Clear indicators on a stream

```
#include <stdio.h>

void clearerr(FILE *stream);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

clock

Report CPU time used

```
#include <time.h>

clock_t clock(void);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

clock_getcpuclockid

Access a process CPU-time clock (**ADVANCED REALTIME**)

CPT
```
#include <time.h>

int clock_getcpuclockid(pid_t pid, clockid_t *clock_id);
```

The *clock_getcpuclockid*( ) function returns the clock ID of the CPU-time clock of the process specified by *pid*.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1d-1999. This function is part of the Advanced Realtime Option Group and may not be available on all implementations.

Issue 6 As a change over IEEE Std 1003.1d-1999, in the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

clock_getres, clock_gettime, clock_settime

Clock and timer functions (**REALTIME**)

TMR
```
#include <time.h>

int clock_getres(clockid_t clock_id, struct timespec *res);
int clock_gettime(clockid_t clock_id, struct timespec *tp);
int clock_settime(clockid_t clock_id, const struct timespec *tp);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. These functions are part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Timers option.

The following changes were made to align with the IEEE P1003.1a draft standard:

- Clarification is added of the effect of resetting the clock resolution.

CPU-time clocks and the *clock_getcpuclockid*( ) function are added for alignment with IEEE Std 1003.1d-1999.

The following changes are added for alignment with IEEE Std 1003.1j-2000:

- The DESCRIPTION is updated as follows:
  - The value returned by *clock_gettime*( ) for CLOCK_MONOTONIC is specified.
  - *clock_settime*( ) failing for CLOCK_MONOTONIC is specified.
  - The effects of *clock_settime*( ) on the *clock_nanosleep*( ) function with respect to CLOCK_REALTIME are specified.
- An [EINVAL] error is added to the ERRORS section, indicating that *clock_settime*( ) fails for CLOCK_MONOTONIC.
- The APPLICATION USAGE section notes that the CLOCK_MONOTONIC clock need not and shall not be set by *clock_settime*( ) since the absolute value of the CLOCK_MONOTONIC clock is meaningless.

clock_nanosleep

High resolution sleep with specifiable clock (**ADVANCED REALTIME**)

CS

```
#include <time.h>

int clock_nanosleep(clockid_t clock_id, int flags,
    const struct timespec *rqtp, struct timespec *rmtp);
```

The *clock_nanosleep*() function causes the current thread to be suspended from execution until either an interval timer or an absolute timer has elapsed.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1j-2000. This function is part of the Advanced Realtime Option Group and may not be available on all implementations.

Issue 6 This is a new function included for alignment with the IEEE Std 1003.1j-2000.

clog, clogf, clogl

Complex natural logarithm functions

```
#include <complex.h>

double complex clog(double complex z);
float complex clogf(float complex z);
long double complex clogl(long double complex z);
```

These functions compute the complex natural (base *e*) logarithm of *z*, with a branch cut along the negative real axis.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 6 These are new functions included for alignment with the ISO/IEC 9899: 1999 standard.

close

Close a file descriptor

```
#include <unistd.h>

int close(int fildes);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The DESCRIPTION related to a STREAMS-based file or pseudo-terminal is marked as part of the XSI STREAMS Option Group.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EIO] error condition is added as an optional error.
- The DESCRIPTION is updated to describe the state of the *fildes* file descriptor as unspecified if an I/O error occurs and an [EIO] error condition is returned.

Text referring to sockets is added to the DESCRIPTION.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that shared memory objects and memory mapped files (and not typed memory objects) are the types of memory objects to which the paragraph on last

closes applies.

, item XSH/TC1/D6/12 is applied, correcting the XSH shaded text relating to the master side of a pseudo-terminal. The reason for the change is that the behavior of pseudo-terminals and regular terminals should be as much alike as possible in this case; the change achieves that and matches historical behavior.

closedir

Close a directory stream

```
#include <dirent.h>

int closedir(DIR *dirp);
```

Derivation First released in Issue 2.

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EINTR] error condition is added as an optional error condition.

closelog, openlog, setlogmask, syslog

Control system log

XSI

```
#include <syslog.h>

void closelog(void);
void openlog(const char *ident, int logopt, int facility);
int setlogmask(int maskpri);
void syslog(int priority, const char *message, ... /* arguments */);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

confstr

Get configurable variables

```
#include <unistd.h>

size_t confstr(int name, char *buf, size_t len);
```

Derivation First released in Issue 4. Derived from ISO/IEC 9945-2: 1993 (POSIX-2).

Issue 6 The Open Group Corrigendum U033/7 is applied. The return value for the case returning the size of the buffer now explicitly states that this includes the terminating null.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated with new arguments which can be used to determine configuration strings for C compiler flags, linker/loader flags, and libraries for each different supported programming environment. This is a change to support data size neutrality.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The DESCRIPTION is updated to include text describing how _CS_PATH can be used to obtain a *PATH* to access the standard utilities.

The macros associated with the *c89* programming models are marked LEGACY and new equivalent macros associated with *c99* are introduced.

conj, conjf, conjl

Complex conjugate functions

```
#include <complex.h>

double complex conj(double complex z);
float complex conjf(float complex z);
long double complex conjl(long double complex z);
```

These functions compute the complex conjugate of *z*, by reversing the sign of its imaginary part.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 6 These are new functions included for alignment with the ISO/IEC 9899: 1999 standard.

connect

Connect a socket

```
#include <sys/socket.h>

int connect(int socket, const struct sockaddr *address,
    socklen_t address_len);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

copysign, copysignf, copysignl

Number manipulation function

```
#include <math.h>

double copysign(double x, double y);
float copysignf(float x, float y);
long double copysignl(long double x, long double y);
```

These functions produce a value with the magnitude of *x* and the sign of *y*.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

cos, cosf, cosl

Cosine function

```
#include <math.h>

double cos(double x);
float cosf(float x);
long double cosl(long double x);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *cosf*() and *cosl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

cosh, coshf, coshl

Hyperbolic cosine functions

```
#include <math.h>

double cosh(double x);
float coshf(float x);
long double coshl(long double x);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *coshf*() and *coshl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

cpow, cpowf, cpowl

Complex power functions

```
#include <complex.h>

double complex cpow(double complex x, double complex y);
float complex cpowf(float complex x, float complex y);
long double complex cpowl(long double complex x,
    long double complex y);
```

These functions compute the complex power function x^y, with a branch cut for the first parameter along the negative real axis.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 6 These are new functions included for alignment with the ISO/IEC 9899: 1999 standard.

cproj, cprojf, cprojl

Complex projection functions

```
#include <complex.h>

double complex cproj(double complex z);
float complex cprojf(float complex z);
long double complex cprojl(long double complex z);
```

These functions compute a projection of *z* onto the Riemann sphere.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 6 These are new functions included for alignment with the ISO/IEC 9899: 1999 standard.

creal, crealf, creall

Complex real functions

```
#include <complex.h>

double creal(double complex z);
float crealf(float complex z);
long double creall(long double complex z);
```

These functions compute the real part of *z*.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 6 These are new functions included for alignment with the ISO/IEC 9899: 1999 standard.

creat

Create a new file or rewrite an existing one

OH

```
#include <sys/stat.h>
#include <fcntl.h>

int creat(const char *path, mode_t mode);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

crypt

String encoding function (**CRYPT**)

XSI

```
#include <unistd.h>

char *crypt(const char *key, const char *salt);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

csin, csinf, csinl

Complex sine functions

```
#include <complex.h>

double complex csin(double complex z);
float complex csinf(float complex z);
long double complex csinl(long double complex z);
```

These functions compute the complex sine of *z*.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 6 These are new functions included for alignment with the ISO/IEC 9899: 1999 standard.

csinh, csinhf, csinhl

Complex hyperbolic sine functions

```
#include <complex.h>

double complex csinh(double complex z);
float complex csinhf(float complex z);
long double complex csinhl(long double complex z);
```

These functions compute the complex hyperbolic sine of *z*.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 6 These are new functions included for alignment with the ISO/IEC 9899: 1999 standard.

csqrt, csqrtf, csqrtl

Complex square root functions

```
#include <complex.h>

double complex csqrt(double complex z);
float complex csqrtf(float complex z);
long double complex csqrtl(long double complex z);
```

These functions compute the complex square root of *z*, with a branch cut along the negative real axis.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 6 These are new functions included for alignment with the ISO/IEC 9899: 1999 standard.

ctan, ctanf, ctanl

Complex tangent functions

```
#include <complex.h>

double complex ctan(double complex z);
float complex ctanf(float complex z);
long double complex ctanl(long double complex z);
```

These functions compute the complex tangent of *z*.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 6 These are new functions included for alignment with the ISO/IEC 9899: 1999 standard.

ctanh, ctanhf, ctanhl

Complex hyperbolic tangent functions

```
#include <complex.h>

double complex ctanh(double complex z);
float complex ctanhf(float complex z);
long double complex ctanhl(long double complex z);
```

These functions compute the complex hyperbolic tangent of *z*.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 6 These are new functions included for alignment with the ISO/IEC 9899: 1999 standard.

ctermid

Generate a pathname for controlling terminal

CX
```
#include <stdio.h>

char *ctermid(char *s);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

ctime, ctime_r

Convert a time value to date and time string

```
#include <time.h>

char *ctime(const time_t *clock);
```
TSF
```
char *ctime_r(const time_t *clock, char *buf);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety. A portable multi-threaded application may only safely call the function when access to the function is serialized.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

dbm_clearerr, dbm_close, dbm_delete, dbm_error, dbm_fetch, dbm_firstkey, dbm_nextkey, dbm_open, dbm_store

Database functions

XSI
```
#include <ndbm.h>

int dbm_clearerr(DBM *db);
void dbm_close(DBM *db);
int dbm_delete(DBM *db, datum key);
int dbm_error(DBM *db);
datum dbm_fetch(DBM *db, datum key);
datum dbm_firstkey(DBM *db);
datum dbm_nextkey(DBM *db);
DBM *dbm_open(const char *file, int open_flags, mode_t file_mode);
int dbm_store(DBM *db, datum key, datum content, int store_mode);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

difftime

Compute the difference between two calendar time values

```
#include <time.h>

double difftime(time_t time1, time_t time0);
```

Derivation First released in Issue 4. Derived from the ISO C standard.

Issue 6 No functional changes are made in this issue.

dirname

Report the parent directory name of a file pathname

XSI
```
#include <libgen.h>

char *dirname(char *path);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

div

Compute the quotient and remainder of an integer division

```
#include <stdlib.h>

div_t div(int numer, int denom);
```

Derivation First released in Issue 4. Derived from the ISO C standard.

Issue 6 No functional changes are made in this issue.

dlclose

Close a *dlopen*() object

XSI
```
#include <dlfcn.h>
int dlclose(void *handle);
```

Derivation First released in Issue 5.

Issue 6 The DESCRIPTION is updated to say that the referenced object is closed ‘‘if this is the last reference to it’’.

dlerror

Get diagnostic information

XSI
```
#include <dlfcn.h>
char *dlerror(void);
```

Derivation First released in Issue 5.

Issue 6 In the DESCRIPTION the note about reentrancy and thread-safety is added. A portable multi-threaded application may only safely call the function when access to the function is serialized.

dlopen

Gain access to an executable object file

XSI
```
#include <dlfcn.h>
void *dlopen(const char *file, int mode);
```

Derivation First released in Issue 5.

Issue 6 , item XSH/TC2/D6/21 is applied, changing the default behavior in the DESCRIPTION when neither RTLD_GLOBAL nor RTLD_LOCAL are specified from implementation-defined to unspecified.

dlsym

Obtain the address of a symbol from a *dlopen*() object

XSI
```
#include <dlfcn.h>
void *dlsym(void *restrict handle, const char *restrict name);
```

It should be noted that the ISO C standard does not require that pointers to functions can be cast back and forth to pointers to data. Indeed, the ISO C standard does not require that an object of type **void *** can hold a pointer to a function. UNIX system implementations, however, do require that an object of type **void *** can hold a pointer to a function. The result of converting a pointer to a function into a pointer to another data type (except **void ***) is still undefined, however. Note that compilers conforming to the ISO C standard are required to generate a warning if a conversion from a **void *** pointer to a function pointer is attempted as in:

```
fptr = (int (*)(int))dlsym(handle, "my_function");
```

Due to the problem noted here, a future version may either add a new function to return function pointers, or the current interface may be deprecated in favor of two new functions: one that returns data pointers and the other that returns function pointers.

Derivation First released in Issue 5.

Issue 6 The **restrict** keyword is added to the *dlsym*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

drand48, erand48, jrand48, lcong48, lrand48, mrand48, nrand48, seed48, srand48

Generate uniformly distributed pseudo-random numbers

XSI
```
#include <stdlib.h>

double drand48(void);
double erand48(unsigned short xsubi[3]);
long jrand48(unsigned short xsubi[3]);
void lcong48(unsigned short param[7]);
long lrand48(void);
long mrand48(void);
long nrand48(unsigned short xsubi[3]);
unsigned short *seed48(unsigned short seed16v[3]);
void srand48(long seedval);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

dup, dup2

Duplicate an open file descriptor

```
#include <unistd.h>

int dup(int fildes);
int dup2(int fildes, int fildes2);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

ecvt, fcvt, gcvt

Convert a floating-point number to a string (**LEGACY**)

XSI
```
#include <stdlib.h>

char *ecvt(double value, int ndigit, int *restrict decpt,
    int *restrict sign);
char *fcvt(double value, int ndigit, int *restrict decpt,
    int *restrict sign);
char *gcvt(double value, int ndigit, char *buf);
```

Derivation First released in Issue 4, Version 2.

Issue 6 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety. A portable multi-threaded application may only safely call the function when access to the function is serialized.

These functions are marked LEGACY and may not be available on all implementations.

The *sprintf*( ) function is preferred over these functions.

The **restrict** keyword is added to the *ecvt*( ) and *fcvt*( ) prototypes for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION is updated to explicitly use ‘‘conversion specification’’ to describe %g, %f, and %e.

encrypt

Encoding function (**CRYPT**)

XSI
```
#include <unistd.h>

void encrypt(char block[64], int edflag);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety. A portable multi-threaded application may only safely call the function when access to the function is serialized.

endgrent, getgrent, setgrent

Group database entry functions

XSI
```
#include <grp.h>

void endgrent(void);
struct group *getgrent(void);
void setgrent(void);
```

Derivation First released in Issue 4, Version 2.

Issue 6 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety. A portable multi-threaded application may only safely call the function when access to the function is serialized.

endhostent, gethostent, sethostent

Network host database functions

```
#include <netdb.h>

void endhostent(void);
struct hostent *gethostent(void);
void sethostent(int stayopen);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 No functional changes since the XNS, Issue 5.2 specification.

endnetent, getnetbyaddr, getnetbyname, getnetent, setnetent

Network database functions

```
#include <netdb.h>

void endnetent(void);
struct netent *getnetbyaddr(uint32_t net, int type);
struct netent *getnetbyname(const char *name);
struct netent *getnetent(void);
void setnetent(int stayopen);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 No functional changes since the XNS, Issue 5.2 specification.

endprotoent, getprotobyname, getprotobynumber, getprotoent, setprotoent

Network protocol database functions

```
#include <netdb.h>

void endprotoent(void);
struct protoent *getprotobyname(const char *name);
struct protoent *getprotobynumber(int proto);
struct protoent *getprotoent(void);
void setprotoent(int stayopen);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 No functional changes since the XNS, Issue 5.2 specification.

endpwent, getpwent, setpwent

User database functions

XSI

```
#include <pwd.h>

void endpwent(void);
struct passwd *getpwent(void);
void setpwent(void);
```

Derivation First released in Issue 4, Version 2.

Issue 6 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety. A portable multi-threaded application may only safely call the function when access to the function is serialized.

endservent, getservbyname, getservbyport, getservent, setservent

Network services database functions

```
#include <netdb.h>

void endservent(void);
struct servent *getservbyname(const char *name, const char *proto);
struct servent *getservbyport(int port, const char *proto);
struct servent *getservent(void);
void setservent(int stayopen);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

endutxent, getutxent, getutxid, getutxline, pututxline, setutxent

User accounting database functions

XSI
```
#include <utmpx.h>

void endutxent(void);
struct utmpx *getutxent(void);
struct utmpx *getutxid(const struct utmpx *id);
struct utmpx *getutxline(const struct utmpx *line);
struct utmpx *pututxline(const struct utmpx *utmpx);
void setutxent(void);
```

Derivation First released in Issue 4, Version 2.

Issue 6 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety. A portable multi-threaded application may only safely call the function when access to the function is serialized.

erf, erff, erfl

Error functions

```
#include <math.h>

double erf(double x);
float erff(float x);
long double erfl(long double x);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *erf*() function is no longer marked as an extension.

The *erfc*() function is split out onto its own reference page.

The *erff*() and *erfl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

erfc, erfcf, erfcl

Complementary error functions

```
#include <math.h>

double erfc(double x);
float erfcf(float x);
long double erfcl(long double x);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *erfc*() function is no longer marked as an extension.

These functions are split out from the *erf*( ) reference page.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

errno

Error return value

```
#include <errno.h>
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 Obsolescent text regarding defining *errno* as:

```
extern int errno
```

is removed.

Text regarding no function setting *errno* to zero to indicate an error is changed to no function shall set *errno* to zero. This is for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC2/D6/23 is applied, adding text to the DESCRIPTION stating that the setting of *errno* after a successful call to a function is unspecified unless the description of the function requires that it will not be modified.

environ, execl, execv, execle, execve, execlp, execvp

Execute a file

```
#include <unistd.h>

extern char **environ;
int execl(const char *path, const char *arg0, ... /*, (char *)0 */);
int execv(const char *path, char *const argv[]);
int execle(const char *path, const char *arg0, ... /*,
    (char *)0, char *const envp[]*/);
int execve(const char *path, char *const argv[], char *const envp[]);
int execlp(const char *file, const char *arg0, ... /*, (char *)0 */);
int execvp(const char *file, char *const argv[]);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, behavior is defined for when the process image file is not a valid executable.
- In this issue, _POSIX_SAVED_IDS is mandated, thus the effective user ID and effective group ID of the new process image are saved (as the saved set-user-ID and the saved set-group-ID) for use by the *setuid*( ) function.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.

- The [ETXTBSY] optional error condition is added.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [EINVAL] mandatory error condition is added.
- The [ELOOP] optional error condition is added.

The description of CPU-time clock semantics is added for alignment with IEEE Std 1003.1d-1999.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics for typed memory.

The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

IEEE PASC Interpretation 1003.1 #132 is applied. This addresses the issue of whether or not an implementation allows SIG_IGN as a SIGCHLD disposition to be inherited across a call to one of the *exec* family of functions or *posix_spawn*( ), leaving it explicitly unspecified.

The DESCRIPTION is updated to make it explicit that the floating-point environment in the new process image is set to the default.

, item XSH/TC1/D6/15 is applied, adding a new paragraph to the DESCRIPTION and text to the end of the APPLICATION USAGE section. This change addresses a security concern, where implementations may want to reopen file descriptors 0, 1, and 2 for programs with the set-user-id or set-group-id file mode bits calling the *exec* family of functions.

, item XSH/TC2/D6/24 is applied, applying changes to the DESCRIPTION, addressing which attributes are inherited by threads, and behavioral requirements for threads attributes.

, item XSH/TC2/D6/25 is applied, updating text in the RATIONALE from ‘‘the process signal mask be unchanged across an *exec*’’ to ‘‘the new process image inherits the signal mask of the thread that called *exec* in the old process image’’.

exit, _Exit, _exit

Terminate a process

```
#include <stdlib.h>

void exit(int status);
void _Exit(int status);

#include <unistd.h>

void _exit(int status);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics for typed memory.

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The *_Exit*( ) function is included.

- The DESCRIPTION is updated.

The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

References to the *wait3*() function are removed.

exp, expf, expl

Exponential function

```
#include <math.h>

double exp(double x);
float expf(float x);
long double expl(long double x);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *expf*() and *expl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

exp2, exp2f, exp2l

Exponential base 2 functions

```
#include <math.h>

double exp2(double x);
float exp2f(float x);
long double exp2l(long double x);
```

These functions compute the base-2 exponential of *x*.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

expm1, expm1f, expm1l

Compute exponential functions

```
#include <math.h>

double expm1(double x);
float expm1f(float x);
long double expm1l(long double x);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The *expm1f*() and *expm1l*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The *expm1*() function is no longer marked as an extension.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

fabs, fabsf, fabsl

Absolute value function

```
#include <math.h>

double fabs(double x);
float fabsf(float x);
long double fabsl(long double x);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *fabsf*() and *fabsl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

fattach

Attach a STREAMS-based file descriptor to a file in the file system name space (**STREAMS**)

XSR

```
#include <stropts.h>

int fattach(int fildes, const char *path);
```

Derivation First released in Issue 4, Version 2.

Issue 6 This function is marked as part of the XSI STREAMS Option Group and need not be supported on all implementations supporting the Single UNIX Specification.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

fchdir

Change working directory

XSI

```
#include <unistd.h>

int fchdir(int fildes);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

fchmod

Change mode of a file

```
#include <sys/stat.h>

int fchmod(int fildes, mode_t mode);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by stating that *fchmod*( ) behavior is unspecified for typed memory objects.

fchown

Change owner and group of a file

```
#include <unistd.h>

int fchown(int fildes, uid_t owner, gid_t group);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The following changes were made to align with the IEEE P1003.1a draft standard:

- Clarification is added that a call to *fchown*( ) may not be allowed on a pipe.

The *fchown*( ) function is defined as mandatory.

fclose

Close a stream

```
#include <stdio.h>

int fclose(FILE *stream);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EFBIG] error is added as part of the large file support extensions.
- The [ENXIO] optional error condition is added.

The DESCRIPTION is updated to note that the stream and any buffer are disassociated whether or not the call succeeds. This is for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC2/D6/28 is applied, updating the [EAGAIN] error in the ERRORS section from ‘‘the process would be delayed’’ to ‘‘the thread would be delayed’’.

fcntl

File control

OH

```
#include <unistd.h>
#include <fcntl.h>

int fcntl(int fildes, int cmd, ...);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, sentences describing behavior when *l_len* is negative are now mandated, and the description of unlock (F_UNLOCK) when *l_len* is non-negative is mandated.
- In the ERRORS section, the [EINVAL] error condition has the case mandated when the *cmd* is invalid, and two [EOVERFLOW] error conditions are added.

The F_GETOWN and F_SETOWN values are added for sockets.

The following changes were made to align with the IEEE P1003.1a draft standard:

- Clarification is added that the extent of the bytes locked is determined prior to the blocking action.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that *fcntl*( ) results are unspecified for typed memory objects.

fdatasync

Synchronize the data of a file (**REALTIME**)

SIO
```
#include <unistd.h>

int fdatasync(int fildes);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Synchronized Input and Output option.

The *fdatasync*( ) function is marked as part of the Synchronized Input and Output option.

fdetach

Detach a name from a STREAMS-based file descriptor (**STREAMS**)

XSR
```
#include <stropts.h>

int fdetach(const char *path);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

fdim, fdimf, fdiml

Compute positive difference between two floating-point numbers

```
#include <math.h>

double fdim(double x, double y);
float fdimf(float x, float y);
long double fdiml(long double x, long double y);
```

These functions determine the positive difference between their arguments.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

fdopen

Associate a stream with a file descriptor

CX
```
#include <stdio.h>

FILE *fdopen(int fildes, const char *mode);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, the use and setting of the *mode* argument are changed to include binary streams.
- In the DESCRIPTION, text is added for large file support to indicate setting of the offset maximum in the open file description.
- All errors identified in the ERRORS section are added.
- In the DESCRIPTION, text is added that the *fdopen*() function may cause *st_atime* to be updated.

The following changes were made to align with the IEEE P1003.1a draft standard:

- Clarification is added that it is the responsibility of the application to ensure that the mode is compatible with the open file descriptor.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that *fdopen*() results are unspecified for typed memory objects.

feclearexcept

Clear floating-point exception

```
#include <fenv.h>

int feclearexcept(int excepts);
```

The *feclearexcept*() function attempts to clear the supported floating-point exceptions represented by *excepts*.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

ISO/IEC 9899: 1999 standard, Technical Corrigendum No. 1 is incorporated.

fegetenv, fesetenv

Get and set current floating-point environment

```
#include <fenv.h>

int fegetenv(fenv_t *envp);
int fesetenv(const fenv_t *envp);
```

The *fegetenv*() function attempts to store the current floating-point environment in the object pointed to by *envp*.

The *fesetenv*() function attempts to establish the floating-point environment represented by the object pointed to by *envp*.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

ISO/IEC 9899: 1999 standard, Technical Corrigendum No. 1 is incorporated.

fegetexceptflag, fesetexceptflag

Get and set floating-point status flags

```
#include <fenv.h>

int fegetexceptflag(fexcept_t *flagp, int excepts);
int fesetexceptflag(const fexcept_t *flagp, int excepts);
```

The *fegetexceptflag*() function attempts to store an implementation-defined representation of the states of the floating-point status flags indicated by the argument *excepts* in the object pointed to by the argument *flagp*.

The *fesetexceptflag*() function attempts to set the floating-point status flags indicated by the argument *excepts* to the states stored in the object pointed to by *flagp*.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

ISO/IEC 9899: 1999 standard, Technical Corrigendum No. 1 is incorporated.

fegetround, fesetround

Get and set current rounding direction

```
#include <fenv.h>

int fegetround(void);
int fesetround(int round);
```

The *fegetround*() function gets the current rounding direction.

The *fesetround*() function establishes the rounding direction represented by its argument *round*.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

ISO/IEC 9899: 1999 standard, Technical Corrigendum No. 1 is incorporated.

feholdexcept

Save current floating-point environment

```
#include <fenv.h>

int feholdexcept(fenv_t *envp);
```

The *feholdexcept*() function saves the current floating-point environment in the object pointed to by *envp*, clears the floating-point status flags, and then installs a non-stop (continue on floating-point exceptions) mode, if available, for all floating-point exceptions.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

feof

Test end-of-file indicator on a stream

```
#include <stdio.h>

int feof(FILE *stream);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

feraiseexcept

Raise floating-point exception

```
#include <fenv.h>

int feraiseexcept(int excepts);
```

The *feraiseexcept*() function attempts to raise the supported floating-point exceptions represented by the argument *excepts*.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

ISO/IEC 9899: 1999 standard, Technical Corrigendum No. 1 is incorporated.

ferror

Test error indicator on a stream

```
#include <stdio.h>

int ferror(FILE *stream);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

fetestexcept

Test floating-point exception flags

```
#include <fenv.h>

int fetestexcept(int excepts);
```

The *fetestexcept*() function determines which of a specified subset of the floating-point exception flags are currently set. The *excepts* argument specifies the floating-point status flags to be queried.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

feupdateenv

Update floating-point environment

```
#include <fenv.h>

int feupdateenv(const fenv_t *envp);
```

The *feupdateenv*() function attempts to save the currently raised floating-point exceptions in its automatic storage, attempts to install the floating-point environment represented by the object pointed to by *envp*, and then attempts to raise the saved floating-point exceptions.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

ISO/IEC 9899: 1999 standard, Technical Corrigendum No. 1 is incorporated.

fflush

Flush a stream

```
#include <stdio.h>

int fflush(FILE *stream);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EFBIG] error is added as part of the large file support extensions.
- The [ENXIO] optional error condition is added.

The RETURN VALUE section is updated to note that the error indicator is set for the stream. This is for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC2/D6/31 is applied, updating the [EAGAIN] error in the ERRORS section from ‘‘the process would be delayed’’ to ‘‘the thread would be delayed’’.

ffs

Find first set bit

XSI

```
#include <strings.h>

int ffs(int i);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

fgetc

Get a byte from a stream

```
#include <stdio.h>

int fgetc(FILE *stream);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EIO] and [EOVERFLOW] mandatory error conditions are added.
- The [ENOMEM] and [ENXIO] optional error conditions are added.

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The DESCRIPTION is updated to clarify the behavior when the end-of-file indicator for the input stream is not set.
- The RETURN VALUE section is updated to note that the error indicator is set for the stream.

, item XSH/TC2/D6/32 is applied, updating the [EAGAIN] error in the ERRORS section from ‘‘the process would be delayed’’ to ‘‘the thread would be delayed’’.

fgetpos

Get current file position information

```
#include <stdio.h>

int fgetpos(FILE *restrict stream, fpos_t *restrict pos);
```

Derivation First released in Issue 4. Derived from the ISO C standard.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EIO] mandatory error condition is added.
- The [EBADF] and [ESPIPE] optional error conditions are added.

An additional [ESPIPE] error condition is added for sockets.

The prototype for *fgetpos*() is changed for alignment with the ISO/IEC 9899: 1999 standard.

fgets

Get a string from a stream

```
#include <stdio.h>

char *fgets(char *restrict s, int n, FILE *restrict stream);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The prototype for *fgets*() is changed for alignment with the ISO/IEC 9899: 1999 standard.

fgetwc

Get a wide-character code from a stream

```
#include <stdio.h>
#include <wchar.h>

wint_t fgetwc(FILE *stream);
```

Derivation First released in Issue 4. Derived from the MSE working draft.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EIO] and [EOVERFLOW] mandatory error conditions are added.
- The [ENOMEM] and [ENXIO] optional error conditions are added.

, item XSH/TC2/D6/33 is applied, updating the [EAGAIN] error in the ERRORS section from ‘‘the process would be delayed’’ to ‘‘the thread would be delayed’’.

fgetws

Get a wide-character string from a stream

```
#include <stdio.h>
#include <wchar.h>

wchar_t *fgetws(wchar_t *restrict ws, int n,
    FILE *restrict stream);
```

Derivation First released in Issue 4. Derived from the MSE working draft.

Issue 6 The prototype for *fgetws*() is changed for alignment with the ISO/IEC 9899: 1999 standard.

fileno

Map a stream pointer to a file descriptor

CX

```
#include <stdio.h>

int fileno(FILE *stream);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EBADF] optional error condition is added.

flockfile, ftrylockfile, funlockfile

Stdio locking functions

TSF

```
#include <stdio.h>

void flockfile(FILE *file);
int ftrylockfile(FILE *file);
void funlockfile(FILE *file);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 These functions are marked as part of the Thread-Safe Functions option. Support for this option is mandatory on all systems supporting the Single UNIX Specification.

floor, floorf, floorl

Floor function

```
#include <math.h>

double floor(double x);
float floorf(float x);
long double floorl(long double x);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *floorf*() and *floorl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

fma, fmaf, fmal

Floating-point multiply-add

```
#include <math.h>

double fma(double x, double y, double z);
float fmaf(float x, float y, float z);
long double fmal(long double x, long double y, long double z);
```

These functions compute (*x* * *y*) + *z*, rounded as one ternary operation: they compute the value (as if) to infinite precision and round once to the result format, according to the rounding mode characterized by the value of FLT_ROUNDS.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

fmax, fmaxf, fmaxl

Determine maximum numeric value of two floating-point numbers

```
#include <math.h>

double fmax(double x, double y);
float fmaxf(float x, float y);
long double fmaxl(long double x, long double y);
```

These functions determine the maximum numeric value of their arguments. NaN arguments are treated as missing data: if one argument is a NaN and the other numeric, then these functions choose the numeric value.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

fmin, fminf, fminl

Determine minimum numeric value of two floating-point numbers

```
#include <math.h>

double fmin(double x, double y);
float fminf(float x, float y);
long double fminl(long double x, long double y);
```

These functions determine the minimum numeric value of their arguments. NaN arguments are treated as missing data: if one argument is a NaN and the other numeric, then these functions choose the numeric value.

Derivation First released in Issue 6. Derived from ISO/IEC 9899: 1999 standard.

fmod, fmodf, fmodl

Floating-point remainder value function

```
#include <math.h>

double fmod(double x, double y);
float fmodf(float x, float y);
long double fmodl(long double x, long double y);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The behavior for when the *y* argument is zero is now defined.

The *fmodf*() and *fmodl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

fmtmsg

Display a message in the specified format on standard error and/or a system console

XSI

```
#include <fmtmsg.h>

int fmtmsg(long classification, const char *label, int severity,
    const char *text, const char *action, const char *tag);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

fnmatch

Match a filename or a pathname

```
#include <fnmatch.h>

int fnmatch(const char *pattern, const char *string, int flags);
```

Derivation First released in Issue 4. Derived from ISO/IEC 9945-2: 1993 (POSIX-2).

Issue 6 No functional changes are made in this issue.

fopen

Open a stream

```
#include <stdio.h>

FILE *fopen(const char *restrict filename, const char *restrict mode);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file description. This change is to support large files.
- In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support large files.
- The [ELOOP] mandatory error condition is added.
- The [EINVAL], [EMFILE], [ENAMETOOLONG], [ENOMEM], and [ETXTBSY] optional error conditions are added.

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The prototype for *fopen*() is updated.
- The DESCRIPTION is updated to note that if the argument *mode* points to a string other than those listed, then the behavior is undefined.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

fork

Create a new process

```
#include <unistd.h>

pid_t fork(void);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following changes were made to align with the IEEE P1003.1a draft standard:

- The effect of *fork*() on a pending alarm call in the child process is clarified.

The description of CPU-time clock semantics is added for alignment with IEEE Std 1003.1d-1999.

The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

, item XSH/TC1/D6/17 is applied, adding text to the DESCRIPTION relating to fork handlers registered by the *pthread_atfork*() function and async-signal safety. If any of the fork handlers calls a function that is not async-signal safe, the behavior is undefined.

fpathconf, pathconf

Get configurable pathname variables

```
#include <unistd.h>

long fpathconf(int fildes, int name);
long pathconf(const char *path, int name);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to include {FILESIZEBITS}.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The _PC_SYMLINK_MAX entry is added to the table in the DESCRIPTION.

The following *pathconf*() variables and their associated names are added for alignment with IEEE Std 1003.1d-1999:

{POSIX_ALLOC_SIZE_MIN}
{POSIX_REC_INCR_XFER_SIZE}
{POSIX_REC_MAX_XFER_SIZE}
{POSIX_REC_MIN_XFER_SIZE}
{POSIX_REC_XFER_ALIGN}

, item XSH/TC1/D6/18 is applied, changing the fourth paragraph of the DESCRIPTION and removing shading and margin markers from the table. This change is needed since implementations are required to support all of these symbols.

, item XSH/TC2/D6/34 is applied, adding the table entry for POSIX2_SYMLINKS in the DESCRIPTION.

, item XSH/TC2/D6/35 is applied, updating the DESCRIPTION and RATIONALE sections to clarify behavior for the {POSIX_ALLOC_SIZE_MIN}, {POSIX_REC_INCR_XFER_SIZE}, {POSIX_REC_MAX_XFER_SIZE}, {POSIX_REC_MIN_XFER_SIZE}, and {POSIX_REC_XFER_ALIGN} variables.

, item XSH/TC2/D6/36 is applied, updating the RETURN VALUE and APPLICATION USAGE sections to state that the results are unspecified if a variable is dependent on an unsupported option, and advising application writers to check for supported options prior to obtaining and using such values.

fpclassify

Classify real floating type

```
#include <math.h>

int fpclassify(real-floating x);
```

The *fpclassify*() macro classifies its argument value as NaN, infinite, normal, subnormal, zero, or into another implementation-defined category.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 6 This is a new function included for alignment with the ISO/IEC 9899: 1999 standard.

fprintf, printf, snprintf, sprintf

Print formatted output

```
#include <stdio.h>

int fprintf(FILE *restrict stream, const char *restrict format, ...);
int printf(const char *restrict format, ...);
int snprintf(char *restrict s, size_t n,
    const char *restrict format, ...);
int sprintf(char *restrict s, const char *restrict format, ...);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The prototypes for *fprintf*(), *printf*(), *snprintf*(), and *sprintf*() are updated, and the XSI shading is removed from *snprintf*().
- The description of *snprintf*() is aligned with the ISO C standard. Note that this supersedes the *snprintf*() description in The Open Group Base Resolution bwg98-006, which changed the behavior from Issue 5.
- The DESCRIPTION is updated.

The DESCRIPTION is updated to use the terms ‘‘conversion specifier’’ and ‘‘conversion specification’’ consistently.

ISO/IEC 9899: 1999 standard, Technical Corrigendum No. 1 is incorporated.

fputc

Put a byte on a stream

```
#include <stdio.h>

int fputc(int c, FILE *stream);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EIO] and [EFBIG] mandatory error conditions are added.
- The [ENOMEM] and [ENXIO] optional error conditions are added.

, item XSH/TC2/D6/37 is applied, updating the [EAGAIN] error in the ERRRORS section from ‘‘the process would be delayed’’ to ‘‘the thread would be delayed’’.

fputs

Put a string on a stream

```
#include <stdio.h>

int fputs(const char *restrict s, FILE *restrict stream);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *fputs*() prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

fputwc

Put a wide-character code on a stream

```
#include <stdio.h>
#include <wchar.h>

wint_t fputwc(wchar_t wc, FILE *stream);
```

Derivation First released in Issue 4. Derived from the MSE working draft.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EFBIG] and [EIO] mandatory error conditions are added.
- The [ENOMEM] and [ENXIO] optional error conditions are added.

, item XSH/TC2/D6/38 is applied, updating the [EAGAIN] error in the ERRORS section from ‘‘the process would be delayed’’ to ‘‘the thread would be delayed’’.

fputws

Put a wide-character string on a stream

```
#include <stdio.h>
#include <wchar.h>

int fputws(const wchar_t *restrict ws, FILE *restrict stream);
```

Derivation First released in Issue 4. Derived from the MSE working draft.

Issue 6 The *fputws*() prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

fread

Binary input

```
#include <stdio.h>

size_t fread(void *restrict ptr, size_t size, size_t nitems,
    FILE *restrict stream);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The *fread*( ) prototype is updated.
- The DESCRIPTION is updated to describe how the bytes from a call to *fgetc*( ) are stored.

free

Free allocated memory

```
#include <stdlib.h>

void free(void *ptr);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 Reference to the *valloc*( ) function is removed.

freeaddrinfo, getaddrinfo

Get address information

```
#include <sys/socket.h>
#include <netdb.h>

void freeaddrinfo(struct addrinfo *ai);
int getaddrinfo(const char *restrict nodename,
    const char *restrict servname,
    const struct addrinfo *restrict hints,
    struct addrinfo **restrict res);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 The **restrict** keyword is added to the *getaddrinfo*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC1/D6/20 is applied, making changes for alignment with IPv6. These include the following:

- Adding AI_V4MAPPED, AI_ALL, and AI_ADDRCONFIG to the allowed values for the *ai_flags* field
- Adding a description of AI_ADDRCONFIG
- Adding a description of the consequences of ignoring the AI_PASSIVE flag

, item XSH/TC2/D6/39 is applied, changing ‘‘corresponding value’’ to ‘‘corresponding error value’’ in the ERRORS section.

freopen

Open a stream

```
#include <stdio.h>

FILE *freopen(const char *restrict filename, const char *restrict mode,
    FILE *restrict stream);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file description. This change is to support large files.
- In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support large files.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.
- The [EINVAL], [ENOMEM], [ENXIO], and [ETXTBSY] optional error conditions are added.

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The *freopen*( ) prototype is updated.
- The DESCRIPTION is updated.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

The DESCRIPTION is updated regarding failure to close, changing the ‘‘file’’ to ‘‘file descriptor’’.

, item XSH/TC2/D6/40 is applied, adding the following sentence to the DESCRIPTION: ‘‘In this case, the file descriptor associated with the stream need not be closed if the call to *freopen*( ) succeeds.’’.

, item XSH/TC2/D6/41 is applied, adding an mandatory [EBADF] error, and an optional [EBADF] error to the ERRORS section.

frexp, frexpf, frexpl

Extract mantissa and exponent from a double precision number

```
#include <math.h>

double frexp(double num, int *exp);
float frexpf(float num, int *exp);
long double frexpl(long double num, int *exp);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *frexpf*( ) and *frexpl*( ) functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

fscanf, scanf, sscanf

Convert formatted input

```
#include <stdio.h>

int fscanf(FILE *restrict stream, const char *restrict format, ... );
int scanf(const char *restrict format, ... );
int sscanf(const char *restrict s, const char *restrict format, ... );
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The Open Group Corrigendum U021/7 and U028/10 are applied. These correct several occurrences of ‘‘characters’’ in the text which have been replaced with the term ‘‘bytes’’.

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The prototypes for *fscanf*( ), *scanf*( ), and *sscanf*( ) are updated.
- The DESCRIPTION is updated.
- The hh, ll, j, t, and z length modifiers are added.
- The a, A, and F conversion characters are added.

The DESCRIPTION is updated to use the terms ‘‘conversion specifier’’ and ‘‘conversion specification’’ consistently.

fseek, fseeko

Reposition a file-position indicator in a stream

```
#include <stdio.h>

int fseek(FILE *stream, long offset, int whence);
CX int fseeko(FILE *stream, off_t offset, int whence);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The *fseeko*( ) function is added.
- The [EFBIG], [EOVERFLOW], and [ENXIO] mandatory error conditions are added.

The following change is incorporated for alignment with the FIPS requirements:

- The [EINTR] error is no longer an indication that the implementation does not report partial transfers.

The DESCRIPTION is updated to explicitly state that *fseek*( ) sets the file-position indicator, and then on error the error indicator is set and *fseek*( ) fails. This is for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC2/D6/42 is applied, updating the [EAGAIN] error in the ERRORS section from ‘‘the process would be delayed’’ to ‘‘the thread would be delayed’’.

fsetpos

Set current file position

```
#include <stdio.h>

int fsetpos(FILE *stream, const fpos_t *pos);
```

Derivation First released in Issue 4. Derived from the ISO C standard.

Issue 6 An additional [ESPIPE] error condition is added for sockets.

The DESCRIPTION is updated to clarify that the error indicator is set for the stream on a read or write error. This is for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC1/D6/21 is applied, deleting an erroneous [EINVAL] error case from the ERRORS section.

, item XSH/TC2/D6/43 is applied, updating the [EAGAIN] error in the ERRORS section from ‘‘the process would be delayed’’ to ‘‘the thread would be delayed’’.

fstat

Get file status

```
#include <sys/stat.h>

int fstat(int fildes, struct stat *buf);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EIO] mandatory error condition is added.
- The [EOVERFLOW] mandatory error condition is added. This change is to support large files.
- The [EOVERFLOW] optional error condition is added.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that shared memory object semantics apply to typed memory objects.

fstatvfs, statvfs

Get file system information

XSI
```
#include <sys/statvfs.h>

int fstatvfs(int fildes, struct statvfs *buf);
int statvfs(const char *restrict path, struct statvfs *restrict buf);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The **restrict** keyword is added to the *statvfs*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

fsync

Synchronize changes to a file

FSC
```
#include <unistd.h>

int fsync(int fildes);
```

Derivation First released in Issue 3.

Issue 6 This function is marked as part of the File Synchronization option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EINVAL] and [EIO] mandatory error conditions are added.

, item XSH/TC2/D6/44 is applied, applying an editorial rewording of the DESCRIPTION. No change in meaning is intended.

ftell, ftello

Return a file offset in a stream

```
#include <stdio.h>

long ftell(FILE *stream);
```
CX
```
off_t ftello(FILE *stream);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The *ftello*( ) function is added.
- The [EOVERFLOW] error conditions are added.

An additional [ESPIPE] error condition is added for sockets.

ftime

Get date and time (**LEGACY**)

XSI
```
#include <sys/timeb.h>

int ftime(struct timeb *tp);
```

Derivation First released in Issue 4, Version 2.

Issue 6 This function is marked LEGACY and may not be available on all implementations.

For applications portability, the *time*( ) function should be used to determine the current time instead of *ftime*( ). Realtime applications should use *clock_gettime*( ) to determine the current time instead of *ftime*( ).

The DESCRIPTION is updated to refer to ‘‘seconds since the Epoch’’ rather than ‘‘seconds since 00:00:00 UTC (Coordinated Universal Time), January 1 1970’’ for consistency with other *time* functions.

ftok

Generate an IPC key

XSI
```
#include <sys/ipc.h>

key_t ftok(const char *path, int id);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

ftruncate

Truncate a file to a specified length

```
#include <unistd.h>

int ftruncate(int fildes, off_t length);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The *truncate*() function is split out into a separate reference page.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is changed to indicate that if the file size is changed, and if the file is a regular file, the S_ISUID and S_ISGID bits in the file mode may be cleared.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The DESCRIPTION text is updated.

XSI-conformant systems are required to increase the size of the file if the file was previously smaller than the size requested.

ftw

Traverse (walk) a file tree

XSI
```
#include <ftw.h>

int ftw(const char *path, int (*fn)(const char *,
    const struct stat *ptr, int flag), int ndirs);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The ERRORS section is updated as follows:

- The wording of the mandatory [ELOOP] error condition is updated.
- A second optional [ELOOP] error condition is added.
- The [EOVERFLOW] mandatory error condition is added.

Text is added to the DESCRIPTION to say that the *ftw*() function need not be reentrant and that the results are unspecified if the application-supplied *fn* function does not preserve the current working directory.

fwide

Set stream orientation

```
#include <stdio.h>
#include <wchar.h>

int fwide(FILE *stream, int mode);
```

Derivation First released in Issue 5. Included for alignment with the ISO/IEC 9899: 1990 standard.

Issue 6 No functional changes are made in this issue.

fwprintf, swprintf, wprintf

Print formatted wide-character output

```
#include <stdio.h>
#include <wchar.h>

int fwprintf(FILE *restrict stream, const wchar_t *restrict format, ...);
int swprintf(wchar_t *restrict ws, size_t n,
    const wchar_t *restrict format, ...);
int wprintf(const wchar_t *restrict format, ...);
```

Derivation First released in Issue 5. Included for alignment with the ISO/IEC 9899: 1990 standard.

Issue 6 The Open Group Corrigendum U040/1 is applied to the RETURN VALUE section, describing the case if *n* or more wide characters are requested to be written using *swprintf*( ).

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The prototypes for *fwprintf*( ), *swprintf*( ), and *wprintf*( ) are updated.
- The DESCRIPTION is updated.
- The `hh`, `ll`, `j`, `t`, and `z` length modifiers are added.
- The `a`, `A`, and `F` conversion characters are added.
- XSI shading is removed from the description of character string representations of infinity and NaN floating-point values.

The DESCRIPTION is updated to use the terms ‘‘conversion specifier’’ and ‘‘conversion specification’’ consistently.

ISO/IEC 9899: 1999 standard, Technical Corrigendum No. 1 is incorporated.

fwrite

Binary output

```
#include <stdio.h>

size_t fwrite(const void *restrict ptr, size_t size, size_t nitems,
    FILE *restrict stream);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The *fwrite*() prototype is updated.
- The DESCRIPTION is updated to clarify how the data is written out using *fputc*().

fwscanf, swscanf, wscanf

Convert formatted wide-character input

```
#include <stdio.h>
#include <wchar.h>

int fwscanf(FILE *restrict stream, const wchar_t *restrict format, ... );
int swscanf(const wchar_t *restrict ws,
    const wchar_t *restrict format, ... );
int wscanf(const wchar_t *restrict format, ... );
```

Derivation First released in Issue 5. Included for alignment with the ISO/IEC 9899: 1990 standard.

Issue 6 The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The prototypes for *fwscanf*() and *swscanf*() are updated.
- The DESCRIPTION is updated.
- The hh, ll, j, t, and z length modifiers are added.
- The a, A, and F conversion characters are added.

The DESCRIPTION is updated to use the terms ‘‘conversion specifier’’ and ‘‘conversion specification’’ consistently.

gai_strerror

Address and name information error description

```
#include <netdb.h>

const char *gai_strerror(int ecode);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 The Open Group Base Resolution bwg2001-009 is applied, which changes the return type from **char *** to **const char ***. This is for coordination with the IPnG Working Group.

getc

Get a byte from a stream

```
#include <stdio.h>

int getc(FILE *stream);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

getc_unlocked, getchar_unlocked, putc_unlocked, putchar_unlocked

Stdio with explicit client locking

TSF

```
#include <stdio.h>

int getc_unlocked(FILE *stream);
int getchar_unlocked(void);
int putc_unlocked(int c, FILE *stream);
int putchar_unlocked(int c);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 These functions are marked as part of the Thread-Safe Functions option. Support for this option is mandatory on all systems supporting the Single UNIX Specification.

The Open Group Corrigendum U030/2 is applied, adding APPLICATION USAGE describing how applications should be written to avoid the case when the functions are implemented as macros.

getchar

Get a byte from a stdin stream

```
#include <stdio.h>

int getchar(void);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

getcontext, setcontext

Get and set current user context

OB XSI

```
#include <ucontext.h>

int getcontext(ucontext_t *ucp);
int setcontext(const ucontext_t *ucp);
```

Derivation First released in Issue 4, Version 2.

Issue 6 , item XSH/TC2/D6/45 is applied, updating the SYNOPSIS and APPLICATION USAGE sections to note that the *getcontext*() and *setcontext*() functions are obsolescent.

getcwd

Get the pathname of the current working directory

```
#include <unistd.h>

char *getcwd(char *buf, size_t size);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [ENOMEM] optional error condition is added.

getdate

Convert user format date and time

XSI
```
#include <time.h>

struct tm *getdate(const char *string);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The DESCRIPTION is updated to refer to ‘‘seconds since the Epoch’’ rather than ‘‘seconds since 00:00:00 UTC (Coordinated Universal Time), January 1 1970’’ for consistency with other *time* functions.

The description of %S is updated so that the valid range is [00,60] rather than [00,61].

The DESCRIPTION is updated to refer to conversion specifications instead of field descriptors for consistency with other functions.

getegid

Get the effective group ID

```
#include <unistd.h>

gid_t getegid(void);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

getenv

Get value of an environment variable

```
#include <stdlib.h>

char *getenv(const char *name);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following changes were made to align with the IEEE P1003.1a draft standard:

- References added to the new *setenv*() and *unsetenv*() functions.

geteuid

Get the effective user ID

```
#include <unistd.h>

uid_t geteuid(void);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

getgid

Get the real group ID

```
#include <unistd.h>

gid_t getgid(void);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

getgrgid, getgrgid_r

Get group database entry for a group ID

```
#include <grp.h>

struct group *getgrgid(gid_t gid);
```

TSF
```
int getgrgid_r(gid_t gid, struct group *grp, char *buffer,
    size_t bufsize, struct group **result);
```

Derivation First released in Issue 1. Derived from System V Release 2.0.

Issue 6 The *getgrgid_r*() function is marked as part of the Thread-Safe Functions option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION describing matching the *gid*.

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety. A portable multi-threaded application may only safely call the function when access to the function is serialized.

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE section, the requirement to set *errno* on error is added.
- The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the buffer from *bufsize* characters to bytes.

getgrnam, getgrnam_r

```
Search group database for a name
#include <grp.h>

struct group *getgrnam(const char *name);
```

TSF
```
int getgrnam_r(const char *name, struct group *grp, char *buffer,
    size_t bufsize, struct group **result);
```

Derivation First released in Issue 1. Derived from System V Release 2.0.

Issue 6 The *getgrnam_r*() function is marked as part of the Thread-Safe Functions option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety. A portable multi-threaded application may only safely call the function when access to the function is serialized.

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE section, the requirement to set *errno* on error is added.
- The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the buffer from *bufsize* characters to bytes.

getgroups

Get supplementary group IDs

```
#include <unistd.h>

int getgroups(int gidsetsize, gid_t grouplist[]);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- A return value of 0 is not permitted, because {NGROUPS_MAX} cannot be 0. This is a FIPS requirement.

The following changes were made to align with the IEEE P1003.1a draft standard:

- An explanation is added that the effective group ID may be included in the supplementary group list.

gethostbyaddr, gethostbyname

Network host database functions

```
#include <netdb.h>

OB  struct hostent *gethostbyaddr(const void *addr, socklen_t len,
        int type);
    struct hostent *gethostbyname(const char *name);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 These functions are marked obsolescent. The *getaddrinfo*( ) and *getnameinfo*( ) functions are preferred over the *gethostbyaddr*( ) and *gethostbyname*( ) functions.

gethostid

Get an identifier for the current host

XSI
```
#include <unistd.h>

long gethostid(void);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

gethostname

Get name of current host

```
#include <unistd.h>

int gethostname(char *name, size_t namelen);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 The Open Group Base Resolution bwg2001-008 is applied, changing the *namelen* parameter from **socklen_t** to **size_t**.

getitimer, setitimer

Get and set value of interval timer

XSI
```
#include <sys/time.h>

int getitimer(int which, struct itimerval *value);
int setitimer(int which, const struct itimerval *restrict value,
    struct itimerval *restrict ovalue);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The **restrict** keyword is added to the *setitimer*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

getlogin, getlogin_r

Get login name

```
#include <unistd.h>

char *getlogin(void);
```
TSF
```
int getlogin_r(char *name, size_t namesize);
```

Derivation First released in Issue 1. Derived from System V Release 2.0.

Issue 6 The *getlogin_r*( ) function is marked as part of the Thread-Safe Functions option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety. A portable multi-threaded application may only safely call the function when access to the function is serialized.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE section, the requirement to set *errno* on error is added.
- The [EMFILE], [ENFILE], and [ENXIO] optional error conditions are added.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

getmsg, getpmsg

Receive next message from a STREAMS file (**STREAMS**)

XSR

```
#include <stropts.h>

int getmsg(int fildes, struct strbuf *restrict ctlptr,
    struct strbuf *restrict dataptr, int *restrict flagsp);
int getpmsg(int fildes, struct strbuf *restrict ctlptr,
    struct strbuf *restrict dataptr, int *restrict bandp,
    int *restrict flagsp);
```

Derivation First released in Issue 4, Version 2.

Issue 6 This function is marked as part of the XSI STREAMS Option Group and need not be supported on all implementations supporting the Single UNIX Specification.

The **restrict** keyword is added to the *getmsg*() and *getpmsg*() prototypes for alignment with the ISO/IEC 9899: 1999 standard.

getnameinfo

Get name information

```
#include <sys/socket.h>
#include <netdb.h>

int getnameinfo(const struct sockaddr *restrict sa, socklen_t salen,
    char *restrict node, socklen_t nodelen, char *restrict service,
    socklen_t servicelen, unsigned flags);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 The **restrict** keyword is added to the *getnameinfo*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC1/D6/23 is applied, making various changes in the SYNOPSIS and DESCRIPTION for alignment with IPv6.

getopt, optarg, opterr, optind, optopt

Command option parsing

```
#include <unistd.h>

int getopt(int argc, char *const argv[], const char *optstring);
extern char *optarg;
extern int optind, opterr, optopt;
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 IEEE PASC Interpretation 1003.2 #150 is applied, changing the words ‘‘not less than *argc*’’ to ‘‘not greater than *argc*’’.

getpeername

Get the name of the peer socket

```
#include <sys/socket.h>

int getpeername(int socket, struct sockaddr *restrict address,
    socklen_t *restrict address_len);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 The **restrict** keyword is added to the *getpeername*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

getpgid

Get the process group ID for a process

XSI

```
#include <unistd.h>

pid_t getpgid(pid_t pid);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

getpgrp

Get the process group ID of the calling process

```
#include <unistd.h>

pid_t getpgrp(void);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

getpid

Get the process ID

```
#include <unistd.h>

pid_t getpid(void);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

getppid

Get the parent process ID

```
#include <unistd.h>

pid_t getppid(void);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

getpriority, setpriority

Get and set the nice value

XSI
```
#include <sys/resource.h>

int getpriority(int which, id_t who);
int setpriority(int which, id_t who, int value);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

getpwnam, getpwnam_r

Search user database for a name

```
#include <pwd.h>

struct passwd *getpwnam(const char *name);
```

TSF
```
int getpwnam_r(const char *name, struct passwd *pwd, char *buffer,
    size_t bufsize, struct passwd **result);
```

Derivation First released in Issue 1. Derived from System V Release 2.0.

Issue 6 The *getpwnam_r*() function is marked as part of the Thread-Safe Functions option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION describing matching the *name*.

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety. A portable multi-threaded application may only safely call the function when access to the function is serialized.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE section, the requirement to set *errno* on error is added.
- The [EMFILE], [ENFILE], and [ENXIO] optional error conditions are added.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the buffer from *bufsize* characters to bytes.

getpwuid, getpwuid_r

Search user database for a user ID

```
#include <pwd.h>

struct passwd *getpwuid(uid_t uid);
TSF int getpwuid_r(uid_t uid, struct passwd *pwd, char *buffer,
    size_t bufsize, struct passwd **result);
```

Derivation First released in Issue 1. Derived from System V Release 2.0.

Issue 6 The *getpwuid_r*( ) function is marked as part of the Thread-Safe Functions option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION describing matching the *uid*.

In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety. A portable multi-threaded application may only safely call the function when access to the function is serialized.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE section, the requirement to set *errno* on error is added.
- The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the buffer from *bufsize* characters to bytes.

getrlimit, setrlimit

Control maximum resource consumption

XSI
```
#include <sys/resource.h>

int getrlimit(int resource, struct rlimit *rlp);
int setrlimit(int resource, const struct rlimit *rlp);
```

Derivation First released in Issue 4, Version 2.

Issue 6 , item XSH/TC1/D6/25 is applied, changing wording for RLIMIT_NOFILE in the DESCRIPTION related to functions that allocate a file descriptor failing with [EMFILE]. Text is added to the APPLICATION USAGE section noting the consequences of a process attempting to set the hard or soft limit for RLIMIT_NOFILE less than the highest currently open file descriptor +1.

, item XSH/TC2/D6/46 is applied, updating the definition of RLIMIT_STACK in the DESCRIPTION section from ‘‘the maximum size of a process stack’’ to ‘‘the maximum size of the initial thread’s stack’’. Text is added to the RATIONALE section.

getrusage

Get information about resource utilization

XSI
```
#include <sys/resource.h>

int getrusage(int who, struct rusage *r_usage);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

gets

Get a string from a stdin stream

```
#include <stdio.h>

char *gets(char *s);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

getsid

Get the process group ID of a session leader

XSI
```
#include <unistd.h>

pid_t getsid(pid_t pid);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

getsockname

Get the socket name

```
#include <sys/socket.h>

int getsockname(int socket, struct sockaddr *restrict address,
    socklen_t *restrict address_len);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 The **restrict** keyword is added to the *getsockname*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

getsockopt

Get the socket options

```
#include <sys/socket.h>

int getsockopt(int socket, int level, int option_name,
    void *restrict option_value, socklen_t *restrict option_len);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 The **restrict** keyword is added to the *getsockopt*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

getsubopt

Parse suboption arguments from a string

XSI

```
#include <stdlib.h>

int getsubopt(char **optionp, char *const *keylistp, char **valuep);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

gettimeofday

Get the date and time

XSI

```
#include <sys/time.h>

int gettimeofday(struct timeval *restrict tp, void *restrict tzp);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The DESCRIPTION is updated to refer to ‘‘seconds since the Epoch’’ rather than ‘‘seconds since 00:00:00 UTC (Coordinated Universal Time), January 1 1970’’ for consistency with other *time* functions.

The **restrict** keyword is added to the *gettimeofday*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

getuid

Get a real user ID

```
#include <unistd.h>

uid_t getuid(void);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

getwc

Get a wide character from a stream

```
#include <stdio.h>
#include <wchar.h>

wint_t getwc(FILE *stream);
```

Derivation First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working draft.

Issue 6 No functional changes are made in this issue.

getwchar

Get a wide character from a stdin stream

```
#include <wchar.h>

wint_t getwchar(void);
```

Derivation First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working draft.

Issue 6 No functional changes are made in this issue.

getwd

Get the current working directory pathname (**LEGACY**)

XSI

```
#include <unistd.h>

char *getwd(char *path_name);
```

Derivation First released in Issue 4, Version 2.

Issue 6 This function is marked LEGACY and may not be available on all implementations. For applications portability, the *getcwd*() function should be used to determine the current working directory instead of *getwd*().

glob, globfree

Generate pathnames matching a pattern

```
#include <glob.h>

int glob(const char *restrict pattern, int flags,
    int(*errfunc)(const char *epath, int eerrno),
    glob_t *restrict pglob);
void globfree(glob_t *pglob);
```

Derivation First released in Issue 4. Derived from ISO/IEC 9945-2: 1993 (POSIX-2).

Issue 6 The **restrict** keyword is added to the *glob*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

gmtime, gmtime_r

Convert a time value to a broken-down UTC time

```
#include <time.h>

struct tm *gmtime(const time_t *timer);
```
TSF
```
struct tm *gmtime_r(const time_t *restrict timer,
    struct tm *restrict result);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *gmtime_r*() function is marked as part of the Thread-Safe Functions option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

The **restrict** keyword is added to the *gmtime_r*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC1/D6/27 is applied, adding the [EOVERFLOW] error.

, item XSH/TC2/D6/48 is applied, updating the error handling for *gmtime_r*().

grantpt

Grant access to the slave pseudo-terminal device

XSI
```
#include <stdlib.h>

int grantpt(int fildes);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

h_errno

Error return value for network database operations

OB
```
#include <netdb.h>
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 Applications should obtain the definition of *h_errno* by the inclusion of the **<netdb.h>** header.

Note that this method of returning errors is used only in connection with obsolescent functions and may be withdrawn in the future.

hcreate, hdestroy, hsearch

Manage hash search table

XSI
```
#include <search.h>

int hcreate(size_t nel);
void hdestroy(void);
ENTRY *hsearch(ENTRY item, ACTION action);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

htonl, htons, ntohl, ntohs

Convert values between host and network byte order

```
#include <arpa/inet.h>

uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 No functional changes since the XNS, Issue 5.2 specification.

hypot, hypotf, hypotl

Euclidean distance function

```
#include <math.h>

double hypot(double x, double y);
float hypotf(float x, float y);
long double hypotl(long double x, long double y);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *hypot*() function is no longer marked as an extension.

The *hypotf*() and *hypotl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

iconv

Codeset conversion function

XSI
```
#include <iconv.h>

size_t iconv(iconv_t cd, char **restrict inbuf,
    size_t *restrict inbytesleft, char **restrict outbuf,
    size_t *restrict outbytesleft);
```

Derivation First released in Issue 4. Derived from the HP-UX Manual.

Issue 6 The SYNOPSIS has been corrected to align with the **<iconv.h>** reference page.

The **restrict** keyword is added to the *iconv*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

iconv_close

Codeset conversion deallocation function

XSI
```
#include <iconv.h>

int iconv_close(iconv_t cd);
```

Derivation First released in Issue 4. Derived from the HP-UX Manual.

Issue 6 No functional changes are made in this issue.

iconv_open

Codeset conversion allocation function

XSI
```
#include <iconv.h>

iconv_t iconv_open(const char *tocode, const char *fromcode);
```

Derivation First released in Issue 4. Derived from the HP-UX Manual.

Issue 6 No functional changes are made in this issue.

if_freenameindex

Free memory allocated by *if_nameindex*( )

```
#include <net/if.h>

void if_freenameindex(struct if_nameindex *ptr);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 No functional changes since the XNS, Issue 5.2 specification.

if_indextoname

Map a network interface index to its corresponding name

```
#include <net/if.h>

char *if_indextoname(unsigned ifindex, char *ifname);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 , item XSH/TC1/D6/28 is applied, changing {IFNAMSIZ} to {IF_NAMESIZ} in the DESCRIPTION.

if_nameindex

Return all network interface names and indexes

```
#include <net/if.h>

struct if_nameindex *if_nameindex(void);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 No functional changes since the XNS, Issue 5.2 specification.

if_nametoindex

Map a network interface name to its corresponding index

```
#include <net/if.h>

unsigned if_nametoindex(const char *ifname);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 No functional changes since the XNS, Issue 5.2 specification.

ilogb, ilogbf, ilogbl

Return an unbiased exponent

```
#include <math.h>

int ilogb(double x);
int ilogbf(float x);
int ilogbl(long double x);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The *ilogb*() function is no longer marked as an extension.

The *ilogbf*() and *ilogbl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The RETURN VALUE section is revised for alignment with the ISO/IEC 9899: 1999 standard.

imaxabs

Return absolute value

```
#include <inttypes.h>

intmax_t imaxabs(intmax_t j);
```

The *imaxabs*() function computes the absolute value of an integer *j*.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

imaxdiv

Return quotient and remainder

```
#include <inttypes.h>

imaxdiv_t imaxdiv(intmax_t numer, intmax_t denom);
```

The *imaxdiv*() function computes *numer* / *denom* and *numer* % *denom* in a single operation.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

index

Character string operations (**LEGACY**)

XSI

```
#include <strings.h>

char *index(const char *s, int c);
```

Derivation First released in Issue 4, Version 2.

Issue 6 This function is marked LEGACY and may not be available on all implementations.

The *strchr*() function is preferred over this function.

For maximum portability, it is recommended to replace the function call to *index*() as follows:

```
#define index(a,b) strchr((a),(b))
```

inet_addr, inet_ntoa

IPv4 address manipulation

```
#include <arpa/inet.h>

in_addr_t inet_addr(const char *cp);
char *inet_ntoa(struct in_addr in);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 No functional changes since the XNS, Issue 5.2 specification.

inet_ntop, inet_pton

Convert IPv4 and IPv6 addresses between binary and text form

```
#include <arpa/inet.h>

const char *inet_ntop(int af, const void *restrict src,
    char *restrict dst, socklen_t size);
int inet_pton(int af, const char *restrict src, void *restrict dst);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 IPv6 extensions are marked.

The **restrict** keyword is added to the *inet_ntop*() and *inet_pton*() prototypes for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC1/D6/29 is applied, adding ‘‘the address must be in network byte order’’ to the end of the fourth sentence of the first paragraph in the DESCRIPTION.

initstate, random, setstate, srandom

Pseudo-random number functions

XSI

```
#include <stdlib.h>

char *initstate(unsigned seed, char *state, size_t size);
long random(void);
char *setstate(const char *state);
void srandom(unsigned seed);
```

Derivation First released in Issue 4, Version 2.

Issue 6 In the DESCRIPTION, duplicate text ‘‘For values greater than or equal to 8 . . .’’ is removed.

insque, remque

Insert or remove an element in a queue

XSI

```
#include <search.h>

void insque(void *element, void *pred);
void remque(void *element);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

ioctl

Control a STREAMS device (**STREAMS**)

XSR

```
#include <stropts.h>

int ioctl(int fildes, int request, ... /* arg */);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The Open Group Corrigendum U028/4 is applied, correcting text in the I_FDINSERT, [EINVAL] case to refer to *ctlbuf*.

This function is marked as part of the XSI STREAMS Option Group and need not be supported on all implementations supporting the Single UNIX Specification.

isalnum

Test for an alphanumeric character

```
#include <ctype.h>

int isalnum(int c);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

isalpha

Test for an alphabetic character

```
#include <ctype.h>

int isalpha(int c);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

isascii

Test for a 7-bit US-ASCII character

XSI

```
#include <ctype.h>

int isascii(int c);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

isastream

Test a file descriptor (**STREAMS**)

XSR

```
#include <stropts.h>

int isastream(int fildes);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

isatty

Test for a terminal device

```
#include <unistd.h>

int isatty(int fildes);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The optional setting of *errno* to indicate an error is added.
- The [EBADF] and [ENOTTY] optional error conditions are added.

isblank

Test for a blank character

```
#include <ctype.h>

int isblank(int c);
```

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

iscntrl

Test for a control character

```
#include <ctype.h>

int iscntrl(int c);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

isdigit

Test for a decimal digit

```
#include <ctype.h>

int isdigit(int c);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

isfinite

Test for finite value

```
#include <math.h>

int isfinite(real-floating x);
```

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

isgraph

Test for a visible character

```
#include <ctype.h>
int isgraph(int c);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

isgreater

Test if x greater than y

```
#include <math.h>
int isgreater(real-floating x, real-floating y);
```

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

isgreaterequal

Test if x greater than or equal to y

```
#include <math.h>
int isgreaterequal(real-floating x, real-floating y);
```

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

isinf

Test for infinity

```
#include <math.h>
int isinf(real-floating x);
```

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

isless

Test if x is less than y

```
#include <math.h>
int isless(real-floating x, real-floating y);
```

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

islessequal

Test if x is less than or equal to y

```
#include <math.h>
int islessequal(real-floating x, real-floating y);
```

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

islessgreater

Test if x is less than or greater than y

```
#include <math.h>
int islessgreater(real-floating x, real-floating y);
```

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

islower

Test for a lowercase letter

```
#include <ctype.h>
int islower(int c);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 An example is added.

isnan

Test for a NaN

```
#include <math.h>
int isnan(real-floating x);
```

Derivation First released in Issue 3.

Issue 6 Re-written for alignment with the ISO/IEC 9899: 1999 standard.

isnormal

Test for a normal value

```
#include <math.h>
int isnormal(real-floating x);
```

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

isprint

Test for a printable character

```
#include <ctype.h>
int isprint(int c);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

ispunct

Test for a punctuation character

```
#include <ctype.h>
int ispunct(int c);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

isspace

Test for a white-space character

```
#include <ctype.h>
int isspace(int c);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

isunordered

Test if arguments are unordered

```
#include <math.h>
int isunordered(real-floating x, real-floating y);
```

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

, item XSH/TC2/D6/50 is applied, correcting the RETURN VALUE to be 1 when *x* or *y* is NaN.

isupper

Test for an uppercase letter

```
#include <ctype.h>
int isupper(int c);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

iswalnum

Test for an alphanumeric wide-character code

```
#include <wctype.h>
int iswalnum(wint_t wc);
```

Derivation First released as a World-wide Portability Interface in Issue 4.

Issue 6 No functional changes are made in this issue.

iswalpha

Test for an alphabetic wide-character code

```
#include <wctype.h>
int iswalpha(wint_t wc);
```

Derivation First released in Issue 4.

Issue 6 No functional changes are made in this issue.

iswblank

Test for a blank wide-character code

```
#include <wctype.h>
int iswblank(wint_t wc);
```

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

iswcntrl

Test for a control wide-character code

```
#include <wctype.h>
int iswcntrl(wint_t wc);
```

Derivation First released in Issue 4.

Issue 6 No functional changes are made in this issue.

iswctype

Test character for a specified class

```
#include <wctype.h>
int iswctype(wint_t wc, wctype_t charclass);
```

Derivation First released as World-wide Portability Interfaces in Issue 4.

Issue 6 The behavior of *n*=0 is now described.

An example is added.

A new function, *iswblank*( ), is added to the list in the APPLICATION USAGE.

iswdigit

Test for a decimal digit wide-character code

```
#include <wctype.h>
int iswdigit(wint_t wc);
```

Derivation First released in Issue 4.

Issue 6 No functional changes are made in this issue.

iswgraph

Test for a visible wide-character code

```
#include <wctype.h>
int iswgraph(wint_t wc);
```

Derivation First released in Issue 4.

Issue 6 No functional changes are made in this issue.

iswlower

Test for a lowercase letter wide-character code

```
#include <wctype.h>
int iswlower(wint_t wc);
```

Derivation First released in Issue 4.

Issue 6 No functional changes are made in this issue.

iswprint

Test for a printable wide-character code

```
#include <wctype.h>
int iswprint(wint_t wc);
```

Derivation First released in Issue 4.

Issue 6 No functional changes are made in this issue.

iswpunct

Test for a punctuation wide-character code

```
#include <wctype.h>
int iswpunct(wint_t wc);
```

Derivation First released in Issue 4.

Issue 6 No functional changes are made in this issue.

iswspace

Test for a white-space wide-character code

```
#include <wctype.h>
int iswspace(wint_t wc);
```

Derivation First released in Issue 4.

Issue 6 No functional changes are made in this issue.

iswupper

Test for an uppercase letter wide-character code

```
#include <wctype.h>

int iswupper(wint_t wc);
```

Derivation First released in Issue 4.

Issue 6 No functional changes are made in this issue.

iswxdigit

Test for a hexadecimal digit wide-character code

```
#include <wctype.h>

int iswxdigit(wint_t wc);
```

Derivation First released in Issue 4.

Issue 6 No functional changes are made in this issue.

isxdigit

Test for a hexadecimal digit

```
#include <ctype.h>

int isxdigit(int c);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

j0, j1, jn

Bessel functions of the first kind

XSI

```
#include <math.h>

double j0(double x);
double j1(double x);
double jn(int n, double x);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The may fail [EDOM] error is removed for the case for NaN.

The RETURN VALUE and ERRORS sections are reworked for alignment of the error handling with the ISO/IEC 9899: 1999 standard.

kill

Send a signal to a process or a group of processes

CX

```
#include <signal.h>

int kill(pid_t pid, int sig);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, the second paragraph is reworded to indicate that the saved set-user-ID of the calling process is checked in place of its effective user ID. This is a FIPS requirement.
- The behavior when *pid* is –1 is now specified. It was previously explicitly unspecified in IEEE Std 1003.1-1988 (POSIX.1). This may require a change for systems that have previously implemented the BSD semantics.

killpg

Send a signal to a process group

XSI
```
#include <signal.h>

int killpg(pid_t pgrp, int sig);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

labs, llabs

Return a long integer absolute value

```
#include <stdlib.h>

long labs(long i);
long long llabs(long long i);
```

Derivation First released in Issue 4. Derived from the ISO C standard.

Issue 6 The *llabs*() function is added for alignment with the ISO/IEC 9899: 1999 standard.

lchown

Change the owner and group of a symbolic link

XSI
```
#include <unistd.h>

int lchown(const char *path, uid_t owner, gid_t group);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

ldexp, ldexpf, ldexpl

Load exponent of a floating-point number

```
#include <math.h>

double ldexp(double x, int exp);
float ldexpf(float x, int exp);
long double ldexpl(long double x, int exp);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *ldexpf*() and *ldexpl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

ldiv, lldiv

Compute quotient and remainder of a long division

```
#include <stdlib.h>

ldiv_t ldiv(long numer, long denom);
lldiv_t lldiv(long long numer, long long denom);
```

Derivation First released in Issue 4. Derived from the ISO C standard.

Issue 6 The *lldiv*() function is added for alignment with the ISO/IEC 9899: 1999 standard.

lgamma, lgammaf, lgammal

Log gamma function

```
#include <math.h>

double lgamma(double x);
float lgammaf(float x);
long double lgammal(long double x);
```

XSI
```
extern int signgam;
```

Derivation First released in Issue 3.

Issue 6 The *lgamma*() function is no longer marked as an extension.

The *lgammaf*() and *lgammal*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

link

Link to a file

```
#include <unistd.h>

int link(const char *path1, const char *path2);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- An explanation is added of action when *path2* refers to a symbolic link.
- The [ELOOP] optional error condition is added.

lio_listio

List directed I/O (**REALTIME**)

AIO

```
#include <aio.h>

int lio_listio(int mode, struct aiocb *restrict const list[restrict],
    int nent, struct sigevent *restrict sig);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

The *lio_listio*( ) function is marked as part of the Asynchronous Input and Output option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs past the offset maximum established in the open file description associated with *aiocbp->aio_fildes*. This change is to support large files.
- The [EBIG] and [EOVERFLOW] error conditions are defined. This change is to support large files.

The **restrict** keyword is added to the *lio_listio*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC2/D6/54 is applied, adding text to the DESCRIPTION making it explicit that the user is required to keep the structure pointed to by *sig->sigev_notify_attributes* valid until the last asynchronous operation finished and the notification has been sent.

listen

Listen for socket connections and limit the queue of incoming connections

```
#include <sys/socket.h>

int listen(int socket, int backlog);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 The DESCRIPTION is updated to describe the relationship of SOMAXCONN and the *backlog* argument.

llrint, llrintf, llrintl

Round to nearest integer value using current rounding direction

```
#include <math.h>

long long llrint(double x);
long long llrintf(float x);
long long llrintl(long double x);
```

These functions round their argument to the nearest integer value, rounding according to the current rounding direction.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

llround, llroundf, llroundl

Round to nearest integer value

```
#include <math.h>

long long llround(double x);
long long llroundf(float x);
long long llroundl(long double x);
```

These functions round their argument to the nearest integer value, rounding halfway cases away from zero, regardless of the current rounding direction.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

localeconv

Return locale-specific information

```
#include <locale.h>

struct lconv *localeconv(void);
```

Derivation First released in Issue 4. Derived from ANSI standard X3.159-1989, Programming Language C (ANSI C).

Issue 6 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

This reference page is updated for alignment with the ISO/IEC 9899: 1999 standard.

ISO/IEC 9899: 1999 standard, Technical Corrigendum No. 1 is incorporated.

, item XSH/TC1/D6/31 is applied, removing references to **int_curr_symbol** and updating the descriptions of **p_sep_by_space** and **n_sep_by_space**. These

changes are for alignment with the ISO C standard.

localtime, localtime_r

Convert a time value to a broken-down local time

```
#include <time.h>

struct tm *localtime(const time_t *timer);
```
TSF
```
struct tm *localtime_r(const time_t *restrict timer,
    struct tm *restrict result);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *localtime_r*() function is marked as part of the Thread-Safe Functions option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

The **restrict** keyword is added to the *localtime_r*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

Examples are added.

, item XSH/TC1/D6/32 is applied, adding the [EOVERFLOW] error.

, item XSH/TC2/D6/56 is applied, adding a requirement that if *localtime_r*() does not set the *tzname* variable, it shall not set the *daylight* or *timezone* variables. On systems supporting XSI, the *daylight*, *timezone*, and *tzname* variables should all be set to provide information for the same timezone. This updates the description of *localtime_r*() to mention *daylight* and *timezone* as well as *tzname*.

lockf

Record locking on files

XSI
```
#include <unistd.h>

int lockf(int fildes, int function, off_t size);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

log, logf, logl

Natural logarithm function

```
#include <math.h>

double log(double x);
float logf(float x);
long double logl(long double x);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *logf*() and *logl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

log10, log10f, log10l

Base 10 logarithm function

```
#include <math.h>

double log10(double x);
float log10f(float x);
long double log10l(long double x);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *log10f*() and *log10l*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

log1p, log1pf, log1pl

Compute a natural logarithm

```
#include <math.h>

double log1p(double x);
float log1pf(float x);
long double log1pl(long double x);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The *log1p*() function is no longer marked as an extension.

The *log1pf*() and *log1pl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

log2, log2f, log2l

Compute base 2 logarithm functions

```
#include <math.h>

double log2(double x);
float log2f(float x);
long double log2l(long double x);
```

These functions compute the base-2 logarithm of their argument x, $\log_2(x)$.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

logb, logbf, logbl

Radix-independent exponent

```
#include <math.h>

double logb(double x);
float logbf(float x);
long double logbl(long double x);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The *logb*() function is no longer marked as an extension.

The *logbf*() and *logbl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

longjmp

Non-local goto

```
#include <setjmp.h>

void longjmp(jmp_buf env, int val);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION now explicitly makes *longjmp*()'s effect on the signal mask unspecified.

The DESCRIPTION is updated for alignment with the ISO/IEC 9899: 1999 standard.

lrint, lrintf, lrintl

Round to nearest integer value using current rounding direction

```
#include <math.h>

long lrint(double x);
long lrintf(float x);
long lrintl(long double x);
```

These functions round their argument to the nearest integer value, rounding according to the current rounding direction.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

lround, lroundf, lroundl

Round to nearest integer value

```
#include <math.h>

long lround(double x);
long lroundf(float x);
long lroundl(long double x);
```

These functions round their argument to the nearest integer value, rounding halfway cases away from zero, regardless of the current rounding direction.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

lsearch, lfind

Linear search and update

XSI

```
#include <search.h>

void *lsearch(const void *key, void *base, size_t *nelp, size_t width,
    int (*compar)(const void *, const void *));
void *lfind(const void *key, const void *base, size_t *nelp,
    size_t width, int (*compar)(const void *, const void *));
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

lseek

Move the read/write file offset

```
#include <unistd.h>

off_t lseek(int fildes, off_t offset, int whence);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EOVERFLOW] error condition is added. This change is to support large files.

An additional [ESPIPE] error condition is added for sockets.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that *lseek*( ) results are unspecified for typed memory objects.

lstat

Get symbolic link status

```
#include <sys/stat.h>

int lstat(const char *restrict path, struct stat *restrict buf);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The following changes were made to align with the IEEE P1003.1a draft standard:

- This function is now mandatory.
- The [ELOOP] optional error condition is added.

The **restrict** keyword is added to the *lstat*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

makecontext, swapcontext

Manipulate user contexts

OB XSI

```
#include <ucontext.h>

void makecontext(ucontext_t *ucp, void (*func)(void),
    int argc, ...);
int swapcontext(ucontext_t *restrict oucp,
    const ucontext_t *restrict ucp);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The **restrict** keyword is added to the *swapcontext*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC2/D6/57 is applied, obsoleting the functions. The prototype makes use of an obsolescent feature of the ISO C standard. Therefore, a strictly conforming POSIX application cannot use this form, and thus, use of *getcontext*( ), *makecontext*( ), and *swapcontext*( ) is marked obsolescent. It is recommended that applications that use these functions be converted to use threads instead.

malloc

A memory allocator

```
#include <stdlib.h>

void *malloc(size_t size);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE section, the requirement to set *errno* to indicate an error is added.
- The [ENOMEM] error condition is added.

mblen

Get number of bytes in a character

```
#include <stdlib.h>

int mblen(const char *s, size_t n);
```

Derivation First released in Issue 4. Aligned with the ISO C standard.

Issue 6 No functional changes are made in this issue.

mbrlen

Get number of bytes in a character (restartable)

```
#include <wchar.h>

size_t mbrlen(const char *restrict s, size_t n,
    mbstate_t *restrict ps);
```

Derivation First released in Issue 5. Included for alignment with the ISO/IEC 9899: 1990 standard.

Issue 6 The *mbrlen*() prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

mbrtowc

Convert a character to a wide-character code (restartable)

```
#include <wchar.h>

size_t mbrtowc(wchar_t *restrict pwc, const char *restrict s,
    size_t n, mbstate_t *restrict ps);
```

Derivation First released in Issue 5. Included for alignment with the ISO/IEC 9899: 1990 standard.

Issue 6 The *mbrtowc*() prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EINVAL] error condition is added.

ISO/IEC 9899: 1999 standard, Technical Corrigendum No. 1 is incorporated.

mbsinit

Determine conversion object status

```
#include <wchar.h>

int mbsinit(const mbstate_t *ps);
```

Derivation First released in Issue 5. Included for alignment with the ISO/IEC 9899: 1990 standard.

Issue 6 No functional changes are made in this issue.

mbsrtowcs

Convert a character string to a wide-character string (restartable)

```
#include <wchar.h>

size_t mbsrtowcs(wchar_t *restrict dst, const char **restrict src,
    size_t len, mbstate_t *restrict ps);
```

Derivation First released in Issue 5. Included for alignment with the ISO/IEC 9899:1990 standard.

Issue 6 The *mbsrtowcs*() prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

The [EINVAL] error condition is marked CX.

mbstowcs

Convert a character string to a wide-character string

```
#include <stdlib.h>

size_t mbstowcs(wchar_t *restrict pwcs, const char *restrict s,
    size_t n);
```

Derivation First released in Issue 4. Aligned with the ISO C standard.

Issue 6 The *mbstowcs*() prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

mbtowc

Convert a character to a wide-character code

```
#include <stdlib.h>

int mbtowc(wchar_t *restrict pwc, const char *restrict s, size_t n);
```

Derivation First released in Issue 4. Aligned with the ISO C standard.

Issue 6 The *mbtowc*() prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

memccpy

Copy bytes in memory

XSI

```
#include <string.h>

void *memccpy(void *restrict s1, const void *restrict s2,
    int c, size_t n);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The **restrict** keyword is added to the *memccpy*() prototype for alignment with the ISO/IEC 9899:1999 standard.

memchr

Find byte in memory

```
#include <string.h>

void *memchr(const void *s, int c, size_t n);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

memcmp

Compare bytes in memory

```
#include <string.h>

int memcmp(const void *s1, const void *s2, size_t n);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

memcpy

Copy bytes in memory

```
#include <string.h>

void *memcpy(void *restrict s1, const void *restrict s2, size_t n);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *memcpy*() prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

memmove

Copy bytes in memory with overlapping areas

```
#include <string.h>

void *memmove(void *s1, const void *s2, size_t n);
```

Derivation First released in Issue 4. Derived from ANSI standard X3.159-1989, Programming Language C (ANSI C).

Issue 6 No functional changes are made in this issue.

memset

Set bytes in memory

```
#include <string.h>

void *memset(void *s, int c, size_t n);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

mkdir

Make a directory

```
#include <sys/stat.h>

int mkdir(const char *path, mode_t mode);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

mkfifo

Make a FIFO special file

```
#include <sys/stat.h>

int mkfifo(const char *path, mode_t mode);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

mknod

Make a directory, a special file, or a regular file

XSI

```
#include <sys/stat.h>

int mknod(const char *path, mode_t mode, dev_t dev);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

mkstemp

Make a unique filename

XSI
```
#include <stdlib.h>

int mkstemp(char *template);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

mktemp

Make a unique filename (**LEGACY**)

XSI
```
#include <stdlib.h>

char *mktemp(char *template);
```

Derivation First released in Issue 4, Version 2.

Issue 6 This function is marked LEGACY and may not be available on all implementations.

Between the time a pathname is created and the file opened, it is possible for some other process to create a file with the same name. The *mkstemp*() function avoids this problem and is preferred over this function.

mktime

Convert broken-down time into time since the Epoch

```
#include <time.h>

time_t mktime(struct tm *timeptr);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1) and ANSI standard X3.159-1989, Programming Language C (ANSI C).

Issue 6 , item XSH/TC2/D6/58 is applied, updating the RETURN VALUE and ERRORS sections to add the optional [EOVERFLOW] error as a CX extension.

, item XSH/TC2/D6/59 is applied, adding the *tzset*() function to the SEE ALSO section.

mlock, munlock

Lock or unlock a range of process address space (**REALTIME**)

MLR
```
#include <sys/mman.h>

int mlock(const void *addr, size_t len);
int munlock(const void *addr, size_t len);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *mlock*() and *munlock*() functions are marked as part of the Range Memory Locking option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Range Memory Locking option.

mlockall, munlockall

Lock/unlock the address space of a process (**REALTIME**)

ML
```
#include <sys/mman.h>

int mlockall(int flags);
int munlockall(void);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *mlockall*() and *munlockall*() functions are marked as part of the Process Memory Locking option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Process Memory Locking option.

mmap

Map pages of memory

MC3
```
#include <sys/mman.h>

void *mmap(void *addr, size_t len, int prot, int flags,
    int fildes, off_t off);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The *mmap*() function is marked as part of the Memory Mapped Files, Shared Memory Objects option and Typed Memory Objects option.

The Open Group Corrigendum U028/6 is applied, changing (**void** *)–1 to MAP_FAILED.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to describe the use of MAP_FIXED.
- The DESCRIPTION is updated to describe the addition of an extra reference to the file associated with the file descriptor passed to *mmap*().
- The DESCRIPTION is updated to state that there may be implementation-defined limits on the number of memory regions that can be mapped.
- The DESCRIPTION is updated to describe constraints on the alignment and size of the *off* argument.
- The [EINVAL] and [EMFILE] error conditions are added.
- The [EOVERFLOW] error condition is added. This change is to support large files.

The following changes are made for alignment with ISO/IEC 9945-1: 1996 (POSIX-1):

- The DESCRIPTION is updated to describe the cases when MAP_PRIVATE and MAP_FIXED need not be supported.

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- Semantics for typed memory objects are added to the DESCRIPTION.
- New [ENOMEM] and [ENXIO] errors are added to the ERRORS section.

, item XSH/TC2/D6/60 is applied, updating the DESCRIPTION and ERRORS sections to add the [EINVAL] error when *len* is zero.

modf, modff, modfl

Decompose a floating-point number

```
#include <math.h>

double modf(double x, double *iptr);
float modff(float value, float *iptr);
long double modfl(long double value, long double *iptr);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *modff*() and *modfl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

mprotect

Set protection of memory mapping

MPR

```
#include <sys/mman.h>

int mprotect(void *addr, size_t len, int prot);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The *mprotect*() function is marked as part of the Memory Protection option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to state that implementations require *addr* to be a multiple of the page size as returned by *sysconf*().
- The [EINVAL] error condition is added.

mq_close

Close a message queue (**REALTIME**)

MSG
```
#include <mqueue.h>

int mq_close(mqd_t mqdes);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *mq_close*( ) function is marked as part of the Message Passing option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

mq_getattr

Get message queue attributes (**REALTIME**)

MSG
```
#include <mqueue.h>

int mq_getattr(mqd_t mqdes, struct mq_attr *mqstat);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *mq_getattr*( ) function is marked as part of the Message Passing option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

, item XSH/TC2/D6/61 is applied, updating the ERRORS section to change the [EBADF] error from mandatory to optional.

mq_notify

Notify process that a message is available (**REALTIME**)

MSG
```
#include <mqueue.h>

int mq_notify(mqd_t mqdes, const struct sigevent *notification);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *mq_notify*( ) function is marked as part of the Message Passing option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

mq_open

Open a message queue (**REALTIME**)

MSG
```
#include <mqueue.h>

mqd_t mq_open(const char *name, int oflag, ...);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *mq_open*( ) function is marked as part of the Message Passing option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

The DESCRIPTION of O_EXCL is updated in response to IEEE PASC Interpretation 1003.1c #48.

, item XSH/TC2/D6/62 is applied, updating the description of the permission bits in the DESCRIPTION section. The change is made for consistency with the *shm_open*( ) and *sem_open*( ) functions.

mq_receive, mq_timedreceive

Receive a message from a message queue (**REALTIME**)

MSG
```
#include <mqueue.h>

ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len,
    unsigned *msg_prio);
```

MSG TMO
```
#include <mqueue.h>
#include <time.h>

ssize_t mq_timedreceive(mqd_t mqdes, char *restrict msg_ptr,
    size_t msg_len, unsigned *restrict msg_prio,
    const struct timespec *restrict abs_timeout);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *mq_receive*( ) function is marked as part of the Message Passing option.

The Open Group Corrigendum U021/4 is applied. The DESCRIPTION is changed to refer to *msg_len* rather than *maxsize*.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In this function it is possible for the return value to exceed the range of the type **ssize_t** (since **size_t** has a larger range of positive values than **ssize_t**). A sentence restricting the size of the **size_t** object is added to the description to resolve this conflict.

The *mq_timedreceive*( ) function is added for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the *mq_timedreceive*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

IEEE PASC Interpretation 1003.1 #109 is applied, correcting the return type for *mq_timedreceive*( ) from **int** to **ssize_t**.

mq_send, mq_timedsend

Send a message to a message queue (**REALTIME**)

MSG
```
#include <mqueue.h>

int mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
    unsigned msg_prio);
```

MSG TMO
```
#include <mqueue.h>
#include <time.h>

int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
    unsigned msg_prio, const struct timespec *abs_timeout);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *mq_send*( ) function is marked as part of the Message Passing option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

The *mq_timedsend*( ) function is added for alignment with IEEE Std 1003.1d-1999.

mq_setattr

Set message queue attributes (**REALTIME**)

MSG
```
#include <mqueue.h>

int mq_setattr(mqd_t mqdes, const struct mq_attr *restrict mqstat,
    struct mq_attr *restrict omqstat);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *mq_setattr*( ) function is marked as part of the Message Passing option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

The **restrict** keyword is added to the *mq_setattr*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

mq_unlink

Remove a message queue (**REALTIME**)

MSG
```
#include <mqueue.h>

int mq_unlink(const char *name);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *mq_unlink*() function is marked as part of the Message Passing option.

The Open Group Corrigendum U021/5 is applied, clarifying that upon unsuccessful completion, the named message queue is unchanged by this function.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

msgctl

XSI message control operations

XSI
```
#include <sys/msg.h>

int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

Derivation First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 6 No functional changes are made in this issue.

msgget

Get the XSI message queue identifier

XSI
```
#include <sys/msg.h>

int msgget(key_t key, int msgflg);
```

Derivation First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 6 No functional changes are made in this issue.

msgrcv

XSI message receive operation

XSI
```
#include <sys/msg.h>

ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp,
    int msgflg);
```

Derivation First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 6 No functional changes are made in this issue.

msgsnd

XSI message send operation

XSI
```
#include <sys/msg.h>

int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

Derivation First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 6 No functional changes are made in this issue.

msync

Synchronize memory with physical storage

MF SIO
```
#include <sys/mman.h>

int msync(void *addr, size_t len, int flags);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The *msync*() function is marked as part of the Memory Mapped Files and Synchronized Input and Output options.

The following changes are made for alignment with ISO/IEC 9945-1: 1996 (POSIX-1):

- The [EBUSY] mandatory error condition is added.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to state that implementations require *addr* to be a multiple of the page size.
- The second [EINVAL] error condition is made mandatory.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding reference to typed memory objects.

munmap

Unmap pages of memory

MC3
```
#include <sys/mman.h>

int munmap(void *addr, size_t len);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The *munmap*() function is marked as part of the Memory Mapped Files, Shared Memory Objects option and Typed Memory Objects option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to state that implementations require *addr* to be a multiple of the page size.

- The [EINVAL] error conditions are added.

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- Semantics for typed memory objects are added to the DESCRIPTION.

nan, nanf, nanl

Return quiet NaN

```
#include <math.h>

double nan(const char *tagp);
float nanf(const char *tagp);
long double nanl(const char *tagp);
```

The function call *nan*("*n-char-sequence*") is equivalent to:

```
strtod("NAN(n-char-sequence)", (char **) NULL);
```

The function call *nan*("") is equivalent to:

```
strtod("NAN()", (char **) NULL)
```

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

nanosleep

High resolution sleep (**REALTIME**)

TMR

```
#include <time.h>

int nanosleep(const struct timespec *rqtp, struct timespec *rmtp);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *nanosleep*( ) function is marked as part of the Timers option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Timers option.

nearbyint, nearbyintf, nearbyintl

Floating-point rounding functions

```
#include <math.h>

double nearbyint(double x);
float nearbyintf(float x);
long double nearbyintl(long double x);
```

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

nextafter, nextafterf, nextafterl, nexttoward, nexttowardf, nexttowardl

Next representable floating-point number

```
#include <math.h>

double nextafter(double x, double y);
float nextafterf(float x, float y);
long double nextafterl(long double x, long double y);
double nexttoward(double x, long double y);
float nexttowardf(float x, long double y);
long double nexttowardl(long double x, long double y);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The *nextafter*( ) function is no longer marked as an extension.

The *nextafterf*( ), *nextafterl*( ), *nexttoward*( ), *nexttowardf*( ), *nexttowardl*( ) functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

nftw

Walk a file tree

XSI

```
#include <ftw.h>

int nftw(const char *path, int (*fn)(const char *,
    const struct stat *, int, struct FTW *), int depth, int flags);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The Open Group Base Resolution bwg97-003 is applied.

The ERRORS section is updated as follows:

- The wording of the mandatory [ELOOP] error condition is updated.
- A second optional [ELOOP] error condition is added.
- The [EOVERFLOW] mandatory error condition is added.

Text is added to the DESCRIPTION to say that the *nftw*( ) function need not be reentrant and that the results are unspecified if the application-supplied *fn* function does not preserve the current working directory.

, item XSH/TC2/D6/64 is applied, changing the argument *depth* to *fd_limit* throughout and changing ‘‘to a maximum of 5 levels deep’’ to ‘‘using a maximum of 5 file descriptors’’ in the EXAMPLES section.

nice

Change the nice value of a process

XSI
```
#include <unistd.h>

int nice(int incr);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

nl_langinfo

Language information

XSI
```
#include <langinfo.h>

char *nl_langinfo(nl_item item);
```

Derivation First released in Issue 2.

Issue 6 No functional changes are made in this issue.

open

Open a file

OH
```
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *path, int oflag, ... );
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, O_CREAT is amended to state that the group ID of the file is set to the group ID of the file's parent directory or to the effective group ID of the process. This is a FIPS requirement.
- In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file description. This change is to support large files.
- In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support large files.
- The [ENXIO] mandatory error condition is added.
- The [EINVAL], [ENAMETOOLONG], and [ETXTBSY] optional error conditions are added.

The DESCRIPTION and ERRORS sections are updated so that items related to the optional XSI STREAMS Option Group are marked.

The following changes were made to align with the IEEE P1003.1a draft standard:

- An explanation is added of the effect of the O_CREAT and O_EXCL flags when the path refers to a symbolic link.

- The [ELOOP] optional error condition is added.

The DESCRIPTION of O_EXCL is updated in response to IEEE PASC Interpretation 1003.1c #48.

opendir

Open a directory

```
#include <dirent.h>

DIR *opendir(const char *dirname);
```

Derivation First released in Issue 2.

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

pause

Suspend the thread until a signal is received

```
#include <unistd.h>

int pause(void);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

pclose

Close a pipe stream to or from a process

CX

```
#include <stdio.h>

int pclose(FILE *stream);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

perror

Write error messages to standard error

```
#include <stdio.h>

void perror(const char *s);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

pipe

Create an interprocess channel

```
#include <unistd.h>

int pipe(int fildes[2]);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to indicate that certain dispositions of *fildes*[0] and *fildes*[1] are unspecified.

poll

Input/output multiplexing

XSI

```
#include <poll.h>

int poll(struct pollfd fds[], nfds_t nfds, int timeout);
```

Derivation First released in Issue 4, Version 2.

Issue 6 Text referring to sockets is added to the DESCRIPTION.

Text relating to the XSI STREAMS Option Group is marked.

The Open Group Corrigendum U055/3 is applied, updating the DESCRIPTION of POLLWRBAND from ‘‘This event only examines bands that have been written to at least once.’’ to ‘‘If any priority band has been written to on this STREAM, this event only examines bands that have been written to at least once.’’

popen

Initiate pipe streams to or from a process

CX

```
#include <stdio.h>

FILE *popen(const char *command, const char *mode);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The optional [EMFILE] error condition is added.

posix_fadvise

File advisory information (**ADVANCED REALTIME**)

ADV

```
#include <fcntl.h>

int posix_fadvise(int fd, off_t offset, off_t len, int advice);
```

The *posix_fadvise*() function advises the implementation on the expected behavior of the application with respect to the data in the file associated with the open file descriptor, *fd*, starting at *offset* and continuing for *len* bytes.

Implementations may use information conveyed by a previous *posix_fadvise*() call to influence the manner in which allocation is performed. For example, if an application did the following calls:

```
fd = open("file");
posix_fadvise(fd, offset, len, POSIX_FADV_SEQUENTIAL);
posix_fallocate(fd, len, size);
```

an implementation might allocate the file contiguously on disk.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1d-1999. This function is part of the Advanced Realtime Option Group and may not be available on all implementations.

In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

, item XSH/TC2/D6/68 is applied, changing the function prototype in the SYNOPSIS section. The previous prototype was not large-file aware, and the standard developers felt it acceptable to make this change before implementations of the functions become widespread.

posix_fallocate

File space control (**ADVANCED REALTIME**)

ADV
```
#include <fcntl.h>

int posix_fallocate(int fd, off_t offset, off_t len);
```

The *posix_fallocate*() function can be used to guarantee no [ENOSPC] errors and to improve performance by prepaying any overhead required for block allocation.

The *posix_fallocate*() function ensures that any required storage for regular file data starting at *offset* and continuing for *len* bytes is allocated on the file system storage media. If the functon returns successfully, subsequent writes to the specified file data will not fail due to the lack of free space on the file system storage media.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1d-1999. This function is part of the Advanced Realtime Option Group and may not be available on all implementations.

In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

, item XSH/TC2/D6/69 is applied, changing the function prototype in the SYNOPSIS section. The previous prototype was not large-file aware, and the standard developers felt it acceptable to make this change before implementations of the functions become widespread.

posix_madvise

Memory advisory information and alignment control (**ADVANCED REALTIME**)

ADV
```
#include <sys/mman.h>

int posix_madvise(void *addr, size_t len, int advice);
```

The *posix_madvise*() function advises the implementation on the expected behavior of the application with respect to the data in the memory starting at address *addr*, and continuing for *len* bytes.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1d-1999. This function is part of the Advanced Realtime Option Group and may not be available on all implementations.

IEEE PASC Interpretation 1003.1 #102 is applied, clarifying that this function should be declared in **<sys/mman.h>** and not **<fcntl.h>**.

posix_mem_offset

Find offset and length of a mapped typed memory block (**ADVANCED REALTIME**)

TYM
```
#include <sys/mman.h>

int posix_mem_offset(const void *restrict addr, size_t len,
    off_t *restrict off, size_t *restrict contig_len,
    int *restrict fildes);
```

The *posix_mem_offset*() function returns in the variable pointed to by *off* a value that identifies the offset (or location), within a memory object, of the memory block currently mapped at *addr*.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1j-2000. This function is part of the Advanced Realtime Option Group and may not be available on all implementations.

posix_memalign

Aligned memory allocation (**ADVANCED REALTIME**)

ADV
```
#include <stdlib.h>

int posix_memalign(void **memptr, size_t alignment, size_t size);
```

The *posix_memalign*() function was added to allow for the allocation of specifically aligned buffers.

The *posix_memalign*() function allocates *size* bytes aligned on a boundary specified by *alignment*, and returns a pointer to the allocated memory in *memptr*.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1d-1999. This function is part of the Advanced Realtime Option Group and may not be available on all implementations.

In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

posix_openpt

Open a pseudo-terminal device

XSI
```
#include <stdlib.h>
#include <fcntl.h>

int posix_openpt(int oflag);
```

This function is a method for portably obtaining a file descriptor of a master terminal device for a pseudo-terminal. The *grantpt*() and *ptsname*() functions can be used to manipulate mode and ownership permissions and to obtain the name of the slave device, respectively.

The standard developers considered the matter of adding a special device for cloning master pseudo-terminals: the **/dev/ptmx** device. However, consensus could not be reached, and it was felt that adding a new function would permit other implementations.

An example that opens a pseudo-terminal and returns the name of the slave device and a file descriptor is given below:

```
#include <fcntl.h>
#include <stdio.h>

int masterfd, slavefd;
char *slavedevice;

masterfd = posix_openpt(O_RDWR|O_NOCTTY);

if (masterfd == -1
    || grantpt (masterfd) == -1
    || unlockpt (masterfd) == -1
    || (slavedevice = ptsname (masterfd)) == NULL)
    return -1;

printf("slave device is: %s\n", slavedevice);

slavefd = open(slave, O_RDWR|O_NOCTTY);
if (slavefd < 0)
    return -1;
```

Derivation First released in Issue 6.

posix_spawn, posix_spawnp

Spawn a process (**ADVANCED REALTIME**)

SPN

```
#include <spawn.h>

int posix_spawn(pid_t *restrict pid, const char *restrict path,
    const posix_spawn_file_actions_t *file_actions,
    const posix_spawnattr_t *restrict attrp,
    char *const argv[restrict], char *const envp[restrict]);
int posix_spawnp(pid_t *restrict pid, const char *restrict file,
    const posix_spawn_file_actions_t *file_actions,
    const posix_spawnattr_t *restrict attrp,
    char *const argv[restrict], char *const envp[restrict]);
```

The *posix_spawn*() and *posix_spawnp*() functions create a new process (child process) from the specified process image.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1d-1999. These functions are part of the Advanced Realtime Option Group and may not be available on all implementations.

IEEE PASC Interpretation 1003.1 #103 is applied, noting that the signal default actions are changed as well as the signal mask in step 2.

IEEE PASC Interpretation 1003.1 #132 is applied. This addresses the issue of whether or not an implementation allows SIG_IGN as a SIGCHLD disposition to be inherited across a call to one of the *exec* family of functions or *posix_spawn*(), leaving it explicitly unspecified.

posix_spawn_file_actions_addclose, posix_spawn_file_actions_addopen

Add close or open action to spawn file actions object (**ADVANCED REALTIME**)

SPN
```
#include <spawn.h>

int posix_spawn_file_actions_addclose(posix_spawn_file_actions_t *
    file_actions, int fildes);
int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t *restrict
    file_actions, int fildes, const char *restrict path,
    int oflag, mode_t mode);
```

These functions add or delete a close or open action to a spawn file actions object.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1d-1999. These functions are part of the Advanced Realtime Option Group and may not be available on all implementations.

IEEE PASC Interpretation 1003.1 #105 is applied, adding a note to the DESCRIPTION that the string pointed to by *path* is copied by the *posix_spawn_file_actions_addopen*( ) function.

posix_spawn_file_actions_adddup2

Add dup2 action to spawn file actions object (**ADVANCED REALTIME**)

SPN
```
#include <spawn.h>

int posix_spawn_file_actions_adddup2(posix_spawn_file_actions_t *
    file_actions, int fildes, int newfildes);
```

The *posix_spawn_file_actions_adddup2*( ) function adds a *dup2*( ) action to the object referenced by *file_actions* that causes the file descriptor *fildes* to be duplicated as *newfildes* (as if *dup2*(*fildes*, *newfildes*) had been called) when a new process is spawned using this file actions object.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1d-1999. This function is part of the Advanced Realtime Option Group and may not be available on all implementations.

IEEE PASC Interpretation 1003.1 #104 is applied, noting that the [EBADF] error can apply to the *newfildes* argument in addition to *fildes*.

posix_spawn_file_actions_destroy, posix_spawn_file_actions_init

Destroy and initialize spawn file actions object (**ADVANCED REALTIME**)

SPN
```
#include <spawn.h>

int posix_spawn_file_actions_destroy(posix_spawn_file_actions_t *
    file_actions);
int posix_spawn_file_actions_init(posix_spawn_file_actions_t *
    file_actions);
```

The *posix_spawn_file_actions_destroy*( ) function destroys the object referenced by *file_actions*; the object becomes, in effect, uninitialized.

The *posix_spawn_file_actions_init*( ) function initializes the object referenced by *file_actions* to contain no file actions for *posix_spawn*( ) or *posix_spawnp*( ) to perform.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1d-1999. These functions are part of the Advanced Realtime Option Group and may not be available on all implementations.

In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

posix_spawnattr_destroy, posix_spawnattr_init

Destroy and initialize spawn attributes object (**ADVANCED REALTIME**)

SPN
```
#include <spawn.h>

int posix_spawnattr_destroy(posix_spawnattr_t *attr);
int posix_spawnattr_init(posix_spawnattr_t *attr);
```

The *posix_spawnattr_destroy*( ) function destroys a spawn attributes object.

The *posix_spawnattr_init*( ) function initializes a spawn attributes object *attr* with the default value for all of the individual attributes used by the implementation.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1d-1999. These functions are part of the Advanced Realtime Option Group and may not be available on all implementations.

IEEE PASC Interpretation 1003.1 #106 is applied, noting that the effect of initializing an already initialized spawn attributes option is undefined.

posix_spawnattr_getflags, posix_spawnattr_setflags

Get and set spawn-flags attribute of spawn attributes object (**ADVANCED REALTIME**)

SPN
```
#include <spawn.h>

int posix_spawnattr_getflags(const posix_spawnattr_t *restrict attr,
    short *restrict flags);
int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
```

The *posix_spawnattr_getflags*( ) function obtains the value of the *spawn-flags* attribute from the attributes object referenced by *attr*.

The *posix_spawnattr_setflags*( ) function sets the *spawn-flags* attribute in an initialized attributes object referenced by *attr*.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1d-1999. These functions are part of the Advanced Realtime Option Group and may not be available on all implementations.

posix_spawnattr_getpgroup, posix_spawnattr_setpgroup

Get and set spawn-pgroup attribute of spawn attributes object (**ADVANCED REALTIME**)

SPN
```
#include <spawn.h>

int posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict attr,
    pid_t *restrict pgroup);
int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);
```

The *posix_spawnattr_getpgroup*() function obtains the value of the *spawn-pgroup* attribute from the attributes object referenced by *attr*.

The *posix_spawnattr_setpgroup*() function sets the *spawn-pgroup* attribute in an initialized attributes object referenced by *attr*.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1d-1999. These functions are part of the Advanced Realtime Option Group and may not be available on all implementations.

posix_spawnattr_getschedparam, posix_spawnattr_setschedparam

Get and set spawn-schedparam attribute of spawn attributes object (**ADVANCED REALTIME**)

SPN PS
```
#include <spawn.h>
#include <sched.h>

int posix_spawnattr_getschedparam(
    const posix_spawnattr_t *restrict attr,
    struct sched_param *restrict schedparam);
int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict attr,
    const struct sched_param *restrict schedparam);
```

The *posix_spawnattr_getschedparam*() function obtains the value of the *spawn-schedparam* attribute from the attributes object referenced by *attr*.

The *posix_spawnattr_setschedparam*() function sets the *spawn-schedparam* attribute in an initialized attributes object referenced by *attr*.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1d-1999. These functions are part of the Advanced Realtime Option Group and may not be available on all implementations.

posix_spawnattr_getschedpolicy, posix_spawnattr_setschedpolicy

Get and set spawn-schedpolicy attribute of spawn attributes object (**ADVANCED REALTIME**)

SPN PS
```
#include <spawn.h>
#include <sched.h>

int posix_spawnattr_getschedpolicy(
    const posix_spawnattr_t *restrict attr,
    int *restrict schedpolicy);
int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
    int schedpolicy);
```

The *posix_spawnattr_getschedpolicy*() function obtains the value of the *spawn-schedpolicy* attribute from the attributes object referenced by *attr*.

The *posix_spawnattr_setschedpolicy*() function sets the *spawn-schedpolicy* attribute in an initialized attributes object referenced by *attr*.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1d-1999. These functions are part of the Advanced Realtime Option Group and may not be available on all implementations.

posix_spawnattr_getsigdefault, posix_spawnattr_setsigdefault

Get and set spawn-sigdefault attribute of spawn attributes object (**ADVANCED REALTIME**)

SPN
```
#include <signal.h>
#include <spawn.h>
```

```
int posix_spawnattr_getsigdefault(
    const posix_spawnattr_t *restrict attr,
    sigset_t *restrict sigdefault);
int posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict attr,
    const sigset_t *restrict sigdefault);
```

The *posix_spawnattr_getsigdefault*() function obtains the value of the *spawn-sigdefault* attribute from the attributes object referenced by *attr*.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1d-1999. These functions are part of the Advanced Realtime Option Group and may not be available on all implementations.

posix_spawnattr_getsigmask, posix_spawnattr_setsigmask

Get and set spawn-sigmask attribute of spawn attributes object (**ADVANCED REALTIME**)

SPN
```
#include <signal.h>
#include <spawn.h>
```

```
int posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict attr,
    sigset_t *restrict sigmask);
int posix_spawnattr_setsigmask(posix_spawnattr_t *restrict attr,
    const sigset_t *restrict sigmask);
```

The *posix_spawnattr_getsigmask*() function obtains the value of the *spawn-sigmask* attribute from the attributes object referenced by *attr*.

The *posix_spawnattr_setsigmask*() function sets the *spawn-sigmask* attribute in an initialized attributes object referenced by *attr*.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1d-1999. These functions are part of the Advanced Realtime Option Group and may not be available on all implementations.

posix_trace_attr_destroy, posix_trace_attr_init

Trace stream attributes object destroy and initialization (**TRACING**)

TRC
```
#include <trace.h>

int posix_trace_attr_destroy(trace_attr_t *attr);
int posix_trace_attr_init(trace_attr_t *attr);
```

The *posix_trace_attr_destroy*() function destroys an initialized trace attributes object.

The *posix_trace_attr_init*() function initializes a trace attributes object *attr* with the default value for all of the individual attributes used by a given implementation.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1q-2000. These functions are part of the Tracing Option Group and may not be available on all implementations.

IEEE PASC Interpretation 1003.1 #123 is applied.

posix_trace_attr_getclockres, posix_trace_attr_getcreatetime, posix_trace_attr_getgenversion, posix_trace_attr_getname, posix_trace_attr_setname

Retrieve and set information about a trace stream (**TRACING**)

TRC
```
#include <time.h>
#include <trace.h>

int posix_trace_attr_getclockres(const trace_attr_t *attr,
    struct timespec *resolution);
int posix_trace_attr_getcreatetime(const trace_attr_t *attr,
    struct timespec *createtime);
```

```
#include <trace.h>

int posix_trace_attr_getgenversion(const trace_attr_t *attr,
    char *genversion);
int posix_trace_attr_getname(const trace_attr_t *attr,
    char *tracename);
int posix_trace_attr_setname(trace_attr_t *attr,
    const char *tracename);
```

The *posix_trace_attr_getclockres*() function copies the clock resolution of the clock used to generate timestamps from the *clock-resolution* attribute of the attributes object pointed to by the *attr* argument into the structure pointed to by the *resolution* argument.

The *posix_trace_attr_getcreatetime*() function copies the trace stream creation time from the *creation-time* attribute of the attributes object pointed to by the *attr* argument into the structure pointed to by the *createtime* argument.

The *posix_trace_attr_getgenversion*() function copies the string containing version information from the *generation-version* attribute of the attributes object pointed to by the *attr* argument into the string pointed to by the *genversion* argument.

The *posix_trace_attr_getname*() function copies the string containing the trace name from the *trace-name* attribute of the attributes object pointed to by the *attr* argument into the string pointed to by the *tracename* argument.

The *posix_trace_attr_setname*() function sets the name in the *trace-name* attribute of the attributes object pointed to by the *attr* argument, using the trace name string supplied by the *tracename* argument.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1q-2000. These functions are part of the Tracing Option Group and may not be available on all implementations.

posix_trace_attr_getinherited, posix_trace_attr_getlogfullpolicy, posix_trace_attr_getstreamfullpolicy, posix_trace_attr_setinherited, posix_trace_attr_setlogfullpolicy, posix_trace_attr_setstreamfullpolicy

Retrieve and set the behavior of a trace stream (**TRACING**)

TRC
```
#include <trace.h>
```

TRC TRI
```
int posix_trace_attr_getinherited(const trace_attr_t *restrict attr,
    int *restrict inheritancepolicy);
```

TRC TRL
```
int posix_trace_attr_getlogfullpolicy(const trace_attr_t *restrict attr,
    int *restrict logpolicy);
```

TRC
```
int posix_trace_attr_getstreamfullpolicy(const trace_attr_t *attr,
    int *streampolicy);
```

TRC TRI
```
int posix_trace_attr_setinherited(trace_attr_t *attr,
    int inheritancepolicy);
```

TRC TRL
```
int posix_trace_attr_setlogfullpolicy(trace_attr_t *attr,
    int logpolicy);
```

TRC
```
int posix_trace_attr_setstreamfullpolicy(trace_attr_t *attr,
    int streampolicy);
```

The *posix_trace_attr_getinherited*() and *posix_trace_attr_setinherited*() functions, respectively, get and set the inheritance policy stored in the *inheritance* attribute for traced processes across the *fork*() and *spawn*() operations.

The *posix_trace_attr_getlogfullpolicy*() and *posix_trace_attr_setlogfullpolicy*() functions, respectively, get and set the trace log full policy stored in the *log-full-policy* attribute of the attributes object pointed to by the *attr* argument.

The *posix_trace_attr_getstreamfullpolicy*() and *posix_trace_attr_setstreamfullpolicy*() functions, respectively, get and set the trace stream full policy stored in the *stream-full-policy* attribute of the attributes object pointed to by the *attr* argument.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1q-2000. These functions are part of the Tracing Option Group and may not be available on all implementations.

posix_trace_attr_getlogsize, posix_trace_attr_getmaxdatasize, posix_trace_attr_getmaxsystemeventsize, posix_trace_attr_getmaxuserevent size, posix_trace_attr_getstreamsize, posix_trace_attr_setlogsize, posix_trace_attr_setmaxdatasize, posix_trace_attr_setstreamsize

Retrieve and set trace stream size attributes (**TRACING**)

TRC
```
#include <sys/types.h>
#include <trace.h>
```

TRC TRL
```
int posix_trace_attr_getlogsize(const trace_attr_t *restrict attr,
    size_t *restrict logsize);
```

```
TRC     int posix_trace_attr_getmaxdatasize(const trace_attr_t *restrict attr,
            size_t *restrict maxdatasize);
        int posix_trace_attr_getmaxsystemeventsize(
            const trace_attr_t *restrict attr,
            size_t *restrict eventsize);
        int posix_trace_attr_getmaxusereventsize(
            const trace_attr_t *restrict attr,
            size_t data_len, size_t *restrict eventsize);
        int posix_trace_attr_getstreamsize(const trace_attr_t *restrict attr,
            size_t *restrict streamsize);
TRC TRL int posix_trace_attr_setlogsize(trace_attr_t *attr,
            size_t logsize);
TRC     int posix_trace_attr_setmaxdatasize(trace_attr_t *attr,
            size_t maxdatasize);
        int posix_trace_attr_setstreamsize(trace_attr_t *attr,
            size_t streamsize);
```

The *posix_trace_attr_getlogsize*( ) function copies the log size, in bytes, from the *log-max-size* attribute of the attributes object pointed to by the *attr* argument into the variable pointed to by the *logsize* argument.

The *posix_trace_attr_setlogsize*( ) function sets the maximum allowed size, in bytes, in the *log-max-size* attribute of the attributes object pointed to by the *attr* argument, using the size value supplied by the *logsize* argument.

The *posix_trace_attr_getmaxdatasize*( ) function copies the maximum user trace event data size, in bytes, from the *max-data-size* attribute of the attributes object pointed to by the *attr* argument into the variable pointed to by the *maxdatasize* argument.

The *posix_trace_attr_getmaxsystemeventsize*( ) function calculates the maximum memory size, in bytes, required to store a single system trace event for the trace stream attributes object pointed to by the *attr* argument, returning the result in the variable pointed to by the *eventsize* argument.

The *posix_trace_attr_getmaxusereventsize*( ) function calculates the maximum memory size, in bytes, required to store a single user trace event generated by a call to *posix_trace_event*( ) with a *data_len* parameter equal to the *data_len* value specified in this call. This value is calculated for the trace stream attributes object pointed to by the *attr* argument and is returned in the variable pointed to by the *eventsize* argument.

The *posix_trace_attr_getstreamsize*( ) function copies the stream size, in bytes, from the *stream-min-size* attribute of the attributes object pointed to by the *attr* argument into the variable pointed to by the *streamsize* argument.

The *posix_trace_attr_setmaxdatasize*( ) function sets the maximum allowed size, in bytes, in the *max-data-size* attribute of the attributes object pointed to by the *attr* argument, using the size value supplied by the *maxdatasize* argument.

The *posix_trace_attr_setstreamsize*( ) function sets the minimum allowed size, in bytes, in the *stream-min-size* attribute of the attributes object pointed to by the *attr* argument, using the size value supplied by the *streamsize* argument.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1q-2000. These functions are part of the Tracing Option Group and may not be available on all implementations.

posix_trace_clear

Clear trace stream and trace log (**TRACING**)

TRC
```
#include <sys/types.h>
#include <trace.h>

int posix_trace_clear(trace_id_t trid);
```

The *posix_trace_clear*( ) function reinitializes the trace stream identified by the argument *trid* as if it were returning from the *posix_trace_create*( ) function, except that the same allocated resources will be reused, the mapping of trace event type identifiers to trace event names is unchanged, and the trace stream status remains unchanged (that is, if it was running, it remains running and if it was suspended, it remains suspended).

Derivation First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

IEEE PASC Interpretation 1003.1 #123 is applied.

posix_trace_close, posix_trace_open, posix_trace_rewind

Trace log management (**TRACING**)

TRC TRL
```
#include <trace.h>

int posix_trace_close(trace_id_t trid);
int posix_trace_open(int file_desc, trace_id_t *trid);
int posix_trace_rewind(trace_id_t trid);
```

The *posix_trace_close*( ) function deallocates the trace log identifier indicated by *trid*, and all of its associated resources.

The *posix_trace_open*( ) function allocates the necessary resources and establishes the connection between a trace log identified by the *file_desc* argument and a trace stream identifier identified by the object pointed to by the *trid* argument.

The *posix_trace_rewind*( ) function resets the current trace event timestamp, which specifies the timestamp of the trace event that will be read by the next call to *posix_trace_getnext_event*( ), to the timestamp of the oldest trace event recorded in the trace log identified by *trid*.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1q-2000. These functions are part of the Tracing Option Group and may not be available on all implementations.

IEEE PASC Interpretation 1003.1 #123 is applied.

posix_trace_create, posix_trace_create_withlog, posix_trace_flush, posix_trace_shutdown

Trace stream initialization, flush, and shutdown from a process (**TRACING**)

TRC
```
#include <sys/types.h>
#include <trace.h>

int posix_trace_create(pid_t pid,
    const trace_attr_t *restrict attr,
    trace_id_t *restrict trid);
```
TRC TRL
```
int posix_trace_create_withlog(pid_t pid,
    const trace_attr_t *restrict attr, int file_desc,
```

```
    trace_id_t *restrict trid);
int posix_trace_flush(trace_id_t trid);
TRC int posix_trace_shutdown(trace_id_t trid);
```

The *posix_trace_create*() function creates an active trace stream.

The *posix_trace_create_withlog*() function is equivalent to *posix_trace_create*(), except that it associates a trace log with the stream.

The *posix_trace_flush*() function initiates a flush operation which copies the contents of the trace stream identified by the argument *trid* into the trace log associated with the trace stream at the creation time.

The *posix_trace_shutdown*() function stops the tracing of trace events in the trace stream identified by *trid*, as if *posix_trace_stop*() had been invoked.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1q-2000. These functions are part of the Tracing Option Group and may not be available on all implementations.

posix_trace_event, posix_trace_eventid_open

Trace functions for instrumenting application code (**TRACING**)

```
TRC #include <sys/types.h>
#include <trace.h>

void posix_trace_event(trace_event_id_t event_id,
    const void *restrictdata_ptr, size_t data_len);
int posix_trace_eventid_open(const char *restrict event_name,
    trace_event_id_t *restrict event_id);
```

The *posix_trace_event*() function records the *event_id* and the user data pointed to by *data_ptr* in the trace stream into which the calling process is being traced and in which *event_id* is not filtered out.

The *posix_trace_eventid_open*() function associates a user trace event name with a trace event type identifier for the calling process.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1q-2000. These functions are part of the Tracing Option Group and may not be available on all implementations.

IEEE PASC Interpretation 1003.1 #123 is applied.

IEEE PASC Interpretation 1003.1 #127 is applied, correcting some editorial errors in the names of the *posix_trace_eventid_open*() function and the *event_id* argument.

posix_trace_eventid_equal, posix_trace_eventid_get_name, posix_trace_trid_eventid_open

Manipulate trace event type identifier (**TRACING**)

TRC
```
#include <trace.h>
```

```
int posix_trace_eventid_equal(trace_id_t trid, trace_event_id_t event1,
    trace_event_id_t event2);
int posix_trace_eventid_get_name(trace_id_t trid,
    trace_event_id_t event, char *event_name);
```
TRC TEF
```
int posix_trace_trid_eventid_open(trace_id_t trid,
    const char *restrict event_name,
    trace_event_id_t *restrict event);
```

The *posix_trace_eventid_equal*() function compares the trace event type identifiers *event1* and *event2* from the same trace stream or the same trace log identified by the *trid* argument.

The *posix_trace_eventid_get_name*() function returns, in the argument pointed to by *event_name*, the trace event name associated with the trace event type identifier identified by the argument *event*, for the trace stream or for the trace log identified by the *trid* argument.

The *posix_trace_trid_eventid_open*() function associates a user trace event name with a trace event type identifier for a given trace stream.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1q-2000. These functions are part of the Tracing Option Group and may not be available on all implementations.

IEEE PASC Interpretations 1003.1 #123 and #129 are applied.

posix_trace_eventset_add, posix_trace_eventset_del, posix_trace_eventset_empty, posix_trace_eventset_fill, posix_trace_eventset_ismember

Manipulate trace event type sets (**TRACING**)

TRC TEF
```
#include <trace.h>
```

```
int posix_trace_eventset_add(trace_event_id_t event_id,
    trace_event_set_t *set);
int posix_trace_eventset_del(trace_event_id_t event_id,
    trace_event_set_t *set);
int posix_trace_eventset_empty(trace_event_set_t *set);
int posix_trace_eventset_fill(trace_event_set_t *set, int what);
int posix_trace_eventset_ismember(trace_event_id_t event_id,
    const trace_event_set_t *restrict set,
    int *restrict ismember);
```

The *posix_trace_eventset_add*() and *posix_trace_eventset_del*() functions, respectively, add or delete the individual trace event type specified by the value of the argument *event_id* to or from the trace event type set pointed to by the argument *set*.

The *posix_trace_eventset_empty*() function initializes the trace event type set pointed to by the *set* argument such that all trace event types defined, both system and user, are excluded from the set.

The *posix_trace_eventset_fill*( ) function initializes the trace event type set pointed to by the argument *set*, such that the set of trace event types defined by the argument *what* are included in the set.

The *posix_trace_eventset_ismember*( ) function tests whether the trace event type specified by the value of the argument *event_id* is a member of the set pointed to by the argument *set*.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1q-2000. These functions are part of the Tracing Option Group and may not be available on all implementations.

posix_trace_eventtypelist_getnext_id, posix_trace_eventtypelist_rewind

Iterate over a mapping of trace event types (**TRACING**)

TRC
```
#include <trace.h>

int posix_trace_eventtypelist_getnext_id(trace_id_t trid,
    trace_event_id_t *restrict event, int *restrict unavailable);
int posix_trace_eventtypelist_rewind(trace_id_t trid);
```

The first time *posix_trace_eventtypelist_getnext_id*( ) is called, the function returns in the variable pointed to by *event* the first trace event type identifier of the list of trace events of the trace stream identified by the *trid* argument. Successive calls to *posix_trace_eventtypelist_getnext_id*( ) return in the variable pointed to by *event* the next trace event type identifier in that same list.

The *posix_trace_eventtypelist_rewind*( ) function resets the next trace event type identifier to be read to the first trace event type identifier from the list of trace events used in the trace stream identified by *trid*.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1q-2000. These functions are part of the Tracing Option Group and may not be available on all implementations.

IEEE PASC Interpretations 1003.1 #123 and #129 are applied.

posix_trace_get_attr, posix_trace_get_status

Retrieve the trace attributes or trace statuses (**TRACING**)

TRC
```
#include <trace.h>

int posix_trace_get_attr(trace_id_t trid, trace_attr_t *attr);
int posix_trace_get_status(trace_id_t trid,
    struct posix_trace_status_info *statusinfo);
```

The *posix_trace_get_attr*( ) function copies the attributes of the active trace stream identified by *trid* into the object pointed to by the *attr* argument.

The *posix_trace_get_status*( ) function returns, in the structure pointed to by the *statusinfo* argument, the current trace status for the trace stream identified by the *trid* argument.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1q-2000. These functions are part of the Tracing Option Group and may not be available on all implementations.

IEEE PASC Interpretation 1003.1 #123 is applied.

posix_trace_get_filter, posix_trace_set_filter

Retrieve and set filter of an initialized trace stream (**TRACING**)

TRC TEF
```
#include <trace.h>

int posix_trace_get_filter(trace_id_t trid, trace_event_set_t *set);
int posix_trace_set_filter(trace_id_t trid,
    const trace_event_set_t *set, int how);
```

The *posix_trace_get_filter*( ) function retrieves, into the argument pointed to by *set*, the actual trace event filter from the trace stream specified by *trid*.

The *posix_trace_set_filter*( ) function changes the set of filtered trace event types after a trace stream identified by the *trid* argument is created.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1q-2000. These functions are part of the Tracing Option Group and may not be available on all implementations.

IEEE PASC Interpretation 1003.1 #123 is applied.

posix_trace_getnext_event, posix_trace_timedgetnext_event, posix_trace_trygetnext_event

Retrieve a trace event (**TRACING**)

TRC
```
#include <sys/types.h>
#include <trace.h>

int posix_trace_getnext_event(trace_id_t trid,
    struct posix_trace_event_info *restrict event,
    void *restrict data, size_t num_bytes,
    size_t *restrict data_len, int *restrict unavailable);
```
TRC TMO
```
int posix_trace_timedgetnext_event(trace_id_t trid,
    struct posix_trace_event_info *restrict event,
    void *restrict data, size_t num_bytes,
    size_t *restrict data_len, int *restrict unavailable,
    const struct timespec *restrict abs_timeout);
```
TRC
```
int posix_trace_trygetnext_event(trace_id_t trid,
    struct posix_trace_event_info *restrict event,
    void *restrict data, size_t num_bytes,
    size_t *restrict data_len, int *restrict unavailable);
```

The *posix_trace_getnext_event*( ) function reports a recorded trace event either from an active trace stream without log or a pre-recorded trace stream identified by the *trid* argument.

The *posix_trace_trygetnext_event*( ) function reports a recorded trace event from an active trace stream without log identified by the *trid* argument.

The *posix_trace_timedgetnext_event*( ) function attempts to get another trace event from an active trace stream without log, as in the *posix_trace_getnext_event*( ) function. However, if no trace event is available from the trace stream, the implied wait is terminated when the timeout specified by the argument *abs_timeout* expires, and the function returns the error [ETIMEDOUT].

Derivation First released in Issue 6. Derived from IEEE Std 1003.1q-2000. These functions are part of the Tracing Option Group and may not be available on all implementations.

IEEE PASC Interpretation 1003.1 #123 is applied.

posix_trace_start, posix_trace_stop

Trace start and stop (**TRACING**)

TRC
```
#include <trace.h>

int posix_trace_start(trace_id_t trid);
int posix_trace_stop (trace_id_t trid);
```

The *posix_trace_start*() and *posix_trace_stop*() functions, respectively, start and stop the trace stream identified by the argument *trid*.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1q-2000. These functions are part of the Tracing Option Group and may not be available on all implementations.

IEEE PASC Interpretation 1003.1 #123 is applied.

posix_typed_mem_get_info

Query typed memory information (**ADVANCED REALTIME**)

TYM
```
#include <sys/mman.h>

int posix_typed_mem_get_info(int fildes,
    struct posix_typed_mem_info *info);
```

Typed memory objects are pools of specialized storage, different from the main memory resource normally used by a processor to hold code and data, that can be mapped into the address space of one or more processes. For more information see XRAT, Section B.2.8.3.

The *posix_typed_mem_get_info*() function returns, in the *posix_tmi_length* field of the **posix_typed_mem_info** structure pointed to by *info*, the maximum length which can be successfully allocated by the typed memory object designated by *fildes*.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1j-2000. This function is part of the Advanced Realtime Option Group and may not be available on all implementations.

posix_typed_mem_open

Open a typed memory object (**ADVANCED REALTIME**)

TYM
```
#include <sys/mman.h>

int posix_typed_mem_open(const char *name, int oflag, int tflag);
```

The *posix_typed_mem_open*() function establishes a connection between the typed memory object specified by the string pointed to by *name* and a file descriptor.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1j-2000. This function is part of the Advanced Realtime Option Group and may not be available on all implementations.

pow, powf, powl

Power function

```
#include <math.h>

double pow(double x, double y);
float powf(float x, float y);
long double powl(long double x, long double y);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *powf*() and *powl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

pselect, select

Synchronous I/O multiplexing

```
#include <sys/select.h>

int pselect(int nfds, fd_set *restrict readfds,
    fd_set *restrict writefds, fd_set *restrict errorfds,
    const struct timespec *restrict timeout,
    const sigset_t *restrict sigmask);
int select(int nfds, fd_set *restrict readfds,
    fd_set *restrict writefds, fd_set *restrict errorfds,
    struct timeval *restrict timeout);
void FD_CLR(int fd, fd_set *fdset);
int FD_ISSET(int fd, fd_set *fdset);
void FD_SET(int fd, fd_set *fdset);
void FD_ZERO(fd_set *fdset);
```

The *pselect*() function examines the file descriptor sets whose addresses are passed in the *readfds*, *writefds*, and *errorfds* parameters to see whether some of their descriptors are ready for reading, are ready for writing, or have an exceptional condition pending, respectively.

The *select*() function is equivalent to the *pselect*() function, except as follows:

- For the *select*() function, the timeout period is given in seconds and microseconds in an argument of type **struct timeval**, whereas for the *pselect*() function the timeout period is given in seconds and nanoseconds in an argument of type **struct timespec**.
- The *select*() function has no *sigmask* argument; it behaves as *pselect*() does when *sigmask* is a null pointer.
- Upon successful completion, the *select*() function may modify the object pointed to by the *timeout* argument.

Derivation First released in Issue 4, Version 2.

Issue 6 The Open Group Corrigendum U026/6 is applied, changing the occurrences of *readfs* and *writefs* in the *select*() DESCRIPTION to be *readfds* and *writefds*.

Text referring to sockets is added to the DESCRIPTION.

The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are marked as part of the XSI STREAMS Option Group.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- These functions are now mandatory.

The *pselect*() function is added for alignment with IEEE Std 1003.1g-2000 and additional detail related to sockets semantics is added to the DESCRIPTION.

The *select*() function now requires inclusion of **<sys/select.h>**.

The **restrict** keyword is added to the *select*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC2/D6/70 is applied, updating the DESCRIPTION to reference the signal mask in terms of the calling thread rather than the process.

pthread_atfork

Register fork handlers

THR
```
#include <pthread.h>

int pthread_atfork(void (*prepare)(void), void (*parent)(void),
    void (*child)(void));
```

Derivation First released in Issue 5. Derived from the POSIX Threads Extension.

Issue 6 The *pthread_atfork*() function is marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

IEEE PASC Interpretation 1003.1c #4 is applied, clarifying that the function should be declared in the **<pthread.h>** header.

The **<pthread.h>** header is added to the SYNOPSIS.

pthread_attr_destroy, pthread_attr_init

Destroy and initialize threads attributes object

THR
```
#include <pthread.h>

int pthread_attr_destroy(pthread_attr_t *attr);
int pthread_attr_init(pthread_attr_t *attr);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_attr_destroy*() and *pthread_attr_init*() functions are marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

IEEE PASC Interpretation 1003.1 #107 is applied, noting that the effect of initializing an already initialized thread attributes object is undefined.

, item XSH/TC2/D6/71 is applied, updating the ERRORS section to add the optional [EINVAL] error for the *pthread_attr_destroy*( ) function, and the optional [EBUSY] error for the *pthread_attr_init*( ) function.

pthread_attr_getdetachstate, pthread_attr_setdetachstate

Get and set detachstate attribute

THR

```
#include <pthread.h>

int pthread_attr_getdetachstate(const pthread_attr_t *attr,
    int *detachstate);
int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_attr_setdetachstate*( ) and *pthread_attr_getdetachstate*( ) functions are marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

, item XSH/TC2/D6/73 is applied, updating the ERRORS section to include the optional [EINVAL] error.

pthread_attr_getguardsize, pthread_attr_setguardsize

Get and set the thread guardsize attribute

XSI

```
#include <pthread.h>

int pthread_attr_getguardsize(const pthread_attr_t *restrict attr,
    size_t *restrict guardsize);
int pthread_attr_setguardsize(pthread_attr_t *attr,
    size_t guardsize);
```

Derivation First released in Issue 5.

Issue 6 In the ERRORS section, a third [EINVAL] error condition is removed as it is covered by the second error condition.

The **restrict** keyword is added to the *pthread_attr_getguardsize*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC2/D6/74 is applied, updating the ERRORS section to remove the [EINVAL] error (“The attribute *attr* is invalid.”), and replacing it with the optional [EINVAL] error.

pthread_attr_getinheritsched, pthread_attr_setinheritsched

Get and set inheritsched attribute (**REALTIME THREADS**)

THR TPS
```
#include <pthread.h>

int pthread_attr_getinheritsched(const pthread_attr_t *restrict attr,
    int *restrict inheritsched);
int pthread_attr_setinheritsched(pthread_attr_t *attr,
    int inheritsched);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Marked as part of the Realtime Threads Feature Group.

Issue 6 The *pthread_attr_getinheritsched*() and *pthread_attr_setinheritsched*() functions are marked as part of the Threads and Thread Execution Scheduling options.

These functions are part of the Realtime Threads Option Group and may not be available on all implementations.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Thread Execution Scheduling option.

The **restrict** keyword is added to the *pthread_attr_getinheritsched*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC2/D6/75 is applied, clarifying the values of *inheritsched* in the DESCRIPTION and adding two optional [EINVAL] errors to the ERRORS section for checking when *attr* refers to an uninitialized thread attribute object.

pthread_attr_getschedparam, pthread_attr_setschedparam

Get and set schedparam attribute

THR
```
#include <pthread.h>

int pthread_attr_getschedparam(const pthread_attr_t *restrict attr,
    struct sched_param *restrict param);
int pthread_attr_setschedparam(pthread_attr_t *restrict attr,
    const struct sched_param *restrict param);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_attr_getschedparam*() and *pthread_attr_setschedparam*() functions are marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the *pthread_attr_getschedparam*() and *pthread_attr_setschedparam*() prototypes for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC2/D6/78 is applied, updating the ERRORS section to include optional errors for the case when *attr* refers to an uninitialized thread attribute

object.

pthread_attr_getschedpolicy, pthread_attr_setschedpolicy

Get and set schedpolicy attribute (**REALTIME THREADS**)

THR TPS
```
#include <pthread.h>

int pthread_attr_getschedpolicy(const pthread_attr_t *restrict attr,
    int *restrict policy);
int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Marked as part of the Realtime Threads Feature Group.

Issue 6 The *pthread_attr_getschedpolicy*() and *pthread_attr_setschedpolicy*() functions are marked as part of the Threads and Thread Execution Scheduling options.

These functions are part of the Realtime Threads Option Group and may not be available on all implementations.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Thread Execution Scheduling option.

The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the *pthread_attr_getschedpolicy*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC2/D6/80 is applied, updating the ERRORS section to include optional errors for the case when *attr* refers to an uninitialized thread attribute object.

pthread_attr_getscope, pthread_attr_setscope

Get and set contentionscope attribute (**REALTIME THREADS**)

THR TPS
```
#include <pthread.h>

int pthread_attr_getscope(const pthread_attr_t *restrict attr,
    int *restrict contentionscope);
int pthread_attr_setscope(pthread_attr_t *attr, int contentionscope);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Marked as part of the Realtime Threads Feature Group.

Issue 6 The *pthread_attr_getscope*() and *pthread_attr_setscope*() functions are marked as part of the Threads and Thread Execution Scheduling options.

These functions are part of the Realtime Threads Option Group and may not be available on all implementations.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Thread Execution Scheduling option.

The **restrict** keyword is added to the *pthread_attr_getscope*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC2/D6/82 is applied, updating the ERRORS section to include optional errors for the case when *attr* refers to an uninitialized thread attribute object.

pthread_attr_getstack, pthread_attr_setstack

Get and set stack attributes

THR
```
#include <pthread.h>
```

TSA TSS
```
int pthread_attr_getstack(const pthread_attr_t *restrict attr,
    void **restrict stackaddr, size_t *restrict stacksize);
int pthread_attr_setstack(pthread_attr_t *attr, void *stackaddr,
    size_t stacksize);
```

The *pthread_attr_getstack*() and *pthread_attr_setstack*() functions, respectively, get and set the thread creation stack attributes *stackaddr* and *stacksize* in the *attr* object.

These functions are appropriate for use by applications in an environment where the stack for a thread must be placed in some particular region of memory.

While it might seem that an application could detect stack overflow by providing a protected page outside the specified stack region, this cannot be done portably. Implementations are free to place the thread's initial stack pointer anywhere within the specified region to accommodate the machine's stack pointer behavior and allocation requirements. Furthermore, on some architectures, such as the IA-64, ''overflow'' might mean that two separate stack pointers allocated within the region will overlap somewhere in the middle of the region.

After a successful call to *pthread_attr_setstack,*() the storage area specified by the *stackaddr* parameter is under the control of the implementation, as described in Section 2.9.8, Use of Application-Managed Thread Stacks.

Derivation — First released in Issue 6. Developed as an XSI extension and brought into the BASE as a response to IEEE PASC Interpretation 1003.1 #101.

Issue 6 — , item XSH/TC2/D6/84 is applied, updating the ERRORS section to include optional errors for the case when *attr* refers to an uninitialized thread attribute object.

pthread_attr_getstackaddr, pthread_attr_setstackaddr

Get and set stackaddr attribute

THR TSA
```
#include <pthread.h>
```

OB
```
int pthread_attr_getstackaddr(const pthread_attr_t *restrict attr,
    void **restrict stackaddr);
int pthread_attr_setstackaddr(pthread_attr_t *attr, void *stackaddr);
```

The specification of the *stackaddr* attribute presents several ambiguities that make portable use of these interfaces impossible, hence they have been marked obsolescent. The description of the single address parameter as a ''stack'' does not specify a particular relationship between the address and the ''stack'' implied by that address. For example, the address may be taken as the low memory address of a buffer intended for use as a stack, or it may be taken as the address

to be used as the initial stack pointer register value for the new thread. These two are not the same except for a machine on which the stack grows ‘‘up’’ from low memory to high, and on which a ‘‘push’’ operation first stores the value in memory and then increments the stack pointer register. Further, on a machine where the stack grows ‘‘down’’ from high memory to low, interpretation of the address as the ‘‘low memory’’ address requires a determination of the intended size of the stack. The Single UNIX Specification, Version 3 introduces the new interfaces *pthread_attr_setstack*() and *pthread_attr_getstack*() to resolve these ambiguities.

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_attr_getstackaddr*() and *pthread_attr_setstackaddr*() functions are marked as part of the Threads and Thread Stack Address Attribute options.

The **restrict** keyword is added to the *pthread_attr_getstackaddr*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

These functions are marked obsolescent.

, item XSH/TC2/D6/86 is applied, updating the ERRORS section to include optional errors for the case when *attr* refers to an uninitialized thread attribute object.

pthread_attr_getstacksize, pthread_attr_setstacksize

Get and set stacksize attribute

THR TSS
```
#include <pthread.h>

int pthread_attr_getstacksize(const pthread_attr_t *restrict attr,
    size_t *restrict stacksize);
int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_attr_getstacksize*() and *pthread_attr_setstacksize*() functions are marked as part of the Threads and Thread Stack Address Attribute options. These functions are mandatory on systems supporting the Single UNIX Specification.

The **restrict** keyword is added to the *pthread_attr_getstacksize*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC2/D6/87 is applied, updating the ERRORS section to include optional errors for the case when *attr* refers to an uninitialized thread attribute object.

pthread_barrier_destroy, pthread_barrier_init

Destroy and initialize a barrier object (**ADVANCED REALTIME THREADS**)

THR BAR
```
#include <pthread.h>

int pthread_barrier_destroy(pthread_barrier_t *barrier);
int pthread_barrier_init(pthread_barrier_t *restrict barrier,
    const pthread_barrierattr_t *restrict attr, unsigned count);
```

Barriers are typically used in parallel DO/FOR loops to ensure that all threads have reached a particular stage in a parallel computation before allowing any to proceed to the next stage.

The *pthread_barrier_destroy*( ) function destroys the barrier referenced by *barrier* and releases any resources used by the barrier.

The *pthread_barrier_init*( ) function allocates any resources required to use the barrier referenced by *barrier* and initializes the barrier with attributes referenced by *attr*.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

These functions are part of the Advanced Realtime Threads Option Group and may not be available on all implementations.

pthread_barrier_wait

Synchronize at a barrier (**ADVANCED REALTIME THREADS**)

THR BAR
```
#include <pthread.h>

int pthread_barrier_wait(pthread_barrier_t *barrier);
```

The *pthread_barrier_wait*( ) function synchronizes participating threads at the barrier referenced by *barrier*.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

pthread_barrierattr_destroy, pthread_barrierattr_init

Destroy and initialize barrier attributes object (**ADVANCED REALTIME THREADS**)

THR BAR
```
#include <pthread.h>

int pthread_barrierattr_destroy(pthread_barrierattr_t *attr);
int pthread_barrierattr_init(pthread_barrierattr_t *attr);
```

The *pthread_barrierattr_destroy*( ) function destroys a barrier attributes object.

The *pthread_barrierattr_init*( ) function initializes a barrier attributes object *attr* with the default value for all of the attributes defined by the implementation.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

These functions are part of the Advanced Realtime Threads Option Group and may not be available on all implementations.

In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

pthread_barrierattr_getpshared, pthread_barrierattr_setpshared

Get and set process-shared attribute of barrier attributes object (**ADVANCED REALTIME THREADS**)

THR
```
#include <pthread.h>
```

BAR TSH
```
int pthread_barrierattr_getpshared(
    const pthread_barrierattr_t *restrict attr,
    int *restrict pshared);
int pthread_barrierattr_setpshared(pthread_barrierattr_t *attr,
```

```
    int pshared);
```

The *pthread_barrierattr_getpshared*() function obtains the value of the *process-shared* attribute from the attributes object referenced by *attr*. The *pthread_barrierattr_setpshared*() function sets the *process-shared* attribute in an initialized attributes object referenced by *attr*.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1j-2000

These functions are part of the Advanced Realtime Threads Option Group and may not be available on all implementations.

pthread_cancel

Cancel execution of a thread

THR
```
#include <pthread.h>

int pthread_cancel(pthread_t thread);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_cancel*() function is marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

pthread_cleanup_pop, pthread_cleanup_push

Establish cancelation handlers

THR
```
#include <pthread.h>

void pthread_cleanup_pop(int execute);
void pthread_cleanup_push(void (*routine)(void*), void *arg);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_cleanup_pop*() and *pthread_cleanup_push*() functions are marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

, item XSH/TC2/D6/88 is applied, updating the DESCRIPTION to describe the consequences of prematurely leaving a code block defined by the *pthread_cleanup_push*() and *pthread_cleanup_pop*() functions.

pthread_cond_broadcast, pthread_cond_signal

Broadcast or signal a condition

THR
```
#include <pthread.h>

int pthread_cond_broadcast(pthread_cond_t *cond);
int pthread_cond_signal(pthread_cond_t *cond);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_cond_broadcast*() and *pthread_cond_signal*() functions are marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

pthread_cond_destroy, pthread_cond_init

Destroy and initialize condition variables

THR
```
#include <pthread.h>

int pthread_cond_destroy(pthread_cond_t *cond);
int pthread_cond_init(pthread_cond_t *restrict cond,
    const pthread_condattr_t *restrict attr);
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_cond_destroy*() and *pthread_cond_init*() functions are marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

IEEE PASC Interpretation 1003.1c #34 is applied, updating the DESCRIPTION.

The **restrict** keyword is added to the *pthread_cond_init*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

pthread_cond_timedwait, pthread_cond_wait

Wait on a condition

THR
```
#include <pthread.h>

int pthread_cond_timedwait(pthread_cond_t *restrict cond,
    pthread_mutex_t *restrict mutex,
    const struct timespec *restrict abstime);
int pthread_cond_wait(pthread_cond_t *restrict cond,
    pthread_mutex_t *restrict mutex);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_cond_timedwait*() and *pthread_cond_wait*() functions are marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

The Open Group Corrigendum U021/9 is applied, correcting the prototype for the *pthread_cond_wait*() function from:

```
int pthread_cond_wait(pthread_cond_t *);
```

to:

```
int pthread_cond_wait(pthread_cond_t *, pthread_mutex_t *);
```

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics for the Clock Selection option.

The ERRORS section has an additional case for [EPERM] in response to IEEE PASC Interpretation 1003.1c #28.

The **restrict** keyword is added to the *pthread_cond_timedwait*( ) and *pthread_cond_wait*( ) prototypes for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC2/D6/89 is applied, updating the DESCRIPTION for consistency with the *pthread_cond_destroy*( ) function that states it is safe to destroy an initialized condition variable upon which no threads are currently blocked.

, item XSH/TC2/D6/90 is applied, updating words in the DESCRIPTION from ‘‘the cancelability enable state’’ to ‘‘the cancelability type’’.

, item XSH/TC2/D6/91 is applied, updating the ERRORS section to remove the error case related to *abstime* from the *pthread_cond_wait*( ) function, and to make the error case related to *abstime* mandatory for *pthread_cond_timedwait*( ) for consistency with other functions.

pthread_condattr_destroy, pthread_condattr_init

Destroy and initialize condition variable attributes object

THR

```
#include <pthread.h>

int pthread_condattr_destroy(pthread_condattr_t *attr);
int pthread_condattr_init(pthread_condattr_t *attr);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_condattr_destroy*( ) and *pthread_condattr_init*( ) functions are marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

pthread_condattr_getclock, pthread_condattr_setclock

Get and set the clock selection condition variable attribute (**ADVANCED REALTIME**)

THR CS

```
#include <pthread.h>

int pthread_condattr_getclock(const pthread_condattr_t *restrict attr,
    clockid_t *restrict clock_id);
int pthread_condattr_setclock(pthread_condattr_t *attr,
    clockid_t clock_id);
```

The *pthread_condattr_getclock*( ) function obtains the value of the *clock* attribute from the attributes object referenced by *attr*. The *pthread_condattr_setclock*( ) function sets the *clock* attribute in an initialized attributes object referenced by *attr*.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1j-2000. These functions are part of the Advanced Realtime Option Group and may not be available on all implementations.

pthread_condattr_getpshared, pthread_condattr_setpshared

Get and set the process-shared condition variable attributes

THR TSH
```
#include <pthread.h>

int pthread_condattr_getpshared(const pthread_condattr_t *restrict attr,
    int *restrict pshared);
int pthread_condattr_setpshared(pthread_condattr_t *attr,
    int pshared);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_condattr_getpshared*() and *pthread_condattr_setpshared*() functions are marked as part of the Threads and Thread Process-Shared Synchronization options.

The **restrict** keyword is added to the *pthread_condattr_getpshared*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

pthread_create

Thread creation

THR
```
#include <pthread.h>

int pthread_create(pthread_t *restrict thread,
    const pthread_attr_t *restrict attr,
    void *(*start_routine)(void*), void *restrict arg);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_create*() function is marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EPERM] mandatory error condition is added.

The thread CPU-time clock semantics are added for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the *pthread_create*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION is updated to make it explicit that the floating-point environment is inherited from the creating thread.

, item XSH/TC1/D6/44 is applied, adding text that the alternate stack is not inherited.

, item XSH/TC2/D6/93 is applied, updating the ERRORS section to remove the mandatory [EINVAL] error (‘‘The value specified by *attr* is invalid’’), and adding the optional [EINVAL] error (‘‘The attributes specified by *attr* are invalid’’).

pthread_detach

Detach a thread

THR
```
#include <pthread.h>

int pthread_detach(pthread_t thread);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_detach*() function is marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

, item XSH/TC2/D6/95 is applied, updating the ERRORS section so that the [EINVAL] and [ESRCH] error cases become optional.

pthread_equal

Compare thread IDs

THR
```
#include <pthread.h>

int pthread_equal(pthread_t t1, pthread_t t2);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_equal*() function is marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

pthread_exit

Thread termination

THR
```
#include <pthread.h>

void pthread_exit(void *value_ptr);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_exit*() function is marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

pthread_getconcurrency, pthread_setconcurrency

Get and set level of concurrency

XSI
```
#include <pthread.h>

int pthread_getconcurrency(void);
int pthread_setconcurrency(int new_level);
```

Derivation First released in Issue 5.

Issue 6 No functional changes are made in this issue.

pthread_getcpuclockid

Access a thread CPU-time clock (**ADVANCED REALTIME THREADS**)

THR TCT
```
#include <pthread.h>
#include <time.h>

int pthread_getcpuclockid(pthread_t thread_id, clockid_t *clock_id);
```

The *pthread_getcpuclockid*() function returns in *clock_id* the clock ID of the CPU-time clock of the thread specified by *thread_id*, if the thread specified by *thread_id* exists.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

This function is part of the Advanced Realtime Threads Option Group and may not be available on all implementations.

In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

pthread_getschedparam, pthread_setschedparam

Dynamic thread scheduling parameters access (**REALTIME THREADS**)

THR TPS
```
#include <pthread.h>

int pthread_getschedparam(pthread_t thread, int *restrict policy,
    struct sched_param *restrict param);
int pthread_setschedparam(pthread_t thread, int policy,
    const struct sched_param *param);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_getschedparam*() and *pthread_setschedparam*() functions are marked as part of the Threads and Thread Execution Scheduling options.

These functions are part of the Realtime Threads Option Group and may not be available on all implementations.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Thread Execution Scheduling option.

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 SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

The **restrict** keyword is added to the *pthread_getschedparam*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

The Open Group Corrigendum U047/1 is applied, making it clear that any temporary adjustments to priority are not reflected as a result of any priority inheritance or the ceiling functions.

IEEE PASC Interpretation 1003.1 #96 is applied, noting that priority values can also be set by a call to the *pthread_setschedprio*() function.

pthread_getspecific, pthread_setspecific

Thread-specific data management

THR
```
#include <pthread.h>

void *pthread_getspecific(pthread_key_t key);
int pthread_setspecific(pthread_key_t key, const void *value);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_getspecific*() and *pthread_setspecific*() functions are marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

IEEE PASC Interpretation 1003.1c #3 (Part 6) is applied, updating the DESCRIPTION.

, item XSH/TC2/D6/96 is applied, updating the ERRORS section so that the [ENOMEM] error case is changed from ‘‘to associate the value with the key’’ to ‘‘to associate the non-NULL value with the key’’.

pthread_join

Wait for thread termination

THR
```
#include <pthread.h>

int pthread_join(pthread_t thread, void **value_ptr);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_join*() function is marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

, item XSH/TC2/D6/97 is applied, updating the ERRORS section so that the [EINVAL] error is made optional and the words ‘‘the implementation has detected’’ are removed from it.

pthread_key_create

Thread-specific data key creation

THR
```
#include <pthread.h>

int pthread_key_create(pthread_key_t *key, void (*destructor)(void*));
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_key_create*() function is marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

IEEE PASC Interpretation 1003.1c #8 is applied, updating the DESCRIPTION.

pthread_key_delete

Thread-specific data key deletion

THR
```
#include <pthread.h>

int pthread_key_delete(pthread_key_t key);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_key_delete*() function is marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

pthread_kill

Send a signal to a thread

THR
```
#include <signal.h>

int pthread_kill(pthread_t thread, int sig);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_kill*() function is marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

pthread_mutex_destroy, pthread_mutex_init

Destroy and initialize a mutex

THR
```
#include <pthread.h>

int pthread_mutex_destroy(pthread_mutex_t *mutex);
int pthread_mutex_init(pthread_mutex_t *restrict mutex,
    const pthread_mutexattr_t *restrict attr);
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_mutex_destroy*() and *pthread_mutex_init*() functions are marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

IEEE PASC Interpretation 1003.1c #34 is applied, updating the DESCRIPTION.

The **restrict** keyword is added to the *pthread_mutex_init*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

pthread_mutex_getprioceiling, pthread_mutex_setprioceiling

Get and set the priority ceiling of a mutex (**REALTIME THREADS**)

THR TPP
```
#include <pthread.h>

int pthread_mutex_getprioceiling(const pthread_mutex_t *restrict mutex,
    int *restrict prioceiling);
int pthread_mutex_setprioceiling(pthread_mutex_t *restrict mutex,
    int prioceiling, int *restrict old_ceiling);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Marked as part of the Realtime Threads Feature Group.

Issue 6 The *pthread_mutex_getprioceiling*( ) and *pthread_mutex_setprioceiling*( ) functions are marked as part of the Threads and Thread Priority Protection options.

These functions are part of the Realtime Threads Option Group and may not be available on all implementations.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Thread Priority Protection option.

The [ENOSYS] error denoting non-support of the priority ceiling protocol for mutexes has been removed. This is since if the implementation provides the functions (regardless of whether _POSIX_PTHREAD_PRIO_PROTECT is defined), they must function as in the DESCRIPTION and therefore the priority ceiling protocol for mutexes is supported.

The **restrict** keyword is added to the *pthread_mutex_getprioceiling*( ) and *pthread_mutex_setprioceiling*( ) prototypes for alignment with the ISO/IEC 9899: 1999 standard.

pthread_mutex_lock, pthread_mutex_trylock, pthread_mutex_unlock

Lock and unlock a mutex

THR
```
#include <pthread.h>

int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_mutex_lock*( ), *pthread_mutex_trylock*( ), and *pthread_mutex_unlock*( ) functions are marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The behavior when attempting to relock a mutex is defined.

, item XSH/TC2/D6/98 is applied, updating the ERRORS section so that the [EDEADLK] error includes detection of a deadlock condition.

pthread_mutex_timedlock

Lock a mutex (**ADVANCED REALTIME**)

THR TMO
```
#include <pthread.h>
#include <time.h>

int pthread_mutex_timedlock(pthread_mutex_t *restrict mutex,
    const struct timespec *restrict abs_timeout);
```

The *pthread_mutex_timedlock*() function locks the mutex object referenced by *mutex*. If the mutex is already locked, the calling thread blocks until the mutex becomes available as in the *pthread_mutex_lock*() function. If the mutex cannot be locked without waiting for another thread to unlock the mutex, this wait terminates when the specified timeout expires.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1d-1999. This function is part of the Advanced Realtime Option Group and may not be available on all implementations.

Issue 6 , item XSH/TC2/D6/99 is applied, marking the last paragraph in the DESCRIPTION as part of the Thread Priority Inheritance option.

, item XSH/TC2/D6/100 is applied, updating the ERRORS section so that the [EDEADLK] error includes detection of a deadlock condition.

pthread_mutexattr_destroy, pthread_mutexattr_init

Destroy and initialize mutex attributes object

THR
```
#include <pthread.h>

int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);
int pthread_mutexattr_init(pthread_mutexattr_t *attr);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_mutexattr_destroy*() and *pthread_mutexattr_init*() functions are marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

IEEE PASC Interpretation 1003.1c #27 is applied, updating the ERRORS section.

pthread_mutexattr_getprioceiling, pthread_mutexattr_setprioceiling

Get and set prioceiling attribute of mutex attributes object (**REALTIME THREADS**)

THR TPP
```
#include <pthread.h>

int pthread_mutexattr_getprioceiling(
    const pthread_mutexattr_t *restrict attr,
    int *restrict prioceiling);
int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,
    int prioceiling);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Marked as part of the Realtime Threads Feature Group.

Issue 6 The *pthread_mutexattr_getprioceiling*( ) and *pthread_mutexattr_setprioceiling*( ) functions are marked as part of the Threads and Thread Priority Protection options.

These functions are part of the Realtime Threads Option Group and may not be available on all implementations.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Thread Priority Protection option.

The [ENOTSUP] error condition has been removed since these functions do not have a *protocol* argument.

The **restrict** keyword is added to the *pthread_mutexattr_getprioceiling*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

pthread_mutexattr_getprotocol, pthread_mutexattr_setprotocol

Get and set protocol attribute of mutex attributes object (**REALTIME THREADS**)

THR
```
#include <pthread.h>
```

TPP|TPI
```
int pthread_mutexattr_getprotocol(
    const pthread_mutexattr_t *restrict attr,
    int *restrict protocol);
int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,
    int protocol);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Marked as part of the Realtime Threads Feature Group.

Issue 6 The *pthread_mutexattr_getprotocol*( ) and *pthread_mutexattr_setprotocol*( ) functions are marked as part of the Threads option and either the Thread Priority Protection or Thread Priority Inheritance options.

These functions are part of the Realtime Threads Option Group and may not be available on all implementations.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Thread Priority Protection or Thread Priority Inheritance options.

The **restrict** keyword is added to the *pthread_mutexattr_getprotocol*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

pthread_mutexattr_getpshared, pthread_mutexattr_setpshared

Get and set process-shared attribute

THR TSH
```
#include <pthread.h>

int pthread_mutexattr_getpshared(
    const pthread_mutexattr_t *restrict attr,
    int *restrict pshared);
int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,
    int pshared);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_mutexattr_getpshared*() and *pthread_mutexattr_setpshared*() functions are marked as part of the Threads and Thread Process-Shared Synchronization options. These functions are mandatory on systems supporting the Single UNIX Specification.

The **restrict** keyword is added to the *pthread_mutexattr_getpshared*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

pthread_mutexattr_gettype, pthread_mutexattr_settype

Get and set a mutex type attribute

XSI
```
#include <pthread.h>

int pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict attr,
    int *restrict type);
int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type);
```

Derivation First released in Issue 5.

Issue 6 The Open Group Corrigendum U033/3 is applied. The SYNOPSIS for *pthread_mutexattr_gettype*() is updated so that the first argument is of type **const pthread_mutexattr_t ***.

The **restrict** keyword is added to the *pthread_mutexattr_gettype*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

pthread_once

Dynamic package initialization

THR
```
#include <pthread.h>

int pthread_once(pthread_once_t *once_control,
    void (*init_routine)(void));
pthread_once_t once_control = PTHREAD_ONCE_INIT;
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_once*() function is marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

The [EINVAL] error is added as a may fail case for if either argument is invalid.

pthread_rwlock_destroy, pthread_rwlock_init

Destroy and initialize a read-write lock object

THR
```
#include <pthread.h>

int pthread_rwlock_destroy(pthread_rwlock_t *rwlock);
int pthread_rwlock_init(pthread_rwlock_t *restrict rwlock,
    const pthread_rwlockattr_t *restrict attr);
```

Derivation First released in Issue 5.

Issue 6 These functions are mandatory on systems supporting the Single UNIX Specification.

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is now part of the Threads option (previously it was part of the Read-Write Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension). The initializer macro is also deleted from the SYNOPSIS.
- The DESCRIPTION is updated as follows:
  - It explicitly notes allocation of resources upon initialization of a read-write lock object.
  - A paragraph is added specifying that copies of read-write lock objects may not be used.
- An [EINVAL] error is added to the ERRORS section for *pthread_rwlock_init*( ), indicating that the *rwlock* value is invalid.

The **restrict** keyword is added to the *pthread_rwlock_init*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

pthread_rwlock_rdlock, pthread_rwlock_tryrdlock

Lock a read-write lock object for reading

THR
```
#include <pthread.h>

int pthread_rwlock_rdlock(pthread_rwlock_t *rwlock);
int pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlock);
```

Derivation First released in Issue 5.

Issue 6 These functions are mandatory on systems supporting the Single UNIX Specification.

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is now part of the Threads option (previously it was part of the Read-Write Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension).

- The DESCRIPTION is updated as follows:
  - — Conditions under which writers have precedence over readers are specified.
  - — Failure of *pthread_rwlock_tryrdlock*( ) is clarified.
  - — A paragraph on the maximum number of read locks is added.
- In the ERRORS sections, [EBUSY] is modified to take into account write priority, and [EDEADLK] is deleted as a *pthread_rwlock_tryrdlock*( ) error.

, item XSH/TC2/D6/101 is applied, updating the ERRORS section so that the [EDEADLK] error includes detection of a deadlock condition.

pthread_rwlock_timedrdlock

Lock a read-write lock for reading

THR TMO
```
#include <pthread.h>
#include <time.h>

int pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict rwlock,
    const struct timespec *restrict abs_timeout);
```

The function is part of the Advanced Realtime Option Group and need not be supported on all implementations.

The *pthread_rwlock_timedrdlock*( ) function applies a read lock to the read-write lock referenced by *rwlock* as in the *pthread_rwlock_rdlock*( ) function. However, if the lock cannot be acquired without waiting for other threads to unlock the lock, this wait terminates when the specified timeout expires.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Isseu 6 , item XSH/TC2/D6/102 is applied, updating the ERRORS section so that the [EDEADLK] error includes detection of a deadlock condition.

pthread_rwlock_timedwrlock

Lock a read-write lock for writing

THR TMO
```
#include <pthread.h>
#include <time.h>

int pthread_rwlock_timedwrlock(pthread_rwlock_t *restrict rwlock,
    const struct timespec *restrict abs_timeout);
```

The function is part of the Advanced Realtime option group and need not be supported on all implementations.

The *pthread_rwlock_timedwrlock*( ) function applies a write lock to the read-write lock referenced by *rwlock* as in the *pthread_rwlock_wrlock*( ) function. However, if the lock cannot be acquired without waiting for other threads to unlock the lock, this wait terminates when the specified timeout expires.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 6 , item XSH/TC2/D6/103 is applied, updating the ERRORS section so that the [EDEADLK] error includes detection of a deadlock condition.

pthread_rwlock_trywrlock, pthread_rwlock_wrlock

Lock a read-write lock object for writing

THR
```
#include <pthread.h>

int pthread_rwlock_trywrlock(pthread_rwlock_t *rwlock);
int pthread_rwlock_wrlock(pthread_rwlock_t *rwlock);
```

Derivation First released in Issue 5.

Issue 6 These functions are mandatory on systems supporting the Single UNIX Specification.

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is now part of the Threads option (previously it was part of the Read-Write Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension).
- The [EDEADLK] error is deleted as a *pthread_rwlock_trywrlock*() error.

, item XSH/TC2/D6/104 is applied, updating the ERRORS section so that the [EDEADLK] error includes detection of a deadlock condition.

pthread_rwlock_unlock

Unlock a read-write lock object

THR
```
#include <pthread.h>

int pthread_rwlock_unlock(pthread_rwlock_t *rwlock);
```

Derivation First released in Issue 5.

Issue 6 These functions are mandatory on systems supporting the Single UNIX Specification.

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is now part of the Threads option (previously it was part of the Read-Write Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension).
- The DESCRIPTION is updated as follows:
  - The conditions under which writers have precedence over readers are specified.
  - The concept of read-write lock owner is deleted.

pthread_rwlockattr_destroy, pthread_rwlockattr_init

Destroy and initialize read-write lock attributes object

THR

```
#include <pthread.h>

int pthread_rwlockattr_destroy(pthread_rwlockattr_t *attr);
int pthread_rwlockattr_init(pthread_rwlockattr_t *attr);
```

Derivation First released in Issue 5.

Issue 6 These functions are mandatory on systems supporting the Single UNIX Specification.

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is now part of the Threads option (previously it was part of the Read-Write Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension).

pthread_rwlockattr_getpshared, pthread_rwlockattr_setpshared

Get and set process-shared attribute of read-write lock attributes object

THR TSH

```
#include <pthread.h>

int pthread_rwlockattr_getpshared(
    const pthread_rwlockattr_t *restrict attr,
    int *restrict pshared);
int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,
    int pshared);
```

Derivation First released in Issue 5.

Issue 6 These functions are mandatory on systems supporting the Single UNIX Specification.

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- The margin code in the SYNOPSIS is changed to THR TSH to indicate that the functionality is now part of the Threads option (previously it was part of the Read-Write Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension).
- The DESCRIPTION notes that additional attributes are implementation-defined.

The **restrict** keyword is added to the *pthread_rwlockattr_getpshared*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

pthread_self

Get calling thread's ID

THR
```
#include <pthread.h>

pthread_t pthread_self(void);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_self*( ) function is marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

pthread_setcancelstate, pthread_setcanceltype, pthread_testcancel

Set cancelability state

THR
```
#include <pthread.h>

int pthread_setcancelstate(int state, int *oldstate);
int pthread_setcanceltype(int type, int *oldtype);
void pthread_testcancel(void);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6 The *pthread_setcancelstate*( ), *pthread_setcanceltype*( ), and *pthread_testcancel*( ) functions are marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

pthread_setschedprio

Dynamic thread scheduling parameters access (**REALTIME THREADS**)

THR TPS
```
#include <pthread.h>

int pthread_setschedprio(pthread_t thread, int prio);
```

The *pthread_setschedprio*( ) function sets the scheduling priority for the thread whose thread ID is given by *thread* to the value given by *prio*.

Derivation First released in Issue 6. Included as a response to IEEE PASC Interpretation 1003.1 #96. This function was added since previously there was no way for a thread to lower its own priority without going to the tail of the threads list for its new priority. This capability is required to bound the duration of the priority inversion encountered by the thread.

This function is marked as part of the Realtime Threads Option Group and may not be available on all implementations.

pthread_sigmask, sigprocmask

Examine and change blocked signals

```
#include <signal.h>

THR  int pthread_sigmask(int how, const sigset_t *restrict set,
         sigset_t *restrict oset);
CX   int sigprocmask(int how, const sigset_t *restrict set,
         sigset_t *restrict oset);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 The *pthread_sigmask*( ) function is marked as part of the Threads option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

The SYNOPSIS for *sigprocmask*( ) is marked as a CX extension to note that the presence of this function in the **<signal.h>** header is an extension to the ISO C standard.

The following changes are made for alignment with ISO/IEC 9945-1: 1996 (POSIX-1):

- The DESCRIPTION is updated to explicitly state the functions which may generate the signal.

The **restrict** keyword is added to the *pthread_sigmask*( ) and *sigprocmask*( ) prototypes for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC2/D6/105 is applied, updating ‘‘process’ signal mask’’ to ‘‘thread’s signal mask’’ in the DESCRIPTION and RATIONALE sections.

pthread_spin_destroy, pthread_spin_init

Destroy or initialize a spin lock object (**ADVANCED REALTIME THREADS**)

```
THR SPI #include <pthread.h>

int pthread_spin_destroy(pthread_spinlock_t *lock);
int pthread_spin_init(pthread_spinlock_t *lock, int pshared);
```

Spin locks represent an extremely low-level synchronization mechanism suitable primarily for use on shared memory multi-processors. It is typically an atomically modified Boolean value that is set to one when the lock is held and to zero when the lock is freed.

When a caller requests a spin lock that is already held, it typically spins in a loop testing whether the lock has become available. Such spinning wastes processor cycles so the lock should only be held for short durations and not across sleep/block operations. Callers should unlock spin locks before calling sleep operations.

The *pthread_spin_destroy*( ) function destroys the spin lock referenced by *lock* and release any resources used by the lock.

The *pthread_spin_init*( ) function allocates any resources required to use the spin lock referenced by *lock* and initializes the lock to an unlocked state.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

These functions are part of the Advanced Realtime Threads Option Group and may not be available on all implementations.

In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

pthread_spin_lock, pthread_spin_trylock

Lock a spin lock object (**ADVANCED REALTIME THREADS**)

THR SPI
```
#include <pthread.h>

int pthread_spin_lock(pthread_spinlock_t *lock);
int pthread_spin_trylock(pthread_spinlock_t *lock);
```

The *pthread_spin_lock*( ) function locks the spin lock referenced by *lock*.

The *pthread_spin_trylock*( ) function locks the spin lock referenced by *lock* if it is not held by any thread. Otherwise, the function fails.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

These functions are part of the Advanced Realtime Threads Option Group and may not be available on all implementations.

In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

, item XSH/TC2/D6/107 is applied, updating the ERRORS section so that the [EDEADLK] error includes detection of a deadlock condition.

pthread_spin_unlock

Unlock a spin lock object (**ADVANCED REALTIME THREADS**)

THR SPI
```
#include <pthread.h>

int pthread_spin_unlock(pthread_spinlock_t *lock);
```

The *pthread_spin_unlock*( ) function releases the spin lock referenced by *lock* which was locked via the *pthread_spin_lock*( ) or *pthread_spin_trylock*( ) functions.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

These functions are part of the Advanced Realtime Threads Option Group and may not be available on all implementations.

In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

ptsname

Get name of the slave pseudo-terminal device

XSI
```
#include <stdlib.h>

char *ptsname(int fildes);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

putc

Put byte on a stream

```
#include <stdio.h>

int putc(int c, FILE *stream);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

putchar

Put byte on stdout stream

```
#include <stdio.h>

int putchar(int c);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

putenv

Change or add a value to environment

XSI

```
#include <stdlib.h>

int putenv(char *string);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

putmsg, putpmsg

Send a message on a STREAM (**STREAMS**)

XSR

```
#include <stropts.h>

int putmsg(int fildes, const struct strbuf *ctlptr,
    const struct strbuf *dataptr, int flags);
int putpmsg(int fildes, const struct strbuf *ctlptr,
    const struct strbuf *dataptr, int band, int flags);
```

Derivation First released in Issue 4, Version 2.

Issue 6 This function is marked as part of the XSI STREAMS Option Group and need not be supported on all implementations supporting the Single UNIX Specification.

puts

Put a string on standard output

```
#include <stdio.h>

int puts(const char *s);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

putwc

Put a wide character on a stream

```
#include <stdio.h>
#include <wchar.h>

wint_t putwc(wchar_t wc, FILE *stream);
```

Derivation First released as a World-wide Portability Interface in Issue 4.

Issue 6 No functional changes are made in this issue.

putwchar

Put a wide character on stdout stream

```
#include <wchar.h>

wint_t putwchar(wchar_t wc);
```

Derivation First released in Issue 4.

Issue 6 No functional changes are made in this issue.

qsort

Sort a table of data

```
#include <stdlib.h>

void qsort(void *base, size_t nel, size_t width,
    int (*compar)(const void *, const void *));
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 , item XSH/TC1/D6/49 is applied, adding the last sentence to the first non-shaded paragraph in the DESCRIPTION, and the following two paragraphs. These changes are for alignment with the ISO C standard.

raise

Send a signal to the executing process

```
#include <signal.h>

int raise(int sig);
```

Derivation First released in Issue 4. Derived from ANSI standard X3.159-1989, Programming Language C (ANSI C).

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE section, the requirement to set *errno* on error is added.
- The [EINVAL] error condition is added.

rand, rand_r, srand

Pseudo-random number generator

```
#include <stdlib.h>

int rand(void);
TSF int rand_r(unsigned *seed);
void srand(unsigned seed);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *rand_r*() function is marked as part of the Thread-Safe Functions option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

pread, read

Read from a file

```
#include <unistd.h>

XSI ssize_t pread(int fildes, void *buf, size_t nbyte, off_t offset);
ssize_t read(int fildes, void *buf, size_t nbyte);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are marked as part of the XSI STREAMS Option Group.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION now states that if *read*() is interrupted by a signal after it has successfully read some data, it returns the number of bytes read. In Issue 3, it was optional whether *read*() returned the number of bytes read, or whether it returned –1 with *errno* set to [EINTR]. This is a FIPS requirement.
- In the DESCRIPTION, text is added to indicate that for regular files, no data transfer occurs past the offset maximum established in the open file description associated with *fildes*. This change is to support large files.
- The [EOVERFLOW] mandatory error condition is added.
- The [ENXIO] optional error condition is added.

Text referring to sockets is added to the DESCRIPTION.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The effect of reading zero bytes is clarified.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that *read*() results are unspecified for typed memory objects.

New RATIONALE is added to explain the atomicity requirements for input and output operations.

The following error conditions are added for operations on sockets: [EAGAIN], [ECONNRESET], [ENOTCONN], and [ETIMEDOUT].

The [EIO] error is changed to ‘‘may fail’’.

The following error conditions are added for operations on sockets: [ENOBUFS] and [ENOMEM].

, item XSH/TC2/D6/108 is applied, updating the [EAGAIN] error in the ERRORS section from ‘‘the process would be delayed’’ to ‘‘the thread would be delayed’’.

readdir, readdir_r

Read directory

```
#include <dirent.h>

struct dirent *readdir(DIR *dirp);
TSF int readdir_r(DIR *restrict dirp, struct dirent *restrict entry,
        struct dirent **restrict result);
```

Derivation First released in Issue 2.

Issue 6 The *readdir_r*( ) function is marked as part of the Thread-Safe Functions option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

The Open Group Corrigendum U026/7 is applied, correcting the prototype for *readdir_r*( ).

The Open Group Corrigendum U026/8 is applied, clarifying the wording of the successful return for the *readdir_r*( ) function.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- A statement is added to the DESCRIPTION indicating the disposition of certain fields in **struct dirent** when an entry refers to a symbolic link.
- The [EOVERFLOW] mandatory error condition is added. This change is to support large files.
- The [ENOENT] optional error condition is added.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

The **restrict** keyword is added to the *readdir_r*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

readlink

Read the contents of a symbolic link

```
#include <unistd.h>

ssize_t readlink(const char *restrict path, char *restrict buf,
    size_t bufsize);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The return type is changed to **ssize_t**, to align with the IEEE P1003.1a draft standard.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- This function is made mandatory.
- In this function it is possible for the return value to exceed the range of the type **ssize_t** (since **size_t** has a larger range of positive values than **ssize_t**). A sentence restricting the size of the **size_t** object is added to the description to resolve this conflict.

The following changes are made for alignment with ISO/IEC 9945-1: 1996 (POSIX-1):

- The FUTURE DIRECTIONS section is changed to None.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

The **restrict** keyword is added to the *readlink*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

readv

Read a vector

XSI

```
#include <sys/uio.h>

ssize_t readv(int fildes, const struct iovec *iov, int iovcnt);
```

Derivation First released in Issue 4, Version 2.

Issue 6 Split out from the *read*( ) reference page.

realloc

Memory reallocator

```
#include <stdlib.h>

void *realloc(void *ptr, size_t size);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE section, if there is not enough available memory, the setting of *errno* to [ENOMEM] is added.

- The [ENOMEM] error condition is added.

realpath

Resolve a pathname

XSI
```
#include <stdlib.h>

char *realpath(const char *restrict file_name,
    char *restrict resolved_name);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The **restrict** keyword is added to the *realpath*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

, item XSH/TC1/D6/51 is applied, adding new text to the DESCRIPTION for the case when *resolved_name* is a null pointer, and changing the [EINVAL] error text.

recv

Receive a message from a connected socket

```
#include <sys/socket.h>

ssize_t recv(int socket, void *buffer, size_t length, int flags);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 No functional changes since the XNS, Issue 5.2 specification.

recvfrom

Receive a message from a socket

```
#include <sys/socket.h>

ssize_t recvfrom(int socket, void *restrict buffer, size_t length,
    int flags, struct sockaddr *restrict address,
    socklen_t *restrict address_len);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

recvmsg

Receive a message from a socket

```
#include <sys/socket.h>

ssize_t recvmsg(int socket, struct msghdr *message, int flags);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

regcomp, regerror, regexec, regfree

Regular expression matching

```
#include <regex.h>

int regcomp(regex_t *restrict preg, const char *restrict pattern,
    int cflags);
size_t regerror(int errcode, const regex_t *restrict preg,
    char *restrict errbuf, size_t errbuf_size);
int regexec(const regex_t *restrict preg, const char *restrict string,
    size_t nmatch, regmatch_t pmatch[restrict], int eflags);
void regfree(regex_t *preg);
```

Derivation First released in Issue 4. Derived from ISO/IEC 9945-2: 1993 (POSIX-2).

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The REG_ENOSYS constant is removed.

The **restrict** keyword is added to the *regcomp*(), *regerror*(), and *regexec*() prototypes for alignment with the ISO/IEC 9899: 1999 standard.

remainder, remainderf, remainderl

Remainder function

```
#include <math.h>

double remainder(double x, double y);
float remainderf(float x, float y);
long double remainderl(long double x, long double y);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The *remainder*() function is no longer marked as an extension.

The *remainderf*() and *remainderl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

remove

Remove a file

```
#include <stdio.h>

int remove(const char *path);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1) and the ISO C standard.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION, RETURN VALUE, and ERRORS sections are updated so that if *path* is not a directory, *remove*() is equivalent to *unlink*(), and if it is a directory, it is equivalent to *rmdir*().

remquo, remquof, remquol

Remainder functions

```
#include <math.h>

double remquo(double x, double y, int *quo);
float remquof(float x, float y, int *quo);
long double remquol(long double x, long double y, int *quo);
```

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

rename

Rename a file

```
#include <stdio.h>

int rename(const char *old, const char *new);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EIO] mandatory error condition is added.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.
- The [ETXTBSY] optional error condition is added.

The following changes were made to align with the IEEE P1003.1a draft standard:

- Details are added regarding the treatment of symbolic links.
- The [ELOOP] optional error condition is added.

rewind

Reset file position indicator in a stream

```
#include <stdio.h>

void rewind(FILE *stream);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

rewinddir

Reset position of directory stream to the beginning of a directory

```
#include <dirent.h>

void rewinddir(DIR *dirp);
```

Derivation First released in Issue 2.

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

rindex

Character string operations (**LEGACY**)

XSI
```
#include <strings.h>

char *rindex(const char *s, int c);
```

Derivation First released in Issue 4, Version 2.

Issue 6 This function is marked LEGACY and may not be available on all implementations.

The *strrchr*( ) function is preferred over this function.

For maximum portability, it is recommended to replace the function call to *rindex*( ) as follows:

```
#define rindex(a,b) strrchr((a),(b))
```

rint, rintf, rintl

Round-to-nearest integral value

```
#include <math.h>

double rint(double x);
float rintf(float x);
long double rintl(long double x);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The *rintf*( ) and *rintl*( ) functions are added.
- The *rint*( ) function is no longer marked as an extension.
- The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

  IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

rmdir

Remove a directory

```
#include <unistd.h>

int rmdir(const char *path);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to indicate the results of naming a symbolic link in *path*.
- The [EIO] mandatory error condition is added.

- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

round, roundf, roundl

Round to nearest integer value in floating-point format

```
#include <math.h>

double round(double x);
float roundf(float x);
long double roundl(long double x);
```

These functions round their argument to the nearest integer value in floating-point format, rounding halfway cases away from zero, regardless of the current rounding direction.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

scalb

Load exponent of a radix-independent floating-point number

OB XSI

```
#include <math.h>

double scalb(double x, double n);
```

Derivation First released in Issue 4, Version 2.

Issue 6 This function is marked obsolescent.

Although this function is not part of the ISO/IEC 9899: 1999 standard, the RETURN VALUE and ERROR sections are updated to align with the error handling in the ISO/IEC 9899: 1999 standard.

scalbln, scalblnf, scalblnl, scalbn, scalbnf, scalbnl

Compute exponent using FLT_RADIX

```
#include <math.h>

double scalbln(double x, long n);
float scalblnf(float x, long n);
long double scalblnl(long double x, long n);
double scalbn(double x, int n);
float scalbnf(float x, int n);
long double scalbnl(long double x, int n);
```

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

sched_get_priority_max, sched_get_priority_min

Get priority limits (**REALTIME**)

PS|TPS
```
#include <sched.h>

int sched_get_priority_max(int policy);
int sched_get_priority_min(int policy);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 These functions are marked as part of the Process Scheduling option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Process Scheduling option.

The [ESRCH] error condition has been removed since these functions do not take a *pid* argument.

sched_getparam

Get scheduling parameters (**REALTIME**)

PS
```
#include <sched.h>

int sched_getparam(pid_t pid, struct sched_param *param);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *sched_getparam*() function is marked as part of the Process Scheduling option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Process Scheduling option.

sched_getscheduler

Get scheduling policy (**REALTIME**)

PS
```
#include <sched.h>

int sched_getscheduler(pid_t pid);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *sched_getscheduler*() function is marked as part of the Process Scheduling option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Process Scheduling option.

sched_rr_get_interval

Get execution time limits (**REALTIME**)

PS|TPS
```
#include <sched.h>

int sched_rr_get_interval(pid_t pid, struct timespec *interval);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *sched_rr_get_interval*() function is marked as part of the Process Scheduling option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Process Scheduling option.

sched_setparam

Set scheduling parameters (**REALTIME**)

PS
```
#include <sched.h>

int sched_setparam(pid_t pid, const struct sched_param *param);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *sched_setparam*() function is marked as part of the Process Scheduling option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Process Scheduling option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, the effect of this function on a thread's scheduling parameters is added.
- Sections describing two-level scheduling and atomicity of the function are added.

The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

IEEE PASC Interpretation 1003.1 #100 is applied, changing text about the target process in the DESCRIPTION to be described in terms of the tail of the thread list for its priority.

sched_setscheduler

Set scheduling policy and parameters (**REALTIME**)

PS
```
#include <sched.h>

int sched_setscheduler(pid_t pid, int policy,
    const struct sched_param *param);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *sched_setscheduler*() function is marked as part of the Process Scheduling option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Process Scheduling option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, the effect of this function on a thread's scheduling parameters is added.
- Sections describing two-level scheduling and atomicity of the function are added.

The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

sched_yield

Yield processor

PS|THR
```
#include <sched.h>

int sched_yield(void);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Issue 6 The *sched_yield*() function is marked as part of the Process Scheduling and Threads options.

seekdir

Set position of directory stream

XSI
```
#include <dirent.h>

void seekdir(DIR *dirp, long loc);
```

Derivation First released in Issue 2.

Issue 6 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

sem_close

Close a named semaphore (**REALTIME**)

SEM

```
#include <semaphore.h>

int sem_close(sem_t *sem);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *sem_close*( ) function is marked as part of the Semaphores option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Semaphores option.

, item XSH/TC2/D6/113 is applied, updating the ERRORS section so that the [EINVAL] error becomes optional.

sem_destroy

Destroy an unnamed semaphore (**REALTIME**)

SEM

```
#include <semaphore.h>

int sem_destroy(sem_t *sem);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *sem_destroy*( ) function is marked as part of the Semaphores option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Semaphores option.

, item XSH/TC2/D6/113 is applied, updating the ERRORS section so that the [EINVAL] error becomes optional.

sem_getvalue

Get the value of a semaphore (**REALTIME**)

SEM

```
#include <semaphore.h>

int sem_getvalue(sem_t *restrict sem, int *restrict sval);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *sem_getvalue*( ) function is marked as part of the Semaphores option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Semaphores option.

The **restrict** keyword is added to the *sem_getvalue*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC2/D6/115 is applied, updating the ERRORS section so that the [EINVAL] error becomes optional.

sem_init

Initialize an unnamed semaphore (**REALTIME**)

SEM
```
#include <semaphore.h>

int sem_init(sem_t *sem, int pshared, unsigned value);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *sem_init*( ) function is marked as part of the Semaphores option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Semaphores option.

, item XSH/TC2/D6/116 is applied, updating the DESCRIPTION to add the *sem_timedwait*( ) function for alignment with IEEE Std 1003.1d-1999.

sem_open

Initialize and open a named semaphore (**REALTIME**)

SEM
```
#include <semaphore.h>

sem_t *sem_open(const char *name, int oflag, ...);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *sem_open*( ) function is marked as part of the Semaphores option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Semaphores option.

, item XSH/TC2/D6/118 is applied, updating the DESCRIPTION to describe the conditions to return the same semaphore address on a call to *sem_open*( ). The words ‘‘and at least one previous successful *sem_open*( ) call for this semaphore has not been matched with a *sem_close*( ) call’’ are added.

sem_post

Unlock a semaphore (**REALTIME**)

SEM
```
#include <semaphore.h>

int sem_post(sem_t *sem);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *sem_post*( ) function is marked as part of the Semaphores option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Semaphores option.

SCHED_SPORADIC is added to the list of scheduling policies for which the thread that is to be unblocked is specified for alignment with IEEE Std 1003.1d-1999.

, item XSH/TC2/D6/119 is applied, updating the ERRORS section so that the [EINVAL] error becomes optional.

sem_timedwait

Lock a semaphore (**ADVANCED REALTIME**)

SEM TMO
```
#include <semaphore.h>
#include <time.h>

int sem_timedwait(sem_t *restrict sem,
    const struct timespec *restrict abs_timeout);
```

The *sem_timedwait*() function locks the semaphore referenced by *sem* as in the *sem_wait*() function. However, if the semaphore cannot be locked without waiting for another process or thread to unlock the semaphore by performing a *sem_post*() function, this wait is terminated when the specified timeout expires.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1d-1999. These functions are part of the Advanced Realtime Option Group and may not be available on all implementations.

Issue 6 , item XSH/TC2/D6/120 is applied, updating the ERRORS section so that the [EINVAL] error becomes optional.

sem_trywait, sem_wait

Lock a semaphore (**REALTIME**)

SEM
```
#include <semaphore.h>

int sem_trywait(sem_t *sem);
int sem_wait(sem_t *sem);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *sem_trywait*() and *sem_wait*() functions are marked as part of the Semaphores option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Semaphores option.

, item XSH/TC2/D6/121 is applied, updating the ERRORS section so that the [EINVAL] error becomes optional.

sem_unlink

Remove a named semaphore (**REALTIME**)

SEM
```
#include <semaphore.h>

int sem_unlink(const char *name);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *sem_unlink*() function is marked as part of the Semaphores option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Semaphores option.

semctl

XSI semaphore control operations

XSI
```
#include <sys/sem.h>

int semctl(int semid, int semnum, int cmd, ...);
```

Derivation First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 6 No functional changes are made in this issue.

semget

Get set of XSI semaphores

XSI
```
#include <sys/sem.h>

int semget(key_t key, int nsems, int semflg);
```

Derivation First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 6 , item XSH/TC2/D6/122 is applied, updating the DESCRIPTION from ‘‘each semaphore in the set shall not be initialized’’ to ‘‘each semaphore in the set need not be initialized’’.

semop

XSI semaphore operations

XSI
```
#include <sys/sem.h>

int semop(int semid, struct sembuf *sops, size_t nsops);
```

Derivation First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 6 No functional changes are made in this issue.

send

Send a message on a socket

```
#include <sys/socket.h>

ssize_t send(int socket, const void *buffer, size_t length, int flags);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

sendmsg

Send a message on a socket using a message structure

```
#include <sys/socket.h>

ssize_t sendmsg(int socket, const struct msghdr *message, int flags);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

sendto

Send a message on a socket

```
#include <sys/socket.h>

ssize_t sendto(int socket, const void *message, size_t length,
    int flags, const struct sockaddr *dest_addr,
    socklen_t dest_len);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

setbuf

Assign buffering to a stream

```
#include <stdio.h>

void setbuf(FILE *restrict stream, char *restrict buf);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The prototype for *setbuf*() is updated for alignment with the ISO/IEC 9899: 1999 standard.

setegid

Set effective group ID

```
#include <unistd.h>

int setegid(gid_t gid);
```

If *gid* is equal to the real group ID or the saved set-group-ID, or if the process has appropriate privileges, *setegid*() sets the effective group ID of the calling process to *gid*; the real group ID, saved set-group-ID, and any supplementary group IDs remain unchanged, and the supplementary group list is not affected in any way.

Derivation First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

setenv

Add or change environment variable

CX
```
#include <stdlib.h>

int setenv(const char *envname, const char *envval, int overwrite);
```

The *setenv*() function updates or adds a variable in the environment of the calling process.

Derivation First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

seteuid

Set effective user ID

```
#include <unistd.h>

int seteuid(uid_t uid);
```

If *uid* is equal to the real user ID or the saved set-user-ID, or if the process has appropriate privileges, *seteuid*() sets the effective user ID of the calling process to *uid*; the real user ID and saved set-user-ID remain unchanged and the supplementary group list is not affected in any way.

Derivation First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

setgid

Set-group-ID

```
#include <unistd.h>

int setgid(gid_t gid);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- Functionality associated with _POSIX_SAVED_IDS is now mandated. This is a FIPS requirement.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The effects of *setgid*() in processes without appropriate privileges are changed.
- A requirement that the supplementary group list is not affected is added.

setjmp

Set jump point for a non-local goto

```
#include <setjmp.h>

int setjmp(jmp_buf env);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

setkey

Set encoding key (**CRYPT**)

XSI

```
#include <stdlib.h>

void setkey(const char *key);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

setlocale

Set program locale

```
#include <locale.h>

char *setlocale(int category, const char *locale);
```

Derivation First released in Issue 3.

Issue 6 , item XSH/TC2/D6/124 is applied, updating the DESCRIPTION to clarify the behavior of:

```
setlocale(LC_ALL, "");
```

setpgid

Set process group ID for job control

```
#include <unistd.h>

int setpgid(pid_t pid, pid_t pgid);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The *setpgid*() function is mandatory since _POSIX_JOB_CONTROL is required to be defined in this issue. This is a FIPS requirement.

, item XSH/TC1/D6/56 is applied, changing the wording in the DESCRIPTION from ‘‘the process group ID of the indicated process shall be used’’ to ‘‘the process ID of the indicated process shall be used’’. This change reverts the wording to as in ISO/IEC 9945-1: 1996 (POSIX-1); it appeared to be an unintentional change.

setpgrp

Set process group ID

XSI
```
#include <unistd.h>

pid_t setpgrp(void);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

setregid

Set real and effective group IDs

XSI
```
#include <unistd.h>

int setregid(gid_t rgid, gid_t egid);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

setreuid

Set real and effective user IDs

XSI
```
#include <unistd.h>

int setreuid(uid_t ruid, uid_t euid);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

setsid

Create session and set process group ID

```
#include <unistd.h>

pid_t setsid(void);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

setsockopt

Set the socket options

```
#include <sys/socket.h>

int setsockopt(int socket, int level, int option_name,
    const void *option_value, socklen_t option_len);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 , item XSH/TC2/D6/125 is applied, updating the SO_LINGER option in the DESCRIPTION to refer to the calling thread rather than the process.

setuid

Set user ID

```
#include <unistd.h>

int setuid(uid_t uid);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The functionality associated with _POSIX_SAVED_IDS is now mandatory. This is a FIPS requirement.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The effects of *setuid*() in processes without appropriate privileges are changed.
- A requirement that the supplementary group list is not affected is added.

setvbuf

Assign buffering to a stream

```
#include <stdio.h>

int setvbuf(FILE *restrict stream, char *restrict buf, int type,
    size_t size);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *setvbuf*() prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

shm_open

Open a shared memory object (**REALTIME**)

SHM

```
#include <sys/mman.h>

int shm_open(const char *name, int oflag, mode_t mode);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *shm_open*() function is marked as part of the Shared Memory Objects option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Shared Memory Objects option.

shm_unlink

Remove a shared memory object (**REALTIME**)

SHM
```
#include <sys/mman.h>

int shm_unlink(const char *name);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *shm_unlink*() function is marked as part of the Shared Memory Objects option.

In the DESCRIPTION, text is added to clarify that reusing the same name after a *shm_unlink*() will not attach to the old shared memory object.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Shared Memory Objects option.

shmat

XSI shared memory attach operation

XSI
```
#include <sys/shm.h>

void *shmat(int shmid, const void *shmaddr, int shmflg);
```

Derivation First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 6 The Open Group Corrigendum U021/13 is applied, changing the expression:

```
(shmaddr-((ptrdiff_t)shmaddr%SHMLBA))
```

to:

```
(shmaddr-((uintptr_t)shmaddr%SHMLBA))
```

shmctl

XSI shared memory control operations

XSI
```
#include <sys/shm.h>

int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

Derivation First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 6 No functional changes are made in this issue.

shmdt

XSI shared memory detach operation

XSI
```
#include <sys/shm.h>

int shmdt(const void *shmaddr);
```

Derivation First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 6 No functional changes are made in this issue.

shmget

Get XSI shared memory segment

XSI
```
#include <sys/shm.h>

int shmget(key_t key, size_t size, int shmflg);
```

Derivation First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 6 No functional changes are made in this issue.

shutdown

Shut down socket send and receive operations

```
#include <sys/socket.h>

int shutdown(int socket, int how);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

sigaction

Examine and change signal action

CX
```
#include <signal.h>

int sigaction(int sig, const struct sigaction *restrict act,
    struct sigaction *restrict oact);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 The Open Group Corrigendum U028/7 is applied. In the paragraph entitled ‘‘Signal Effects on Other Functions’’, a reference to *sigpending*( ) is added.

In the DESCRIPTION, the text ‘‘Signal Generation and Delivery’’, ‘‘Signal Actions’’, and ‘‘Signal Effects on Other Functions’’ are moved to a separate section.

Text describing functionality from the Realtime Signals option is marked.

The following changes are made for alignment with ISO/IEC 9945-1: 1996 (POSIX-1):

- The [ENOTSUP] error condition is added.

The **restrict** keyword is added to the *sigaction*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

References to the *wait3*( ) function are removed.

The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an extension over the ISO C standard.

sigaddset

Add a signal to a signal set

CX
```
#include <signal.h>

int sigaddset(sigset_t *set, int signo);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an extension over the ISO C standard.

sigaltstack

Set and get signal alternate stack context

XSI
```
#include <signal.h>

int sigaltstack(const stack_t *restrict ss, stack_t *restrict oss);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The **restrict** keyword is added to the *sigaltstack*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC1/D6/58 is applied, updating the first sentence to include ‘‘for the current thread’’.

sigdelset

Delete a signal from a signal set

CX
```
#include <signal.h>

int sigdelset(sigset_t *set, int signo);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an extension over the ISO C standard.

sigemptyset

Initialize and empty a signal set

CX
```
#include <signal.h>

int sigemptyset(sigset_t *set);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an extension over the ISO C standard.

sigfillset

Initialize and fill a signal set

CX
```
#include <signal.h>

int sigfillset(sigset_t *set);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an extension over the ISO C standard.

sighold, sigignore, sigpause, sigrelse, sigset

Signal management

XSI
```
#include <signal.h>

int sighold(int sig);
int sigignore(int sig);
int sigpause(int sig);
int sigrelse(int sig);
void (*sigset(int sig, void (*disp)(int)))(int);
```

Derivation First released in Issue 4, Version 2.

Issue 6 References to the *wait3*( ) function are removed.

The XSI functions are split out into their own reference page.

siginterrupt

Allow signals to interrupt functions

XSI
```
#include <signal.h>

int siginterrupt(int sig, int flag);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

sigismember

Test for a signal in a signal set

CX
```
#include <signal.h>

int sigismember(const sigset_t *set, int signo);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an extension over the ISO C standard.

siglongjmp

Non-local goto with signal handling

CX
```
#include <setjmp.h>

void siglongjmp(sigjmp_buf env, int val);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 The DESCRIPTION is rewritten in terms of *longjmp*().

The SYNOPSIS is marked CX since the presence of this function in the **<setjmp.h>** header is an extension over the ISO C standard.

signal

Signal management

```
#include <signal.h>

void (*signal(int sig, void (*func)(int)))(int);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The DESCRIPTION is updated for alignment with the ISO/IEC 9899: 1999 standard.

References to the *wait3*() function are removed.

The *sighold*(), *sigignore*(), *sigrelse*(), and *sigset*() functions are split out onto their own reference page.

signbit

Test sign

```
#include <math.h>

int signbit(real-floating x);
```

The *signbit*() macro determines whether the sign of its argument value is negative.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

sigpending

Examine pending signals

CX
```
#include <signal.h>

int sigpending(sigset_t *set);
```

Derivation First released in Issue 3.

Issue 6 The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an extension over the ISO C standard.

sigqueue

Queue a signal to a process (**REALTIME**)

RTS
```
#include <signal.h>

int sigqueue(pid_t pid, int signo, const union sigval value);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Issue 6 The *sigqueue*() function is marked as part of the Realtime Signals Extension option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Realtime Signals Extension option.

sigsetjmp

Set jump point for a non-local goto

CX
```
#include <setjmp.h>

int sigsetjmp(sigjmp_buf env, int savemask);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 The DESCRIPTION is reworded in terms of *setjmp*().

The SYNOPSIS is marked CX since the presence of this function in the **<setjmp.h>** header is an extension over the ISO C standard.

sigsuspend

Wait for a signal

CX
```
#include <signal.h>

int sigsuspend(const sigset_t *sigmask);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 The text in the RETURN VALUE section has been changed from ‘‘suspends process execution’’ to ‘‘suspends thread execution’’. This reflects IEEE PASC Interpretation 1003.1c #40.

Text in the APPLICATION USAGE section has been replaced.

The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an extension over the ISO C standard.

sigtimedwait, sigwaitinfo

Wait for queued signals (**REALTIME**)

RTS
```
#include <signal.h>

int sigtimedwait(const sigset_t *restrict set,
    siginfo_t *restrict info,
    const struct timespec *restrict timeout);
int sigwaitinfo(const sigset_t *restrict set,
    siginfo_t *restrict info);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Issue 6 These functions are marked as part of the Realtime Signals Extension option.

The Open Group Corrigendum U035/3 is applied. The SYNOPSIS of the *sigwaitinfo*( ) function has been corrected so that the second argument is of type **siginfo_t ***.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Realtime Signals Extension option.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that the CLOCK_MONOTONIC clock, if supported, is used to measure timeout intervals.

The **restrict** keyword is added to the *sigtimedwait*( ) and *sigwaitinfo*( ) prototypes for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC2/D6/130 is applied, restoring wording in the RETURN VALUE section to that in the original base document (‘‘An implementation should only check for this error if no signal is pending in *set* and it is necessary to wait’’).

sigwait

Wait for queued signals

CX
```
#include <signal.h>

int sigwait(const sigset_t *restrict set, int *restrict sig);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Issue 6 The **restrict** keyword is added to the *sigwait*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC2/D6/131 is applied, updating the DESCRIPTION section to state that if more than a single thread is blocked in *sigwait*( ), it is unspecified which of the waiting threads returns, and that if a signal is generated for a specific thread only that thread shall return.

sin, sinf, sinl

Sine function

```
#include <math.h>

double sin(double x);
float sinf(float x);
long double sinl(long double x);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *sinf*() and *sinl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

sinh, sinhf, sinhl

Hyperbolic sine function

```
#include <math.h>

double sinh(double x);
float sinhf(float x);
long double sinhl(long double x);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *sinhf*() and *sinhl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

sleep

Suspend execution for an interval of time

```
#include <unistd.h>

unsigned sleep(unsigned seconds);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

sockatmark

Determine whether a socket is at the out-of-band mark

```
#include <sys/socket.h>

int sockatmark(int s);
```

The *sockatmark*() function determines whether the socket specified by the descriptor *s* is at the out-of-band data mark .

The *sockatmark*() function replaces the historical SIOCATMARK command to *ioctl*() which implemented the same functionality on many implementations. Using a wrapper function follows the adopted conventions to avoid specifying commands to the *ioctl*() function, other than those now included to support XSI STREAMS. The *sockatmark*() function could be implemented as follows:

```
#include <sys/ioctl.h>

int sockatmark(int s)
{
    int val;
    if (ioctl(s,SIOCATMARK,&val)==-1)
        return(-1);
    return(val);
}
```

The use of [ENOTTY] to indicate an incorrect descriptor type matches the historical behavior of SIOCATMARK.

Derivation First released in Issue 6. Derived from IEEE Std 1003.1g-2000.

socket

Create an endpoint for communication

```
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

socketpair

Create a pair of connected sockets

```
#include <sys/socket.h>

int socketpair(int domain, int type, int protocol,
    int socket_vector[2]);
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

sqrt, sqrtf, sqrtl

Square root function

```
#include <math.h>

double sqrt(double x);
float sqrtf(float x);
long double sqrtl(long double x);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *sqrtf*() and *sqrtl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

stat

Get file status

```
#include <sys/stat.h>

int stat(const char *restrict path, struct stat *restrict buf);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EIO] mandatory error condition is added.
- The [ELOOP] mandatory error condition is added.
- The [EOVERFLOW] mandatory error condition is added. This change is to support large files.
- The [ENAMETOOLONG] and the second [EOVERFLOW] optional error conditions are added.

The following changes were made to align with the IEEE P1003.1a draft standard:

- Details are added regarding the treatment of symbolic links.
- The [ELOOP] optional error condition is added.

The **restrict** keyword is added to the *stat*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

stderr, stdin, stdout

Standard I/O streams

```
#include <stdio.h>

extern FILE *stderr, *stdin, *stdout;
```

Derivation First released in Issue 1.

Issue 6 A note that *stderr* is expected to be open for reading and writing is added to the DESCRIPTION.

strcasecmp, strncasecmp

Case-insensitive string comparisons

XSI

```
#include <strings.h>

int strcasecmp(const char *s1, const char *s2);
int strncasecmp(const char *s1, const char *s2, size_t n);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

strcat

Concatenate two strings

```
#include <string.h>

char *strcat(char *restrict s1, const char *restrict s2);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *strcat*() prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

strchr

String scanning operation

```
#include <string.h>

char *strchr(const char *s, int c);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

strcmp

Compare two strings

```
#include <string.h>

int strcmp(const char *s1, const char *s2);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

strcoll

String comparison using collating information

```
#include <string.h>

int strcoll(const char *s1, const char *s2);
```

Derivation First released in Issue 3.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EINVAL] optional error condition is added.

An example is added.

strcpy

Copy a string

```
#include <string.h>

char *strcpy(char *restrict s1, const char *restrict s2);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *strcpy*() prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

strcspn

Get length of a complementary substring

```
#include <string.h>

size_t strcspn(const char *s1, const char *s2);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The Open Group Corrigendum U030/1 is applied. The text of the RETURN VALUE section is updated to indicate that the computed segment length is returned, not the *s1* length.

strdup

Duplicate a string

XSI

```
#include <string.h>

char *strdup(const char *s1);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

strerror, strerror_r

Get error message string

```
#include <string.h>

char *strerror(int errnum);
TSF int strerror_r(int errnum, char *strerrbuf, size_t buflen);
```

Derivation First released in Issue 3.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE section, the fact that *errno* may be set is added.
- The [EINVAL] optional error condition is added.

The *strerror_r*() function is added in response to IEEE PASC Interpretation 1003.1c #39.

The *strerror_r*() function is marked as part of the Thread-Safe Functions option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

strfmon

Convert monetary value to a string

```
XSI #include <monetary.h>

ssize_t strfmon(char *restrict s, size_t maxsize,
    const char *restrict format, ...);
```

Derivation First released in Issue 4.

Issue 6 The **restrict** keyword is added to the *strfmon*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

strftime

Convert date and time to a string

```
#include <time.h>

size_t strftime(char *restrict s, size_t maxsize,
    const char *restrict format, const struct tm *restrict timeptr);
```

Derivation First released in Issue 3.

Issue 6 The Open Group Corrigendum U033/8 is applied. The %V conversion specifier is changed from ‘‘Otherwise, it is week 53 of the previous year, and the next week is week 1’’ to ‘‘Otherwise, it is the last week of the previous year, and the next week is week 1’’.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The %C, %D, %e, %h, %n, %r, %R, %t, and %T conversion specifiers are added.

- The modified conversion specifiers are added for consistency with the ISO/IEC 9945-2: 1993 (POSIX-2) *date* utility.

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The *strftime*() prototype is updated.
- The DESCRIPTION is extensively revised.
- The %z conversion specifier is added.

A new example is added.

, item XSH/TC1/D6/60 is applied, describing the consequences of creating or modifying broken-down time structures.

strlen

Get string length

```
#include <string.h>

size_t strlen(const char *s);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

strncat

Concatenate a string with part of another

```
#include <string.h>

char *strncat(char *restrict s1, const char *restrict s2, size_t n);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *strncat*() prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

strncmp

Compare part of two strings

```
#include <string.h>

int strncmp(const char *s1, const char *s2, size_t n);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

strncpy

Copy part of a string

```
#include <string.h>

char *strncpy(char *restrict s1, const char *restrict s2, size_t n);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *strncpy*() prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

strpbrk

Scan string for byte

```
#include <string.h>

char *strpbrk(const char *s1, const char *s2);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

strptime

Date and time conversion

XSI

```
#include <time.h>

char *strptime(const char *restrict buf, const char *restrict format,
    struct tm *restrict tm);
```

Derivation First released in Issue 4.

Issue 6 The Open Group Corrigendum U033/5 is applied. The %r specifier description is reworded.

The **restrict** keyword is added to the *strptime*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

The Open Group Corrigendum U047/2 is applied, adding APPLICATION USAGE that states that the effect of multiple calls to *strptime*() using the same **tm** structure is unspecified.

The DESCRIPTION is updated to use the terms ‘‘conversion specifier’’ and ‘‘conversion specification’’ for consistency with *strftime*().

strrchr

String scanning operation

```
#include <string.h>

char *strrchr(const char *s, int c);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

strspn

Get length of a substring

```
#include <string.h>

size_t strspn(const char *s1, const char *s2);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

strstr

Find a substring

```
#include <string.h>

char *strstr(const char *s1, const char *s2);
```

Derivation First released in Issue 3. Included for alignment with ANSI standard X3.159-1989, Programming Language C (ANSI C).

Issue 6 No functional changes are made in this issue.

strtod, strtof, strtold

Convert string to a double-precision number

```
#include <stdlib.h>

double strtod(const char *restrict nptr, char **restrict endptr);
float strtof(const char *restrict nptr, char **restrict endptr);
long double strtold(const char *restrict nptr, char **restrict endptr);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is added if no conversion could be performed.

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The *strtod*( ) function is updated.
- The *strtof*( ) and *strtold*( ) functions are added.
- The DESCRIPTION is extensively revised.

ISO/IEC 9899: 1999 standard, Technical Corrigendum No. 1 is incorporated.

strtoimax, strtoumax

Convert string to integer type

```
#include <inttypes.h>

intmax_t strtoimax(const char *restrict nptr, char **restrict endptr,
    int base);
uintmax_t strtoumax(const char *restrict nptr, char **restrict endptr,
    int base);
```

These functions are equivalent to the *strtol*( ), *strtoll*( ), *strtoul*( ), and *strtoull*( ) functions, except that the initial portion of the string is converted to **intmax_t** and **uintmax_t** representation, respectively.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

strtok, strtok_r

Split string into tokens

```
#include <string.h>

char *strtok(char *restrict s1, const char *restrict s2);
TSF char *strtok_r(char *restrict s, const char *restrict sep,
        char **restrict lasts);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *strtok_r*() function is marked as part of the Thread-Safe Functions option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety. A portable multi-threaded application may only safely call the function when access to the function is serialized.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

The **restrict** keyword is added to the *strtok*() and *strtok_r*() prototypes for alignment with the ISO/IEC 9899: 1999 standard.

strtol, strtoll

Convert string to a long integer

```
#include <stdlib.h>

long strtol(const char *restrict str, char **restrict endptr, int base);
long long strtoll(const char *restrict str, char **restrict endptr,
    int base)
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is added if no conversion could be performed.

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The *strtol*() prototype is updated.
- The *strtoll*() function is added.

strtoul, strtoull

Convert string to an unsigned long

```
#include <stdlib.h>

unsigned long strtoul(const char *restrict str,
    char **restrict endptr, int base);
unsigned long long strtoull(const char *restrict str,
    char **restrict endptr, int base);
```

Derivation First released in Issue 4. Derived from ANSI standard X3.159-1989, Programming Language C (ANSI C).

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EINVAL] error condition is added for when the value of *base* is not supported.

  In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is added if no conversion could be performed.

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The *strtoul*( ) prototype is updated.
- The *strtoull*( ) function is added.

strxfrm

String transformation

```
#include <string.h>

size_t strxfrm(char *restrict s1, const char *restrict s2, size_t n);
```

Derivation First released in Issue 3. Included for alignment with the ISO C standard.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is added if no conversion could be performed.

The *strxfrm*( ) prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

swab

Swap bytes

XSI

```
#include <unistd.h>

void swab(const void *restrict src, void *restrict dest,
    ssize_t nbytes);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The **restrict** keyword is added to the *swab*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

symlink

Make symbolic link to a file

```
#include <unistd.h>

int symlink(const char *path1, const char *path2);
```

Derivation First released in Issue 4, Version 2.

Issue 6 The following changes were made to align with the IEEE P1003.1a draft standard:

- The DESCRIPTION text is updated.
- The [ELOOP] optional error condition is added.

sync

Schedule file system updates

XSI

```
#include <unistd.h>

void sync(void);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

sysconf

Get configurable system variables

```
#include <unistd.h>

long sysconf(int name);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 The symbol CLK_TCK is obsolescent and removed. It is replaced with the phrase ‘‘clock ticks per second’’.

The symbol {PASS_MAX} is removed.

The following changes were made to align with the IEEE P1003.1a draft standard:

- Table entries are added for the following variables: _SC_REGEXP, _SC_SHELL, _SC_REGEX_VERSION, _SC_SYMLOOP_MAX.

The following *sysconf*() variables and their associated names are added for alignment with IEEE Std 1003.1d-1999:

_POSIX_ADVISORY_INFO
_POSIX_CPUTIME
_POSIX_SPAWN
_POSIX_SPORADIC_SERVER
_POSIX_THREAD_CPUTIME
_POSIX_THREAD_SPORADIC_SERVER
_POSIX_TIMEOUTS

The following changes are made to the DESCRIPTION for alignment with IEEE Std 1003.1j-2000:

- A statement expressing the dependency of support for some system variables on implementation options is added.
- The following system variables are added:

  _POSIX_BARRIERS
  _POSIX_CLOCK_SELECTION
  _POSIX_MONOTONIC_CLOCK
  _POSIX_READER_WRITER_LOCKS
  _POSIX_SPIN_LOCKS
  _POSIX_TYPED_MEMORY_OBJECTS

The following system variables are added for alignment with IEEE Std 1003.2d-1994:

_POSIX2_PBS
_POSIX2_PBS_ACCOUNTING
_POSIX2_PBS_LOCATE
_POSIX2_PBS_MESSAGE
_POSIX2_PBS_TRACK

The following *sysconf*( ) variables and their associated names are added for alignment with IEEE Std 1003.1q-2000:

_POSIX_TRACE
_POSIX_TRACE_EVENT_FILTER
_POSIX_TRACE_INHERIT
_POSIX_TRACE_LOG

The macros associated with the *c89* programming models are marked LEGACY, and new equivalent macros associated with *c99* are introduced.

, item XSH/TC1/D6/62 is applied, updating the DESCRIPTION to denote that the _PC* and _SC* symbols are now required to be supported. A corresponding change has been made in the Base Definitions volume of IEEE Std 1003.1-2001. The deletion in the second paragraph removes some duplicated text. Additional symbols that were erroneously omitted from this reference page have been added.

, item XSH/TC1/D6/63 is applied, making it clear in the RETURN VALUE section that the value returned for *sysconf*(_SC_OPEN_MAX) may change if a call to *setrlimit*( ) adjusts the RLIMIT_NOFILE soft limit.

, item XSH/TC2/D6/134 is applied, updating the DESCRIPTION to remove an erroneous entry for _POSIX_SYMLOOP_MAX. This corrects an error in .

, item XSH/TC2/D6/135 is applied, removing _POSIX_FILE_LOCKING, _POSIX_MULTI_PROCESS, _POSIX2_C_VERSION, and _XOPEN_XCU_VERSION (and their associated _SC_* variables) from the DESCRIPTION and APPLICATION USAGE sections.

, item XSH/TC2/D6/136 is applied, adding _POSIX_SS_REPL_MAX, _POSIX_TRACE_EVENT_NAME_MAX, _POSIX_TRACE_NAME_MAX, _POSIX_TRACE_SYS_MAX, and _POSIX_TRACE_USER_EVENT_MAX (and their associated _SC_* variables) to the DESCRIPTION. The RETURN VALUE and APPLICATION USAGE sections are updated to note that if variables are dependent on unsupported options, the results are unspecified.

, item XSH/TC2/D6/137 is applied, removing _REGEX_VERSION and _SC_REGEX_VERSION.

system

Issue a command

```
#include <stdlib.h>

int system(const char *command);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following changes were made to align with the IEEE P1003.1a draft standard:

- The DESCRIPTION is adjusted to reflect the behavior on systems that do not support the Shell option.

tan, tanf, tanl

Tangent function

```
#include <math.h>

double tan(double x);
float tanf(float x);
long double tanl(long double x);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *tanf*() and *tanl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

, item XSH/TC1/D6/64 is applied, correcting the last paragraph in the RETURN VALUE section.

tanh, tanhf, tanhl

Hyperbolic tangent functions

```
#include <math.h>

double tanh(double x);
float tanhf(float x);
long double tanhl(long double x);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *tanhf*() and *tanhl*() functions are added for alignment with the ISO/IEC 9899: 1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899: 1999 standard.

IEC 60559: 1989 standard floating-point extensions over the ISO/IEC 9899: 1999 standard are marked.

tcdrain

Wait for transmission of output

```
#include <termios.h>

int tcdrain(int fildes);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, the final paragraph is no longer conditional on _POSIX_JOB_CONTROL. This is a FIPS requirement.
- The [EIO] error is added.

tcflow

Suspend or restart the transmission or reception of data

```
#include <termios.h>

int tcflow(int fildes, int action);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EIO] error is added.

tcflush

Flush non-transmitted output data, non-read input data, or both

```
#include <termios.h>

int tcflush(int fildes, int queue_selector);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 The Open Group Corrigendum U035/1 is applied. In the ERRORS and APPLICATION USAGE sections, references to *tcflow*() are replaced with *tcflush*().

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, the final paragraph is no longer conditional on _POSIX_JOB_CONTROL. This is a FIPS requirement.
- The [EIO] error is added.

tcgetattr

Get the parameters associated with the terminal

```
#include <termios.h>

int tcgetattr(int fildes, struct termios *termios_p);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 In the DESCRIPTION, the rate returned as the input baud rate shall be the output rate. Previously, the number zero was also allowed but was obsolescent.

tcgetpgrp

Get the foreground process group ID

```
#include <unistd.h>

pid_t tcgetpgrp(int fildes);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, text previously conditional on support for _POSIX_JOB_CONTROL is now mandatory. This is a FIPS requirement.

tcgetsid

Get process group ID for session leader for controlling terminal

XSI

```
#include <termios.h>

pid_t tcgetsid(int fildes);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

tcsendbreak

Send a ‘‘break’’ for a specific duration

```
#include <termios.h>

int tcsendbreak(int fildes, int duration);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, text previously conditional on _POSIX_JOB_CONTROL is now mandated. This is a FIPS requirement.

- The [EIO] error is added.

tcsetattr

Set the parameters associated with the terminal

```
#include <termios.h>

int tcsetattr(int fildes, int optional_actions,
    const struct termios *termios_p);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, text previously conditional on _POSIX_JOB_CONTROL is now mandated. This is a FIPS requirement.
- The [EIO] error is added.

In the DESCRIPTION, the text describing use of *tcsetattr*() from a process which is a member of a background process group is clarified.

tcsetpgrp

Set the foreground process group ID

```
#include <unistd.h>

int tcsetpgrp(int fildes, pid_t pgid_id);
```

Derivation First released in Issue 3. Included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION and ERRORS sections, text previously conditional on _POSIX_JOB_CONTROL is now mandated. This is a FIPS requirement.

The Open Group Corrigendum U047/4 is applied, adding a new paragraph to the end of the DESCRIPTION. This change is for consistency with IEEE Std 1003.1-1996 (POSIX.1), Section 7.2.

tdelete, tfind, tsearch, twalk

Manage a binary search tree

XSI

```
#include <search.h>

void *tdelete(const void *restrict key, void **restrict rootp,
    int(*compar)(const void *, const void *));
void *tfind(const void *key, void *const *rootp,
    int(*compar)(const void *, const void *));
void *tsearch(const void *key, void **rootp,
    int (*compar)(const void *, const void *));
void twalk(const void *root,
    void (*action)(const void *, VISIT, int));
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The **restrict** keyword is added to the *tdelete*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

telldir

Current location of a named directory stream

XSI
```
#include <dirent.h>

long telldir(DIR *dirp);
```

Derivation First released in Issue 2.

Issue 6 No functional changes are made in this issue.

tempnam

Create a name for a temporary file

XSI
```
#include <stdio.h>

char *tempnam(const char *dir, const char *pfx);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

tgamma, tgammaf, tgammal

Compute gamma function

```
#include <math.h>

double tgamma(double x);
float tgammaf(float x);
long double tgammal(long double x);
```

These functions compute the *gamma*() function of *x*.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

, item XSH/TC1/D6/65 is applied, correcting the third paragraph in the RETURN VALUE section.

time

Get time

```
#include <time.h>

time_t time(time_t *tloc);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The EXAMPLES, RATIONALE, and FUTURE DIRECTIONS sections are added.

timer_create

Create a per-process timer (**REALTIME**)

TMR
```
#include <signal.h>
#include <time.h>

int timer_create(clockid_t clockid, struct sigevent *restrict evp,
    timer_t *restrict timerid);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *timer_create*( ) function is marked as part of the Timers option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Timers option.

CPU-time clocks are added for alignment with IEEE Std 1003.1d-1999.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding the requirement for the CLOCK_MONOTONIC clock under the Monotonic Clock option.

The **restrict** keyword is added to the *timer_create*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC2/D6/138 is applied, updating the DESCRIPTION and APPLICATION USAGE sections to describe the case when a timer is created with the notification method set to SIGEV_THREAD.

timer_delete

Delete a per-process timer (**REALTIME**)

TMR
```
#include <time.h>

int timer_delete(timer_t timerid);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. This is part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *timer_delete*( ) function is marked as part of the Timers option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Timers option.

, item XSH/TC2/D6/139 is applied, updating the ERRORS section so that the [EINVAL] error becomes optional.

timer_getoverrun, timer_gettime, timer_settime

Per-process timers (**REALTIME**)

TMR
```
#include <time.h>

int timer_getoverrun(timer_t timerid);
int timer_gettime(timer_t timerid, struct itimerspec *value);
int timer_settime(timer_t timerid, int flags,
    const struct itimerspec *restrict value,
    struct itimerspec *restrict ovalue);
```

Derivation First released in Issue 5. Included for alignment with the POSIX Realtime Extension. These functions are part of the Realtime Option Group and may not be available on all implementations.

Issue 6 The *timer_getoverrun*(), *timer_gettime*(), and *timer_settime*() functions are marked as part of the Timers option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Timers option.

The [EINVAL] error condition is updated to include the following: ‘‘and the *it_value* member of that structure did not specify zero seconds and nanoseconds.’’ This change is for IEEE PASC Interpretation 1003.1 #89.

The DESCRIPTION for *timer_getoverrun*() is updated to clarify that ‘‘If no expiration signal has been delivered for the timer, or if the Realtime Signals Extension is not supported, the return value of *timer_getoverrun*() is unspecified’’.

The **restrict** keyword is added to the *timer_settime*() prototype for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC2/D6/141 is applied, updating the ERRORS section to include an optional [EINVAL] error for the case when a timer is created with the notification method set to SIGEV_THREAD. APPLICATION USAGE text is also added.

times

Get process and waited-for child process times

```
#include <sys/times.h>

clock_t times(struct tms *buffer);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

tmpfile

Create a temporary file

```
#include <stdio.h>

FILE *tmpfile(void);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support large files.
- The [EMFILE] optional error condition is added.

tmpnam

Create a name for a temporary file

```
#include <stdio.h>

char *tmpnam(char *s);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The DESCRIPTION is expanded for alignment with the ISO/IEC 9899: 1999 standard.

, item XSH/TC2/D6/142 is applied, updating the DESCRIPTION to allow implementations of the *tempnam*( ) function to call *tmpnam*( ).

toascii

Translate integer to a 7-bit ASCII character

XSI

```
#include <ctype.h>

int toascii(int c);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

tolower

Transliterate uppercase characters to lowercase

```
#include <ctype.h>

int tolower(int c);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

toupper

Transliterate lowercase characters to uppercase

```
#include <ctype.h>

int toupper(int c);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

towctrans

Wide-character transliteration

```
#include <wctype.h>

wint_t towctrans(wint_t wc, wctrans_t desc);
```

Derivation First released in Issue 5. Derived from the ISO/IEC 9899: 1990 standard.

Issue 6 No functional changes are made in this issue.

towlower

Transliterate uppercase wide-character code to lowercase

```
#include <wctype.h>

wint_t towlower(wint_t wc);
```

Derivation First released in Issue 4.

Issue 6 No functional changes are made in this issue.

towupper

Transliterate lowercase wide-character code to uppercase

```
#include <wctype.h>

wint_t towupper(wint_t wc);
```

Derivation First released in Issue 4.

Issue 6 No functional changes are made in this issue.

trunc, truncf, truncl

Round to truncated integer value

```
#include <math.h>

double trunc(double x);
float truncf(float x);
long double truncl(long double x);
```

These functions round their argument to the integer value, in floating format, nearest to but no larger in magnitude than the argument.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

truncate

Truncate a file to a specified length

XSI

```
#include <unistd.h>

int truncate(const char *path, off_t length);
```

Derivation First released in Issue 4, Version 2.

Issue 6 This reference page is split out from the *ftruncate*() reference page.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

ttyname, ttyname_r

Find pathname of a terminal

```
#include <unistd.h>

char *ttyname(int fildes);
TSF int ttyname_r(int fildes, char *name, size_t namesize);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *ttyname_r*() function is marked as part of the Thread-Safe Functions option. Support for this option is mandatory on systems supporting the Single UNIX Specification.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The statement that *errno* is set on error is added.
- The [EBADF] and [ENOTTY] optional error conditions are added.

daylight, timezone, tzname, tzset

Set timezone conversion information

```
#include <time.h>

XSI extern int daylight;
    extern long timezone;
CX  extern char *tzname[2];
    void tzset(void);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The example is corrected.

ualarm

Set the interval timer

```
OB XSI #include <unistd.h>

useconds_t ualarm(useconds_t useconds, useconds_t interval);
```

Derivation First released in Issue 4, Version 2.

Issue 6 This function is marked obsolescent.

ulimit

Get and set process limits

XSI
```
#include <ulimit.h>

long ulimit(int cmd, ...);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

umask

Set and get file mode creation mask

```
#include <sys/stat.h>

mode_t umask(mode_t cmask);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

uname

Get name of current system

```
#include <sys/utsname.h>

int uname(struct utsname *name);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

ungetc

Push byte back into input stream

```
#include <stdio.h>

int ungetc(int c, FILE *stream);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes are made in this issue.

ungetwc

Push wide-character code back into input stream

```
#include <stdio.h>
#include <wchar.h>

wint_t ungetwc(wint_t wc, FILE *stream);
```

Derivation First released in Issue 4. Derived from the MSE working draft.

Issue 6 The [EILSEQ] optional error condition is marked CX.

unlink

Remove a directory entry

```
#include <unistd.h>

int unlink(const char *path);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, the effect is specified if *path* specifies a symbolic link.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.
- The [ETXTBSY] optional error condition is added.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

unlockpt

Unlock a pseudo-terminal master/slave pair

XSI

```
#include <stdlib.h>

int unlockpt(int fildes);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

unsetenv

Remove environment variable

CX

```
#include <stdlib.h>

int unsetenv(const char *name);
```

Derivation First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

usleep

Suspend execution for an interval

OB XSI

```
#include <unistd.h>

int usleep(useconds_t useconds);
```

Applications are recommended to use *nanosleep*() if the Timers option is supported, or *setitimer*(), *timer-create*(), *timer_delete*(), *timer_getoverrun*(), *timer_gettime*(), or *timer_settime*() instead of this function.

Derivation First released in Issue 4, Version 2.

Issue 6 This function is marked obsolescent.

, item XSH/TC2/D6/144 is applied, updating the DESCRIPTION from ‘‘process’ signal mask’’ to ‘‘thread’s signal mask’’, and adding a statement that the *usleep*( ) function need not be reentrant.

utime

Set file access and modification times

```
#include <utime.h>

int utime(const char *path, const struct utimbuf *times);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

utimes

Set file access and modification times (**LEGACY**)

XSI

```
#include <sys/time.h>

int utimes(const char *path, const struct timeval times[2]);
```

Derivation First released in Issue 4, Version 2.

Issue 6 This function is marked LEGACY and may not be available on all implementations.

For applications portability, the *utime*( ) function should be used to set file access and modification times instead of *utimes*( ).

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

va_arg, va_copy, va_end, va_start

Handle variable argument list

```
#include <stdarg.h>

type va_arg(va_list ap, type);
void va_copy(va_list dest, va_list src);
void va_end(va_list ap);
void va_start(va_list ap, argN);
```

Derivation First released in Issue 4. Derived from the ISO C standard.

Issue 6 The *va_copy*( ) function is added to copy a variable argument list. This is added for alignment with the ISO/IEC 9899: 1999 standard.

vfork

Create new process; share virtual memory

OB XSI
```
#include <unistd.h>

pid_t vfork(void);
```

Conforming applications are recommended not to depend on *vfork*(), but to use *fork*() instead. The *vfork*() function may be withdrawn in a future version.

The use of *vfork*() for any purpose except as a prelude to an immediate call to a function from the *exec* family, or to *_exit*(), is not advised.

Derivation First released in Issue 4, Version 2.

Issue 6 Marked obsolescent.

vfprintf, vprintf, vsnprintf, vsprintf

Format output of a stdarg argument list

```
#include <stdarg.h>
#include <stdio.h>

int vfprintf(FILE *restrict stream, const char *restrict format,
    va_list ap);
int vprintf(const char *restrict format, va_list ap);
int vsnprintf(char *restrict s, size_t n, const char *restrict format,
    va_list ap);
int vsprintf(char *restrict s, const char *restrict format, va_list ap);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *vfprintf*(), *vprintf*(), *vsnprintf*(), and *vsprintf*() functions are updated to include the **restrict** keyword for alignment with the ISO/IEC 9899: 1999 standard.

vfscanf, vscanf, vsscanf

Format input of a stdarg list

```
#include <stdarg.h>
#include <stdio.h>

int vfscanf(FILE *restrict stream, const char *restrict format,
    va_list arg);
int vscanf(const char *restrict format, va_list arg);
int vsscanf(const char *restrict s, const char *restrict format,
    va_list arg);
```

The *vscanf*(), *vfscanf*(), and *vsscanf*() functions are equivalent to the *scanf*(), *fscanf*(), and *sscanf*() functions, respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined in the **<stdarg.h>** header.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

vfwprintf, vswprintf, vwprintf

Wide-character formatted output of a stdarg argument list

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>

int vfwprintf(FILE *restrict stream, const wchar_t *restrict format,
    va_list arg);
int vswprintf(wchar_t *restrict ws, size_t n,
    const wchar_t *restrict format, va_list arg);
int vwprintf(const wchar_t *restrict format, va_list arg);
```

Derivation First released in Issue 5. Included for alignment with the ISO/IEC 9899: 1990 standard.

Issue 6 The *vfwprintf*( ), *vswprintf*( ), and *vwprintf*( ) prototypes are updated to include the **restrict** keyword for alignment with the ISO/IEC 9899: 1999 standard.

vfwscanf, vswscanf, vwscanf

Wide-character formatted input of a stdarg list

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>

int vfwscanf(FILE *restrict stream, const wchar_t *restrict format,
    va_list arg);
int vswscanf(const wchar_t *restrict ws, const wchar_t *restrict format,
    va_list arg);
int vwscanf(const wchar_t *restrict format, va_list arg);
```

The *vfwscanf*( ), *vswscanf*( ), and *vwscanf*( ) functions are equivalent to the *fwscanf*( ), *swscanf*( ), and *wscanf*( ) functions, respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined in the **<stdarg.h>** header.

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

wait, waitpid

Wait for a child process to stop or terminate

```
#include <sys/wait.h>

pid_t wait(int *stat_loc);
pid_t waitpid(pid_t pid, int *stat_loc, int options);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The processing of the SIGCHLD signal and the [ECHILD] error is clarified.

The semantics of WIFSTOPPED(*stat_val*), WIFEXITED(*stat_val*), and WIFSIGNALED(*stat_val*) are defined with respect to *posix_spawn*( ) or *posix_spawnp*( ) for alignment with IEEE Std 1003.1d-1999.

The DESCRIPTION is updated for alignment with the ISO/IEC 9899: 1999 standard.

waitid

Wait for a child process to change state

XSI
```
#include <sys/wait.h>

int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes are made in this issue.

wcrtomb

Convert a wide-character code to a character (restartable)

```
#include <stdio.h>

size_t wcrtomb(char *restrict s, wchar_t wc, mbstate_t *restrict ps);
```

Derivation First released in Issue 5. Included for alignment with the ISO/IEC 9899: 1990 standard.

Issue 6 In the DESCRIPTION, a note on using this function in a threaded application is added.

The *wcrtomb*() prototype is updated to include the **restrict** keyword for alignment with the ISO/IEC 9899: 1999 standard.

wcscat

Concatenate two wide-character strings

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *restrict ws1, const wchar_t *restrict ws2);
```

Derivation First released in Issue 4. Derived from the MSE working draft.

Issue 6 The Open Group Corrigendum U040/2 is applied. In the RETURN VALUE section, *s1* is changed to *ws1*.

The *wcscat*() prototype is updated to include the **restrict** keyword for alignment with the ISO/IEC 9899: 1999 standard.

wcschr

Wide-character string scanning operation

```
#include <wchar.h>

wchar_t *wcschr(const wchar_t *ws, wchar_t wc);
```

Derivation First released in Issue 4. Derived from the MSE working draft.

Issue 6 No functional changes are made in this issue.

wcscmp

Compare two wide-character strings

```
#include <wchar.h>

int wcscmp(const wchar_t *ws1, const wchar_t *ws2);
```

Derivation First released in Issue 4. Derived from the MSE working draft.

Issue 6 No functional changes are made in this issue.

wcscoll

Wide-character string comparison using collating information

```
#include <wchar.h>

int wcscoll(const wchar_t *ws1, const wchar_t *ws2);
```

Derivation First released in Issue 4. Derived from the MSE working draft.

Issue 6 No functional changes are made in this issue.

wcscpy

Copy a wide-character string

```
#include <wchar.h>

wchar_t *wcscpy(wchar_t *restrict ws1, const wchar_t *restrict ws2);
```

Derivation First released in Issue 4. Derived from the MSE working draft.

Issue 6 The *wcscpy*() prototype is updated to include the **restrict** keyword for alignment with the ISO/IEC 9899: 1999 standard.

wcscspn

Get length of a complementary wide substring

```
#include <wchar.h>

size_t wcscspn(const wchar_t *ws1, const wchar_t *ws2);
```

Derivation First released in Issue 4. Derived from the MSE working draft.

Issue 6 No functional changes are made in this issue.

wcsftime

Convert date and time to a wide-character string

```
#include <wchar.h>

size_t wcsftime(wchar_t *restrict wcs, size_t maxsize,
    const wchar_t *restrict format, const struct tm *restrict timeptr);
```

Derivation First released in Issue 4.

Issue 6 The *wcsftime*() prototype is updated to include the **restrict** keyword for alignment with the ISO/IEC 9899: 1999 standard.

wcslen

Get wide-character string length

```
#include <wchar.h>

size_t wcslen(const wchar_t *ws);
```

Derivation First released in Issue 4. Derived from the MSE working draft.

Issue 6 No functional changes are made in this issue.

wcsncat

Concatenate a wide-character string with part of another

```
#include <wchar.h>

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict ws2,
    size_t n);
```

Derivation First released in Issue 4. Derived from the MSE working draft.

Issue 6 The *wcsncat*() prototype is updated to include the **restrict** keyword for alignment with the ISO/IEC 9899: 1999 standard.

wcsncmp

Compare part of two wide-character strings

```
#include <wchar.h>

int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
```

Derivation First released in Issue 4. Derived from the MSE working draft.

Issue 6 No functional changes are made in this issue.

wcsncpy

Copy part of a wide-character string

```
#include <wchar.h>

wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict ws2,
    size_t n);
```

Derivation First released in Issue 4. Derived from the MSE working draft.

Issue 6 The *wcsncpy*() prototype is updated to include the **restrict** keyword for alignment with the ISO/IEC 9899: 1999 standard.

wcspbrk

Scan wide-character string for a wide-character code

```
#include <wchar.h>

wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);
```

Derivation First released in Issue 4. Derived from the MSE working draft.

Issue 6 No functional changes are made in this issue.

wcsrchr

Wide-character string scanning operation

```
#include <wchar.h>

wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);
```

Derivation First released in Issue 4. Derived from the MSE working draft.

Issue 6 No functional changes are made in this issue.

wcsrtombs

Convert a wide-character string to a character string (restartable)

```
#include <wchar.h>

size_t wcsrtombs(char *restrict dst, const wchar_t **restrict src,
    size_t len, mbstate_t *restrict ps);
```

Derivation First released in Issue 5. Included for alignment with the ISO/IEC 9899: 1990 standard.

Issue 6 In the DESCRIPTION, a note on using this function in a threaded application is added.

The *wcsrtombs*() prototype is updated to include the **restrict** keyword for alignment with the ISO/IEC 9899: 1999 standard.

wcsspn

Get length of a wide substring

```
#include <wchar.h>

size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);
```

Derivation First released in Issue 4. Derived from the MSE working draft.

Issue 6 No functional changes are made in this issue.

wcsstr

Find a wide-character substring

```
#include <wchar.h>

wchar_t *wcsstr(const wchar_t *restrict ws1, const wchar_t *restrict ws2);
```

Derivation First released in Issue 5. Included for alignment with the ISO/IEC 9899: 1990 standard.

Issue 6 The *wcsstr*() prototype is updated to include the **restrict** keyword for alignment with the ISO/IEC 9899: 1999 standard.

wcstod, wcstof, wcstold

Convert a wide-character string to a double-precision number

```
#include <wchar.h>

double wcstod(const wchar_t *restrict nptr, wchar_t **restrict endptr);
float wcstof(const wchar_t *restrict nptr, wchar_t **restrict endptr);
long double wcstold(const wchar_t *restrict nptr,
    wchar_t **restrict endptr);
```

Derivation First released in Issue 4. Derived from the MSE working draft.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is added if no conversion could be performed.

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The *wcstod*( ) prototype is updated to include the **restrict** keyword.
- The *wcstof*( ) and *wcstold*( ) functions are added.
- If the correct value for *wcstod*( ) would cause underflow, the return value changed from 0 (as specified in Issue 5) to the smallest normalized positive number.
- The DESCRIPTION, RETURN VALUE, and APPLICATION USAGE sections are extensively updated.

ISO/IEC 9899: 1999 standard, Technical Corrigendum No. 1 is incorporated.

wcstoimax, wcstoumax

Convert wide-character string to integer type

```
#include <stddef.h>
#include <inttypes.h>

intmax_t wcstoimax(const wchar_t *restrict nptr,
    wchar_t **restrict endptr, int base);
uintmax_t wcstoumax(const wchar_t *restrict nptr,
    wchar_t **restrict endptr, int base);
```

Derivation First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

wcstok

Split wide-character string into tokens

```
#include <wchar.h>

wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict ws2,
    wchar_t **restrict ptr);
```

Derivation First released in Issue 4.

Issue 6 The *wcstok*( ) prototype is updated to include the **restrict** keyword for alignment with the ISO/IEC 9899: 1999 standard.

wcstol, wcstoll

Convert a wide-character string to a long integer

```
#include <wchar.h>

long wcstol(const wchar_t *restrict nptr, wchar_t **restrict endptr,
    int base);
long long wcstoll(const wchar_t *restrict nptr,
    wchar_t **restrict endptr, int base);
```

Derivation First released in Issue 4. Derived from the MSE working draft.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is added if no conversion could be performed.

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The *wcstol*( ) prototype is updated to include the **restrict** keyword.
- The *wcstoll*( ) function is added.

wcstombs

Convert a wide-character string to a character string

```
#include <stdlib.h>

size_t wcstombs(char *restrict s, const wchar_t *restrict pwcs,
    size_t n);
```

Derivation First released in Issue 4. Derived from the ISO C standard.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION states the effect of when *s* is a null pointer.
- The [EILSEQ] error condition is added.

The *wcstombs*( ) prototype is updated to include the **restrict** keyword for alignment with the ISO/IEC 9899: 1999 standard.

wcstoul, wcstoull

Convert a wide-character string to an unsigned long

```
#include <wchar.h>

unsigned long wcstoul(const wchar_t *restrict nptr,
    wchar_t **restrict endptr, int base);
unsigned long long wcstoull(const wchar_t *restrict nptr,
    wchar_t **restrict endptr, int base);
```

Derivation First released in Issue 4. Derived from the MSE working draft.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EINVAL] error condition is added for when the value of *base* is not supported.

  In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is added if no conversion could be performed.

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- The *wcstoul*( ) prototype is updated to include the **restrict** keyword.
- The *wcstoull*( ) function is added.

wcswcs

Find a wide substring (**LEGACY**)

XSI
```
#include <wchar.h>

wchar_t *wcswcs(const wchar_t *ws1, const wchar_t *ws2);
```

Derivation First released in Issue 4. Derived from the MSE working draft.

Issue 6 This function is marked LEGACY and may not be available on all implementations.

This function was not included in the final ISO C Amendment 1 (MSE). Application developers are strongly encouraged to use the *wcsstr*( ) function instead.

wcswidth

Number of column positions of a wide-character string

XSI
```
#include <wchar.h>

int wcswidth(const wchar_t *pwcs, size_t n);
```

Derivation First released in Issue 4. Derived from the MSE working draft.

Issue 6 The Open Group Corrigendum U021/11 is applied. The function is marked as an extension.

wcsxfrm

Wide-character string transformation

```
#include <wchar.h>

size_t wcsxfrm(wchar_t *restrict ws1, const wchar_t *restrict ws2,
    size_t n);
```

Derivation First released in Issue 4. Derived from the MSE working draft.

Issue 6 In previous versions, this function was required to return −1 on error.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is added if no conversion could be performed.

The *wcsxfrm*( ) prototype is updated to include the **restrict** keyword for alignment with the ISO/IEC 9899: 1999 standard.

wctob

Wide-character to single-byte conversion

```
#include <stdio.h>
#include <wchar.h>

int wctob(wint_t c);
```

Derivation First released in Issue 5. Included for alignment with the ISO/IEC 9899: 1990 standard.

Issue 6 No functional changes are made in this issue.

wctomb

Convert a wide-character code to a character

```
#include <stdlib.h>

int wctomb(char *s, wchar_t wchar);
```

Derivation First released in Issue 4. Derived from ANSI standard X3.159-1989, Programming Language C (ANSI C).

Issue 6 In the DESCRIPTION, a note about reentrancy and thread-safety is added. A portable multi-threaded application may only safely call the function when access to the function is serialized.

wctrans

Define character mapping

```
#include <wctype.h>

wctrans_t wctrans(const char *charclass);
```

Derivation First released in Issue 5. Derived from the ISO/IEC 9899: 1990 standard.

Issue 6 No functional changes are made in this issue.

wctype

Define character class

```
#include <wctype.h>

wctype_t wctype(const char *property);
```

Derivation First released in Issue 4.

Issue 6 No functional changes are made in this issue.

wcwidth

Number of column positions of a wide-character code

XSI

```
#include <wchar.h>

int wcwidth(wchar_t wc);
```

Derivation First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working draft.

Issue 6 The Open Group Corrigendum U021/12 is applied, noting that this function was removed from the final ISO C Amendment 1 (MSE). This function is marked as an extension since this function was not included in the ISO/IEC 9899: 1999 standard.

wmemchr

Find a wide character in memory

```
#include <wchar.h>

wchar_t *wmemchr(const wchar_t *ws, wchar_t wc, size_t n);
```

Derivation First released in Issue 5. Included for alignment with the ISO/IEC 9899: 1990 standard.

Issue 6 No functional changes are made in this issue.

wmemcmp

Compare wide characters in memory

```
#include <wchar.h>

int wmemcmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
```

Derivation First released in Issue 5. Included for alignment with the ISO/IEC 9899: 1990 standard.

Issue 6 No functional changes are made in this issue.

wmemcpy

Copy wide characters in memory

```
#include <wchar.h>

wchar_t *wmemcpy(wchar_t *restrict ws1, const wchar_t *restrict ws2,
    size_t n);
```

Derivation First released in Issue 5. Included for alignment with the ISO/IEC 9899: 1990 standard.

Issue 6 The *wmemcpy*() prototype is updated to include the **restrict** keyword for alignment with the ISO/IEC 9899: 1999 standard.

wmemmove

Copy wide characters in memory with overlapping areas

```
#include <wchar.h>

wchar_t *wmemmove(wchar_t *ws1, const wchar_t *ws2, size_t n);
```

Derivation First released in Issue 5. Included for alignment with the ISO/IEC 9899: 1990 standard.

Issue 6 No functional changes are made in this issue.

wmemset

Set wide characters in memory

```
#include <wchar.h>

wchar_t *wmemset(wchar_t *ws, wchar_t wc, size_t n);
```

Derivation First released in Issue 5. Included for alignment with the ISO/IEC 9899: 1990 standard.

Issue 6 No functional changes are made in this issue.

wordexp, wordfree

Perform word expansions

```
#include <wordexp.h>

int wordexp(const char *restrict words, wordexp_t *restrict pwordexp,
    int flags);
void wordfree(wordexp_t *pwordexp);
```

Derivation First released in Issue 4. Derived from ISO/IEC 9945-2: 1993 (POSIX-2).

Issue 6 The **restrict** keyword is added to the *wordexp*( ) prototype for alignment with the ISO/IEC 9899: 1999 standard.

pwrite, write

Write on a file

```
#include <unistd.h>
```

XSI

```
ssize_t pwrite(int fildes, const void *buf, size_t nbyte,
    off_t offset);
ssize_t write(int fildes, const void *buf, size_t nbyte);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The DESCRIPTION states that the *write*( ) function does not block the thread. Previously this said ‘‘process’’ rather than ‘‘thread’’.

The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are marked as part of the XSI STREAMS Option Group.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION now states that if *write*( ) is interrupted by a signal after it has successfully written some data, it returns the number of bytes written. In IEEE Std 1003.1-1988 (POSIX.1), it was optional whether *write*( ) returned the number of bytes written, or whether it returned −1 with *errno* set to [EINTR]. This is a FIPS requirement.
- The following changes are made to support large files:
  - For regular files, no data transfer occurs past the offset maximum established in the open file description associated with the *fildes*.
  - A second [EFBIG] error condition is added.

- The [EIO] error condition is added.
- The [EPIPE] error condition is added for when a pipe has only one end open.
- The [ENXIO] optional error condition is added.

Text referring to sockets is added to the DESCRIPTION.

The following changes are made to align with the IEEE P1003.1a draft standard:

- The effect of reading zero bytes is clarified.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that *write*() results are unspecified for typed memory objects.

The following error conditions are added for operations on sockets: [EAGAIN], [EWOULDBLOCK], [ECONNRESET], [ENOTCONN], and [EPIPE].

The [EIO] error is changed to ‘‘may fail’’.

The [ENOBUFS] error is added for sockets.

The following error conditions are added for operations on sockets: [EACCES], [ENETDOWN], and [ENETUNREACH].

, item XSH/TC2/D6/146 is applied, updating text in the ERRORS section from ‘‘a SIGPIPE signal is generated to the calling process’’ to ‘‘a SIGPIPE signal shall also be sent to the thread’’.

writev

Write a vector

XSI
```
#include <sys/uio.h>

ssize_t writev(int fildes, const struct iovec *iov, int iovcnt);
```

Derivation First released in Issue 4, Version 2.

Issue 6 Split out from the *write*() reference page.

y0, y1, yn

Bessel functions of the second kind

XSI
```
#include <math.h>

double y0(double x);
double y1(double x);
double yn(int n, double x);
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The RETURN VALUE and ERRORS sections are reworked for alignment of the error handling with the ISO/IEC 9899: 1999 standard.

, item XSH/TC2/D6/148 is applied, updating the RETURN VALUE and ERRORS sections. The changes are made for consistency with the general rules stated in ‘‘Treatment of Error Conditions for Mathematical Functions’’ in the Base Definitions volume of IEEE Std 1003.1-2001.

Chapter 14

Utilities Migration

This chapter contains a section for each interface defined in XCU, Issue 6. Each section contains the SYNOPSIS and identifies syntax and semantic changes made to the interface in Issue 6 (if any), complete with examples where appropriate. Only changes that might affect an application programmer are identified.

ISO/IEC Terminology Changes

In order for the XCU document to also be an ISO/IEC standard, a number of terminology changes in the normative text were made, notably use of the term ‘‘shall’’ instead of the term ‘‘will’’ for requirements, and avoiding the term ‘‘must’’ for application requirements. Although these terminology changes affected most utilities, no functional change is intended and these changes are not further noted in this chapter.

Eight-Bit Bytes

This issue of the specification mandates that a byte has eight bits. As such, text describing behavior on systems with bytes consisting of more than eight bits has been removed. This change is not further noted in this chapter.

admin

Create and administer SCCS files (**DEVELOPMENT**)

XSI
```
admin –i[name][–n][–a login][–d flag][–e login][–f flag][–m mrlist]
    [–r rel][–t[name][–y[comment]] newfile
admin –n[–a login][–d flag][–e login][–f flag][–m mrlist][–t[name]]
    [–y[comment]] newfile ...
admin [–a login][–d flag][–m mrlist][–r rel][–t[name]] file ...
admin –h file ...
admin –z file ...
```

Derivation First released in Issue 2.

Issue 6 The grammar is updated.

The Open Group Base Resolution bwg2001-007 is applied, adding new text to the INPUT FILES section warning that the maximum lines recorded in the file is 99 999.

The Open Group Base Resolution bwg2001-009 is applied, replacing the text related to the –**h** option for the *admin* utility with the following:

> Check the structure of the SCCS file and compare the newly-computed checksum with the checksum that is stored in the SCCS file. If the newly-computed checksum does not match the checksum in the SCCS file, a diagnostic message shall be written.

alias

Define or display aliases

UP
```
alias [alias-name[=string] ...]
```

Derivation First released in Issue 4.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

No functional changes in this issue.

ar

Create and maintain library archives

SD
```
ar –d[–v] archive file ...
```

XSI
```
ar –m[–abiv][posname] archive file ...
```

XSI
```
ar –p[–v][–s]archive [file ...]
```

XSI
```
ar –q[–cv] archive file ...
```

XSI `ar –r[–cuv][–abi][posname]archive file ...`

XSI `ar –t[–v][–s]archive [file ...]`

XSI `ar –x[–v][–sCT]archive [file ...]`

Derivation First released in Issue 2.

Issue 6 This utility is marked as part of the Software Development Utilities option.

The STDOUT description is changed for the –**v** option to align with the IEEE P1003.2b draft standard.

The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

IEEE PASC Interpretation 1003.2 #198 is applied, changing the description to consistently use ‘‘file’’ to refer to a file in the file system hierarchy, ‘‘archive’’ to refer to the archive being operated upon by the *ar* utility, and ‘‘file in the archive’’ to refer to a copy of a file that is contained in the archive.

asa

Interpret carriage-control characters

FR `asa [ file ... ]`

Derivation First released in Issue 4.

Issue 6 This utility is marked as part of the FORTRAN Runtime Utilities option.

No functional changes in this issue.

at

Execute commands at a later time

UP `at [–m][–f file][–q queuename] –t time_arg`

`at [–m][–f file][–q queuename] timespec ...`

`at –r at_job_id ...`

`at –l –q queuename`

`at –l [at_job_id ...]`

Derivation First released in Issue 2.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- If –**m** is not used, the job’s standard output and standard error are provided to the user by mail.

The effects of using the –**q** and –**t** options as defined in the IEEE P1003.2b draft standard are specified.

awk

Pattern scanning and processing language

awk **[**–F *ERE*]**[**–v *assignment*] ... *program* **[***argument* ...]

awk **[**–F *ERE*] –f *progfile* ... **[**–v *assignment*] ...**[***argument* ...]

Derivation First released in Issue 2.

Issue 6 The *awk* utility is aligned with the IEEE P1003.2b draft standard.

IEEE PASC Interpretation 1003.2 #211 is applied, adding the sentence ‘‘An occurrence of two consecutive backslashes shall be interpreted as just a single literal backslash character.’’ into the description of the **sub** string function.

basename

Return non-directory portion of a pathname

basename *string* **[***suffix*]

Derivation First released in Issue 2.

Issue 6 IEEE PASC Interpretation 1003.2 #164 is applied, adding a new step 1. as follows:

1. If *string* is a null string, it is unspecified whether the resulting string is ’.’ or a null string. In either case, skip steps 2 through 6.

batch

Schedule commands to be executed in a batch queue

UP *batch*

Derivation First released in Issue 2.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

The NAME is changed to align with the IEEE P1003.2b draft standard.

bc

Arbitrary-precision arithmetic language

bc **[**–l] **[***file* ...]

Derivation First released in Issue 4.

Issue 6 Updated to align with the IEEE P1003.2b draft standard, which included resolution of several interpretations of ISO/IEC 9945-2: 1993 (POSIX-2).

bg

Run jobs in the background

UP
```
bg [job_id ...]
```

Derivation First released in Issue 4.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

The JC margin marker on the SYNOPSIS is removed since support for Job Control is mandatory in this issue. This is a FIPS requirement.

No functional changes in this issue.

c99

Compile standard C programs

CD
```
c99 [–c][–D name[=value]]...[–E][–g][–I directory] ... [–L directory]
    ... [–o outfile][–Ooptlevel][–s][–U name]... operand ...
```

Derivation First released in Issue 6. Included for alignment with the ISO/IEC 9899: 1999 standard.

Issue 6 This is a new utility created in Issue 6 to allow applications a front end to a compiler conforming to the ISO/IEC 9899: 1999 standard.

On systems providing POSIX Conformance (see XBD, Chapter 2, Conformance), *c99* is required only with the C-Language Development option; XSI-conformant systems always provide *c99*.

, item XCU/TC1/D6/12 is applied, correcting the EXTENDED DESCRIPTION of –**l c** and –**l m**. Previously, the text did not take into account the presence of the *c99* math headers.

, item XCU/TC1/D6/13 is applied, changing the reference to the **libxnet** library to **libxnet.a**.

, item XCU/TC2/D6/5 is applied, updating the OPTIONS section, so that the names of files contained in the directory specified by the –**L** option are not assumed to end in the **.a** suffix. The set of library prefixes is also updated.

, item XCU/TC2/D6/6 is applied, removing the lead underscore from the POSIX_V6_WIDTH_RESTRICTED_ENVS variable in the EXTENDED DESCRIPTION and the EXAMPLES sections.

cal

Print a calendar

XSI
```
cal [[month] year ]
```

Derivation First released in Issue 2.

Issue 6 The DESCRIPTION is updated to allow for traditional behavior for years before the adoption of the Gregorian calendar.

cat

Concatenate and print files

`cat [–u][file ...]`

Derivation First released in Issue 2.

Issue 6 No functional changes in this issue.

cd

Change the working directory

`cd [–L] [–P] [directory]`

`cd –`

Derivation First released in Issue 2.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The *cd* –, *PWD*, and *OLDPWD* are added.

The –**L** and –**P** options are added to align with the IEEE P1003.2b draft standard. This also includes the introduction of a new description to include the effect of these options.

, item XCU/TC1/D6/14 is applied, changing the SYNOPSIS to make it clear that the –**L** and –**P** options are mutually-exclusive.

cflow

Generate a C-language flowgraph (**DEVELOPMENT**)

XSI
```
cflow [–r][–d num][–D name[=def]] ... [–i incl][–I dir] ...
    [–U dir] ... file ...
```

Derivation First released in Issue 2.

Issue 6 No functional changes in this issue.

chgrp

Change the file group ownership

`chgrp –hR group file ...`

`chgrp –R [–H | –L | –P ] group file ...`

Derivation First released in Issue 2.

Issue 6 New options –**H**, –**L**, and –**P** are added to align with the IEEE P1003.2b draft standard. These options affect the processing of symbolic links.

IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS section to ‘‘Default.’’.

, item XCU/TC1/D6/15 is applied, changing the SYNOPSIS to make it clear that –**h** and –**R** are optional.

chmod

Change the file modes

`chmod [`–`R] mode file ...`

Derivation First released in Issue 2.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- Octal modes have been kept and made mandatory despite being marked obsolescent in ISO/IEC 9945-2: 1993 (POSIX-2).

IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS section to ‘‘Default.’’.

The Open Group Base Resolution bwg2001-010 is applied (as XSI shaded text) as follows:

- The table is extended to include the entry 1000 S_ISVTX.
- The **t perm** symbol is added.
- The following text is added to the EXTENDED DESCRIPTION:

  > The **perm** symbol **t** shall specify the S_ISVTX bit and shall apply to directories only. The effect when using it with any other file type is unspecified. It can be used with the **who** symbols **o**, **a**, or with no **who** symbol. It shall not be an error to specify a **who** symbol of **u** or **g** in conjunction with the **perm** symbol **t**; it shall ignored for **u** and **g**.

, item XCU/TC1/D6/16 is applied, changing the XSI shaded text in the EXTENDED DESCRIPTION from:

> ‘‘The **perm** symbol **t** shall specify the S_ISVTX bit and shall apply to directories only. The effect when using it with any other file type is unspecified. It can be used with the **who** symbols **o**, **a**, or with no **who** symbol. It shall not be an error to specify a **who** symbol of **u** or **g** in conjunction with the **perm** symbol **t**; it shall be ignored for **u** and **g**.’’

to:

> ‘‘The **perm** symbol **t** shall specify the S_ISVTX bit. When used with a file of type directory, it can be used with the **who** symbol **a**, or with no **who** symbol. It shall not be an error to specify a **who** symbol of **u**, **g**, or **o** in conjunction with the **perm** symbol **t**, but the meaning of these combinations is unspecified. The effect when using the **perm** symbol **t** with any file type other than directory is unspecified.’’

This change is to permit historical behavior.

chown

Change the file ownership

`chown –hR` *owner*`[:`*group*`]` *file* `...`

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

Derivation First released in Issue 2.

Issue 6 New options –**h**, –**H**, –**L**, and –**P** are added to align with the IEEE P1003.2b draft standard. These options affect the processing of symbolic links.

IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS section is changed to ‘‘Default.’’.

, item XCU/TC1/D6/17 is applied, changing the SYNOPSIS to make it clear that –**h** and –**R** are optional.

cksum

Write file checksums and sizes

`cksum [`*file* `...]`

Derivation First released in Issue 4.

Issue 6 No functional changes in this issue.

cmp

Compare two files

`cmp [ –l | –s ]` *file1 file2*

Derivation First released in Issue 2.

Issue 6 No functional changes in this issue.

comm

Select or reject lines common to two files

`comm [–123]` *file1 file2*

Derivation First released in Issue 2.

Issue 6 No functional changes in this issue.

command

Execute a simple command

`command [–p]` *command_name* `[`*argument* `...]`

UP `command [ –v | –V ]` *command_name*

Derivation First released in Issue 4.

Issue 6 No functional changes in this issue.

compress

Compress data

XSI `compress [–fv][–b bits][file ...]`

`compress [–cfv][–b bits][file]`

Derivation First released in Issue 4.

Issue 6 An error case is added for systems not supporting adaptive Lempel-Ziv coding.

cp

Copy files

`cp [–fip] source_file target_file`

`cp [–fip] source_file ... target`

`cp –R [–H | –L | –P][–fip] source_file ... target`

OB `cp –r [–H | –L | –P][–fip] source_file ... target`

Derivation First released in Issue 2.

Issue 6 The –**r** option is marked obsolescent.

The new options –**H**, –**L**, and –**P** are added to align with the IEEE P1003.2b draft standard. These options affect the processing of symbolic links.

IEEE PASC Interpretation 1003.2 #194 is applied, adding a description of the –**P** option.

crontab

Schedule periodic background work

UP `crontab [file]`

`crontab [ –e | –l | –r ]`

Derivation First released in Issue 2.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

No functional changes in this issue.

csplit

Split files based on context

UP `csplit [–ks][–f prefix][–n number] file arg1 ...argn`

Derivation First released in Issue 2.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

The APPLICATION USAGE section is added.

The description of regular expression operands is changed to align with the IEEE P1003.2b draft standard.

ctags

Create a tags file (**DEVELOPMENT**, **FORTRAN**)

UP
```
ctags [–a][–f tagsfile] pathname ...
ctags –x pathname ...
```

Derivation First released in Issue 4.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

The OUTPUT FILES section is changed to align with the IEEE P1003.2b draft standard.

IEEE PASC Interpretation 1003.2 #168 is applied, changing ‘‘create’’ to ‘‘write’’ in the DESCRIPTION.

cut

Cut out selected fields of each line of a file

```
cut –b list [–n] [file ...]
cut –c list [file ...]
cut –f list [–d delim][–s][file ...]
```

Derivation First released in Issue 2.

Issue 6 The OPTIONS section is changed to align with the IEEE P1003.2b draft standard.

cxref

Generate a C-language program cross-reference table (**DEVELOPMENT**)

XSI
```
cxref [–cs][–o file][–w num] [–D name[=def]]...[–I dir]...
    [–U name]... file ...
```

Derivation First released in Issue 2.

Issue 6 No functional changes in this issue.

date

Write the date and time

```
date [–u] [+format]
```

XSI
```
date [–u] mmddhhmm[[cc]yy]
```

Derivation First released in Issue 2.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The setting of system date and time is described, including how to interpret two-digit year values if a century is not given.
- The %EX modified conversion specification is added.

The Open Group Corrigendum U048/2 is applied, correcting the examples.

The DESCRIPTION is updated to refer to conversion specifications, instead of field descriptors for consistency with the *LC_TIME* category.

A clarification is made such that the current year is the default if the *yy* argument is omitted when setting the system date and time.

dd

Convert and copy a file

`dd [`*`operand`* `...]`

Derivation First released in Issue 2.

Issue 6 Changes are made to *swab* conversion and SIGINT handling to align with the IEEE P1003.2b draft standard.

IEEE PASC Interpretation 1003.2 #209 is applied, clarifying the interaction between *dd* **of**=*file* and **conv**=**notrunc**.

delta

Make a delta (change) to an SCCS file (**DEVELOPMENT**)

XSI `delta [`–nps`][`–g *list*`][`–m *mrlist*`][`–r *SID*`][`–y`[`*comment*`]] *file*...`

Derivation First released in Issue 2.

Issue 6 The APPLICATION USAGE section is added.

The Open Group Base Resolution bwg2001-007 is applied as follows:

- The use of '–' as a file argument is clarified.
- The use of STDIN is added.
- The ASYNCHRONOUS EVENTS section is updated to remove the implicit requirement that implementations re-signal themselves when catching a normally fatal signal.
- New text is added to the INPUT FILES section warning that the maximum lines recorded in the file is 99 999.

New text is added to the EXTENDED DESCRIPTION and APPLICATION USAGE sections regarding how the system date and time may be taken into account, and the *TZ* environment variable is added to the ENVIRONMENT VARIABLES section as per The Open Group Base Resolution bwg2001-007.

df

Report free disk space

UP XSI `df [–k][–P|–t][file...]`

Derivation First released in Issue 2.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

No functional changes in this issue.

diff

Compare two files

`diff [–c| –e| –f| –C n][–br] file1 file2`

Derivation First released in Issue 2.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The –**f** option is added.

The output format for –**c** or –**C** format is changed to align with changes to the IEEE P1003.2b draft standard resulting from IEEE PASC Interpretation 1003.2 #71.

, item XCU/TC1/D6/20 is applied, changing the STDOUT section. This changes the specification of *diff* –**c** so that it agrees with existing practice when contexts contain zero lines or one line.

dirname

Return the directory portion of pathname

`dirname string`

Derivation First released in Issue 2.

Issue 6 No functional changes in this issue.

du

Estimate file space usage

UP `du [–a | –s][–kx][–H | –L][file ...]`

Derivation First released in Issue 2.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

The APPLICATION USAGE section is added.

This utility is reinstated, as the LEGACY marking was incorrect in Issue 5.

The obsolescent −**r** option has been removed.

The Open Group Corrigendum U025/3 is applied. The *du* utility had incorrectly been marked LEGACY.

The −**H** and −**L** options for symbolic links are added as described in the IEEE P1003.2b draft standard.

echo

Write arguments to standard output

`echo [`*`string ...`*`]`

Derivation First released in Issue 2.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- A set of character sequences is defined as *string* operands.
- *LC_CTYPE* is added to the list of environment variables affecting *echo*.
- In the OPTIONS section, implementations shall not support any options.

, item XCU/TC1/D6/21 is applied, so that the *echo* utility can accommodate historical BSD behavior.

ed

Edit text

`ed [`−p *`string`*`][`−s`][`*`file`*`]`

Derivation First released in Issue 2.

Issue 6 The obsolescent single-minus form has been removed.

A second APPLICATION USAGE note has been added.

The Open Group Corrigendum U025/2 is applied, correcting the description of the Edit section.

The *ed* utility is updated to align with the IEEE P1003.2b draft standard. This includes addition of the treatment of the SIGQUIT signal, changes to *ed* addressing, and changes to processing when end-of-file is detected and when terminal disconnect is detected.

, item XCU/TC1/D6/22 is applied, adding the text: ‘‘Any line modified by the *command list* shall be unmarked.’’ to the **G** command. This change corresponds to a similar change made to the **g** command in the first version of IEEE Std 1003.1-2001.1-2001.

env

Set the environment for command invocation

`env [–i][name=value]... [utility [argument...]]`

Derivation First released in Issue 2.

Issue 6 No functional changes in this issue.

ex

Text editor

UP `ex [–rR][–l][–s | –v][–c command][–t tagstring][–w size][file ...]`

Derivation First released in Issue 2.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

The obsolescent SYNOPSIS is removed, removing the **+***command* and – options.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the *map* command description, the sequence ***#****digit* is added.
- The **directory**, **edcompatible**, **redraw**, and **slowopen** edit options are added.

The *ex* utility is extensively changed for alignment with the IEEE P1003.2b draft standard. This includes changes as a result of the IEEE PASC Interpretations 1003.2 #31, #38, #49, #50, #51, #52, #55, #56, #57, #61, #62, #63, #64, #65, and #78.

The –**l** option is removed.

, item XCU/TC1/D6/23 is applied, correcting a URL.

, item XCU/TC2/D6/8 is applied, making an editorial correction in the EXTENDED DESCRIPTION.

expand

Convert tabs to spaces

UP `expand [–t tablist][file ...]`

Derivation First released in Issue 4.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

The APPLICATION USAGE section is added.

The obsolescent SYNOPSIS is removed.

The *LC_CTYPE* environment variable description is updated to align with the IEEE P1003.2b draft standard.

expr

Evaluate arguments as an expression

`expr` *operand*

Derivation First released in Issue 2.

Issue 6 The *expr* utility is aligned with the IEEE P1003.2b draft standard, to include resolution of IEEE PASC Interpretation 1003.2 #104.

false

Return false value

`false`

Derivation First released in Issue 2.

Issue 6 No functional changes in this issue.

fc

Process the command history list

UP `fc [`–r`][`–e *editor*`] [`*first*`[`*last*`]]`

`fc –l[`–nr`] [`*first*`[`*last*`]]`

`fc –s[`*old*`=`*new*`][`*first*`]`

Derivation First released in Issue 4.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

In the ENVIRONMENT VARIABLES section, the text ‘‘user’s home directory’’ is updated to ‘‘directory referred to by the *HOME* environment variable’’.

fg

Run jobs in the foreground

UP `fg [`*job_id*`]`

Derivation First released in Issue 4.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

The APPLICATION USAGE section is added.

The JC marking is removed from the SYNOPSIS since job control is mandatory is this issue.

No functional changes in this issue.

file

Determine file type

UP `file [–dhi][–M file][–m file] file ...`

Derivation First released in Issue 4.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

Options and an EXTENDED DESCRIPTION are added as specified in the IEEE P1003.2b draft standard.

IEEE PASC Interpretation 1003.2 #178 is applied, adding the following text to the EXTENDED DESCRIPTION:

x The test shall succeed if the file is large enough to contain a value of the type specified starting at the offset specified.

IEEE PASC Interpretation 1003.2 #192 is applied, replacing the second paragraph in the EXTENDED DESCRIPTION within the *value* item with:

> If the specifier from the type field is s or **string**, then interpret the value as a string. Otherwise, interpret it as a number. If the value is a string, then the test shall succeed only when a string value exactly matches the bytes from the file.

, item XCU/TC1/D6/25 is applied, making major changes to address ambiguities raised in defect reports.

, item XCU/TC1/D6/26 is applied, making it clear in the OPTIONS section that the –**m**, –**d**, and –**M** options do not comply with Guideline 11 of the Utility Syntax Guidelines.

, item XCU/TC2/D6/10 is applied, clarifying the specification characters.

, item XCU/TC2/D6/11 is applied, allowing application writers to create portable magic files that can match characters in strings, and allowing common extensions found in existing implementations.

find

Find files

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

Derivation First released in Issue 2.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The –**perm [–]***onum* primary is supported.

The *find* utility is aligned with the IEEE P1003.2b draft standard, to include processing of symbolic links and changes to the description of the **atime**, **ctime**, and **mtime** operands.

IEEE PASC Interpretation 1003.2 #210 is applied, extending the –**exec** operand.

, item XCU/TC2/D6/13 is applied, updating the RATIONALE section to be consistent with the normative text.

fold

Filter for folding lines

```
fold [–bs][–w width][file...]
```

Derivation First released in Issue 4.

Issue 6 No functional changes in this issue.

fort77

FORTRAN compiler (**FORTRAN**)

FD
```
fort77 [–c][–g][–L directory]... [–O optlevel][–o outfile][–s][–w]
    operand...
```

Derivation First released in Issue 4.

Issue 6 This utility is marked as part of the FORTRAN Development Utilities option. This utility may not be present on all systems supporting the Single UNIX Specification.

No functional changes in this issue.

fuser

List process IDs of all processes that have one or more files open

XSI
```
fuser [ –cfu ] file ...
```

Derivation First released in Issue 5.

Issue 6 No functional changes in this issue.

gencat

Generate a formatted message catalog

XSI
```
gencat catfile msgfile...
```

Derivation First released in Issue 3.

Issue 6 No functional changes in this issue.

get

Get a version of an SCCS file (**DEVELOPMENT**)

XSI
```
get [–begkmnlLpst][–c cutoff][–i list][–r SID][–x list] file...
```

Derivation First released in Issue 2.

Issue 6 The obsolescent SYNOPSIS is removed, removing the –**lp** option.

The Open Group Corrigendum U025/5 is applied, correcting text in the OPTIONS section.

The Open Group Corrigendum U048/1 is applied.

The EXTENDED DESCRIPTION section is updated to make partial SID handling unspecified, reflecting common usage, and to clarify SID ranges as per The Open Group Base Resolution bwg2001-007.

The Open Group Interpretation PIN4C.00014 is applied, adding the string `"lines"` to the required output on *stdout*.

New text is added to the EXTENDED DESCRIPTION and APPLICATION USAGE sections regarding how the system date and time may be taken into account, and the *TZ* environment variable is added to the ENVIRONMENT VARIABLES section as per The Open Group Base Resolution bwg2001-007.

getconf

Get configuration values

`getconf [ –v specification ] system_var`

`getconf [ –v specification ] path_var pathname`

Derivation First released in Issue 4.

Issue 6 The Open Group Corrigendum U029/4 is applied, correcting the example command in the last paragraph of the OPTIONS section.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- Operands are added to determine supported programming environments.

This reference page is updated for alignment with the ISO/IEC 9899: 1999 standard. Specifically, new macros for *c99* programming environments are introduced.

XSI marked *system_var* (XBS5_*) values are marked LEGACY.

, item XCU/TC1/D6/27 is applied, correcting the descriptions of *path_var* and *system_var* in the OPERANDS section.

getopts

Parse utility options

`getopts optstring name [arg...]`

Derivation First released in Issue 4.

Issue 6 No functional changes in this issue.

grep

Search a file for a pattern

```
grep [–E| –F][–c| –l| –q][–insvx] –e pattern_list...
    [–f pattern_file]...[file...]
```

```
grep [–E| –F][–c| –l| –q][–insvx][–e pattern_list]...
    –f pattern_file...[file...]
```

```
grep [–E| –F][–c| –l| –q][–insvx] pattern_list[file...]
```

Derivation First released in Issue 2.

Issue 6 The Open Group Corrigendum U029/5 is applied, correcting the SYNOPSIS.

No functional changes in this issue.

hash

Remember or report utility locations

XSI `hash [utility...]`

`hash –r`

Derivation First released in Issue 2.

Issue 6 No functional changes in this issue.

head

Copy the first part of files

`head [–n number][file...]`

Derivation First released in Issue 4.

Issue 6 The obsolescent –**number** form is withdrawn.

iconv

Codeset conversion

`iconv [–cs] –f fromcode –t tocode [file ...]`

`iconv –l`

Application writers should note that the *iconv* utility can be used portably only when it is provided with two charmap files as option-arguments. This is because a single charmap provided by the user cannot reliably be joined with the names in a system-provided character set description.

Derivation First released in Issue 3.

Issue 6 This utility has been rewritten to align with the IEEE P1003.2b draft standard. Specifically, the ability to use charmap files for conversion has been added.

, item XCU/TC1/D6/29 is applied, making changes to address inconsistencies with the *iconv*( ) function in the System Interfaces volume of IEEE Std 1003.1-2001.

id

Return user identity

`id [user]`

`id –G[–n] [user]`

`id –g[–nr] [user]`

`id –u[–nr] [user]`

Derivation First released in Issue 2.

Issue 6 No functional changes in this issue.

ipcrm

Remove an XSI message queue, semaphore set, or shared memory segment identifier

XSI
```
ipcrm [ –q msgid | –Q msgkey | –s semid | –S semkey |
    –m shmid | –M shmkey ] ...
```

Derivation First released in Issue 5.

Issue 6 No functional changes in this issue.

ipcs

Report XSI interprocess communication facilities status

XSI
```
ipcs [–qms][–a | –bcopt]
```

Derivation First released in Issue 5.

Issue 6 The Open Group Corrigendum U020/1 is applied, correcting the SYNOPSIS.

The Open Group Corrigenda U032/1 and U032/2 are applied, clarifying the output format.

The Open Group Base Resolution bwg98-004 is applied, changing NSEMS (**a**,**t**) to NSEMS (**a**,**b**) in the STDOUT section.

jobs

Display status of jobs in the current session

UP
```
jobs [–l| –p][job_id...]
```

Derivation First released in Issue 4.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

The JC shading is removed as job control is mandatory in this issue.

No functional changes in this issue.

join

Relational database operator

```
join [–a file_number | –v file_number][–e string][–o list][–t char]
    [–1 field][–2 field] file1 file2
```

Derivation First released in Issue 2.

Issue 6 The obsolescent –**j** options and the multi-argument –**o** option are withdrawn in this issue.

kill

Terminate or signal processes

```
kill –s signal_name pid ...
kill –l [exit_status]
```

XSI

```
kill [–signal_name] pid ...
kill [–signal_number] pid ...
```

Derivation First released in Issue 2.

Issue 6 The obsolescent versions of the SYNOPSIS were turned into non-obsolescent features of the XSI option, corresponding to a similar change in the *trap* special built-in.

lex

Generate programs for lexical tasks (**DEVELOPMENT**)

CD

```
lex [–t][–n|–v][file ...]
```

Derivation First released in Issue 2.

Issue 6 This utility is marked as part of the C-Language Development Utilities option.

The obsolescent –**c** option is withdrawn in this issue.

link

Call *link*( ) function

XSI

```
link file1 file2
```

Derivation First released in Issue 5.

Issue 6 No functional changes in this issue.

ln

Link files

```
ln [–fs] source_file target_file
ln [–fs] source_file ... target_dir
```

Derivation First released in Issue 2.

Issue 6 The *ln* utility is updated to include symbolic link processing as defined in the IEEE P1003.2b draft standard.

locale

Get locale-specific information

`locale [–a| –m]`

`locale [–ck] name...`

Derivation First released in Issue 4.

Issue 6 No functional changes in this issue.

localedef

Define locale environment

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

Derivation First released in Issue 4.

Issue 6 The –**u** option is added, as specified in the IEEE P1003.2b draft standard.

, item XCU/TC2/D6/15 is applied, rewording text in the OPERANDS section describing the ability to create public locales.

, item XCU/TC2/D6/16 is applied, making the text consistent with the descriptions of **WIDTH** and **WIDTH_DEFAULT** in the Base Definitions volume of IEEE Std 1003.1-2001.

logger

Log messages

`logger string ...`

Derivation First released in Issue 4.

Issue 6 No functional changes in this issue.

logname

Return the user's login name

`logname`

Derivation First released in Issue 2.

Issue 6 No functional changes in this issue.

lp

Send files to a printer

`lp [–c][–d dest][–n copies][–msw][–o option]... [–t title][file...]`

Derivation First released in Issue 2.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, the requirement to associate a unique request ID, and the normal generation of a banner page is added.
- In the OPTIONS section:

— The –**d** *dest* description is expanded, but references to *lpstat* are removed.

— The –**m**, –**o**, –**s**, –**t**, and –**w** options are added.

- In the ENVIRONMENT VARIABLES section, *LC_TIME* may now affect the execution.
- The STDOUT section is added.

The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

ls

List directory contents

XSI `ls [–CFRacdilqrtu1][–H | –L ][–fgmnopsx][`*file*`...]`

Derivation First released in Issue 2.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the –**F** option, other symbols are allowed for other file types.

Treatment of symbolic links is added, as defined in the IEEE P1003.2b draft standard.

The Open Group Base Resolution bwg2001-010 is applied (as XSH shaded text) as follows:

`T` If in <*other permissions*> and the file is a directory, search permission is not granted to others, and the restricted deletion flag is set.

`t` If in <*other permissions*> and the file is a directory, search permission is granted to others, and the restricted deletion flag is set.

m4

Macro processor (**DEVELOPMENT**)

XSI `m4 [–s][–D` *name*`[=`*val*`]]...[–U` *name*`]... `*file*`...`

Derivation First released in Issue 2.

Issue 6 In the EXTENDED DESCRIPTION, the **eval** text is updated to include a `'&'` character in the excepted list.

The EXTENDED DESCRIPTION of **divert** is updated to clarify that there are only nine diversion buffers.

The Open Group Base Resolution bwg2000-006 is applied, adding the following text to the end of the last paragraph before the list of built-in macros in the EXTENDED DESCRIPTION:

> Except for the first argument to the **eval** built-in macro, all numeric built-in macro arguments shall be interpreted as decimal values. The string values produced as the defining text of the **decr**, **divnum**, **incr**, **index**, **len**, and **sysval** built-in macros shall be in the form of a decimal-constant as defined in the C language.

mailx

Process messages

Derivation First released in Issue 2.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The –**F** option is added.
- The **allnet**, **debug**, and **sendwait** internal variables are added.
- The **C**, **ec**, **fo**, **F**, and **S** *mailx* commands are added.

In the DESCRIPTION and ENVIRONMENT VARIABLES sections, text stating ‘‘*HOME* directory’’ is replaced by ‘‘directory referred to by the *HOME* environment variable’’.

The *mailx* utility is aligned with the IEEE P1003.2b draft standard, which included various clarifications to resolve IEEE PASC Interpretations submitted for ISO/IEC 9945-2: 1993 (POSIX-2). In particular, the changes here address IEEE PASC Interpretations 1003.2 #10, #11, #103, #106, #108, #114, #115, #122, and #129.

The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

, item XCU/TC1/D6/32 is applied, applying a change to the EXTENDED DESCRIPTION, raised by IEEE PASC Interpretation 1003.2 #122, which was overlooked in the first version of IEEE Std 1003.1-2001.1-2001.

, item XCU/TC2/D6/17 is applied, updating the EXTENDED DESCRIPTION (Internal Variables in *mailx*). The system start-up file is changed from ‘‘implementation-defined’’ to ‘‘unspecified’’ for consistency with other text in the EXTENDED DESCRIPTION.

make

Maintain, update, and regenerate groups of programs (**DEVELOPMENT**)

SD
```
make [–einpqrst][–f makefile]...[ –k| –S][macro=value]...
    [target_name...]
```

Derivation First released in Issue 2.

Issue 6 This utility is marked as part of the Software Development Utilities option.

The Open Group Corrigendum U029/1 is applied, correcting a typographical error in the SPECIAL TARGETS section.

In the ENVIRONMENT VARIABLES section, the *PROJECTDIR* description is updated from ‘‘otherwise, the home directory of a user of that name is examined’’ to ‘‘otherwise, the value of *PROJECTDIR* is treated as a user name and that user’s initial working directory is examined’’.

It is specified whether the command line is related to the makefile or to the *make* command, and the macro processing rules are updated to align with the IEEE P1003.2b draft standard.

PASC Interpretation 1003.2 #193 is applied, clarifying the wording regarding macro expansion and evaluation of macros.

man

Display system documentation

`man [–k] name...`

Derivation First released in Issue 4.

Issue 6 No functional changes in this issue.

mesg

Permit or deny messages

UP `mesg [y|n]`

Derivation First released in Issue 2.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

No functional changes in this issue.

mkdir

Make directories

`mkdir [–p][–m mode] dir...`

Derivation First released in Issue 2.

Issue 6 No functional changes in this issue.

mkfifo

Make FIFO special files

`mkfifo [–m mode] file...`

Derivation First released in Issue 3.

Issue 6 No functional changes in this issue.

more

Display files on a page-by-page basis

UP `more [–ceisu][–n number][–p command][–t tagstring][file ...]`

Derivation First released in Issue 4.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

The obsolescent SYNOPSIS is removed.

The utility has been extensively reworked for alignment with the IEEE P1003.2b draft standard:

- Changes have been made as a result of IEEE PASC Interpretations 1003.2 #37 and #109.
- The *more* utility should be able to handle underlined and emboldened displays of characters that are wider than a single column position.

mv

Move files

`mv [–fi] source_file target_file`

`mv [–fi] source_file... target_file`

Derivation First released in Issue 2.

Issue 6 The *mv* utility is changed to describe processing of symbolic links as specified in the IEEE P1003.2b draft standard.

The APPLICATION USAGE section is added.

newgrp

Change to a new group

UP `newgrp [–l][group`

Derivation First released in Issue 2.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

The obsolescent SYNOPSIS is removed.

The text describing supplemental groups is no longer conditional on {NGROUPS_MAX} being greater than 1. This is because {NGROUPS_MAX} now has a minimum value of 8. This is a FIPS requirement.

nice

Invoke a utility with an altered nice value

UP `nice [–n increment] utility [argument...]`

Derivation First released in Issue 4.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

The obsolescent SYNOPSIS is removed.

nl

Line numbering filter

XSI
```
nl [–p][–b type][–d delim][–f type][–h type][–i incr][–l num][–n format]
    [–s sep][–v startnum][–w width][file]
```

Derivation First released in Issue 2.

Issue 6 The obsolescent behavior allowing the options to be intermingled with the optional *file* operand is removed.

nm

Write the name list of an object file (**DEVELOPMENT**)

UP SD XSI
```
nm [–APv][–efox][ –g| –u][–t format] file...
```

Derivation First released in Issue 2.

Issue 6 This utility is marked as supported when both the User Portability Utilities option and the Software Development Utilities option are supported.

nohup

Invoke a utility immune to hangups

```
nohup utility [argument...]
```

Derivation First released in Issue 2.

Issue 6 No functional changes in this issue.

od

Dump files in various formats

```
od [–v][–A address_base][–j skip][–N count][–t type_string]...
    [file...]
```

XSI
```
od [–bcdosx][file] [[+]offset[.][b]]
```

Derivation First released in Issue 2.

Issue 6 The *od* utility is changed to remove the assumption that **short** was a two-byte entity, as per the revisions in the IEEE P1003.2b draft standard.

paste

Merge corresponding or subsequent lines of files

```
paste [–s][–d list] file...
```

Derivation First released in Issue 2.

Issue 6 No functional changes in this issue.

patch

Apply changes to files

UP

```
patch [–blNR][ –c| –e| –n][–d dir][–D define][–i patchfile]
    [–o outfile][–p num][–r rejectfile][file]
```

Derivation First released in Issue 4.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

The description of the –**D** option and the steps in Filename Determination (in the EXTENDED DESCRIPTION) are changed to match historical practice as defined in the IEEE P1003.2b draft standard.

, item XCU/TC1/D6/34 is applied, clarifying the way that the *patch* utility performs **ifdef** selection for the –**D** option.

pathchk

Check pathnames

```
pathchk [–p] pathname...
```

Derivation First released in Issue 4.

Issue 6 No functional changes in this issue.

pax

Portable archive interchange

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

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

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

```
pax –r –w[–diklntuvX][–H|–L][–p string]...[–s replstr]...
    [file...] directory
```

Derivation First released in Issue 4.

Issue 6 The *pax* utility is aligned with the IEEE P1003.2b draft standard:

- Support has been added for symbolic links in the options and interchange formats.
- A new format has been devised, based on extensions to **ustar**.
- References to the ‘‘extended’’ *tar* and *cpio* formats derived from IEEE Std 1003.1-1990 (POSIX.1) have been changed to remove the ‘‘extended’’ adjective because this could cause confusion with the extended *tar* header added in this revision. (All references to *tar* are actually to **ustar**).

IEEE PASC Interpretation 1003.2 #168 is applied, clarifying that *mkdir*( ) and *mkfifo*( ) calls can ignore an [EEXIST] error when extracting an archive.

The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

IEEE PASC Interpretation 1003.2 #180 is applied, clarifying how extracted files are created when in **read** mode.

IEEE PASC Interpretation 1003.2 #181 is applied, clarifying the description of the –**t** option.

IEEE PASC Interpretation 1003.2 #195 is applied.

IEEE PASC Interpretation 1003.2 #206 is applied, clarifying the handling of links for the –**H**, –**L**, and –**l** options.

, item XCU/TC1/D6/35 is applied, adding the process ID of the *pax* process into certain fields. This change provides a method for the implementation to ensure that different instances of *pax* extracting a file named **/a/b/foo** will not collide when processing the extended header information associated with **foo**.

, item XCU/TC1/D6/36 is applied, changing –**x B** to –**x** *pax* in the OPTIONS section.

, item XCU/TC2/D6/20 is applied, updating the SYNOPSIS to be consistent with the normative text.

, item XCU/TC2/D6/21 is applied, updating the DESCRIPTION to describe the behavior when files to be linked are symbolic links and the system is not capable of making hard links to symbolic links.

, item XCU/TC2/D6/22 is applied, updating the OPTIONS section to describe the behavior for how multiple –**odelete=***pattern* options are to be handled.

, item XCU/TC2/D6/23 is applied, updating the **write** option within the OPTIONS section.

, item XCU/TC2/D6/24 is applied, adding a paragraph into the OPTIONS section that states that specifying more than one of the mutually-exclusive options (–**H** and –**L**) is not considered an error and that the last option specified will determine the behavior of the utility.

, item XCU/TC2/D6/25 is applied, removing the *ctime* paragraph within the EXTENDED DESCRIPTION. There is a contradiction in the definition of the *ctime* keyword for the *pax* extended header, in that the *st_ctime* member of the **stat** structure does not refer to a file creation time. No field in the standard **stat** structure from **<sys/stat.h>** includes a file creation time.

, item XCU/TC2/D6/26 is applied, making it clear that *typeflag* 1 (**ustar** Interchange Format) applies not only to files that are hard-linked, but also to files that are aliased via symlinks.

, item XCU/TC2/D6/27 is applied, clarifying the *cpio c_nlink* field.

pr

Print files

```
pr [+page][–column][–adFmrt][–e[char][gap]][–h header][–i[char][gap]]
    [–l lines][–n[char][width]][–o offset][–s[char]][–w width][–fp]
    [file...]
```

XSI

Derivation First released in Issue 2.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The –**p** option is added.

printf

Write formatted output

```
printf format[argument...]
```

Derivation First released in Issue 4.

Issue 6 No functional changes in this issue.

prs

Print an SCCS file (**DEVELOPMENT**)

```
prs [–a][–d dataspec][–r[SID]] file...
```

XSI

```
prs [ –e| –l] –c cutoff [–d dataspec] file...
```

XSI

```
prs [ –e| –l] –r[SID][–d dataspec]file...
```

XSI

Derivation First released in Issue 2.

Issue 6 The normative text is reworded to emphasize the term ‘‘shall’’ for implementation requirements. No functional change is intended by this change.

The Open Group Base Resolution bwg2001-007 is applied, updating the table in STDOUT with a note that line statistics are capped at 99 999 for the **:Li:**, **:Ld:**, **:Lu:**, and **:DL:** keywords.

The Open Group Interpretation PIN4C.00009 is applied, clarifying that under Data Keywords in the STDOUT section, the *dataspec* for each selected delta does not include a trailing <newline>. The *dataspec* is thus now:

```
:Dt:\t:DL:\nMRs:\n:MR:COMMENTS:\n:C:
```

ps

Report process status

```
ps [–aA][–defl][–G grouplist][–o format]...[–p proclist][–t termlist]
[–U userlist][–g grouplist][–n namelist][–u userlist]
```

UP XSI

Derivation First released in Issue 2.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

pwd

Return working directory name

```
pwd [−L | −P ]
```

Derivation First released in Issue 2.

Issue 6 The −**P** and −**L** options are added to describe actions relating to symbolic links as specified in the IEEE P1003.2b draft standard.

qalter

Alter batch job

BE

```
qalter [−a date_time][−A account_string][−c interval][−e path_name]
    [−h hold_list][−j join_list][−k keep_list][−l resource_list]
    [−m mail_options][−M mail_list][−N name][−o path_name]
    [−p priority][−r y|n][−S path_name_list][−u user_list]
    job_identifier ...
```

This utility is part of the Batch Environment Services and Utilities option and may not be available on all implementations.

Derivation Derived from IEEE Std 1003.2d-1994.

Issue 6 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

IEEE PASC Interpretation 1003.2 #182 is applied, clarifying the description of the −**a** option.

qdel

Delete batch jobs

BE

```
qdel job_identifier ...
```

This utility is part of the Batch Environment Services and Utilities option and may not be available on all implementations.

Derivation Derived from IEEE Std 1003.2d-1994.

Issue 6 The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

qhold

Hold batch jobs

BE
```
qhold [–h hold_list] job_identifier ...
```

This utility is part of the Batch Environment Services and Utilities option and may not be available on all implementations.

Derivation Derived from IEEE Std 1003.2d-1994.

Issue 6 The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

qmove

Move batch jobs

BE
```
qmove destination job_identifier ...
```

This utility is part of the Batch Environment Services and Utilities option and may not be available on all implementations.

Derivation Derived from IEEE Std 1003.2d-1994.

Issue 6 The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

qmsg

Send message to batch jobs

BE
```
qmsg [–E][–O] message_string job_identifier ...
```

This utility is part of the Batch Environment Services and Utilities option and may not be available on all implementations.

Derivation Derived from IEEE Std 1003.2d-1994.

Issue 6 The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

qrerun

Rerun batch jobs

BE
```
qrerun job_identifier ...
```

This utility is part of the Batch Environment Services and Utilities option and may not be available on all implementations.

Derivation Derived from IEEE Std 1003.2d-1994.

Issue 6 The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

qrls

Release batch jobs

BE
```
qrls [–h hold_list] job_identifier ...
```

This utility is part of the Batch Environment Services and Utilities option and may not be available on all implementations.

Derivation Derived from IEEE Std 1003.2d-1994.

Issue 6 The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

qselect

Select batch jobs

BE
```
qselect [–a [op]date_time][–A account_string][–c [op]interval]
    [–h hold_list][–l resource_list][–N name][–p [op]priority]
    [–q destination][–r y|n][–s states][–u user_list]
```

This utility is part of the Batch Environment Services and Utilities option and may not be available on all implementations.

Derivation Derived from IEEE Std 1003.2d-1994.

Issue 6 No functional changes in this issue.

qsig

Signal batch jobs

BE
```
qsig [–s signal] job_identifier ...
```

This utility is part of the Batch Environment Services and Utilities option and may not be available on all implementations.

Derivation Derived from IEEE Std 1003.2d-1994.

Issue 6 The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

qstat

Show status of batch jobs

BE
```
qstat [–f] job_identifier ...
qstat –Q [–f] destination ...
qstat –B [–f] server_name ...
```

This utility is part of the Batch Environment Services and Utilities option and may not be available on all implementations.

Derivation Derived from IEEE Std 1003.2d-1994.

Issue 6 IEEE PASC Interpretation 1003.2 #191 is applied, removing the following ENVIRONMENT VARIABLES listed as affecting *qstat*: *COLUMNS*, *LINES*, *LOGNAME*, *TERM*, and *TZ*.

The *LC_TIME* entry is also removed from the ENVIRONMENT VARIABLES section.

qsub

Submit a script

BE
```
qsub [–a date_time][–A account_string][–c interval]
    [–C directive_prefix][–e path_name][–h][–j join_list][–k keep_list]
    [–m mail_options][–M mail_list][–N name]
    [–o path_name][–p priority][–q destination][–r y|n]
    [–S path_name_list][–u user_list][–v variable_list][–V]
    [–z][script]
```

This utility is part of the Batch Environment Services and Utilities option and may not be available on all implementations.

Derivation Derived from IEEE Std 1003.2d-1994.

Issue 6 The **–l** option has been removed as there is no portable description of the resources that are allowed or required by the batch job.

read

Read a line from standard input

```
read [–r] var...
```

Derivation First released in Issue 2.

Issue 6 No functional changes in this issue.

renice

Set nice values of running processes

UP
```
renice –n increment [–g | –p | –u] ID ...
```

Derivation First released in Issue 4.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

The APPLICATION USAGE section is added.

The obsolescent forms of the SYNOPSIS are removed.

Text previously conditional on POSIX_SAVED_IDS is mandatory in this issue. This is a FIPS requirement.

rm

Remove directory entries

`rm [–fiRr] file...`

Derivation First released in Issue 2.

Issue 6 Text is added to clarify actions relating to symbolic links as specified in the IEEE P1003.2b draft standard.

rmdel

Remove a delta from an SCCS file (**DEVELOPMENT**)

XSI `rmdel –r SID file...`

Derivation First released in Issue 2.

Issue 6 No functional changes are made in this issue.

rmdir

Remove directories

`rmdir [–p] dir...`

Derivation First released in Issue 2.

Issue 6 No functional changes are made in this issue.

sact

Print current SCCS file-editing activity (**DEVELOPMENT**)

XSI `sact file...`

Derivation First released in Issue 2.

Issue 6 The normative text is reworded to emphasize the term ‘‘shall’’ for implementation requirements. No functional change is intended by this change.

sccs

Front end for the SCCS subsystem (**DEVELOPMENT**)

XSI `sccs [–r][–d path][–p path] command [options...][operands...]`

Derivation First released in Issue 4.

Issue 6 In the ENVIRONMENT VARIABLES section, the *PROJECTDIR* description is updated from ‘‘otherwise, the home directory of a user of that name is examined’’ to ‘‘otherwise, the value of *PROJECTDIR* is treated as a user name and that user’s initial working directory is examined’’.

sed

Stream editor

`sed [–n] script[file...]`

`sed [–n][–e script]...[–f script_file]...[file...]`

Derivation First released in Issue 2.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- Implementations are required to support at least ten *wfile* arguments in an editing command.

The EXTENDED DESCRIPTION is changed to align with the IEEE P1003.2b draft standard.

IEEE PASC Interpretation 1003.2 #190 is applied.

IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the meaning of the backslash escape sequences in a replacement string for a BRE.

sh

Shell, the standard command language interpreter

`sh [–abCefhimnuvx][–o option][+abCefhmnuvx][+o option]`
`[command_file [argument...]]`

`sh –c[–abCefhimnuvx][–o option][+abCefhimnuvx][+o option]command_string`
`[command_name [argument...]]`

`sh –s[–abCefhimnuvx][–o option][+abCefhimnuvx][+o option][argument]`

Derivation First released in Issue 2.

Issue 6 The Open Group Corrigendum U029/2 is applied, correcting the second SYNOPSIS.

The Open Group Corrigendum U027/3 is applied, correcting a typographical error.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The option letters derived from the *set* special built-in are also accepted with a leading plus sign (`'+'`).
- Large file extensions are added:
  - — Pathname expansion does not fail due to the size of a file.
  - — Shell input and output redirections have an implementation-defined offset maximum that is established in the open file description.

In the ENVIRONMENT VARIABLES section, the text ‘‘user’s home directory’’ is updated to ‘‘directory referred to by the *HOME* environment variable’’.

Descriptions for the *ENV* and *PWD* environment variables are included to align with the IEEE P1003.2b draft standard.

sleep

Suspend execution for an interval

`sleep` *time*

Derivation First released in Issue 2.

Issue 6 No functional changes in this issue.

sort

Sort, merge, or sequence check text files

`sort [`–m`][`–o *output*`][`–bdfinru`][`–t *char*`][`–k *keydef*`]... [`*file*`...]`

`sort` –c `[`–bdfinru`][`–t *char*`][`–k *keydef*`][`*file*`]`

Derivation First released in Issue 2.

Issue 6 IEEE PASC Interpretation 1003.2 #174 is applied, updating the DESCRIPTION of comparisons.

IEEE PASC Interpretation 1003.2 #168 is applied.

split

Split files into pieces

UP `split [`–l *line_count*`][`–a *suffix_length*`][`*file*`[`*name*`]]`

`split` –b *n*`[`k|m`][`–a *suffix_length*`][`*file*`[`*name*`]]`

Derivation First released in Issue 2.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

The APPLICATION USAGE section is added.

The obsolescent SYNOPSIS is removed.

strings

Find printable strings in files

UP `strings [`–a`][`–t *format*`][`–n *number*`][`*file*`...]`

Derivation First released in Issue 4.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

The obsolescent SYNOPSIS is removed.

strip

Remove unnecessary information from executable files (**DEVELOPMENT**)

SD `strip file...`

Derivation First released in Issue 2.

Issue 6 This utility is marked as part of the Software Development Utilities option. This utility may not be present on all systems supporting the Single UNIX Specification.

No functional changes in this issue.

stty

Set the options for a terminal

`stty [ –a| –g]`

`stty operands`

Derivation First released in Issue 2.

Issue 6 The legacy items **iuclc**(–**iuclc**), **xcase**, **olcuc**(–**olcuc**), **lcase**(–**lcase**), and **LCASE**(–**LCASE**), are removed.

, item XCU/TC1/D6/37 is applied, applying IEEE PASC Interpretation 1003.2 #133, fixing an error in the OPERANDS section for the Combination Modes **nl**(–**nl**).

tabs

Set terminal tabs

UP XSI `tabs [ –n| –a| –a2| –c| –c2| –c3| –f| –p| –s| –u][+m[n]] [–T type]`

`tabs [–T type][ +[n]] n1[,n2,...]`

Derivation First released in Issue 2.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

No functional changes in this issue.

tail

Copy the last part of a file

`tail [–f][ –c number| –n number][file]`

Derivation First released in Issue 2.

Issue 6 The obsolescent SYNOPSIS lines and associated text are removed.

talk

Talk to another user

UP `talk` *address* **[***terminal***]**

Derivation First released in Issue 4.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

No functional changes in this issue.

tee

Duplicate standard input

`tee` **[**–ai**][***file*`...`**]**

Derivation First released in Issue 2.

Issue 6 IEEE PASC Interpretation 1003.2 #168 is applied.

test

Evaluate expression

`test` **[***expression***]**

`[` **[***expression***]** `]`

Derivation First released in Issue 2.

Issue 6 The –**h** operand is added for symbolic links, and access permission requirements are clarified for the –**r**, –**w**, and –**x** operands to align with the IEEE P1003.2b draft standard.

The –**L** and –**S** operands are added for symbolic links and sockets.

time

Time a simple command

UP `time` **[**–p**]** *utility* **[***argument*`...`**]**

Derivation First released in Issue 2.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

No functional changes in this issue.

touch

Change file access and modification times

```
touch [–acm][ –r ref_file| –t time] file...
```

Derivation First released in Issue 2.

Issue 6 The obsolescent *date_time* operand is removed.

The Open Group Corrigendum U027/1 is applied. This extends the range of valid time past the Epoch to at least the time 0 hours, 0 minutes, 0 seconds, January 1, 2038, Coordinated Universal Time. This is a new requirement on POSIX implementations.

The range for double leap seconds is changed from [00,61] to [00,60] to align with the ISO/IEC 9899: 1999 standard.

tput

Change terminal characteristics

UP

```
tput [–T type] operand...
```

Derivation First released in Issue 4.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

No functional changes in this issue.

tr

Translate characters

```
tr [–c | –C][–s] string1 string2
tr –s [–c | –C] string1
tr –d [–c | –C] string1
tr –ds [–c | –C] string1 string2
```

Derivation First released in Issue 2.

Issue 6 The **–C** operand is added, and the description of the **–c** operand is changed to align with the IEEE P1003.2b draft standard.

true

Return true value

```
true
```

Derivation First released in Issue 2.

Issue 6 No functional changes in this issue.

tsort

Topological sort

XSI `tsort [file]`

Derivation First released in Issue 2.

Issue 6 No functional changes in this issue.

tty

Return user's terminal name

`tty`

Derivation First released in Issue 2.

Issue 6 The obsolescent **–s** option is removed.

type

Write a description of command type

XSI `type name...`

Derivation First released in Issue 2.

Issue 6 No functional changes in this issue.

ulimit

Set or report file size limit

XSI `ulimit [–f][blocks]`

Derivation First released in Issue 2.

Issue 6 No functional changes in this issue.

umask

Get or set the file mode creation mask

`umask [–S][mask]`

Derivation First released in Issue 2.

Issue 6 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The octal mode is supported.

unalias

Remove alias definitions

UP
```
unalias alias-name...
unalias –a
```

Derivation First released in Issue 4.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

No functional changes in this issue.

uname

Return system name

```
uname [–snrvma]
```

Derivation First released in Issue 2.

Issue 6 No functional changes in this issue.

uncompress

Expand compressed data

XSI
```
uncompress [–cfv][file...]
```

Derivation First released in Issue 4.

Issue 6 No functional changes in this issue.

unexpand

Convert spaces to tabs

UP
```
unexpand [ –a| –t tablist][file...]
```

Derivation First released in Issue 4.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

The definition of the *LC_CTYPE* environment variable is changed to align with the IEEE P1003.2b draft standard.

unget

Undo a previous get of an SCCS file (**DEVELOPMENT**)

XSI `unget [–ns][–r SID] file...`

Derivation First released in Issue 2.

Issue 6 No functional changes are made in this issue.

uniq

Report or filter out repeated lines in a file

`uniq [–c|–d|–u][–f fields][–s char][input_file [output_file]]`

Derivation First released in Issue 2.

Issue 6 The obsolescent SYNOPSIS and associated text are removed.

unlink

Call the *unlink*( ) function

XSI `unlink file`

Derivation First released in Issue 5.

Issue 6 No functional changes in this issue.

uucp

System-to-system copy

XSI `uucp [–cCdfjmr][–n user] source-file... destination-file`

Derivation First released in Issue 2.

Issue 6 The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

The UN margin codes and associated shading are removed from the –**C**, –**f**, –**j**, –**n**, and –**r** options in response to The Open Group Base Resolution bwg2001-003.

uudecode

Decode a binary file

UP `uudecode [–o outfile][file]`

Derivation First released in Issue 4.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

The –**o** *outfile* option is added, as specified in the IEEE P1003.2b draft standard.

, item XCU/TC2/D6/35 is applied, clarifying in the DESCRIPTION that the initial mode used if either of the *op* characters is `'+'` or `'–'` is unspecified.

uuencode

Encode a binary file

UP
```
uuencode [–m][file] decode_pathname
```

Derivation First released in Issue 4.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

The Base64 algorithm and the ability to output to **/dev/stdout** are added as specified in the IEEE P1003.2b draft standard.

uustat

uucp status inquiry and job control

XSI
```
uustat [ –q| –k jobid| –r jobid]
uustat [–s system][–u user]
```

Derivation First released in Issue 2.

Issue 6 The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

The UN margin code and associated shading are removed from the –**q** option in response to The Open Group Base Resolution bwg2001-003.

uux

Remote command execution

XSI
```
uux [–np] command–string
uux [–jnp] command–string
```

Derivation First released in Issue 2.

Issue 6 The obsolescent SYNOPSIS is removed.

The UN margin code and associated shading are removed from the –**j** option in response to The Open Group Base Resolution bwg2001-003.

val

Validate SCCS files (**DEVELOPMENT**)

XSI
```
val –
val [–s][–m name][–r SID][–y type] file...
```

Derivation First released in Issue 2.

Issue 6 The Open Group Corrigendum U025/4 is applied, correcting a typographical error in the EXIT STATUS.

The normative text is reworded to emphasize the term ‘‘shall’’ for implementation requirements. No functional change is intended by this change.

vi

Screen-oriented (visual) display editor

UP `vi [–rR][–l][–c command][–t tagstring][–w size][file ...]`

Derivation First released in Issue 2.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

The APPLICATION USAGE section is added.

The obsolescent SYNOPSIS is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The **reindent** command description is added.

The *vi* utility has been extensively rewritten for alignment with the IEEE P1003.2b draft standard.

IEEE PASC Interpretations 1003.2 #57, #62, #63, #64, #78, and #188 are applied.

IEEE PASC Interpretation 1003.2 #207 is applied, clarifying the description of the **R** command in a manner similar to the descriptions of other text input mode commands such as **i**, **o**, and **O**.

wait

Await process completion

`wait [pid...]`

Derivation First released in Issue 2.

Issue 6 No functional changes in this issue.

wc

Word, line, and byte or character count

`wc [–c|–m][–lw][file...]`

Derivation First released in Issue 2.

Issue 6 No functional changes in this issue.

what

Identify SCCS files (**DEVELOPMENT**)

XSI
```
what [–s] file...
```

Derivation First released in Issue 2.

Issue 6 No functional changes in this issue.

who

Display who is on the system

UP
```
who [–mTu]
```

XSI
```
who [–mu]–s[–bHlprt][file]
who [–mTu][–abdHlprt][file]
who –q [file]
who am i
who am I
```

Derivation First released in Issue 2.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

write

Write to another user

UP
```
write user_name [terminal]
```

Derivation First released in Issue 2.

Issue 6 This utility is marked as part of the User Portability Utilities option. Support for the User Portability Utilities option is mandatory on implementations supporting the Single UNIX Specification.

xargs

Construct argument lists and invoke utility

XSI
```
xargs [–t][–p]][–E eofstr][–I replstr][–L number][–n number [–x]]
    [–s size][utility [argument...]]
```

Derivation First released in Issue 2.

Issue 6 The obsolescent –**e**, –**i**, and –**l** options are removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The –**p** option is added.
- In the INPUT FILES section, the file **/dev/tty** is used to read responses required by the –**p** option.
- The STDERR section is updated to describe the –**p** option.

The description of the –**E** option is aligned with ISO/IEC 9945-2: 1993 (POSIX-2).

yacc

Yet another compiler compiler (**DEVELOPMENT**)

CD `yacc [–dltv][–b file_prefix][–p sym_prefix] grammar`

Derivation First released in Issue 2.

Issue 6 This utility is marked as part of the C-Language Development Utilities option.

Minor changes have been added to align with the IEEE P1003.2b draft standard.

IEEE PASC Interpretation 1003.2 #177 is applied, changing the comment on **RCURL** from the **}%** token to the **%}**.

zcat

Expand and concatenate data

XSI `zcat [file...]`

Derivation First released in Issue 4.

Issue 6 No functional changes in this issue.

Chapter 15

Headers Migration

This chapter contains a section for each header defined in XBD, Issue 6. Each section contains the SYNOPSIS and identifies syntax and semantic changes made to the header in Issue 6 (if any), complete with examples where appropriate. Only changes that might affect an application programmer are identified.

aio.h

Asynchronous input and output (**REALTIME**)

AIO
```
#include <aio.h>
```

Derivation 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 and need not be available in all implementations.

The description of the constants is expanded.

No functional changes in this issue except that the **restrict** keyword is added to the prototype for *lio_listio*( ):

```
int lio_listio(int, struct aiocb *restrict const[restrict], int,
    struct sigevent *restrict);
```

arpa/inet.h

Definitions for internet operations

```
#include <arpa/inet.h>
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 This is a new header included for alignment with the XNS, Issue 5.2 specification. It contains definitions to support internet operations.

This header defines the types **in_port_t** and **in_addr_t** and the **in_addr** structure.

The INET_ADDRSTRLEN and INET6_ADDRSTRLEN macros are defined.

The following are defined as macros or functions or both: *htonl*( ), *htons*( ), *ntohl*( ), *ntohs*( ).

This header includes declarations for the following functions: *inet_addr*( ), *inet_ntoa*( ), *inet_ntop*( ), and *inet_pton*( ).

A change over the XNS, Issue 5.2 specification is that the **restrict** keyword is added to the prototypes for *inet_ntop*( ) and *inet_pton*( ):

```
const char *inet_ntop(int, const void *restrict,
      char *restrict, socklen_t);
int   inet_pton(int, const char *restrict, void *restrict);
```

assert.h

Verify program assertion

```
#include <assert.h>
```

Derivation 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.

complex.h

Complex arithmetic

```
#include <complex.h>
```

Derivation First released in Issue 6. Included for alignment with the ISO/IEC 9899: 1999 standard.

Issue 6 This is a new header included for alignment with the ISO/IEC 9899: 1999 standard. Complex types have been added to the ISO/IEC 9899: 1999 standard as part of the effort to make C suitable and attractive for general numerical programming. Complex arithmetic is used heavily in certain important application areas.

This header defines the macros complex and _Complex_I.

This header includes declarations for the following functions: *cabs*( ), *cabsf*( ), *cabsl*( ), *cacos*( ), *cacosf*( ), *cacosh*( ), *cacoshf*( ), *cacoshl*( ), *cacosl*( ), *carg*( ), *cargf*( ), *cargl*( ), *casin*( ), *casinf*( ), *casinh*( ), *casinhf*( ), *casinhl*( ), *casinl*( ), *catan*( ), *catanf*( ), *catanh*( ), *catanhf*( ), *catanhl*( ), *catanl*( ), *ccos*( ), *ccosf*( ), *ccosh*( ), *ccoshf*( ), *ccoshl*( ), *ccosl*( ), *cexp*( ), *cexpf*( ), *cexpl*( ), *cimag*( ), *cimagf*( ), *cimagl*( ), *clog*( ), *clogf*( ), *clogl*( ), *conj*( ), *conjf*( ), *conjl*( ), *cpow*( ), *cpowf*( ), *cpowl*( ), *cproj*( ), *cprojf*( ), *cprojl*( ), *creal*( ), *crealf*( ), *creall*( ), *csin*( ), *csinf*( ), *csinh*( ), *csinhf*( ), *csinhl*( ), *csinl*( ), *csqrt*( ), *csqrtf*( ), *csqrtl*( ), *ctan*( ), *ctanf*( ), *ctanh*( ), *ctanhf*( ), *ctanhl*( ), *ctanl*( ).

cpio.h

Cpio archive values

XSI
```
#include <cpio.h>
```

Derivation First released in Issue 3. Derived from IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 No functional changes in this issue.

The SEE ALSO is updated to refer to *pax*, since the *cpio* utility is not included in XCU, Issue 6.

ctype.h

Character types

```
#include <ctype.h>
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes in this issue.

Extensions beyond the ISO C standard are marked.

dirent.h

Format of directory entries

```
#include <dirent.h>
```

Derivation First released in Issue 2.

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*( ):

```
int readdir_r(DIR *restrict, struct dirent *restrict,
    struct dirent **restrict);
```

dlfcn.h

Dynamic linking

XSI

```
#include <dlfcn.h>
```

Derivation First released in Issue 5.

Issue 6 No functional changes in this issue except that the **restrict** keyword is added to the prototype for *dlsym*( ):

```
void *dlsym(void *restrict, const char *restrict);
```

errno.h

System error numbers

```
#include <errno.h>
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

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.

fcntl.h

File control options

```
#include <fcntl.h>
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The following changes are made for alignment with ISO/IEC 9945-1: 1996 (POSIX-1):

- O_DSYNC and O_RSYNC are marked as part of the Synchronized Input and Output option and need not be available in all implementations.

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. Note however that IEEE PASC Interpretation 1003.1 #102 is applied, moving the prototype for *posix_madvise*() to **<sys/mman.h>**:

```
int posix_fadvise(int, off_t, off_t, int);
int posix_fallocate(int, off_t, off_t);
```

, item XBD/TC2/D6/18 is applied, updating the protoypes for *posix_fadvise*() and *posix_fallocate*() to be large file-aware, using **off_t** instead of **size_t**.

fenv.h

Floating-point environment

```
#include <fenv.h>
```

Derivation — First released in Issue 6. Included for alignment with the ISO/IEC 9899: 1999 standard.

Issue 6 — This is a new header included for alignment with the ISO/IEC 9899: 1999 standard. This header provides definitions for the floating-point environment.

This header defines the data types **fenv_t** and **fexcept_t**.

This header defines the following macros and constants:

FE_DIVBYZERO
FE_INEXACT
FE_INVALID
FE_OVERFLOW
FE_UNDERFLOW
FE_ALL_EXCEPT
FE_DOWNWARD
FE_TONEAREST
FE_TOWARDZERO
FE_UPWARD
FE_DFL_ENV

This header includes declarations for the following functions: *feclearexcept*(), *fegetexceptflag*(), *feraiseexcept*(), *fesetexceptflag*(), *fetestexcept*(), *fegetround*(), *fesetround*(), *fegetenv*(), *feholdexcept*(), *fesetenv*(), *feupdateenv*().

The following change has been made for alignment with the ISO/IEC 9899: 1999 standard, Defect Report 202:

- The return types for *feclearexcept*(), *fegetexcepflag*(), *feraiseexcept*(), *fesetexceptflag*(), *fegetenv*(), *fesetenv*(), and *feupdateenv*() are changed from **void** to **int**.

float.h

Floating types

```
#include <float.h>
```

Derivation First released in Issue 4. Derived from the ISO C standard.

Issue 6 No functional changes in this issue, although the description of the operations with floating-point values has been reworked more closely to align with the description and layout of the ISO/IEC 9899: 1999 standard.

fmtmsg.h

Message display structures

XSI
```
#include <fmtmsg.h>
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes in this issue.

fnmatch.h

Filename-matching types

```
#include <fnmatch.h>
```

Derivation First released in Issue 4. Derived from ISO/IEC 9945-2: 1993 (POSIX-2).

Issue 6 The constant FNM_NOSYS is marked obsolescent.

ftw.h

File tree traversal

XSI
```
#include <ftw.h>
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes in this issue.

glob.h

Pathname pattern-matching types

```
#include <glob.h>
```

Derivation First released in Issue 4. Derived from ISO/IEC 9945-2: 1993 (POSIX-2).

Issue 6 The **restrict** keyword is added to the prototype for *glob*( ):

```
int glob(const char *restrict, int,
    int (*)(const char *, int), glob_t *restrict);
```

The constant GLOB_NOSYS is marked obsolescent.

, item XBD/TC1/D6/8 is applied, correcting the *glob*( ) prototype definition by removing the **restrict** qualifier from the function pointer argument.

grp.h

Group structure

```
#include <grp.h>
```

Derivation First released in Issue 1.

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 and need not be available in all implementations.

iconv.h

Codeset conversion facility

XSI
```
#include <iconv.h>
```

Derivation First released in Issue 4.

Issue 6 The **restrict** keyword is added to the prototype for *iconv*():

```
size_t iconv(iconv_t, char **restrict, size_t *restrict,
       char **restrict, size_t *restrict);
```

inttypes.h

Fixed size integer types

```
#include <inttypes.h>
```

Derivation First released in Issue 5.

Issue 6 The purpose of this header 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.

The Open Group Base Resolution bwg97-006 is applied.

This reference page is updated to align with the ISO/IEC 9899: 1999 standard.

iso646.h

Alternative spellings

```
#include <iso646.h>
```

Derivation First released in Issue 5. Derived from ISO C Amendment 1 (MSE).

Issue 6 No functional changes in this issue.

langinfo.h

Language information constants

```
#include <langinfo.h>
```

Derivation First released in Issue 2.

Issue 6 The constants YESSTR and NOSTR are removed. These were previously marked LEGACY in Issue 5.

, item XBD/TC1/D6/9 is applied, adding a sentence to the‘‘Meaning’’ column entry for the CRNCYSTR constant. This change is to accommodate historic practice.

libgen.h

Definitions for pattern matching functions

XSI
```
#include <libgen.h>
```

Derivation First released in Issue 4, Version 2.

Issue 6 The __**loc1** symbol and the *regcmp*() and *regex*() functions are removed. These were previously marked LEGACY in Issue 5.

limits.h

Implementation-defined constants

```
#include <limits.h>
```

Derivation First released in Issue 1.

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}.

{_POSIX_TZNAME_MAX} is changed from 3 to 6.

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.

, item XBD/TC1/D6/10 is applied, correcting the value of {_POSIX_CHILD_MAX} from 6 to 25. This is for FIPS 151-2 alignment.

, item XBD/TC2/D6/19 is applied, updating the values for {INT_MAX}, {UINT_MAX}, and {INT_MIN} to be CX extensions over the ISO C standard, and correcting {WORD_BIT} from 16 to 32.

, item XBD/TC2/D6/20 is applied, removing {CHARCLASS_NAME_MAX} from the ‘‘Other Invariant Values’’ section (it also occurs under ‘‘Runtime Increasable Values’’).

locale.h

Category macros

```
#include <locale.h>
```

Derivation First released in Issue 3.

Issue 6 The **lconv** structure is expanded with the following new members for alignment with the ISO/IEC 9899: 1999 standard: **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**.

Extensions beyond the ISO C standard are marked.

math.h

Mathematical declarations

```
#include <math.h>
```

Derivation First released in Issue 1.

Issue 6 This reference page is updated to align with the ISO/IEC 9899: 1999 standard.

The following types are added: **float_t** and **double_t**.

The following constants and macros are added:

HUGE_VALF
HUGE_VALD
INFINITY
NAN
FP_INFINITE
FP_NAN
FP_NORMAL
FP_SUBNORMAL
FP_ZERO
FP_FAST_FMA
FP_FAST_FMAF
FP_FAST_FMAL
FP_ILOGB0
FP_ILOGBNAN
MATH_ERRNO
MATH_ERREXCEPT
math_errhandling

The following macros are added:

fpclassify(), *isfinite*(), *isinf*(), *isnan*(), *isnormal*(), *signbit*(), *isgreater*(), *isgreaterequal*(), *isless*(), *islessequal*(), *islessgreater*(), *isunordered*()

The following function declarations are added:

acosf(), *acoshf*(), *acoshl*(), *acosl*(), *asinf*(), *asinhf*(), *asinhl*(), *asinl*(), *atan2f*(), *atan2l*(), *atanf*(), *atanhf*(), *atanhl*(), *atanl*(), *cbrt*(), *cbrtf*(), *cbrtl*(), *ceilf*(), *ceill*(), *copysign*(), *copysignf*(), *copysignl*(), *cosf*(), *coshf*(), *coshl*(), *cosl*(), *erfcf*(), *erfcl*(), *erff*(), *erfl*(), *exp2*(), *exp2f*(), *exp2l*(), *expf*(), *expl*(), *expm1f*(), *expm1l*(), *fabsf*(), *fabsl*(), *fdim*(), *fdimf*(), *fdiml*(), *floorf*(), *floorl*(), *fma*(), *fmaf*(), *fmal*(), *fmax*(), *fmaxf*(), *fmaxl*(), *fmin*(), *fminf*(), *fminl*(), *fmodf*(), *fmodl*(), *frexpf*(), *frexpl*(), *hypotf*(), *hypotl*(), *ilogbf*(), *ilogbl*(), *lrint*(), *lrintf*(), *lrintl*(), *lround*(), *lroundf*(), *lroundl*(), *ldexpf*(), *ldexpl*(), *lgammaf*(), *lgammal*(), *log10f*(), *log10l*(), *log1pf*(), *log1pl*(), *log2*(), *log2f*(), *log2l*(), *logbf*(), *logbl*(), *logf*(), *logl*(), *llrint*(), *llrintf*(), *llrintl*(), *llround*(), *llroundf*(), *llroundl*(), *modff*(), *modfl*(), *nan*(), *nanf*(), *nanl*(), *nearbyint*(), *nearbyintf*(), *nearbyintl*(), *nextafterf*(), *nextafterl*(), *nexttoward*(), *nexttowardf*(), *nexttowardl*(), *powf*(), *powl*(), *remainderf*(), *remainderl*(), *remquo*(), *remquof*(), *remquol*(), *rintf*(), *rintl*(), *round*(), *roundf*(), *roundl*(), *scalbln*(), *scalblnf*(), *scalblnl*(), *scalbn*(), *scalbnf*(), *scalbnl*(), *sinf*(), *sinhf*(), *sinhl*(), *sinl*(), *sqrtf*(), *sqrtl*(), *tanf*(), *tanhf*(), *tanhl*(), *tanl*(), *tgamma*(), *tgammaf*(), *tgammal*(), *trunc*(), *truncf*(), *truncl*()

, item XBD/TC2/D6/21 is applied, making it clear that the meaning of the FP_FAST_FMA macro is unspecified if the macro is undefined.

monetary.h

Monetary types

XSI
```
#include <monetary.h>
```

Derivation First released in Issue 4.

Issue 6 The **restrict** keyword is added to the prototype for *strfmon*():

```
ssize_t strfmon(char *restrict, size_t,
    const char *restrict, ...);
```

mqueue.h

Message queues (**REALTIME**)

MSG
```
#include <mqueue.h>
```

Derivation 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 and need not be available in all implementations.

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*():

```
int     mq_setattr(mqd_t, const struct mq_attr *restrict,
        struct mq_attr *restrict);
ssize_t mq_timedreceive(mqd_t, char *restrict, size_t,
        unsigned *restrict, const struct timespec *restrict);
```

ndbm.h

Definitions for ndbm database operations

XSI
```
#include <ndbm.h>
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes in this issue.

net/if.h

Sockets local interfaces

```
#include <net/if.h>
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 This is a new header included for alignment with the XNS, Issue 5.2 specification. It contains information for sockets local interfaces. There have been no changes over the XNS, Issue 5.2 specification.

This header declares the **if_nameindex** structure and the IF_NAMESIZE macro for manipulating interface names.

The following functions are declared: *if_nametoindex*( ), *if_indextoname*( ), *if_nameindex*( ), and *if_freenameindex*( ).

netdb.h

Definitions for network database operations

```
#include <netdb.h>
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 This is a new header included for alignment with 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.

, item XBD/TC1/D6/11 is applied, adding a description of the NI_NUMERICSCOPE macro and correcting the *getnameinfo*( ) function prototype. These changes are for alignment with IPv6.

netinet/in.h

Internet address family

```
#include <netinet/in.h>
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 This is a new header included for alignment with 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.

, item XBD/TC1/D6/12 is applied, adding **const** qualifiers to the *in6addr_any* and *in6addr_loopback* external variables.

, item XBD/TC2/D6/22 is applied, making it clear which structure members are in network byte order.

netinet/tcp.h

Definitions for the Internet Transmission Control Protocol (TCP)

```
#include <netinet/tcp.h>
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 This is a new header included for alignment with the XNS, Issue 5.2 specification.

nl_types.h

Data types

XSI

```
#include <nl_types.h>
```

Derivation First released in Issue 2.

Issue 6 No functional changes in this issue.

poll.h

Definitions for the *poll*( ) function

XSI

```
#include <poll.h>
```

Derivation 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.

pthread.h

Threads

THR

```
#include <pthread.h>
```

Derivation 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*( ), *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 and need not be available in all implementations.

The **restrict** keyword is added to the following prototypes:

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*( ), *pthread_sigmask*( )

as follows:

```
int pthread_attr_getguardsize(const pthread_attr_t *restrict,
    size_t *restrict);
int pthread_attr_getinheritsched(const pthread_attr_t *restrict,
    int *restrict);
int pthread_attr_getschedparam(const pthread_attr_t *restrict,
    struct sched_param *restrict);
int pthread_attr_getschedpolicy(const pthread_attr_t *restrict,
    int *restrict);
int pthread_attr_getscope(const pthread_attr_t *restrict,
    int *restrict);
int pthread_attr_getstack(const pthread_attr_t *restrict,
    void **restrict, size_t *restrict);
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_setschedparam(pthread_attr_t *restrict,
    const struct sched_param *restrict);
int pthread_barrier_init(pthread_barrier_t *restrict,
    const pthread_barrierattr_t *restrict, unsigned);
int pthread_barrierattr_getpshared( \
    const pthread_barrierattr_t *restrict, int *restrict);
```

```
int pthread_cond_init(pthread_cond_t *restrict,
    const pthread_condattr_t *restrict);
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_getclock(
    const pthread_condattr_t *restrict, clockid_t *restrict);
int pthread_condattr_getpshared(
    const pthread_condattr_t *restrict, int *restrict);
int pthread_create(pthread_t *restrict,
    const pthread_attr_t *restrict, void *(*)(void *),
    void *restrict);
int pthread_getschedparam(pthread_t, int *restrict,
    struct sched_param *restrict);
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_setprioceiling(pthread_mutex_t *restrict,
    int, int *restrict);
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);
int pthread_mutexattr_gettype(
    const pthread_mutexattr_t *restrict, int *restrict);
int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *, int);
int pthread_mutexattr_setprotocol(pthread_mutexattr_t *, int);
int pthread_rwlock_init(pthread_rwlock_t *restrict,
    const pthread_rwlockattr_t *restrict);
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_rwlockattr_getpshared(
    const pthread_rwlockattr_t *restrict, int *restrict);
```

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.

, item XBD/TC1/D6/13 is applied, correcting shading errors that were in contradiction with the System Interfaces volume of IEEE Std 1003.1-2001.

pwd.h

Password structure

```
#include <pwd.h>
```

Derivation First released in Issue 1.

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 and need not be available in all implementations.

regex.h

Regular expression matching types

```
#include <regex.h>
```

Derivation First released in Issue 4. Derived from ISO/IEC 9945-2: 1993 (POSIX-2).

Issue 6 The REG_ENOSYS constant is marked obsolescent.

The **restrict** keyword is added to the prototypes for *regcomp*( ), *regerror*( ), and *regexec*( ):

```
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);
```

A statement is added that the **size_t** type is defined as described in **<sys/types.h>**.

sched.h

Execution scheduling (**REALTIME**)

PS
```
#include <sched.h>
```

Derivation 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 and need not be available in all implementations.

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**.

, items XSH/TC1/D6/52 and XSH/TC1/D6/53 are applied, aligning the function prototype shading and margin codes with the System Interfaces volume of IEEE Std 1003.1-2001.

, item XBD/TC2/D6/23 is applied, updating the DESCRIPTION to differentiate between thread and process execution.

search.h

Search tables

XSI
```
#include <search.h>
```

Derivation 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*( ).

semaphore.h

Semaphores (**REALTIME**)

SEM
```
#include <semaphore.h>
```

Derivation 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 and need not be available in all implementations.

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*( ).

setjmp.h

Stack environment declarations

```
#include <setjmp.h>
```

Derivation First released in Issue 1.

Issue 6 No functional changes in this issue.

Extensions beyond the ISO C standard are marked.

signal.h

Signals

```
#include <signal.h>
```

Derivation First released in Issue 1.

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 changed to RTS to denote that the semantics are part of the Realtime Signals Extension option and need not be available in all implementations.

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 marked.

, item XBD/TC1/D6/14 is applied, changing the descriptive text for members of **struct sigaction**.

, item XBD/TC1/D6/15 is applied, correcting the definition of the *sa_sigaction* member of **struct sigaction**.

, item XBD/TC2/D6/24 is applied, reworking the ordering of the **siginfo_t** type structure in the DESCRIPTION. This is an editorial change and no normative change is intended.

spawn.h

Spawn (**ADVANCED REALTIME**)

SPN

```
#include <spawn.h>
```

Derivation First released in Issue 6. Included for alignment with IEEE Std 1003.1d-1999.

Issue 6 This is a new header included for alignment with IEEE Std 1003.1d-1999.

The following change is made over IEEE Std 1003.1d-1999:

- The **restrict** keyword is added to the following prototypes:

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*( ), *posix_spawnp*( )

as follows:

```
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_addopen(
    posix_spawn_file_actions_t *restrict, int,
    const char *restrict, int, mode_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_setsigdefault(posix_spawnattr_t *restrict,
    const sigset_t *restrict);
int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict,
    const struct sched_param *restrict);
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]);
```

stdarg.h

Handle variable argument list

```
#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);
```

Derivation First released in Issue 4. Derived from ANSI standard X3.159-1989, Programming Language C (ANSI C).

Issue 6 The new function *va_copy*() is added to copy a variable argument list. This is added for alignment with the ISO/IEC 9899: 1999 standard.

stdbool.h

Boolean type and values

```
#include <stdbool.h>
```

Derivation First released in Issue 6. Included for alignment with the ISO/IEC 9899: 1999 standard.

Issue 6 This is a new header included for alignment with the ISO/IEC 9899: 1999 standard. This header contains a **typedef** for bool, and macros for true, false, and __bool_true_false_are_defined.

stddef.h

Standard type definitions

```
#include <stddef.h>
```

Derivation First released in Issue 4. Derived from ANSI standard X3.159-1989, Programming Language C (ANSI C).

Issue 6 No functional changes in this issue.

stdint.h

Integer types

```
#include <stdint.h>
```

Derivation First released in Issue 6. Included for alignment with the ISO/IEC 9899: 1999 standard.

Issue 6 This is a new header included for alignment with the ISO/IEC 9899: 1999 standard.

This header is a subset of **<inttypes.h>**, the rationale being that this subset is more suitable for use in environments that might not support the formatted I/O functions, thereby permitting those environments to avoid defining a large number of macros.

ISO/IEC 9899: 1999 standard, Technical Corrigendum No. 1 is incorporated.

stdio.h

Standard buffered input/output

```
#include <stdio.h>
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

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 Thread-Safe Functions option and need not be available in all implementations.

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 marked.

stdlib.h

Standard library definitions

```
#include <stdlib.h>
```

Derivation First released in Issue 3.

Issue 6 The Open Group Corrigendum U021/1 is applied, correcting the prototype for *realpath*() to be consistent with the reference page. The **restrict** keyword is also added to the prototype for *realpath*():

```
char *realpath(const char *restrict, char *restrict);
```

The Open Group Corrigendum U028/13 is applied, correcting the prototype for *putenv*() to be consistent with the reference page:

```
int putenv(char *);
```

The *rand_r*() function is marked as part of the Thread-Safe Functions option and need not be available in all implementations.

Function prototypes for *setenv*() and *unsetenv*() are added:

```
int setenv(const char *, const char *, int);
int unsetenv(const char *);
```

The *posix_memalign*() function is added for alignment with IEEE Std 1003.1d-1999:

```
int posix_memalign(void **, size_t, size_t);
```

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 individual reference pages should be consulted for information on alternative recommended interfaces.

The *ttyslot*() and *valloc*() functions are removed as they were previously marked LEGACY.

Extensions beyond the ISO C standard are marked.

string.h

String operations

```
#include <string.h>
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 The *strtok_r*() function is marked as part of the Thread-Safe Functions option and need not be available in all implementations.

This reference page is updated to align with the ISO/IEC 9899: 1999 standard.

strings.h

String operations

XSI
```
#include <strings.h>
```

Derivation 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. The individual reference pages should be consulted for information on alternative recommended interfaces.

stropts.h

STREAMS interface (**STREAMS**)

XSR
```
#include <stropts.h>
```

Derivation First released in Issue 4, Version 2.

Issue 6 This header is marked as part of the XSI STREAMS Option Group and may not be available on all implementations.

The **restrict** keyword is added to the prototypes for *getmsg*( ) and *getpmsg*( ):

```
int getmsg(int, struct strbuf *restrict,
    struct strbuf *restrict, int *restrict);
int getpmsg(int, struct strbuf *restrict,
    struct strbuf *restrict, int *restrict, int *restrict);
```

sys/ipc.h

XSI interprocess communication access structure

XSI
```
#include <sys/ipc.h>
```

Derivation First released in Issue 2. Derived from System V Release 2.0.

Issue 6 No functional changes in this issue.

sys/mman.h

Memory management declarations

```
#include <sys/mman.h>
```

Derivation First released in Issue 4, Version 2.

Issue 6 The **<sys/mman.h>** header is marked as dependent on support for either the Memory Mapped Files, Process Memory Locking, or Shared Memory Objects 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 following flags are added:

  POSIX_TYPED_MEM_ALLOCATE
  POSIX_TYPED_MEM_ALLOCATE_CONTIG
  POSIX_TYPED_MEM_MAP_ALLOCATABLE

- The **posix_tmi_length** structure is added.

- The *posix_mem_offset*(), *posix_typed_mem_get_info*(), and *posix_typed_mem_open*() functions are added, with the **restrict** keyword added to the prototype for *posix_mem_offset*():

```
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);
```

IEEE PASC Interpretation 1003.1 #102 is applied, adding the prototype for *posix_madvise*():

```
int posix_madvise(void *, size_t, int);
```

, item XSH/TC1/D6/34 is applied, changing the margin code for the *mmap*() function from MF|SHM to MC3 (notation for MF|SHM|TYM).

, item XSH/TC1/D6/36 is applied, changing the margin code for the *munmap*() function from MF|SHM to MC3 (notation for MF|SHM|TYM).

sys/msg.h

XSI message queue structures

XSI
```
#include <sys/msg.h>
```

Derivation First released in Issue 2. Derived from System V Release 2.0.

Issue 6 No functional changes in this issue.

sys/resource.h

Definitions for XSI resource operations

XSI
```
#include <sys/resource.h>
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes in this issue.

sys/select.h

Select types

```
#include <sys/select.h>
```

Derivation First released in Issue 6. Derived from IEEE Std 1003.1g-2000.

Issue 6 This is a new header included for alignment with IEEE Std 1003.1g-2000. It includes many of the definitions that were previously found in the **<sys/time.h>** header. That header is now allowed to include the **<sys/select.h>** header.

A change from the Single UNIX Specification is that 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.

The *pselect*() function is new. The **restrict** keyword has been added to the prototypes for *select*() and *pselect*():

```
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);
```

sys/sem.h

XSI semaphore facility

XSI
```
#include <sys/sem.h>
```

Derivation First released in Issue 2. Derived from System V Release 2.0.

Issue 6 No functional changes in this issue.

sys/shm.h

XSI shared memory facility

XSI
```
#include <sys/shm.h>
```

Derivation First released in Issue 2. Derived from System V Release 2.0.

Issue 6 No functional changes in this issue.

sys/socket.h

Main sockets header

```
#include <sys/socket.h>
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 This is a new header included for alignment with the XNS, Issue 5.2 specification.

Changes over the XNS, Issue 5.2 specification include that the **restrict** keyword is added to the prototypes for *accept*(), *getpeername*(), *getsockname*(), *getsockopt*(), and *recvfrom*():

```
int     accept(int, struct sockaddr *restrict,
        socklen_t *restrict);
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);
ssize_t recvfrom(int, void *restrict, size_t, int,
        struct sockaddr *restrict, socklen_t *restrict);
```

sys/stat.h

Data returned by the *stat*( ) function

```
#include <sys/stat.h>
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

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.
- The *lstat*( ) function is now mandatory.

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*( ):

```
int lstat(const char *restrict, struct stat *restrict);
int stat(const char *restrict, struct stat *restrict);
```

, item XBD/TC2/D6/25 is applied, adding to the DESCRIPTION that the **timespec** structure may be defined as described in the **<time.h>** header.

sys/statvfs.h

VFS File System information structure

XSI

```
#include <sys/statvfs.h>
```

Derivation First released in Issue 4, Version 2.

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*( ):

```
int statvfs(const char *restrict, struct statvfs *restrict);
```

, item XBD/TC1/D6/18 is applied, changing the description of ST_NOSUID from ‘‘Does not support *setuid*( )/*setgid*( ) semantics’’ to ‘‘Does not support the semantics of the ST_ISUID and ST_ISGID file mode bits’’.

sys/time.h

Time types

XSI
```
#include <sys/time.h>
```

Derivation First released in Issue 4, Version 2.

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. The individual reference page should be consulted for information on alternative recommended interfaces.

sys/timeb.h

Additional definitions for date and time

XSI
```
#include <sys/timeb.h>
```

Derivation First released in Issue 4, Version 2.

Issue 6 The *ftime*() function is marked LEGACY. The individual reference page should be consulted for information on alternative recommended interfaces.

sys/times.h

File access and modification times structure

```
#include <sys/times.h>
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes in this issue.

sys/types.h

Data types

```
#include <sys/types.h>
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

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 marked THR.

, item XBD/TC2/D6/26 is applied, adding **pthread_t** to the list of types that are not required to be arithmetic types, thus allowing **pthread_t** to be defined as a structure.

sys/uio.h

Definitions for vector I/O operations

XSI
```
#include <sys/uio.h>
```

Derivation First released in Issue 4, Version 2.

Issue 6 Text referring to scatter/gather I/O is added to the DESCRIPTION.

sys/un.h

Definitions for UNIX domain sockets

```
#include <sys/un.h>
```

Derivation First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 6 This is a new header included for alignment with the XNS, Issue 5.2 specification.

sys/utsname.h

System name structure

```
#include <sys/utsname.h>
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6 No functional changes in this issue.

sys/wait.h

Declarations for waiting

```
#include <sys/wait.h>
```

Derivation First released in Issue 3. Entry included for alignment with IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 The *wait3*( ) function is removed.

syslog

Definitions for system error logging

XSI
```
#include <syslog.h>
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes in this issue.

tar.h

Extended tar definitions

```
#include <tar.h>
```

Derivation First released in Issue 3. Derived from IEEE Std 1003.1-1988 (POSIX.1).

Issue 6 No functional changes in this issue.

The SEE ALSO section now refers to *pax* since XCU, Issue 6 no longer contains the *tar* utility.

termios.h

Define values for termios

```
#include <termios.h>
```

Derivation First released in Issue 3. Entry included for alignment with ISO/IEC 9945-1: 1996 (POSIX-1).

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.

tgmath.h

Type-generic macros

```
#include <tgmath.h>
```

Derivation First released in Issue 6. Included for alignment with the ISO/IEC 9899: 1999 standard.

Issue 6 This is a new header included for alignment with the ISO/IEC 9899: 1999 standard.

This header includes the **<math.h>** and **<complex.h>** and defines several type-generic macros.

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.

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.

time.h

Time types

```
#include <time.h>
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

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 ISO/IEC 9945-1: 1996 (POSIX-1):

- The POSIX timer-related functions are marked as part of the Timers option and need not be available in all implementations.

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 marked.

trace.h

Tracing

TRC
```
#include <trace.h>
```

Derivation First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 6 This is a new header included for alignment with IEEE Std 1003.1q-2000.

, item XSH/TC1/D6/40 is applied, adding the TRL margin code to the *posix_trace_flush*( ) function, for alignment with the System Interfaces volume of IEEE Std 1003.1-2001.

ucontext.h

User context

XSI
```
#include <ucontext.h>
```

Derivation First released in Issue 4, Version 2.

Issue 6 , item XBD/TC2/D6/28 is applied, updating the *getcontext*( ), *makecontext*( ), *setcontext*( ), and *swapcontext*( ) functions to be obsolescent.

ulimit.h

Ulimit commands

XSI
```
#include <ulimit.h>
```

Derivation First released in Issue 3.

Issue 6 No functional changes in this issue.

unistd.h

Standard symbolic constants and types

```
#include <unistd.h>
```

Derivation First released in Issue 1. Derived from Issue 1 of the SVID.

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 following symbols:

_SC_XOPEN_LEGACY
_SC_XOPEN_REALTIME
_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 1003.1-2001.

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:

```
int gethostname(char *, size_t);
```

A new section is added for System-Wide Options.

Function prototypes for *setegid*( ) and *seteuid*( ) are added:

```
int setegid(gid_t);
int seteuid(uid_t);
```

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 individual reference page should be consulted for information on alternative recommended interfaces.

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 1003.1-2001 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**.

, item XBD/TC1/D6/2 is applied, changing ‘‘Thread Stack Address Size’’ to ‘‘Thread Stack Size Attribute’’.

, item XBD/TC1/D6/20 is applied, adding the _POSIX_IPV6, _SC_V6, and _SC_RAW_SOCKETS symbols.

, item XBD/TC1/D6/21 is applied, correcting the description in ‘‘Constants for Functions’’ for the _CS_POSIX_V6_LP64_OFF64_CFLAGS, _CS_POSIX_V6_LP64_OFF64_LDFLAGS, and _CS_POSIX_V6_LP64_OFF64_LIBS symbols.

, item XBD/TC1/D6/22 is applied, removing the shading for the _PC* and _SC* constants, since these are mandatory on all implementations.

, item XBD/TC1/D6/23 is applied, adding the _PC_SYMLINK_MAX and _SC_SYMLOOP_MAX constants.

, item XBD/TC1/D6/24 is applied, correcting the shading and margin code for the *fsync*( ) function.

, item XBD/TC1/D6/25 is applied, adding the following text to the APPLICATION USAGE section: ‘‘New applications should not use _XOPEN_SHM or _XOPEN_ENH_I18N’’.

, item XBD/TC2/D6/29 is applied, clarifying the requirements for when constants for Options and Option Groups can be defined or undefined.

, item XBD/TC2/D6/30 is applied, changing the _V6_ILP32_OFF32, _V6_ILP32_OFFBIG, _V6_LP64_OFF64, and _V6_LPBIG_OFFBIG symbols to _POSIX_V6_ILP32_OFF32, _POSIX_V6_ILP32_OFFBIG, _POSIX_V6_LP64_OFF64, and _POSIX_V6_LPBIG_OFFBIG, respectively. This is for consistency with the *sysconf*( ) and *c99* reference pages.

, item XBD/TC2/D6/31 is applied, adding that the format of names of programming environments can be obtained using the *getconf* –**v** option.

, item XBD/TC2/D6/32 is applied, deleting the _SC_FILE_LOCKING, _SC_2_C_VERSION, and _SC_XOPEN_XCU_VERSION constants.

, item XBD/TC2/D6/33 is applied, adding _SC_SS_REPL_MAX, _SC_TRACE_EVENT_NAME_MAX, _SC_TRACE_NAME_MAX, _SC_TRACE_SYS_MAX, and _SC_TRACE_USER_EVENT_MAX to the list of symbolic constants for *sysconf*( ).

, item XBD/TC2/D6/34 is applied, updating the prototype for the *symlink*( ) function to match that in the System Interfaces volume of IEEE Std 1003.1-2001.

, item XBD/TC2/D6/35 is applied, adding _PC_2_SYMLINKS to the symbolic constants list for *pathconf*( ). This corresponds to the definition of POSIX2_SYMLINKS in the Shell and Utilities volume of IEEE Std 1003.1-2001.

utime.h

Access and modification times structure

```
#include <utime.h>
```

Derivation 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.

utmpx.h

User accounting database definitions

XSI
```
#include <utmpx.h>
```

Derivation First released in Issue 4, Version 2.

Issue 6 No functional changes in this issue.

wchar.h

Wide-character handling

```
#include <wchar.h>
```

Derivation First released in Issue 4.

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.

wctype.h

Wide-character classification and mapping utilities

```
#include <wctype.h>
```

Derivation First released in Issue 5. Derived from ISO C Amendment 1 (MSE).

Issue 6 The *iswblank*() function is added for alignment with the ISO/IEC 9899: 1999 standard:

```
int iswblank(wint_t);
```

wordexp.h

Word-expansion types

```
#include <wordexp.h>
```

Derivation First released in Issue 4. Derived from ISO/IEC 9945-2: 1993 (POSIX-2).

Issue 6 The **restrict** keyword is added to the prototype for *wordexp*():

```
int wordexp(const char *restrict, wordexp_t *restrict, int);
```

The WRDE_NOSYS constant is marked obsolescent.

Chapter 16

ISO C Migration

By Finnbarr P. Murphy, Compaq Computer Corporation

Finnbarr P. Murphy is a software engineer in the Business Critical Systems Group (BCSG) at Compaq Computer Corporation in Nashua, New Hampshire. He can be contacted via email at Finnbarr.Murphy@Compaq.com.

16.1 Introduction

The original ISO/IEC C language programming standard (the ISO/IEC 9899: 1990 standard) was adopted by the International Organization for Standardization (ISO) and the International Electotechnical Commission (IEC) in 1990. Subsequently, two technical corrigenda (TC1 and TC2) were approved together with the normative ISO C Amendment 1 (MSE), Multibyte Support Extension.

At the end of 1993, there was general agreement that work should start on the next revision of the standard. The revised standard (C99) was sent for FCD ballot in August 1998, and adopted by ISO/IEC in 1999 as the ISO/IEC 9899: 1999 standard.

This chapter is intended to provide the reader with a good, but not exhaustive, overview of the differences between the two revisions of the standard. Thus the reader is strongly advised to reference the ISO/IEC 9899: 1999 standard for specific details.

16.2 Language Changes

A significant number of changes occurred in the standard, including new keywords and types, type qualifiers, better floating-point support, and support for complex numbers.

16.2.1 New Keywords

The following new keywords were defined:

- **inline**
- **restrict**
- **_Bool**
- **_Complex**
- **_Imaginary**
- **long long**

16.2.2 New Types

Two new types were added:

- **_Bool**
- **long long**

The **long long** type is an integer type with at least 64 bits of precision.

Note: In some programming models such as LP64 and ILP64, **long long** and **long** are equivalent. In the others—for example, LLP64—**long long** is larger than **long**.

16.2.3 Type Qualifiers

Type qualifiers are now idempotent. If a type qualifier appears more than once (either directly or indirectly) in a type specification, it is as if it appeared only once. Thus **const const int** *fpm*; and **const int** *fpm*; are equivalent.

restrict is a new type qualifier which enables programs to be written so that compilers can produce significantly faster executables. It is intended to be used only with pointers. Objects referenced through a **restrict-**qualified pointer are special in that all references to the object must directly or indirectly use the value of the **restrict-**qualified pointer. It is intended to facilitate better alias analysis by compilers. In the absence of this qualifier, other pointers can alias the object and prevent compiler optimizations since a compiler may not be able to determine that different pointers are being used to reference different objects. Note that a restricted pointer and a non-restricted pointer can be aliases.

A number of function definitions were modified to take advantage of the **restrict** qualifier. A typical example is the *fopen*() function which was changed from:

```
FILE *fopen(const char *filename, const char *mode);
```

to:

```
FILE *fopen(const char *restrict filename,
    const char *restrict mode);
```

Changed functions include:

fgetpos()	*freopen*()	*memcpy*()	*strncpy*()	*vwprintf*()
fgets()	*fwprintf*()	*setbuf*()	*strxfrm*()	*wcstod*()
fgetws()	*fwrite*()	*setvbuf*()	*swprintf*()	*wcstol*()
fopen()	*fwscanf*()	*strcat*()	*swscanf*()	*wcstombs*()
fputs()	*mbstowcs*()	*strcpy*()	*vfwprintf*()	*wcstoul*()
fread()	*mbtowc*()	*strncat*()	*vswprintf*()	*wprintf*()

16.2.4 Boolean

The standard now supports a boolean type **_Bool** which is an integer type which can hold either 0 or 1.

The header **<stdbool.h>** also defines the macro bool which expands to **_Bool**, true which expands to the integer constant 1, and false which expands to the integer constant 0.

16.2.5 Universal Character Names

Prior to this revision of the standard, ‘‘native’’ characters, in the form of multibyte and wide characters, could be used in string literals and character constants, but not as part of an identifier.

This standard introduced the concept of a universal character name (UCN) that may be used in identifiers, character constants, and string literals to designate characters that are not in the basic character set.

The two forms of a UCN are:

\u*nnnn* where *nnnn* is *hex-quad*
\U*nnnnnnnn* where *nnnnnnnn* is *hex-quad hex-quad*

A *hex-quad* consists of 4 hexadecimal digits.

The UNC \U*nnnnnnnn* designates the character whose eight-digit short identifier as specified by the ISO/IEC 10646-1: 2000 standard is *nnnnnnnn*.

Similarly, the UCN \u*nnnn* can be used to designate a given character whose four-digit short identifier as specified by the ISO/IEC 10646-1: 2000 standard is *nnnn* (and whose eight-digit short identifier is 0000*nnnn*).

There are a number of disallowed characters; that is, those in the basic character set, and code positions reserved in the ISO/IEC 10646-1: 2000 standard for control and DELETE characters and UTF-16.

Note: A strictly conforming program may use only the extended characters listed in Annex I (Universal Character Names for Identifiers) and may not begin an identifier with an extended digit. Also, use of native characters in comments has always been strictly conforming, though what happens when such a program is printed in a different locale is unspecified.

16.2.6 inline

The **inline** keyword is intended to provide users with a portable way to suggest to implementations that inlining a function might result in program optimizations.

It is a function-specifier that can be used only in function declarations. It was adopted from C++ but extended in such a way that it can be implemented with existing linker technology. The translation unit that contains the definition of an inline function is the unit that provides the external definition for the function. If a function is declared inline in one translation unit, it need not be declared inline in every other translation unit.

16.2.7 Predefined Identifiers

Predefined identifiers are variables that have block scope.

The standard defined one predefined identifier __func__ which is declared implicitly by the compiler as if, immediately following the opening brace of each function definition, the following declaration was included in the source code:

```
static const char __func__[] = "function-name";
```

where *function-name* is the name of the lexically-enclosing function. This enables a function name to be obtained at runtime.

The *assert*() macro now includes the identifier __func__ in the output to *stderr*:

```
void assert(scalar expression);
```

Note that the parameter type of the *assert*( ) macro was also changed from **int** to **scalar**.

16.2.8 Compound Literals

Compound literals (also known as anonymous aggregates) provide a mechanism for specifying constants of aggregate or union type. This eliminates the requirement for temporary variables when an aggregate or union value may only be needed once. Compound literals are primary expressions which can also be combined with designated initializers to form an even more convenient aggregate or union constant notation.

Compound literals are created using the notation:

```
( type-name ) { initializer-list }
```

For example:

```
int *ap = (int a[]) {1. 2, 3 };
```

Note that a trailing comma before the closing brace is permitted.

16.2.9 Designated Initializers

Designated initializers provide a mechanism for initializing aggregates such as sparse arrays, a common requirement in numerical programming. This mechanism also allows initialization of sparse structures and initialization of unions via any member, regardless of whether or not it is the first member.

Initializers have a named notation for initializing members. For array elements, the element is designated by [*const-expression*], for **struct** and **union** members by a dot member-name notation.

For example:

```
struct s { int a; int b; };
struct s mystruct = {.b = 2};    // initialize member b
struct {int a[3], b[3]} w[] = { [0].a = {1}, [1].b = 2 };
```

If an initializer is present, any members not explicitly set are zeroed out. Initializers for **auto** aggregates can be non-constant expressions.

16.3 Decimal Integer Constants

The default type of a decimal integer constant is either **int**, **long**, or **long long** (previously **int**, **long**, and **unsigned long**), depending on which type is large enough to hold the value without overflow.

The standard added LL to specify **long long**, and ULL to specify **unsigned long long**.

16.3.1 String Literals

The standard defines a number of macros as expanding into character string literals that are frequently needed as wide strings.

One example is the format specifier macros in **<inttypes.h>**. Rather than specifying two forms of each macro, one character string literal and one wide string literal, the decision was made to define the result of concatenating a character string literal and a wide string literal as a wide string literal.

16.4 Implicit Declarations

Implicit declaration of functions is no longer permitted by the standard. There must be a least one type specifier otherwise a diagnostic is issued. However, after issuing the diagnostic, an implementation may choose to assume an implicit declaration and continue translation in order to support existing source code.

For example, the declaration *fpm*(); was valid in previous revisions of the standard (equivalent to **int** *fpm*();) but is now invalid.

16.4.1 sizeof

With the addition of variable length arrays, the *sizeof* operator is a constant expression only if the type of the operand is not a variable length array type.

Note: It is still possible to determine the number of elements in a variable length array *vla* with *sizeof*(*vla*)/*sizeof*(*vla*[0]).

16.4.2 Multiplicative Operators

In previous revisions of the standard, division of integers involving negative operands could round upward or downward in an implementation-defined manner. The standard now mandates that, as in Fortran, the result always truncates toward zero.

For example, both of the following truncate towards zero:

```
-22 / 7 = -3
-22 % 7 = -1
```

This was done to facilitate porting of code from Fortran to C.

16.4.3 Enumeration Specifiers

A common extension to many C implementations is to allow a trailing comma after the list of enumeration constants. The standard now permits this.

16.5 Variable Length Array

A new array type, called a variable length array type, was added to the standard. The number of elements specified in the declaration of a variable length array type is not specified by the source code; rather it is a computed value determined at runtime.

Multi-dimensional variable-length arrays are permitted.

Some things cannot be declared as a variable length array type, including:

- File scope identifiers
- Arrays declared using either **static** or **extern** storage class specifiers
- Structure and union members

The rationale behind this new array type was that some standard method to support runtime array sizing was considered crucial for C's acceptance in the numerical computing world. Before this revision of the standard, the size expression was required to be an integer constant expression.

16.5.1 Array Declarations

The **static** storage class specifier and the type-qualifiers **restrict**, **const**, or **volatile** can now be used inside the square brackets of an array type declaration, but only in the outermost array type derivation of a function parameter.

```
int foo(const int a[static 10]);
```

In the above example, the **static** keyword will guarantee that the pointer to the array *a* is not NULL, and points to an object of the appropriate type.

16.5.2 Array Type Compatibility

Array type compatibility was extended so that variable length arrays are compatible with both an array of known constant size and an array with an incomplete type.

16.5.3 Incomplete Array Structure Members

The last member of a structure with more than one member can now be an incomplete array type. This incomplete member is called a flexible array member.

Consider the following example

```
struct s { int n;
           double d[];
};

size_t sz = sizeof( struct s );
struct s *sp = malloc(sz + 10);
```

The structure pointer **sp** behaves as if the structure **s** had been declared as

```
struct s { int n;
           double d[10];
};
```

The size of the structure is equal to the offset of the last element of an otherwise identical structure that replaces the flexible array member with an array of unspecified length. When a `'.'` or a `'->'` operator point to a structure with a flexible array member and the right operand names that member, it behaves as if that member were replaced with the longest array with the

same element type that would not make the structure larger than the object being accessed.

The offset of the array remains that of the flexible array member, even if this would differ from that of the replacement array. If this array would have no elements, it behaves as if it had one element. However, behavior is undefined if any attempt is made to access that element or to generate a pointer one past it.

16.5.4 Blocks

A common coding practice is to always use compound statements for every selection and iteration statement to guard against inadvertent problems when changes are made to the source code.

Because this can lead to surprising behavior in connection with certain uses of compound literals, the concept of a block was expanded in this revision of the standard.

As in C++, all selection and iteration statements, and their associated substatements, are now defined to be blocks, even if they are not also compound statements. If compound literals are defined in selection or iteration statements, their lifetimes are limited to the implied enclosing block.

16.5.5 The for Statement

The standard now permits loop counter variables as part of a **for** statement. Such a variable is in a new scope (so it does not affect any other variable of the same name), is destroyed at the end of the loop, and must have **auto** or **register** storage class.

```
for (int i = 0; i < 10; i++)
printf("Loop number: %d\n", i);
```

16.5.6 errno

For underflow, *errno* is no longer required to be set to [EDOM] or [ERANGE].

16.6 Comments

Support for `//`-style comments was added due to their utility and widespread existing practice, especially in dual C/C++ translators. This is a quiet change which could cause different semantics between this standard and C89. Consider the following example:

```
a = b //*divisor:*/ f
    + e;
```

According to this standard this is the same as:

```
a = b + e;
```

but in previous revisions of the standard it was the same as:

```
a = b / f + e;
```

16.6.1 Hexadecimal Floating-Point Constants

Because hexadecimal notation more clearly expresses the significance of floating constants, the standard now supports hexadecimal floating-point constants.

The binary-exponent part is required, instead of being optional as it is for decimal notation, to avoid ambiguity resulting from an `'f'` suffix being mistaken as a hexadecimal digit. The exponent indicates the power of 2 by which the significant part is to be scaled.

16.6.2 Predefined Macros

New predefined macros include:

`__STDC_VERSION__`	Defined to be 199901L to indicate the current revision of the standard.
`__STDC_HOSTED__`	Defined as 1 if the implementation is hosted; otherwise, 0.

16.6.3 Source File Inclusion

The number of significant characters in header and source file names was raised from six to eight, and digits are now allowed.

16.6.4 Translation-Time Arithmetic

The standard now mandates that translation-time arithmetic be done using **intmax_t** or **uintmax_t**, which must comprise at least 64 bits and must match the execution environment.

Previously, a translator was permitted to evaluate expressions using the **long** integer or **unsigned long** integer arithmetic native to the translation environment.

16.6.5 Minimum Maximum Line Length

The minimum maximum line length was increased from 254 to 4095.

16.6.6 Case-Sensitive Identifiers

All identifiers are now case-sensitive. In previous revisions of the standard, it was implementation-defined whether an implementation ignored the case of external identifiers.

16.6.7 #line Directive

This directive now allows the specification of a line number up to 2**31–1. Previously the limit was 32 767.

16.6.8 Empty Argument Macros

Empty arguments are now explicitly allowed. In previous revisions of the standard, this resulted in undefined behavior. Stringification (# operator) of an empty argument yields the empty string, concatenation (## operator) of an empty argument with a non-empty argument produces the non-empty argument, and concatenation of two empty arguments produces nothing.

16.6.9 Pragmas

Some **pragma** directives have been standardized. Directives whose first preprocessing token is **STDC** are reserved for standardized directives.

As an alternative syntax for a **pragma** directive, the preprocessing operator **_Pragma** is specified. This has the advantage that it can be used in a macro replacement list.

16.6.10 Translation Limits

A number of the program translation limits were significantly increased.

The number of significant initial characters in an internal identifier or a macro name was increased from 31 to 63.

The number of significant characters in an external identifier has increased from 6 to 31 case-sensitive characters.

Note that each universal character name (UCN) specifying a short identifier of 0000FFFF or less is considered to be 6 characters, while a long UCN counts as 10 characters.

While an implementation is not obliged to remember more than the first 63 characters of an identifier with internal linkage, or the first 31 characters of an identifier with external linkage, the programmer is effectively prohibited from intentionally creating two different identifiers that are the same within the appropriate length.

The minimum maximum limit of cases in a switch statement was increased to 1 023.

16.6.11 Token Pasting

The standard replaced non-digit with identifier-non-digit in the grammar to allow the token pasting operator, ##, to work as expected with characters which are not part of the basic character set.

16.6.12 Variadic Macros

The standard extended the functionality of the punctuator `"..."` (ellipsis; denoting a variable number of trailing arguments) to function-like macros. For replacement, the variable arguments (including the separating commas) are ‘‘collected’’ into one single extra argument that can be referenced as __VA_ARGS__ within the macro’s replacement list.

For example:

```
#define MyLog(...) fprintf(stderr, __VA_ARGS__)

main()
{
    int array_bound = 10;
    int array_index = 11;
    ......
    MyLog("ERROR: Index out of bound: %d %d\n", array_index,
        array_bound);
    ......
}
```

There must be at least one argument to match the ellipsis. This requirement avoids problems that might occur when the trailing arguments are included in a list of arguments to another macro or function.

16.6.13 va_copy( )

In previous revisions of the standard, it was not possible to backtrack and examine one or more arguments a second time when processing a variable argument list. The only way to do this was to reprocess the variable argument list.

The *va_copy*( ) macro provides a mechanism for copying the **va_list** object used to represent processing of the arguments. Calling the *va_copy*( ) macro exactly duplicates the **va_list** object.

Note: A separate call to the *va_end*( ) macro is required to remove the new **va_list** object.

16.7 Headers

The following new headers were added to the standard:

<complex.h> Defines a number of macros and functions for use with the three complex arithmetic types defined in the standard.

<fenv.h> Defines a number of types, macros, and functions that can be used to test, control, and access an implementation's floating-point environment.

<inttypes.h> Defines a type and a number of macros and functions for manipulating integers; **<stdint.h>** is a subset of this header.

<stdbool.h> Defines a number of macros for accessing the new Boolean type **_Bool** and writing Boolean tests.

<tgmath.h> Defines a large number of type-generic macros that invoke the correct math function from **<math.h>** or from **<complex.h>** depending upon their argument types.

16.8 Integer Types

The purpose of the **<inttypes.h>** header is to provide a set of integer types whose definitions are consistent across platforms. Consistent use of these integer types should greatly increase the portability of source code across platforms.

The header **<stdint.h>** is a subset of **<inttypes.h>** and may be more suitable for use in freestanding environments, which might not support the formatted I/O functions. It declares sets of integer types having specified widths and corresponding macros that specify limits of the declared types and construct suitable constants.

The following categories of integer types are defined:

- Types having exact widths
- Types having at least certain specified widths
- Fastest types having at least certain specified widths
- Types wide enough to hold pointers to objects
- Types having greatest width

16.8.1 Exact-Width Integer Types

The **typedef** name **intN_t** designates a signed integer type with width *N* bits, no padding bits, and a two's complement representation. The **typedef** name **uintN_t** designates an unsigned integer type with width *N*.

For example, **int16_t** is an unsigned integer type with a width of exactly 16 bits.

Exact-width types are optional. However, if an implementation provides integer types with widths of 8, 16, 32, or 64 bits, it must define the corresponding **typedef** names.

16.8.2 Minimum-Width Integer Types

The **typedef** names **int_leastN_t** and **uint_leastN_t**, respectively, designate signed and unsigned integer types with a width of at least *N* bits, such that no signed integer type with lesser size has at least the specified width.

For example, **uint_least16_t** denotes an unsigned integer type with a width of at least 16 bits.

The following types are mandatory:

int_least8_t	**int_least32_t**	**uint_least8_t**	**uint_least32_t**
int_least16_t	**int_least64_t**	**uint_least16_t**	**uint_least64_t**

16.8.3 Fastest Minimum-Width Integer Types

The **typedef** names **int_fastN_t** and **uint_fastN_t**, respectively, designate the (usually) fastest signed and unsigned integer types with a width of at least *N* bits.

The following types are mandatory:

int_fast8_t	**int_fast32_t**	**uint_fast8_t**	**uint_fast32_t**
int_fast16_t	**int_fast64_t**	**uint_fast16_t**	**uint_fast64_t**

16.8.4 Integer Types Capable of Holding Object Pointers

The optional **intptr_t** and **uintptr_t** types, respectively, designate a signed and 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.

16.8.5 Greatest-Width Integer Types

The **intmax_t** and **uintmax_t** types, respectively, designate a signed integer type capable of representing any value of any signed integer type, and an unsigned integer type capable of representing any value of any unsigned integer type.

For each type declared in **<stdint.h>** conversion macros, which expand to the correct format specifiers, are defined for use with the formatted input/output functions such as *fprintf*( ) and *fscanf*( ).

The *fprintf*( ) macros for signed integers are:

PRIdFASTN	PRIdMAX	PRIdPRT	PRIiLEASTN	PRIiN
PRIdLEASTN	PRIdN	PRIiFASTN	PRIiMAX	PRIiPRT

The PRId macros each expand to a string literal suitable for use as a d print conversion specifier, plus any needed qualifiers, to convert values of the types **int8_t**, **int16_t**, **int32_t**, or **int64_t**, respectively.

The PRIdLEAST macros each expand to a string literal suitable for use as a d print conversion specifier, plus any needed qualifiers, to convert values of the types **int_least8_t**, **int_least16_t**, **int_least32_t**, or **int_least64_t**, respectively.

The PRIdFAST macros each expand to a string literal suitable for use as a d print conversion specifier, plus any needed qualifiers, to convert values of the types **int_fast8_t**, **int_fast16_t**, **int_fast32_t**, or **int_fast64_t**, respectively.

The PRIdMAX macro expands to a string literal suitable for use as a d print conversion specifier, plus any needed qualifiers, to convert values of the type **intmax_t**.

The PRIdPTR macro expands to a string literal suitable for use as a d print conversion specifier, plus any needed qualifiers, to convert values of the type **intptr_t**.

The following example shows part of one possible implementation of the these macros in an LP64 programming model:

```
#define PRId8        "hhd"
#define PRId16       "hd"
#define PRId32       "d"
#define PRId64       "ld"

#define PRIdFAST8    "hhd"
#define PRIdFAST16   "hd"
#define PRIdFAST32   "d"
#define PRIdFAST64   "ld"

#define PRIdLEAST8   "hhd"
#define PRIdLEAST16  "hd"
#define PRIdLEAST32  "d"
#define PRIdLEAST64  "ld"
```

Corresponding *fprintf*( ) macros are defined for unsigned integers (PRIo, PRIu, PRIx, and PRIX).

For *fscanf*( ), the macro names start with SCN instead of PRN. (SCNd and SCNi for signed integers; SCNo, SCNu, SCNx for unsigned integers.)

A new structure type **imaxdiv_t** is defined. It is the type of structure returned by *imaxdiv*( ).

The following function computes the absolute value of an integer *j*:

```
intmax_t imaxabs(intmax_t j);
```

The following function computes both the quotient and remainder in a single operation:

```
imaxdiv_t imaxdiv(intmax_t numer, intmax_t denom);
```

The following functions are equivalent to the *strtol* family of functions, except that the initial portion of the string is converted to **intmax_t** and **uintmax_t** representation, respectively:

```
intmax_t strtoimax(const char *restrict nptr,
    char **restrict endptr, int base);
uintmax_t strtoumax(const char *restrict nptr,
    char **restrict endptr, int base);
```

The following functions are equivalent to the *wcstol* family of functions, except that the initial portion of the wide string is converted to **intmax_t** and **uintmax_t** representation, respectively:

```
intmax_t wcstoimax(const wchar_t *restrict nptr,
    wchar_t **restrict endptr, int base);
uintmax_t wcstoumax(const wchar_t *restrict nptr,
```

```
wchar_t **restrict endptr, int base);
```

16.8.6 Limits of Specified-Width Integer Types

The standard specifies the minimum and maximum limits for all of the types declared in the **<stdint.h>** header.

For example, the minimum value of an exact-width unsigned integer type is {UINTn_MAX}, where *n* is an unsigned decimal integer with no leading zeros, whose value is exactly 2**n*–1.

16.8.7 Macros

The macros *INTN_C*( ) and *UINTN_C*( ) expand to a signed integer constant whose type and value is **int_leastN_t**, and an unsigned integer constant whose type and value is **uint_leastN_t**, respectively.

The macro *INTMAX_C*( ) expands to a integer constant whose type is **intmax_t**, and *UINTMAX_C*( ) expands to an unsigned integer constant whose type **uintmax_t**.

16.9 Complex Numbers

Support for complex numbers and complex number arithmetic is new, and was added as part of the effort to make the C language more attractive for general numerical programming. The underlying implementation of the complex types is explicitly stated to be Cartesian, rather than polar, for consistency with other programming languages. Thus values are interpreted as radians, not degrees.

The header **<complex.h>** contains the macro definitions and function declarations that support complex arithmetic.

Two new type specifiers were defined:

_Complex

_Imaginary (Only if an implementation supports a pure imaginary type.)

A new type qualifier complex (actually a macro which expands to _Complex) is used to denote a number as being a complex number.

Three complex types were defined:

- **float complex**
- **double complex**
- **long double complex**

The corresponding real type is the type obtained by deleting the type qualifier **complex** from the complex type name.

A complex type has the same representation and alignment requirements as an array type containing exactly two elements of the corresponding real type; the first element is equal to the real part, and the second element to the imaginary part, of the complex number.

There is no special syntax for constants; instead there is a new macro _Complex_I, which has a complex value whose real part is zero and whose imaginary part is *x*. Note that _Complex_I*_Complex_I has a value of –1, but the type of that value is **complex**.

The standard reserves the keyword _**Imaginary** for use as a type-specifier in conjunction with the pure imaginary type. The macros imaginary and Imaginary_I are defined only if an implementation supports a pure **imaginary** type. Such support is optional. See Annex G of the standard for further details. If defined, they expand to _Imaginary and a constant expression of type **const float** _**Imaginary** with the value of the imaginary unit.

The macro I expands to _Imaginary_I, if defined, else to _Complex_I. Thus a complex number constant (3.0 +4.0*i*) could be written as either 3.0+4.0*I or 3.0+4.0 *_Complex_I.

The choice of `'I'` instead of `'i'` for the imaginary unit was because of the widespread use of the identifier `'i'` for other purposes. A program can use a different identifier (for example, `'z'`) for the imaginary unit by undefining `'I'` and defining `'z'` as follows:

```
#include <complex.h>
#undef I
#define z _Imaginary_z
```

Annex G, which is marked informative, specifies complex arithmetic intended to be compatible with the IEC 60559: 1989 standard real floating-point arithmetic, This annex was designated as informative because of insufficient prior art for normative status. An implementation claiming such conformance should define __STDC_IEC_559_COMPLEX to be 1.

The **pragma STDC CX_LIMITED_RANGE** can be used to indicate (ON) that the usual mathematical formulas for complex arithmetic may be used. Such formulas are problematic because of overflow, underflow, and handling of infinities.

The following new functions relate to complex arithmetic.

16.9.1 Trigonometric Functions

The complex arc cosine functions compute the complex arc cosine of *z*, with branch cuts outside the interval [–1,+1] along the real axis.

```
double complex cacos(double complex z);
float complex cacosf(float complex z);
long double complex cacosl(long double complex z);
```

The complex arc sine functions compute the complex arc sine of *z*, with branch cuts outside the interval [–1,+1] along the real axis.

```
double complex casin(double complex z);
float complex casinf(float complex z);
long double complex casinl(long double complex z);
```

The complex arc tangent functions compute the complex arc tangent of *z*, with branch cuts outside the interval [–*i*,+*i*] along the imaginary axis.

```
double complex catan(double complex z);
float complex catanf(float complex z);
long double complex catanl(long double complex z);
```

The complex cosine functions compute the complex cosine of *z*.

```
double complex ccos(double complex z);
float complex ccosf(float complex z);
long double complex ccosl(long double complex z);
```

The complex sine functions compute the complex sine of *z*.

```
double complex csin(double complex z);
float complex csinf(float complex z);
long double complex csinl(long double complex z);
```

The complex tangent functions compute the complex tangent of *z*.

```
double complex ctan(double complex z);
float complex ctanf(float complex z);
long double complex ctanl(long double complex z);
```

16.9.2 Hyperbolic Functions

The complex arc hyperbolic cosine functions compute the complex arc hyperbolic cosine of *z*, with a branch cut at values less than 1 along the real axis.

```
double complex cacosh(double complex z);
float complex cacoshf(float complex z);
long double complex cacoshl(long double complex z);
```

The complex arc hyperbolic sine functions compute the complex arc hyperbolic sine of *z*, with branch cuts outside the interval [–*i*,+*i*] along the imaginary axis.

```
double complex casinh(double complex z);
float complex casinhf(float complex z);
long double complex casinhl(long double complex z);
```

The complex arc hyperbolic tangent functions compute the complex arc hyperbolic tangent of *z*, with branch cuts outside the interval [–1,+1] along the real axis.

```
double complex catanh(double complex z);
float complex catanhf(float complex z);
long double complex catanhl(long double complex z);
```

The complex hyperbolic cosine functions compute the complex hyperbolic cosine of *z*.

```
double complex ccosh(double complex z);
float complex ccoshf(float complex z);
long double complex ccoshl(long double complex z);
```

The complex hyperbolic sine functions compute the complex hyperbolic sine of *z*.

```
double complex csinh(double complex z);
float complex csinhf(float complex z);
long double complex csinhl(long double complex z);
```

The complex hyperbolic tangent functions compute the complex hyperbolic tangent of *z*.

```
double complex ctanh(double complex z);
float complex ctanhf(float complex z);
long double complex ctanhl(long double complex z);
```

16.9.3 Exponential and Logarithmic Functions

The complex exponential functions compute the complex base-e exponential of *z*.

```
double complex cexp(double complex z);
float complex cexpf(float complex z);
long double complex cexpl(long double complex z);
```

The complex natural logarithm functions compute the complex natural logarithm of *z*, with a branch cut along the negative real axis.

```
double complex clog(double complex z);
float complex clogf(float complex z);
long double complex clogl(long double complex z);
```

16.9.4 Power and Absolute-Value Functions

The complex absolute value functions compute the modulus of *x*.

```
double complex cabs(double complex x);
float complex cabsf(float complex x);
long double complex cabsl(long double complex x);
```

The complex power functions compute the complex power function *x****y*, with a branch cut for the first parameter along the negative real axis.

```
double complex cpow(double complex x, double complex y);
float complex cpowf(float complex x, float complex y);
long double complex cpowl(long double complex x,
    long double complex y);
```

The complex square root functions compute the complex square root of *z*, with a branch cut along the negative real axis.

```
double complex csqrt(double complex z);
float complex csqrtf(float complex z);
long double complex csqrtl(long double complex z);
```

16.9.5 Manipulation Functions

The complex argument functions compute the argument of *z*, with a branch cut along the negative real axis.

```
double carg(double complex z);
float cargf(float complex z);
long double cargl(long double complex z);
```

The complex imaginary functions compute the imaginary part of *z*.

```
double cimag(double complex z);
float cimagf(float complex z);
long double cimagl(long double complex z);
```

The complex conjugate functions compute the complex conjugate of *z*, by reversing the sign of its imaginary part.

```
double complex conj(double complex z);
float complex conjf(float complex z);
long double complex conjl(long double complex z);
```

The complex projection functions compute a projection of *z* onto the Riemann sphere.

```
double complex cproj(double complex z);
float complex cprojf(float complex z);
long double complex cprojl(long double complex z);
```

The complex real functions compute the real part of *z*.

```
double creal(double complex z);
float crealf(float complex z);
long double creall(long double complex z);
```

Note that no errors are defined for any of the above functions.

16.10 Other Mathematical Changes

The standard extended the mathematical support via **<math.h>** by providing versions of functions to support **float** and **long double** as well as the existing double floating type functions.

The functions *ecvt*( ), *fcvt*( ), and *gcvt*( ) were dropped from the standard since their capability is available using the *sprintf*( ) function.

The **pragma STDC FP_CONTACT** indicates to an implementation whether it is allowed (ON) or disallowed (OFF) to contract expressions; that is, evaluated as though an expression is an atomic operation, thereby omitting certain rounding errors.

The macro NAN is defined only if an implementation supports quiet NaNs.

16.10.1 Classification Macros

The following are defined for use with classification macros:

FP_NAN	The floating-point number *x* is ‘‘Not a Number’’.
FP_INFINITE	The value of the number is either plus or minus infinity.
FP_ZERO	The value of the number is either plus or minus zero.
FP_SUBNORMAL	The number is in denormalized format.
FP_NORMAL	There is nothing special about the number.

The macro *fpclassify*( ) classifies its argument as either NaN, infinite, normal, subnormal, zero, or into another implementation-defined category.

```
int fpclassify(real-floating x);
```

The macro *isfinite*( ) determines whether its argument has a finite value (zero, subnormal, or normal, and not infinite or NaN).

```
int isfinite(real-floating x);
```

The macro *isinf*( ) determines whether its argument value is an infinity (positive or negative).

```
int isinf(real-floating x);
```

The macro *isnam*( ) determines whether its argument value is a NaN.

```
int isnan(real-floating x);
```

The macro *isnormal*( ) determines whether its argument value is normal (neither zero, subnormal, infinite, nor NaN).

```
int isnormal(real-floating x);
```

The macro *signbit*( ) determines whether the sign of its argument value is negative.

```
int signbit(real-floating x);
```

16.10.2 Trigonometric Functions

The following functions compute the arc cosine, arc sin, and arctan of *x*, respectively:

```
float acosf(float x);
long double acosl(long double x);
float asinf(float x);
long double asinl(long double x);
long double tanl(long double x);
```

The following functions compute the arc tangent of *y*/*x*:

```
float atan2f(float y, float x);
long double atan2l(long double y, long double x);
```

The following functions compute the cosine, sin, and tangent of *x*, respectively:

```
float cosf(float x);
long double cosl(long double x);
float sinf(float x);
long double sinl(long double x);
float tanf(float x);
long double tanl(long double x);
```

16.10.3 Hyperbolic Functions

The following functions compute the arc hyperbolic cosine of *x*:

```
float acoshf(float x);
long double acoshl(long double x);
```

The following functions compute the arc hyperbolic sine of *x*:

```
float asinhf(float x);
long double asinhl(long double x);
```

The following functions compute the arc hyperbolic tangent of *x*:

```
float atanhf(float x);
long double atanhl(long double x);
```

The following functions compute the hyperbolic cosine of *x*:

```
float coshf(float x);
long double coshl(long double x);
```

The following functions compute the hyperbolic sine of *x*:

```
float sinhf(float x);
long double sinhl(long double x);
```

The following functions compute the hyperbolic tangent of *x*:

```
float tanhf(float x);
long double tanhl(long double x);
```

16.10.4 Exponential and Logarithmic Functions

The following functions compute the base-e exponential of *x*:

```
float expf(float x);
long double expl(long double x);
```

The following functions compute the base-2 exponential of *x*:

```
double exp2(double x);
float exp2f(float x);
long double exp2l(long double x);
```

The following functions compute the base-e exponential of (*x*–1):

```
float expm1f(float x);
long double expm1l(long double x);
```

The following functions break a floating-point number into a normalized fraction and an integral power of 2:

```
float frexpf(float value, int *exp);
long double frexpl(long double value, int *exp);
```

The following functions extract the exponent of *x*:

```
int ilogbf(float x);
int ilogbl(long double x);
```

The following functions multiply a floating-point number by an integral power of 2:

```
float ldexpf(float x, int exp);
long double ldexpl(long double x, int exp);
```

The following functions compute the natural logarithm of *x*:

```
float logf(float x);
long double logl(long double x);
```

The following functions compute the base-10 logarithm of *x*:

```
float log10f(float x);
long double log10l(long double x);
```

The following functions compute the natural logarithm of (*x*+1):

```
float log1pf(float x);
long double log1pl(long double x);
```

The following functions compute the base-2 logarithm of *x*:

```
double log2(double x);
float log2f(float x);
long double log2l(long double x);
```

The following functions extract the exponent of *x*:

```
float logbf(float x);
long double logbl(long double x);
```

The following functions break *x* into integral and fractional parts:

```
float modff(float x, float *iptr);
long double modfl(long double x, long double *iptr);
```

The following functions compute *x**FLT_RADIX***n* efficiently:

```
double scalbn(double x, int n);
float scalbnf(float x, int n);
long double scalbnl(long double x, int n);
double scalbln(double x, long int n);
float scalblnf(float x, long int n);
long double scalblnl(long double x, long int n);
```

The following functions compute the real cube root of *x*:

```
double cbrt(double x);
float cbrtf(float x);
long double cbrtl(long double x);
```

The following functions compute the absolute value of *x*:

```
float fabsf(float x);
long double fabsl(long double x);
```

The following functions compute the square root of the sum of the squares of *x* and *y*:

```
float hypotf(float x, float y);
long double hypotl(long double x, long double y);
```

The following functions compute *x* raised to the power of *y*:

```
float powf(float x, float y);
long double powl(long double x, long double y);
```

The following functions compute the non-negative square root of *x*:

```
float sqrtf(float x);
long double sqrtl(long double x);
```

The following functions compute the error function of *x*:

```
float erff(float x);
long double erfl(long double x);
```

The following functions compute the natural logarithm of the absolute value of the gamma function of *x*:

```
float lgammaf(float x);
long double lgammal(long double x);
```

The following functions compute the (true) gamma function of *x*:

```
double tgamma(double x);
float tgammaf(float x);
long double tgammal(long double x);
```

16.10.5 Nearest Integer Functions

The following functions compute the smallest integer value not less than *x*:

```
float ceilf(float x);
long double ceill(long double x);
```

The following functions compute the largest integer value not greater than *x*:

```
float floorf(float x);
long double floorl(long double x);
```

The following functions round *x* to an integer value in floating-point format, using the current rounding direction and without raising the inexact floating-point exception:

```
double nearbyint(double x);
float nearbyintf(float x);
long double nearbyintl(long double x);
```

The following functions round *x* to an integer value in floating-point format, using the current rounding direction and may raise the inexact floating-point exception if the result differs in value from the argument:

```
float rintf(float x);
long double rintl(long double x);
```

The following functions round *x* to the nearest integer value, rounding according to the current rounding direction:

```
long int lrint(double x);
long int lrintf(float x);
long int lrintl(long double x);
long long int llrint(double x);
long long int llrintf(float x);
long long int llrintl(long double x);
```

The following functions round *x* to the nearest integer value in floating-point format, rounding halfway cases away from zero, regardless of the current rounding direction:

```
double round(double x);
float roundf(float x);
long double roundl(long double x);
```

The following functions round *x* to the nearest integer value, rounding halfway cases away from zero, regardless of the current rounding direction:

```
long int lround(double x);
long int lroundf(float x);
long int lroundl(long double x);
long long int llround(double x);
long long int llroundf(float x);
long long int llroundl(long double x);
```

The following functions round *x* to the integer value, in floating format, nearest to but no larger in magnitude than *x*:

```
double trunc(double x);
float truncf(float x);
long double truncl(long double x);
```

16.10.6 Remainder Functions

The following functions compute the floating-point remainder of *x*/*y*:

```
float fmodf(float x, float y);
long double fmodl(long double x, long double y);
```

The following functions compute the IEC 60559: 1989 standard remainder *x* REM *y*:

```
float remainderf(float x, float y);
long double remainderl(long double x, long double y);
```

The following functions shall compute the same remainder as the remainder family of functions, but in a different manner:

```
double remquo(double x, double y, int *quo);
float remquof(float x, float y, int *quo);
long double remquol(long double x, long double y, int *quo);
```

16.10.7 Manipulation Functions

The following functions produce a value with the magnitude of *and the sign of y:*

```
double copysign(double x, double y);
float copysignf(float x, float y);
long double copysignl(long double x, long double y);
```

The following functions return a quiet NaN, if available, with content indicated by *tagp*:

```
double nan(const char *tagp);
float nanf(const char *tagp);
long double nanl(const char *tagp);
```

The following functions determine the next representable value:

```
float nextafterf(float x, float y);
long double nextafterl(long double x, long double y);
```

The following functions are equivalent to the *nextafter* functions, except that the second parameter has type **long double** and the functions return *y* converted to the type of the function if *x* equals *y*:

```
double nexttoward(double x, long double y);
float nexttowardf(float x, long double y);
long double nexttowardl(long double x, long double y);
```

The following functions determine the positive difference between their arguments:

```
double fdim(double x, double y);
float fdimf(float x, float y);
long double fdiml(long double x, long double y);
```

The following functions determine the maximum numeric value of their arguments:

```
double fmax(double x, double y);
float fmaxf(float x, float y);
long double fmaxl(long double x, long double y);
```

If the optional macros FP_FAST_FMA, FP_FAST_FMAF, and FP_FAST_FMAL are defined, it indicates that the corresponding *fma*() function executes at least as fast as a multiply and an add of double operands.

The following functions determine the minimum numeric value of their arguments:

```
double fmin(double x, double y);
float fminf(float x, float y);
long double fminl(long double x, long double y);
```

The following functions compute (*x***y*)+*z*, rounded as one ternary operation:

```
double fma(double x, double y, double z);
float fmaf(float x, float y, float z);
long double fmal(long double x, long double y,
    long double z);
```

16.10.8 Comparison Macros

The *isgreater*( ) macro tests whether *x* is greater than *y*.

```
int isgreater(real-floating x, real-floating y);
```

The *isgreaterequal*( ) macro tests whether *x* is greater than or equal to *y*.

```
int isgreaterequal(real-floating x, real-floating y);
```

The *isless*( ) macro tests whether *x* is less than *y*.

```
int isless(real-floating x, real-floating y);
```

The *islessequal*( ) macro tests whether *x* is less than or equal to *y*.

```
int islessequal(real-floating x, real-floating y);
```

The *islessgreater*( ) macro tests whether *x* is less than or greater than *y*.

```
int islessgreater(real-floating x, real-floating y);
```

The *isunordered*( ) macro tests whether *x* and *y* are unordered.

```
int isunordered(real-floating x, real-floating y);
```

Note: Annex F (normative) was added to specify the IEC 60559: 1989 standard floating-point arithmetic. An implementation that defines __STDC_IEC_559__ must conform to the specification detailed in this annex.

16.11 Floating-Point Environment Support

The header **<fenv.h>** declares the types, and defines the macros and functions that support access to an implementation's floating-point environment.

Two types are declared:

fenv_t Represents the entire floating-point environment.

fexcept_t Represents the collective floating-point status flags.

The **pragma** directive has three reserved forms, all starting with the preprocessor token **STDC**. These are used to specify certain characteristics of the floating-point support to comply with the IEC 60559: 1989 standard.

The **pragma STDC FENV_ACCESS** provides the means of informing an implementation when a program might access the floating-point environment.

For example:

```
double a;
#pragma STDC FENV_ACCESS ON
a = 1.0 + 2.0;
#pragma STDC FENV_ACCESS OFF
```

16.11.1 Exceptions

The following function clears the supported floating-point exceptions:

```
void feclearexcept(int excepts);
```

The following function stores an implementation-dependent representation of the states of the floating-point status flags:

```
void fegetexceptflag(fexcept_t *flagp, int excepts);
```

The following function raises the supported floating-point exceptions represented by its argument:

```
void feraiseexcept(int excepts);
```

The following function sets the floating-point status flags:

```
void fesetexceptflag(const fexcept_t *flagp, int excepts);
```

The following function tests which of a specified subset of the floating-point exception flags are currently set:

```
int fetestexcept(int excepts);
```

Each of the following floating-point exception macros is defined if an implementation supports these functions:

FE_DIVBYZERO
FE_INEXACT
FE_INVALID
FE_OVERFLOW
FE_UNDERFLOW
FE_ALL_EXCEPT (Bitwise OR of all the other macros.)

Additional implementation-defined floating-point exceptions, with macro definitions beginning with FE_ and an uppercase letter, may also be defined by an implementation.

16.11.2 Rounding

The following functions respectively set and return the current rounding direction:

```
int fesetround(int round);
int fegetround(void);
```

Each of the following floating-point macros is defined if an implementation supports these functions:

FE_DOWNWARD
FE_TONEAREST
FE_TOWARDZERO
FE_UPWARD

Additional implementation-defined rounding directions, with macro definitions beginning with FE_ and an uppercase letter, may also be defined by an implementation.

16.11.3 Environment

The following functions respectively set and return the floating-point environment function:

```
void fesetenv(const fenv_t *envp);
void fegetenv(fenv_t *envp);
```

The following function saves the currently raised floating-point exception(s), installs the floating-point environment represented by the object pointed to by *envp*, and then raises the saved floating-point exception(s):

```
void feupdateenv(const fenv_t *envp);
```

The following function saves the current floating-point environment in the object pointed to by *envp*, clears the floating-point status flags, and then installs a non-stop (continue on floating-point exceptions) mode, if available, for all floating-point exceptions:

```
int feholdexcept(fenv_t *envp);
```

The macro FE_DFL_ENV represents the default floating-point environment; that is, the one installed at program startup. It can be used as an argument with the above functions and is of type ***const fenv_t**.

16.12 Type-Generic Math

Type-generic macros may enable the writing of more portable code, and reduce need for casting and suffixing when porting applications to new platforms.

The header **<tgmath.h>** includes the headers **<math.h>** and **<complex.h>** and defines numerous type-generic macros. Except for *modf*, there is a type-generic macro for each of the functions in **<math.h>** and **<complex.h>** that do not have an `'f'` (**float**) or `'l'` (**long double**) suffix and have one or more parameters whose corresponding real type is **double**.

Such parameters are called generic parameters.

Use of a type-generic macro invokes a function whose corresponding real type and type domain are determined by the arguments for the generic parameters. The real type is determined as follows:

1. First, if any argument for generic parameters is a **long double**, the real type is **long double**.
2. Otherwise, if any argument for generic parameters is a double or an integer type, the real type is **double**.
3. Otherwise, the real type is **float**.

Type-generic macros that accept complex arguments also accept imaginary arguments. If an argument is imaginary, the macro expands to an expression whose type is **real**, **imaginary**, or **complex**, as appropriate for the particular function.

16.12.1 Unsuffixed Functions With a C-Prefixed Counterpart

For each unsuffixed function in **<math.h>** for which there is a function in **<complex.h>** 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 **<math.h>**.

For example, the type-generic macro for *tan*( ) and *ctan*( ) is *tan*.

If at least one argument for a generic parameter is complex, then use of the macro invokes a complex function; otherwise, a real function is invoked.

16.12.2 Unsuffixed Functions Without a C-Prefixed Counterpart

For each unsuffixed function in **<math.h>** for which there is not a function in **<complex.h>** with the same name but having a `'c'` prefix, the corresponding type-generic macro for both functions has the same name as the function in **<math.h>**. 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. Examples of such functions include *fdim*( ) and *lround*( ).

For each unsuffixed function in **<complex.h>** for which there is not a function in **<math.h>** with the same name but without a `'c'` prefix, the corresponding type-generic macro for both functions has the same name as the function in **<complex.h>**. Use of the macro with any real or complex argument invokes a complex function.

16.13 Other Library Changes

A number of new functions were added to the standard, prototypes for many functions now contain the new keyword **restrict** as part of some parameter declarations, and a number of functions had their definition clarified or extended.

atoll( )

A numeric conversion function for the conversion of a string to a **long long int** representation.

```
long long int atoll(const char *nptr);
```

_Exit( )

This function causes normal program termination to occur and control to be returned to the host environment without triggering signals or *atexit*( ) registered functions.

This function name (rather than *_exit*( )) was chosen to avoid potential conflict with existing practice.

fpos_t

The description of **fpos_t** was changed to exclude array type objects.

isblank()

This function tests whether *c* is a character of class **blank** in a program's current locale.

```
int isblank(int c);
```

iswblank()

This function tests whether *wc* is a wide-character which is a member of the class **blank** in the program's current locale.

```
int iswblank(wint_t wc);
```

llabs()

In a similar manner to its counterparts *abs*() and *lals*(), this function computes the absolute value of an integer.

```
long long int llabs(long long int j);
```

lldiv()

In a similar manner to its counterparts *div*() and *ldiv*(), this function retuns a structure of type **lldiv_t** which contains both the quotient and the remainder, each of which is of type **long long int**.

```
lldiv_t lldiv(long long int numer, long long int denom);
```

localeconv()

The standard added the following members to the **lconv** structure (defined in **<locale.h>**) to assign with long-standing POSIX practice and to permit additional flexibility with internationally formatted monetary quantities:

char p_cs_precedes Set to 1 or 0 if the **currency_symbol** respectively precedes or succeeds the value for a non-negative locally formatted monetary quantity.

char n_cs_precedes Set to 1 or 0 if the **currency_symbol** respectively precedes or succeeds the value for a negative locally formatted monetary quantity.

char p_sep_by_space Set to a value indicating the separation of the **currency_symbol**, the sign string, and the value for a non-negative locally formatted monetary quantity.

char n_sep_by_space Set to a value indicating the separation of the **currency_symbol**, the sign string, and the value for a negative locally formatted monetary quantity.

char p_sign_posn Set to a value indicating the positioning of the **positive_sign** for a non-negative locally formatted monetary quantity.

char n_sign_posn Set to a value indicating the positioning of the **negative_sign** for a negative locally formatted monetary quantity.

printf(), fprintf(), sprintf()

New length modifiers were added to the standard:

`hh` Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **signed char** or **unsigned char** argument; or that a following `n` conversion specifier applies to a pointer to a **signed char** argument. This modifier enables character types to be treated the same as all other integer types.

`ll` Added to support the new **long long int** type. Specifies that a following `d`, `I`, `o`, `u`, `x`, or `X` conversion specifier applies to a **long long int** or **unsigned long long int** argument; or that a following `n` conversion specifier applies to a pointer to a **long long int** argument.

The maximum number of characters that can be produced by any single conversion was increased from 509 characters (C89) to 4 095 characters.

realloc()

The description of this function was changed to make it clear that the pointed-to object is deallocated, a new object is allocated, and the content of the new object is the same as that of the old object up to the lesser of the two sizes.

scanf(), fscanf(), sscanf()

The `hh` and `ll` length modifiers (see *printf*() above) were added.

Also the conversion modifiers `a` and `A` were added with `A` being equivalent to `a`.

These conversion modifiers match an optionally signed floating-point number, infinity, or NaN, whose format is the same as expected for the subject sequence of the *strtod*() function. The corresponding argument shall be a pointer to floating.

The behavior of the *sscanf*() function on encountering the end of a string has been clarified.

setvbuf()

The function prototype was changed to include the **restrict** type qualifier:

```
int setvbuf(FILE *restrict stream, char *restrict buf,
    int type, size_t size);
```

In previous revisions of the standard it was not clear about what, if anything, *size* means when *buf* is a null pointer. The standard now warns that *size* might not be ignored, so portable programs should supply a reasonable value.

snprintf()

This function was added to the standard to address the problem of *sprintf*() potentially overrunning an output buffer. It is equivalent in functionality to *sprintf*() except that it performs bounds checking on the output array. Extra characters are discarded and a null character is written at the end of the characters actually written to the array.

strftime( )

The definition of this function was changed to incorporate additional conversion specifiers defined in IEEE Std 1003.1-1996 (POSIX.1), including %C, %D, %e, %F, %g, %G, %h, %n, %r, %R, %t, %T, %u, and %V, as well as the E and O modifiers.

strtod( ), strtof( ), strtold ( )

The following two functions were added to the standard:

```
float strtof(const char *restrict nptr,
    char **restrict endptr);
long double strtold (const char *restrict nptr,
    char **restrict endptr);
```

In a similar manner to their counterpart, the *strtod*( ) function, these functions convert the initial portion of the string pointed to by *nptr* to **float** and **long double** representation, respectively. Support for subject sequences relating to floating-point (NaN, INF, and so on) was also added.

strtoll( ), strtoull( )

The following two functions were added to the standard:

```
long long int strtoll(const char *restrict nptr,
    char **restrict endptr, int base);
unsigned long int strtoull(const char *restrict nptr,
    char **restrict endptr, int basef5);
```

In a similar manner to their counterparts, the *strtol*( ) and *strtoul*( ) functions, these functions convert the initial portion of the string pointed to by *nptr* to **long long int** and **unsigned long int** representation, respectively.

tmpnam( )

The previous standard had a serious flaw regarding this function. If the function were called fewer than {TMP_MAX} times but was unable to generate a suitable string because every potential string named an existing file, there was no way to report failure and no undefined behavior; hence there was no option other than to never return.

This standard resolved this issue by allowing the function to return a null pointer when it cannot generate a suitable string and by specifying that {TMP_MAX} is the number of potential strings, any or all of which may name existing files and thus not be suitable return values.

Note: This is a quiet change in the standard. Programs that call this function without checking for a null return value may produce undefined behavior.

ungetc( )

The standard deprecated the use of this function on a binary stream at the beginning of the file.

vfscanf()

The following functions are functionally the same as *scanf*(), *fscanf*(), and *sscanf*() respectively, except that instead of being called with a variable number of arguments, they are called with an argument list:

```
int vscanf(const char *restrict format, va_list arg);
int vfscanf(FILE *restrict stream,
    const char *restrict format, va_list arg);
int vsscanf(const char *restrict s,
    const char *restrict format, va_list arg);
```

vfwscanf()

The following functions are functionally the same as *fwscanf*(), *swscanf*(), and *wscanf*() respectively, except that instead of being called with a variable number of arguments, they are called with an argument list:

```
int vfwscanf(FILE *restrict stream,
    const wchar_t *restrict format, va_list arg);
int vswscanf(const wchar_t *restrict s,
    const wchar_t *restrict format, va_list arg);
int vwscanf(const wchar_t *restrict format, va_list arg);
```

16.13.1 Wide-String Numeric Conversion Functions

The following functions were added to the existing wide-string numeric conversion functions:

```
float wcstof(const wchar_t *restrict nptr,
    wchar_t **restrict endptr);
long double wcstold(const wchar_t *restrict nptr,
    wchar_t **restrict endptr);
long long int wcstoll(const wchar_t *restrict nptr,
    wchar_t **restrict endptr, int base);
long long int wcstoull(const wchar_t *restrict nptr,
    wchar_t **restrict endptr, int base);
```

16.14 Annexes

A number of new normative and informative annexes were added to the standard and some exiting annexes were modified.

Annexes A, B, and E were modified to include the new keywords, universal character names, types, implementation limits, macros and functions, and other changes to the C language.

Annex F (normative) was added to specify the IEC 60559: 1989 standard floating-point arithmetic. An implementation that defines __STDC_IEC_559__ must conform to the specification detailed in this annex.

Annex G (informative) was added to specify recommended IEC 60559: 1989 standard-compatible complex arithmetic. An implementation that defines __STDC_IEC_559_COMPLEX__ should conform to the specification detailed in this annex. It is non-normative because there were few existing implementations at the time this standard was approved.

Annex H (informative) describes the extent of support in this standard for language-independent arithmetic as specified in the ISO/IEC 10967-1: 1994 standard. This annex was added, however, because all programming languages covered by ISO/IEC JTC1 SC22 standards are expected to review the ISO/IEC 10967-1: 1994 standard and incorporate and further define the binding between that standard and each programming language.

Information about the ISO C Working Group (JTC1/SC22/WG14) can be found at: *wwwold.dkuug.dk/JTC1/SC22/WG14/*.