This is an extract from:

*A Source Book from The Open Group*

**The Authorized Guide to the Single UNIX Specification, Version 4**

*The Open Group*

A Source Book from The Open Group

The Authorized Guide to the Single UNIX Specification, Version 4

Published in the U.K. by The Open Group, September 2010.

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 11

# System Interfaces Migration

## 11.1 Introduction

This chapter contains a section for each system interface defined in XSH, Issue 7. Each section contains the SYNOPSIS and gives the derivation of the interface. For interfaces new to Issue 7, a brief description is included, complete with examples where appropriate. For interfaces carried forward from Issue 6, syntax and semantic changes made to the interface in Issue 7 are identified (if any). Only changes that might affect an application programmer are included.

## 11.2 System Interfaces

**_Exit, _exit**

Purpose: Terminate a process.

Synopsis:

```
#include <stdlib.h>

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 7: Austin Group Interpretation 1003.1-2001 #085 is applied, clarifying the text regarding flushing of streams and closing of temporary files.

Functionality relating to the Asynchronous Input and Output, Memory Mapped Files, and Semaphores options is moved to the Base.

**_longjmp, _setjmp**

Purpose: Non-local goto.

OB XSI Synopsis:

```
#include <setjmp.h>

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

Derivation: First released in Issue 4, Version 2.

Issue 7: The *_longjmp*() and *_setjmp*() functions are marked obsolescent. Applications should use *siglongjmp*() and *sigsetjmp*() respectively.

**_tolower**

Purpose: Transliterate uppercase characters to lowercase.

OB XSI Synopsis:

```
#include <ctype.h>

int _tolower(int c);
```

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

Issue 7: The *_tolower*() function is marked obsolescent. Applications should use the *tolower*() function instead.

**_toupper**

Purpose: Transliterate lowercase characters to uppercase.

OB XSI Synopsis:

```
#include <ctype.h>

int _toupper(int c);
```

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

Issue 7: The *_toupper*() function is marked obsolescent. Applications should use the *toupper*() function instead.

**a64l, l64a**

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

XSI Synopsis:

```
#include <stdlib.h>

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

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

**abort**

Purpose: Generate an abnormal process abort.

Synopsis:

```
#include <stdlib.h>

void abort(void);
```

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

Issue 7: No functional changes are made in this issue.

**abs**

Purpose: Return an integer absolute value.

Synopsis:

```
#include <stdlib.h>

int abs(int i);
```

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

Issue 7: No functional changes are made in this issue.

**accept**

Purpose: Accept a new connection on a socket.

Synopsis:
```
#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 Commands and Utilities, Issue 5 (XCU5).

Issue 7: SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

Austin Group Interpretation 1003.1-2001 #044 is applied, changing the "may fail" [ENOBUFS] and [ENOMEM] errors to become "shall fail" errors.

Functionality relating to XSI STREAMS is marked obsolescent.

**access, faccessat**

Purpose: Determine accessibility of a file relative to directory file descriptor.

Synopsis:
```
#include <unistd.h>

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

The *faccessat*() function is equivalent to the *access*() function, except in the case where *path* specifies a relative path. In this case the file whose accessibility is to be determined is located relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the file descriptor was opened without O_SEARCH, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with O_SEARCH, the function does not perform the check.

The AT_EACCESS flag can be used to specify that checks for accessibility are performed using the effective user and group IDs instead of the real user and group ID.

The purpose of the *faccessat*() function is to enable the checking of the accessibility of files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *access*(), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *faccessat*() function it can be guaranteed that the file tested for accessibility is located relative to the desired directory.

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

Issue 7: Austin Group Interpretation 1003.1-2001 #046 is applied.

Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The *faccessat*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

**acos, acosf, acosl**

Purpose: Arc cosine functions.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**acosh, acoshf, acoshl**

Purpose: Inverse hyperbolic cosine functions.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**aio_cancel**

Purpose: Cancel an asynchronous I/O request.

Synopsis:
```
#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.

Issue 7: The *aio_cancel*() function is moved from the Asynchronous Input and Output option to the Base.

**aio_error**

Purpose: Retrieve errors status for an asynchronous I/O operation.

Synopsis:
```
#include <aio.h>

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

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

Issue 7: Austin Group Interpretation 1003.1-2001 #045 is applied, clarifying that the behavior is undefined if the **aiocb** structure pointed to by *aiocbp* is not associated with an operation that has been scheduled.

SD5-XSH-ERN-148 is applied, clarifying that when *aio_error*() fails it returns –1 and sets *errno*.

The *aio_error*() function is moved from the Asynchronous Input and Output option to the Base.

**aio_fsync**

Purpose: Asynchronous file synchronization.

Synopsis:
```
#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.

Issue 7: The *aio_fsync*() function is moved from the Asynchronous Input and Output option to the Base.

**aio_read**

Purpose: Asynchronous read from a file.

Synopsis:
```
#include <aio.h>

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

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

Issue 7: Austin Group Interpretation 1003.1-2001 #082 is applied.

The *aio_read*() function is moved from the Asynchronous Input and Output option to the Base.

**aio_return**

Purpose: Retrieve return status of an asynchronous I/O operation.

Synopsis:
```
#include <aio.h>

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

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

Issue 7: SD5-XSH-ERN-148 is applied, clarifying that when *aio_return*() fails it returns –1 and sets *errno*.

The *aio_return*() function is moved from the Asynchronous Input and Output option to the Base.

**aio_suspend**

Purpose: Wait for an asynchronous I/O request.

Synopsis:
```
#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.

Issue 7: The *aio_suspend*() function is moved from the Asynchronous Input and Output option to the Base.

**aio_write**

Purpose: Asynchronous write to a file.

Synopsis:
```
#include <aio.h>

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

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

Issue 7: Austin Group Interpretation 1003.1-2001 #082 is applied.

The *aio_write*() function is moved from the Asynchronous Input and Output option to the Base.

**alarm**

Purpose: Schedule an alarm signal.

Synopsis:
```
#include <unistd.h>

unsigned alarm(unsigned seconds);
```

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

Issue 7: No functional changes are made in this issue.

**alphasort, scandir**

Purpose: Scan a directory.

Synopsis:
```
#include <dirent.h>

int alphasort(const struct dirent **d1,
    const struct dirent **d2);
int scandir(const char *dir, struct dirent ***namelist,
    int (*sel)(const struct dirent *),
    int (*compar)(const struct dirent **,
    const struct dirent **));
```

The *alphasort*() function can be used as the comparison function for the *scandir*() function to sort the directory entries, *d1* and *d2*, into alphabetical order. Sorting happens as if by calling the *strcoll*() function on the *d_name* element of the **dirent** structures passed as the two parameters. If the *strcoll*() function fails, the return value of *alphasort*() is unspecified.

The *scandir*() function scans the directory *dir*, calling the function referenced by *sel* on each directory entry. Entries for which the function referenced by *sel* returns non-zero are stored in strings allocated as if by a call to *malloc*(), and sorted as if by a call to *qsort*() with the comparison function *compar*, except that *compar* need not provide total ordering. The strings are collected in array *namelist* which is allocated as if by a call to *malloc*().

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Issue 7: First released in Issue 7.

**asctime, asctime_r**

Purpose: Convert date and time to a string.

OB Synopsis:

```
#include <time.h>

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

OB CX

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

Issue 7: Austin Group Interpretation 1003.1-2001 #053 is applied, marking these functions obsolescent. Applications should use the *strftime*() function instead.

The *asctime_r*() function is moved from the Thread-Safe Functions option to the Base.

**asin, asinf, asinl**

Purpose: Arc sine function.

Synopsis:

```
#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 7: No functional changes are made in this issue.

**asinh, asinhf, asinhl**

Purpose: Inverse hyperbolic sine functions.

Synopsis:

```
#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 7: No functional changes are made in this issue.

**assert**

Purpose: Insert program diagnostics.

Synopsis:

```
#include <assert.h>

void assert(scalar expression);
```

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

Issue 7: No functional changes are made in this issue.

**atan, atanf, atanl**

Purpose: Arc tangent function.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**atan2, atan2f, atan2l**

Purpose: Arc tangent functions.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**atanh, atanhf, atanhl**

Purpose: Inverse hyperbolic tangent functions.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**atexit**

Purpose: Register a function to run at process termination.

Synopsis:
```
#include <stdlib.h>

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

Derivation: First released in Issue 4. Derived from the IEEE Std 1003.1b-1993.

Issue 7: No functional changes are made in this issue.

**atof**

Purpose: Convert a string to a double-precision number.

Synopsis:
```
#include <stdlib.h>

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

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

Issue 7: No functional changes are made in this issue.

**atoi**

Purpose: Convert a string to an integer.

Synopsis:
```
#include <stdlib.h>

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

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

Issue 7: No functional changes are made in this issue.

**atol, atoll**

Purpose: Convert a string to a long integer.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**basename**

Purpose: Return the last component of a pathname.

XSI Synopsis:
```
#include <libgen.h>

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

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

**bind**

Purpose: Bind a name to a socket.

Synopsis:
```
#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 Commands and Utilities, Issue 5 (XCU5).

Issue 7: Austin Group Interpretation 1003.1-2001 #044 is applied, changing the "may fail" [ENOBUFS] error to become a "shall fail" error.

Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

SD5-XSH-ERN-185 is applied, specifying asynchronous behavior for *bind*() when O_NONBLOCK is set for the socket.

An example is added.

**bsearch**

Purpose: Binary search a sorted table.

Synopsis:
```
#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 7: The EXAMPLES section is revised.

**btowc**

Purpose: Single byte to wide character conversion.

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

wint_t btowc(int c);
```

Derivation: First released in Issue 5. Included for alignment with .

Issue 7: No functional changes are made in this issue.

**cabs, cabsf, cabsl**

Purpose: Return a complex absolute value.

Synopsis:
```
#include <complex.h>

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

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

Issue 7: No functional changes are made in this issue.

**cacos, cacosf, cacosl**

Purpose: Complex arc cosine functions.

Synopsis:
```
#include <complex.h>

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

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

Issue 7: No functional changes are made in this issue.

**cacosh, cacoshf, cacoshl**

Purpose: Complex arc hyperbolic cosine functions.

Synopsis:
```
#include <complex.h>

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

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

Issue 7: No functional changes are made in this issue.

**calloc**

Purpose: A memory allocator.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**carg, cargf, cargl**

Purpose: Complex argument functions.

Synopsis:
```
#include <complex.h>

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

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

Issue 7: No functional changes are made in this issue.

**casin, casinf, casinl**

Purpose: Complex arc sine functions.

Synopsis:
```
#include <complex.h>

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

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

Issue 7: No functional changes are made in this issue.

**casinh, casinhf, casinhl**

Purpose: Complex arc hyperbolic sine functions.

Synopsis:
```
#include <complex.h>

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

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

Issue 7: No functional changes are made in this issue.

**catan, catanf, catanl**

Purpose: Complex arc tangent functions.

Synopsis:
```
#include <complex.h>

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

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

Issue 7: No functional changes are made in this issue.

**catanh, catanhf, catanhl**

Purpose: Complex arc hyperbolic tangent functions.

Synopsis:
```
#include <complex.h>

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

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

Issue 7: No functional changes are made in this issue.

**catclose**

Purpose: Close a message catalog descriptor.

Synopsis:
```
#include <nl_types.h>

int catclose(nl_catd catd);
```

Derivation: First released in Issue 2.

Issue 7: The *catclose*( ) function is moved from the XSI option to the Base.

**catgets**

Purpose: Read a program message.

Synopsis:
```
#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 7: Austin Group Interpretation 1003.1-2001 #044 is applied, changing the "may fail" [EINTR] and [ENOMSG] errors to become "shall fail" errors, updating the RETURN VALUE section, and updating the DESCRIPTION to note that the results are undefined if *catd* is not a value returned by *catopen*( ) for a message catalog still open in the process.

The *catgets*( ) function is moved from the XSI option to the Base.

**catopen**

Purpose: Open a message catalog.

Synopsis:
```
#include <nl_types.h>

nl_catd catopen(const char *name, int oflag);
```

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

The *catopen*( ) function is moved from the XSI option to the Base.

**cbrt, cbrtf, cbrtl**

Purpose: Cube root functions.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**ccos, ccosf, ccosl**

Purpose: Complex cosine functions.

Synopsis:
```
#include <complex.h>

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

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

Issue 7: No functional changes are made in this issue.

**ccosh, ccoshf, ccoshl**

Purpose: Complex hyperbolic cosine functions.

Synopsis:
```
#include <complex.h>

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

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

Issue 7: No functional changes are made in this issue.

**ceil, ceilf, ceill**

Purpose: Ceiling value function.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**cexp, cexpf, cexpl**

Purpose: Complex exponential functions.

Synopsis:
```
#include <complex.h>

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

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

Issue 7: No functional changes are made in this issue.

**cfgetispeed**

Purpose: Get input baud rate.

Synopsis:
```
#include <termios.h>

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

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

Issue 7: No functional changes are made in this issue.

**cfgetospeed**

Purpose: Get output baud rate.

Synopsis:
```
#include <termios.h>

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

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

Issue 7: No functional changes are made in this issue.

**cfsetispeed**

Purpose: Set input baud rate.

Synopsis:
```
#include <termios.h>

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

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

Issue 7: No functional changes are made in this issue.

**cfsetospeed**

Purpose: Set output baud rate.

Synopsis:
```
#include <termios.h>

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

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

Issue 7: No functional changes are made in this issue.

**chdir**

Purpose: Change working directory.

Synopsis:
```
#include <unistd.h>

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

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

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

**chmod, fchmodat**

Purpose: Change mode of a file relative to directory file descriptor.

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

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

The *fchmodat*() function is equivalent to the *chmod*() function except in the case where *path* specifies a relative path. In this case the file to be changed is determined relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the file descriptor was opened without O_SEARCH, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with O_SEARCH, the function does not perform the check.

The AT_SYMLINK_NOFOLLOW flag can be used to specify that if *path* names a symbolic link, then the mode of the symbolic link is changed.

The purpose of the *fchmodat*() function is to enable changing the mode of files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *chmod*(), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *fchmodat*() function it can be guaranteed that the changed file is located relative to the desired directory.

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

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The *fchmodat*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

**chown, fchownat**

Purpose: Change owner and group of a file relative to directory file descriptor.

Synopsis:

```
#include <unistd.h>

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

The *fchownat*( ) function is equivalent to the *chown*( ) and *lchown*( ) functions except in the case where *path* specifies a relative path. In this case the file to be changed is determined relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the file descriptor was opened without O_SEARCH, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with O_SEARCH, the function does not perform the check.

The AT_SYMLINK_NOFOLLOW flag controls whether *fchownat*( ) behaves like *chown*( ) or *lchown*( ): if AT_SYMLINK_NOFOLLOW is set and *path* names a symbolic link, ownership of the symbolic link is changed.

The purpose of the *fchownat*( ) function is to enable changing ownership of files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *chown*( ) or *lchown*( ), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *fchownat*( ) function it can be guaranteed that the changed file is located relative to the desired directory.

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

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The *fchownat*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

**cimag, cimagf, cimagl**

Purpose: Complex imaginary functions.

Synopsis:

```
#include <complex.h>

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

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

Issue 7: No functional changes are made in this issue.

**clearerr**

Purpose: Clear indicators on a stream.

Synopsis:
```
#include <stdio.h>
void clearerr(FILE *stream);
```

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

Issue 7: No functional changes are made in this issue.

**clock**

Purpose: Report CPU time used.

Synopsis:
```
#include <time.h>
clock_t clock(void);
```

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

Issue 7: No functional changes are made in this issue.

**clock_getcpuclockid**

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

CPT Synopsis:
```
#include <time.h>
int clock_getcpuclockid(pid_t pid, clockid_t *clock_id);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: No functional changes are made in this issue.

**clock_getres, clock_gettime, clock_settime**

Purpose: Clock and timer functions.

CX Synopsis:
```
#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.

Issue 7: Functionality relating to the Clock Selection option is moved to the Base.

The *clock_getres*( ), *clock_gettime*( ), and *clock_settime*( ) functions are moved from the Timers option to the Base.

**clock_nanosleep**

Purpose: High resolution sleep with specifiable clock.

CX Synopsis:

```
#include <time.h>

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

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: The *clock_nanosleep*() function is moved from the Clock Selection option to the Base.

**clog, clogf, clogl**

Purpose: Complex natural logarithm functions.

Synopsis:

```
#include <complex.h>

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

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

Issue 7: No functional changes are made in this issue.

**close**

Purpose: Close a file descriptor.

Synopsis:

```
#include <unistd.h>

int close(int fildes);
```

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

Issue 7: Functionality relating to the XSI STREAMS option is marked obsolescent.

Functionality relating to the Asynchronous Input and Output and Memory Mapped Files options is moved to the Base.

Austin Group Interpretation 1003.1-2001 #139 is applied, clarifying that the requirement for *close*() on a socket to block for up to the current linger interval is not conditional on the O_NONBLOCK setting.

**closedir**

Purpose: Close a directory stream.

Synopsis:

```
#include <dirent.h>

int closedir(DIR *dirp);
```

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**closelog, openlog, setlogmask, syslog**

Purpose: Control system log.

XSI Synopsis:

```
#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 7: No functional changes are made in this issue.

**confstr**

Purpose: Get configurable variables.

Synopsis:

```
#include <unistd.h>

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

Derivation: First released in Issue 4. Derived from the .

Issue 7: Austin Group Interpretation 1003.1-2001 #047 is applied, adding the _CS_V7_ENV variable.

Austin Group Interpretations 1003.1-2001 #166 is applied to permit an additional compiler flag to enable threads.

The V6 variables for the supported programming environments are marked obsolescent.

The variables for the supported programming environments are updated to be V7.

The LEGACY variables and obsolescent values are removed.

**conj, conjf, conjl**

Purpose: Complex conjugate functions.

Synopsis:

```
#include <complex.h>

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

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

Issue 7: No functional changes are made in this issue.

**connect**

Purpose: Connect a socket.

Synopsis:
```
#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 Commands and Utilities, Issue 5 (XCU5).

Issue 7: Austin Group Interpretation 1003.1-2001 #035 is applied, clarifying the description of connected sockets.

Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

Austin Group Interpretation 1003.1-2001 #188 is applied, changing the method used to reset a peer address for a datagram socket.

**copysign, copysignf, copysignl**

Purpose: Number manipulation function.

Synopsis:
```
#include <math.h>

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

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

Issue 7: No functional changes are made in this issue.

**cos, cosf, cosl**

Purpose: Cosine function.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**cosh, coshf, coshl**

Purpose: Hyperbolic cosine functions.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**cpow, cpowf, cpowl**

Purpose: Complex power functions.

Synopsis:
```
#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);
```

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

Issue 7: No functional changes are made in this issue.

**cproj, cprojf, cprojl**

Purpose: Complex projection functions.

Synopsis:
```
#include <complex.h>

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

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

Issue 7: No functional changes are made in this issue.

**creal, crealf, creall**

Purpose: Complex real functions.

Synopsis:
```
#include <complex.h>

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

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

Issue 7: No functional changes are made in this issue.

**creat**

Purpose: Create a new file or rewrite an existing one.

OH Synopsis:
```
#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 7: No functional changes are made in this issue.

**crypt**

Purpose: String encoding function (**CRYPT**).

XSI Synopsis:

```
#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 7: SD5-XSH-ERN-178 is applied, clarifying the required contents of the *salt* argument.

**csin, csinf, csinl**

Purpose: Complex sine functions.

Synopsis:

```
#include <complex.h>

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

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

Issue 7: No functional changes are made in this issue.

**csinh, csinhf, csinhl**

Purpose: Complex hyperbolic sine functions.

Synopsis:

```
#include <complex.h>

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

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

Issue 7: No functional changes are made in this issue.

**csqrt, csqrtf, csqrtl**

Purpose: Complex square root functions.

Synopsis:

```
#include <complex.h>

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

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

Issue 7: No functional changes are made in this issue.

**ctan, ctanf, ctanl**

Purpose: Complex tangent functions.

Synopsis:
```
#include <complex.h>

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

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

Issue 7: No functional changes are made in this issue.

**ctanh, ctanhf, ctanhl**

Purpose: Complex hyperbolic tangent functions.

Synopsis:
```
#include <complex.h>

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

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

Issue 7: No functional changes are made in this issue.

**ctermid**

Purpose: Generate a pathname for the controlling terminal.

CX Synopsis:
```
#include <stdio.h>

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

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

Issue 7: Austin Group Interpretation 1003.1-2001 #148 is applied, clarifying the thread-safety requirements for the *ctermid*( ) function.

**ctime, ctime_r**

Purpose: Convert a time value to a date and time string.

OB Synopsis:
```
#include <time.h>

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

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

Issue 7: SD5-XSH-ERN-25 is applied, noting in APPLICATION USAGE that attempts to use *ctime*( ) or *ctime_r*( ) for times before the Epoch or for times beyond the year 9999 produce undefined results.

Austin Group Interpretation 1003.1-2001 #053 is applied, marking these functions obsolescent. Applications should use *strftime*( ) to generate strings from broken-down times. Values for the broken-down time structure can be obtained by calling *gmtime*( ) or *localtime*( ).

The *ctime_r*( ) function is moved from the Thread-Safe Functions option to the

Base.

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

Purpose: Database functions.

XSI Synopsis:

```
#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 7: Austin Group Interpretation 1003.1-2001 #042 is applied so that the DESCRIPTION permits newer implementations of the Berkeley DB interface.

**difftime**

Purpose: Compute the difference between two calendar time values.

Synopsis:

```
#include <time.h>

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

Derivation: First released in Issue 4. Derived from the IEEE Std 1003.1i-1995.

Issue 7: No functional changes are made in this issue.

**dirfd**

Purpose: Extract the file descriptor used by a DIR stream.

Synopsis:

```
#include <dirent.h>

int dirfd(DIR *dirp);
```

The *dirfd*() function returns a file descriptor referring to the same directory as the *dirp* argument. This file descriptor is closed by a call to *closedir*().

The *dirfd*() function is intended to be a mechanism by which an application may obtain a file descriptor to use for the *fchdir*() function.

This interface was introduced because the Base Definitions volume of IEEE Std 1003.1-2001 does not make public the **DIR** data structure. Applications tend to use the *fchdir*() function on the file descriptor returned by this interface, and this has proven useful for security reasons; in particular, it is a better technique than others where directory names might change.

The description uses the term ''a file descriptor'' rather than ''the file descriptor''. The implication intended is that an implementation that does not use an *fd* for

*diropen*( ) could still *open*( ) the directory to implement the *dirfd*( ) function. Such a descriptor must be closed later during a call to *closedir*( ).

An implementation that does not support file descriptors referring to directories may fail with [ENOTSUP].

If it is necessary to allocate an *fd* to be returned by *dirfd*( ), it should be done at the time of a call to *opendir*( ).

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Issue 7: First released in Issue 7.

**dirname**

Purpose: Report the parent directory name of a file pathname.

XSI Synopsis:

```
#include <libgen.h>

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

Derivation: First released in Issue 4, Version 2.

Issue 7: The EXAMPLES section is revised.

**div**

Purpose: Compute the quotient and remainder of an integer division.

Synopsis:

```
#include <stdlib.h>

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

Derivation: First released in Issue 4. Derived from the IEEE Std 1003.1i-1995.

Issue 7: No functional changes are made in this issue.

**dlclose**

Purpose: Close a *dlopen*( ) object.

Synopsis:

```
#include <dlfcn.h>

int dlclose(void *handle);
```

Derivation: First released in Issue 5.

Issue 7: The *dlopen*( ) function is moved from the XSI option to Base.

**dlerror**

Purpose: Get diagnostic information.

Synopsis:

```
#include <dlfcn.h>

char *dlerror(void);
```

Derivation: First released in Issue 5.

Issue 7: The *dlerror*( ) function is moved from the XSI option to the Base.

**dlopen**

Purpose: Gain access to an executable object file.

Synopsis:
```
#include <dlfcn.h>

void *dlopen(const char *file, int mode);
```

Derivation: First released in Issue 5.

Issue 7: The *dlopen*( ) function is moved from the XSI option to the Base.

The EXAMPLES section is updated to refer to *dlsym*( ).

**dlsym**

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

Synopsis:
```
#include <dlfcn.h>

void *dlsym(void *restrict handle, const char *restrict name);
```

Derivation: First released in Issue 5.

Issue 7: The *dlsym*( ) function is moved from the XSI option to the Base.

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

Purpose: Generate uniformly distributed pseudo-random numbers.

XSI Synopsis:
```
#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 7: No functional changes are made in this issue.

**dup, dup2**

Purpose: Duplicate an open file descriptor.

Synopsis:
```
#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.

Issue 7: SD5-XSH-ERN-187 is applied, clarifying several aspects of the behavior of *dup2*( ).

**duplocale**

Purpose: Duplicate a locale object.

CX Synopsis:

```
#include <locale.h>

locale_t duplocale(locale_t locobj);
```

The *duplocale*() function creates duplicate copy of the locale object referenced by the *locobj* argument.

The use of the *duplocale*() function is recommended for situations where a locale object is being used in multiple places, and it is possible that the lifetime of the locale object might end before all uses are finished. Another reason to duplicate a locale object is if a slightly modified form is needed. This can be achieved by a call to *newlocale*() following the *duplocale*() call.

As with the *newlocale*() function, handles for locale objects created by the *duplocale*() function should be released by a corresponding call to *freelocale*().

The following example shows a code fragment to create a slightly altered version of an existing locale object. The function takes a locale object and a locale name and it replaces the *LC_TIME* category data in the locale object with that from the named locale.

```
#include <locale.h>
...
locale_t
with_changed_lc_time (locale_t obj, const char *name)
{
    locale_t retval = duplocale (obj);
    if (retval != (locale_t) 0)
    {
        locale_t changed = newlocale (LC_TIME_MASK, name,
            retval);
        if (changed == (locale_t) 0)
        /* An error occurred. Free all allocated resources. */
            freelocale (retval);
        retval = changed;
    }
    return retval; }
}
```

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 4.

Issue 7: First released in Issue 7.

**encrypt**

Purpose: Encoding function (**CRYPT**).

XSI Synopsis:

```
#include <unistd.h>

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

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

Issue 7: No functional changes are made in this issue.

**endgrent, getgrent, setgrent**

Purpose: Group database entry functions.

XSI Synopsis:

```
#include <grp.h>

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

Derivation: First released in Issue 4, Version 2.

Issue 7: SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

**endhostent, gethostent, sethostent**

Purpose: Network host database functions.

Synopsis:

```
#include <netdb.h>

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

Derivation: First released in Issue 6. Derived from the Commands and Utilities, Issue 5 (XCU5).

Issue 7: No functional changes are made in this issue.

**endnetent, getnetbyaddr, getnetbyname, getnetent, setnetent**

Purpose: Network database functions.

Synopsis:

```
#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 Commands and Utilities, Issue 5 (XCU5).

Issue 7: No functional changes are made in this issue.

**endprotoent, getprotobyname, getprotobynumber, getprotoent, setprotoent**

Purpose: Network protocol database functions.

Synopsis:
```
#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 Commands and Utilities, Issue 5 (XCU5).

Issue 7: No functional changes are made in this issue.

**endpwent, getpwent, setpwent**

Purpose: User database functions.

XSI Synopsis:
```
#include <pwd.h>

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

Derivation: First released in Issue 4, Version 2.

Issue 7: SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

The EXAMPLES section is revised.

**endservent, getservbyname, getservbyport, getservent, setservent**

Purpose: Network services database functions.

Synopsis:
```
#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 Commands and Utilities, Issue 5 (XCU5).

Issue 7: SD5-XBD-ERN-14 is applied, clarifying the way in which port numbers are converted to and from network byte order.

**endutxent, getutxent, getutxid, getutxline, pututxline, setutxent**

Purpose: User accounting database functions.

XSI Synopsis:

```
#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 7: No functional changes are made in this issue.

**erf, erff, erfl**

Purpose: Error functions.

Synopsis:

```
#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 7: No functional changes are made in this issue.

**erfc, erfcf, erfcl**

Purpose: Complementary error functions.

Synopsis:

```
#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 7: No functional changes are made in this issue.

**errno**

Purpose: Error return value.

Synopsis:

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

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

Issue 7: No functional changes are made in this issue.

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

Purpose: Execute a file.

Synopsis:

```
#include <unistd.h>

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

The *fexecve*( ) function is equivalent to the *execve*( ) function except that the file to be executed is determined by the file descriptor *fd* instead of a pathname. The file offset of *fd* is ignored.

The purpose of the *fexecve*( ) function is to enable executing a file which has been verified to be the intended file. It is possible to actively check the file by reading from the file descriptor and be sure that the file is not exchanged for another between the reading and the execution. Alternatively, an function like *openat*( ) can be used to open a file which has been found by reading the content of a directory using *readdir*( ).

Since execute permission is checked by *fexecve*( ), the file descriptor *fd* need not have been opened with the O_EXEC flag. However, if the file to be executed denies read and write permission for the process preparing to do the *exec*, the only way to provide the *fd* to *fexecve*( ) will be to use the O_EXEC flag when opening *fd*. In this case, the application will not be able to perform a checksum test since it will not be able to read the contents of the file.

Note that when a file descriptor is opened with O_RDONLY, O_RDWR, or O_WRONLY mode, the file descriptor can be used to read, read and write, or write the file, respectively, even if the mode of the file changes after the file was opened. Using the O_EXEC open mode is different; *fexecve*( ) will ignore the mode that was used when the file descriptor was opened and the *exec* will fail if the mode of the file associated with *fd* does not grant execute permission to the calling process at the time *fexecve*( ) is called.

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

Issue 7: Austin Group Interpretation 1003.1-2001 #047 is applied, adding a warning for *execle*( ), *execve*( ) and *fexecve*( ) to the APPLICATION USAGE that the new process might be invoked in a nonconforming environment if the *envp* array does not contain implementation-defined variables required by the implementation to provide a conforming environment. See the _CS_V7_ENV entry in **<unistd.h>**, and *confstr*( ), for details.

Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The *fexecve*( ) function is added from The Open Group Technical Standard, 2006,

Extended API Set Part 2.

Functionality relating to the Asynchronous Input and Output, Memory Mapped Files, Threads, and Timers options is moved to the Base.

Changes are made related to support for finegrained timestamps.

**exit**

Purpose: Terminate a process.

Synopsis:
```
#include <stdlib.h>

void exit(int status);
```

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

Issue 7: Austin Group Interpretation 1003.1-2001 #085 is applied, deleting the reference to removal of files created by *tmpfile*( ).

**exp, expf, expl**

Purpose: Exponential function.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**exp2, exp2f, exp2l**

Purpose: Exponential base 2 functions.

Synopsis:
```
#include <math.h>

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

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

Issue 7: No functional changes are made in this issue.

**expm1, expm1f, expm1l**

Purpose: Compute exponential functions.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**fabs, fabsf, fabsl**

Purpose: Absolute value function.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**fattach**

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

OB XSR Synopsis:
```
#include <stropts.h>

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

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The *fattach*( ) function is marked obsolescent.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

**fchdir**

Purpose: Change working directory.

Synopsis:
```
#include <unistd.h>

int fchdir(int fildes);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: The *fchdir*( ) function is moved from the XSI option to the Base.

**fchmod**

Purpose: Change mode of a file.

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

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

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

**fchown**

Purpose: Change owner and group of a file.

Synopsis:
```
#include <unistd.h>

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

Derivation: First released in Issue 4, Version 2.

Issue 7: Functionality relating to XSI STREAMS is marked obsolescent.

**fclose**

Purpose: Close a stream.

Synopsis:
```
#include <stdio.h>

int fclose(FILE *stream);
```

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

Issue 7: Austin Group Interpretation 1003.1-2001 #002 is applied, clarifying the interaction of file descriptors and streams.

The [ENOSPC] error condition is updated and the [ENOMEM] error is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Changes are made related to support for finegrained timestamps.

**fcntl**

Purpose: File control.

Synopsis:
```
#include <fcntl.h>

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

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

Issue 7: Austin Group Interpretation 1003.1-2001 #150 is applied, clarifying the file status flags returned when *cmd* is F_GETFL.

Austin Group Interpretation 1003.1-2001 #171 is applied, adding the F_DUPFD_CLOEXEC flag.

The optional **<unistd.h>** header is removed from this function, since **<fcntl.h>** now defines SEEK_SET, SEEK_CUR, and SEEK_END as part of the Base.

**fdatasync**

Purpose: Synchronize the data of a file (**REALTIME**).

SIO Synopsis:
```
#include <unistd.h>

int fdatasync(int fildes);
```

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

Issue 7: No functional changes are made in this issue.

**fdetach**

Purpose: Detach a name from a STREAMS-based file descriptor (f3STREAMSfP).

OB XSR Synopsis:

```
#include <stropts.h>

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

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The *fdetach*() function is marked obsolescent.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

**fdim, fdimf, fdiml**

Purpose: Compute positive difference between two floating-point numbers.

Synopsis:

```
#include <math.h>

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

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

Issue 7: No functional changes are made in this issue.

**fdopen**

Purpose: Associate a stream with a file descriptor.

CX Synopsis:

```
#include <stdio.h>

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

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

Issue 7: SD5-XSH-ERN-149 is applied, adding the {STREAM_MAX} [EMFILE] error condition.

Changes are made related to support for finegrained timestamps.

**fdopendir, opendir**

Purpose: Open directory associated with file descriptor.

Synopsis:

```
#include <dirent.h>

DIR *fdopendir(int fd);
DIR *opendir(const char *dirname);
```

The *fdopendir*() function is equivalent to the *opendir*() function except that the directory is specified by a file descriptor rather than by a name. The file offset associated with the file descriptor at the time of the call determines which entries are returned.

Upon successful return from *fdopendir*(), the file descriptor is under the control of the system, and if any attempt is made to close the file descriptor, or to modify the state of the associated description, other than by means of *closedir*(), *readdir*(), *readdir_r*(), or *rewinddir*(), the behavior is undefined. Upon calling *closedir*() the file descriptor is closed.

The purpose of the *fdopendir*() function is to enable opening files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *opendir*(), resulting in unspecified behavior.

The following example program searches through a given directory looking for files whose name does not begin with a dot and whose size is larger than 1 MiB.

```
#include <stdio.h>
#include <dirent.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <stdint.h>
#include <stdlib.h>
#include <unistd.h>

int
main(int argc, char *argv[])
{
    struct stat statbuf;
    DIR *d;
    struct dirent *dp;
    int dfd, ffd;

    if ((d = fdopendir((dfd = open("./tmp", O_RDONLY))))
            == NULL) {
        fprintf(stderr, "Cannot open ./tmp directory\n");
        exit(1);
    }
    while ((dp = readdir(d)) != NULL) {
        if (dp->d_name[0] == '.')
            continue;
        /* there is a possible race condition here as the file
         * could be renamed between the readdir and the open */
        if ((ffd = openat(dfd, dp->d_name, O_RDONLY)) == -1) {
            perror(dp->d_name);
            continue;
        }
        if (fstat(ffd, &statbuf) == 0 && statbuf.st_size >
               (1024*1024)) {
            /* found it ... */
            printf("%s: %jdK\n", dp->d_name,
                (intmax_t)(statbuf.st_size / 1024));
        }
        close(ffd);
    }
    closedir(d); // note this implicitly closes dfd
    return 0;
}
```

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

The *fdopendir*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

An additional example is added.

**feclearexcept**

Purpose: Clear floating-point exception.

Synopsis:
```
#include <fenv.h>

int feclearexcept(int excepts);
```

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

Issue 7: No functional changes are made in this issue.

**fegetenv, fesetenv**

Purpose: Get and set current floating-point environment.

Synopsis:
```
#include <fenv.h>

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

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

Issue 7: No functional changes are made in this issue.

**fegetexceptflag, fesetexceptflag**

Purpose: Get and set floating-point status flags.

Synopsis:
```
#include <fenv.h>

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

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

Issue 7: No functional changes are made in this issue.

**fegetround, fesetround**

Purpose: Get and set current rounding direction.

Synopsis:
```
#include <fenv.h>

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

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

Issue 7: No functional changes are made in this issue.

**feholdexcept**

Purpose: Save current floating-point environment.

Synopsis:
```
#include <fenv.h>
int feholdexcept(fenv_t *envp);
```

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

Issue 7: No functional changes are made in this issue.

**feof**

Purpose: Test end-of-file indicator on a stream.

Synopsis:
```
#include <stdio.h>
int feof(FILE *stream);
```

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

Issue 7: No functional changes are made in this issue.

**feraiseexcept**

Purpose: Raise floating-point exception.

Synopsis:
```
#include <fenv.h>
int feraiseexcept(int excepts);
```

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

Issue 7: No functional changes are made in this issue.

**ferror**

Purpose: Test error indicator on a stream.

Synopsis:
```
#include <stdio.h>
int ferror(FILE *stream);
```

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

Issue 7: No functional changes are made in this issue.

**fetestexcept**

Purpose: Test floating-point exception flags.

Synopsis:
```
#include <fenv.h>
int fetestexcept(int excepts);
```

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

Issue 7: No functional changes are made in this issue.

**feupdateenv**

Purpose: Update floating-point environment.

Synopsis:
```
#include <fenv.h>
int feupdateenv(const fenv_t *envp);
```

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

Issue 7: No functional changes are made in this issue.

**fflush**

Purpose: Flush a stream.

Synopsis:
```
#include <stdio.h>
int fflush(FILE *stream);
```

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

Issue 7: Austin Group Interpretation 1003.1-2001 #002 is applied, clarifying the interaction of file descriptors and streams.

The [ENOSPC] error condition is updated and the [ENOMEM] error is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

The EXAMPLES section is revised.

Changes are made related to support for finegrained timestamps.

**ffs**

Purpose: Find first set bit.

XSI Synopsis:
```
#include <strings.h>
int ffs(int i);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

**fgetc**

Purpose: Get a byte from a stream.

Synopsis:
```
#include <stdio.h>
int fgetc(FILE *stream);
```

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

Issue 7: Austin Group Interpretation 1003.1-2001 #051 is applied, updating the list of functions that mark the last data access timestamp for update.

**fgetpos**

Purpose: Get current file position information.

Synopsis:
```
#include <stdio.h>

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

Derivation: First released in Issue 4. Derived from the IEEE Std 1003.1i-1995.

Issue 7: No functional changes are made in this issue.

**fgets**

Purpose: Get a string from a stream.

Synopsis:
```
#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 7: Austin Group Interpretation 1003.1-2001 #051 is applied, updating the list of functions that mark the last data access timestamp for update.

**fgetwc**

Purpose: Get a wide-character code from a stream.

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

wint_t fgetwc(FILE *stream);
```

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

Issue 7: Austin Group Interpretation 1003.1-2001 #051 is applied, clarifying the RETURN VALUE section.

Changes are made related to support for finegrained timestamps.

**fgetws**

Purpose: Get a wide-character string from a stream.

Synopsis:
```
#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 7: Austin Group Interpretation 1003.1-2001 #051 is applied, clarifying the RETURN VALUE section.

Changes are made related to support for finegrained timestamps.

**fileno**

Purpose: Map a stream pointer to a file descriptor.

CX Synopsis:

```
#include <stdio.h>

int fileno(FILE *stream);
```

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

Issue 7: SD5-XBD-ERN-99 is applied, changing the definition of the [EBADF] error.

**flockfile, ftrylockfile, funlockfile**

Purpose: Stdio locking functions.

CX Synopsis:

```
#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 7: The *flockfile*( ), *ftrylockfile*( ), and *funlockfile*( ) functions are moved from the Thread-Safe Functions option to the Base.

**floor, floorf, floorl**

Purpose: Floor function.

Synopsis:

```
#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 7: No functional changes are made in this issue.

**fma, fmaf, fmal**

Purpose: Floating-point multiply-add.

Synopsis:

```
#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);
```

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

Issue 7: ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #57 (SD5-XSH-ERN-69) is applied, adding a ''may fail'' range error for non-MX systems.

**fmax, fmaxf, fmaxl**

Purpose: Determine maximum numeric value of two floating-point numbers.

Synopsis:
```
#include <math.h>

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

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

Issue 7: Austin Group Interpretation 1003.1-2001 #007 is applied, adding MX shading where the text refers to NaNs.

**fmemopen**

Purpose: Open a memory buffer stream.

CX Synopsis:
```
#include <stdio.h>

FILE *fmemopen(void *restrict buf, size_t size,
    const char *restrict mode);
```

The *fmemopen*( ) function associates the buffer given by the *buf* and *size* arguments with a stream.

This interface has been introduced to eliminate many of the errors encountered in the construction of strings, notably overflowing of strings. This interface prevents overflow.

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Issue 7: First released in Issue 7.

**fmin, fminf, fminl**

Purpose: Determine minimum numeric value of two floating-point numbers.

Synopsis:
```
#include <math.h>

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

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

Issue 7: Austin Group Interpretation 1003.1-2001 #008 is applied, adding MX shading where the text refers to NaNs.

**fmod, fmodf, fmodl**

Purpose: Floating-point remainder value function.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**fmtmsg**

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

XSI Synopsis:

```
#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 7: No functional changes are made in this issue.

**fnmatch**

Purpose: Match a filename or a pathname.

Synopsis:

```
#include <fnmatch.h>

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

Derivation: First released in Issue 4. Derived from the .

Issue 7: No functional changes are made in this issue.

**fopen**

Purpose: Open a stream.

Synopsis:

```
#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 7: Austin Group Interpretation 1003.1-2001 #025 is applied, clarifying the file creation mode.

Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

Austin Group Interpretation 1003.1-2001 #159 is applied, clarifying requirements for the flags set on the open file description.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XSH-ERN-149 is applied, changing the {STREAM_MAX} [EMFILE] error condition from a "may fail" to a "shall fail".

Changes are made related to support for finegrained timestamps.

**fork**

Purpose: Create a new process.

Synopsis:
```
#include <unistd.h>

pid_t fork(void);
```

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

Issue 7: Austin Group Interpretation 1003.1-2001 #080 is applied, clarifying the status of asynchronous input and asynchronous output operations and asynchronous control lists in the DESCRIPTION.

Functionality relating to the Asynchronous Input and Output, Memory Mapped Files, Timers, and Threads options is moved to the Base.

Functionality relating to message catalog descriptors is moved from the XSI option to the Base.

**fpathconf, pathconf**

Purpose: Get configurable pathname variables.

Synopsis:
```
#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 the IEEE Std 1003.1-1988 (POSIX.1).

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

Austin Group Interpretation 1003.1-2001 #160 is applied, clarifying the requirements related to variables that have no limit.

Changes are made related to support for finegrained timestamps.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

**fpclassify**

Purpose: Classify real floating type.

Synopsis:
```
#include <math.h>

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

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

Issue 7: No functional changes are made in this issue.

**dprintf, fprintf, printf, snprintf, sprintf**

Purpose: Print formatted output.

Synopsis:

```
#include <stdio.h>

CX  int dprintf(int fildes, const char *restrict format, ...);
    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, ...);
```

The *dprintf*( ) function is equivalent to the *fprintf*( ) function, except that *dprintf*( ) writes output to the file associated with the file descriptor specified by the *fildes* argument rather than placing output on a stream.

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

Issue 7: Austin Group Interpretation 1003.1-2001 #161 is applied, updating the DESCRIPTION of the 0 flag.

Austin Group Interpretation 1003.1-2001 #170 is applied, changing the [EILSEQ] error condition from a "may fail" to a "shall fail".

ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #68 (SD5-XSH-ERN-70) is applied, revising the description of `g` and `G`.

The *dprintf*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Functionality relating to the `%n$` form of conversion specification and the <apostrophe> flag is moved from the XSI option to the Base.

Changes are made related to support for finegrained timestamps.

**fputc**

Purpose: Put a byte on a stream.

Synopsis:

```
#include <stdio.h>

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

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

Issue 7: Changes are made related to support for finegrained timestamps.

**fputs**

Purpose: Put a string on a stream.

Synopsis:

```
#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 7: Changes are made related to support for finegrained timestamps.

**fputwc**

Purpose: Put a wide-character code on a stream.

Synopsis:
```
#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 7: Changes are made related to support for finegrained timestamps.

**fputws**

Purpose: Put a wide-character string on a stream.

Synopsis:
```
#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 7: Changes are made related to support for finegrained timestamps.

**fread**

Purpose: Binary input.

Synopsis:
```
#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 7: Changes are made related to support for finegrained timestamps.

**free**

Purpose: Free allocated memory.

Synopsis:
```
#include <stdlib.h>

void free(void *ptr);
```

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

Issue 7: The DESCRIPTION is updated to clarify that if the pointer returned is not by a function that allocates memory as if by *malloc*( ), then the behavior is undefined.

**freeaddrinfo, getaddrinfo**

Purpose: Get address information.

Synopsis:
```
#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 Commands and Utilities, Issue 5 (XCU5).

Issue 7: Austin Group Interpretation 1003.1-2001 #013 is applied, removing the [EAI_OVERFLOW] error code.

Austin Group Interpretation 1003.1-2001 #146 is applied, eliminating the use of "may" in relation to the *hints* argument.

An example is added.

**freelocale**

Purpose: Free resources allocated for a locale object.

CX Synopsis:

```
#include <locale.h>

void freelocale(locale_t locobj);
```

The *freelocale*( ) function causes the resources allocated for a locale object returned by a call to the *newlocale*( ) or *duplocale*( ) functions to be released.

The following example shows a code fragment to free a locale object created by *newlocale*( ):

```
#include <locale.h>
...
/* Every locale object allocated with newlocale() should be
 * freed using freelocale():
 */
locale_t loc;
/* Get the locale. */
loc = newlocale (LC_CTYPE_MASK | LC_TIME_MASK, "locname", NULL);
/* ... Use the locale object ... */
...
/* Free the locale object resources. */
freelocale (loc);
```

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 4.

Issue 7: First released in Issue 7.

**freopen**

Purpose: Open a stream.

Synopsis:

```
#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 7: Austin Group Interpretation 1003.1-2001 #043 is applied, clarifying that the *freopen*( ) function allocates a file descriptor as per *open*( ).

Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

Austin Group Interpretation 1003.1-2001 #159 is applied, clarifying requirements for the flags set on the open file description.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XSH-ERN-150 is applied, clarifying the DESCRIPTION.

SD5-XSH-ERN-219 is applied, adding advice to the APPLICATION USAGE relating to the use of a NULL *filename* argument.

**frexp, frexpf, frexpl**

Purpose: Extract mantissa and exponent from a double precision number.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**fscanf, scanf, sscanf**

Purpose: Convert formatted input.

Synopsis:
```
#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 7: Austin Group Interpretation 1003.1-2001 #170 is applied, changing the [EILSEQ] error condition from a "may fail" to a "shall fail".

SD5-XSH-ERN-9 is applied, correcting *fscanf*( ) to *scanf*( ) in the DESCRIPTION.

SD5-XSH-ERN-132 is applied, adding the assignment-allocation character `'m'`.

Functionality relating to the `%n$` form of conversion specification is moved from the XSI option to the Base.

Changes are made related to support for finegrained timestamps.

**fseek, fseeko**

Purpose: Reposition a file-position indicator in a stream.

Synopsis:
```
#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 7: Changes are made related to support for finegrained timestamps.

**fsetpos**

Purpose: Set current file position.

Synopsis:
```
#include <stdio.h>

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

Derivation: First released in Issue 4. Derived from the IEEE Std 1003.1i-1995.

Issue 7: SD5-XSH-ERN-220 is applied, changing the first [EPIPE] to [ESPIPE].

**fstat**

Purpose: Get file status.

Synopsis:
```
#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 7: XSH-SD5-ERN-161 is applied, updating the DESCRIPTION to clarify to which file types *st_nlink* applies.

Changes are made related to support for finegrained timestamps.

**fstatat, lstat, stat**

Purpose: Get file status.

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

int fstatat(int fd, const char *restrict path,
    struct stat *restrict buf, int flag);
int lstat(const char *restrict path,
    struct stat *restrict buf);
int stat(const char *restrict path,
    struct stat *restrict buf);
```

The *fstatat*() function is equivalent to the *stat*() and *lstat*() functions, except in the case where *path* specifies a relative path. In this case the status is retrieved from a file relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the file descriptor was opened without O_SEARCH, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with O_SEARCH, the function does not perform the check.

The AT_SYMLINK_NOFOLLOW flag controls whether *fchownat*() behaves like *stat*() or *lstat*(): if AT_SYMLINK_NOFOLLOW is set and *path* names a symbolic link, the status of the symbolic link is returned.

The purpose of the *fstatat*() function is to obtain the status of files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *stat*() or *lstat*(), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *fstatat*() function it can be guaranteed that the file for which status is returned is located relative to the desired directory.

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

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

XSH-SD5-ERN-161 is applied, updating the DESCRIPTION to clarify to which file types *st_nlink* applies.

The *fstatat*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

The *lstat*() function is now required to return meaningful data for symbolic links in all **stat** structure fields, except for the permission bits of *st_mode*.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

**fstatvfs, statvfs**

Purpose: Get file system information.

Synopsis:
```
#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 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

SD5-XSH-ERN-68 is applied, correcting the EXAMPLES section.

The *fstatvfs*() and *statvfs*() functions are moved from the XSI option to the Base.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

**fsync**

Purpose: Synchronize changes to a file.

FSC Synopsis:
```
#include <unistd.h>

int fsync(int fildes);
```

Derivation: First released in Issue 3.

Issue 7: No functional changes are made in this issue.

**ftell, ftello**

Purpose: Return a file offset in a stream.

Synopsis:

```
#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 7: No functional changes are made in this issue.

**ftok**

Purpose: Generate an IPC key.

XSI Synopsis:

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

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

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

**ftruncate**

Purpose: Truncate a file to a specified length.

Synopsis:

```
#include <unistd.h>

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

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #056 is applied, revising the ERRORS section (although the [EINVAL] ''may fail'' error was subsequently removed during review of the XSI option).

Functionality relating to the Memory Protection and Memory Mapped Files options is moved to the Base.

The DESCRIPTION is updated so that a call to *ftruncate*( ) when the file is smaller than the size requested will increase the size of the file. Previously, non-XSI-conforming implementations were allowed to increase the size of the file or fail.

Changes are made related to support for finegrained timestamps.

**ftw**

Purpose: Traverse (walk) a file tree.

OB XSI Synopsis:

```
#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 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The *ftw*() function is marked obsolescent. Applications should use the *nftw*() function instead.

**futimens, utimensat, utimes**

Purpose: Set file access and modification times.

Synopsis:

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

int futimens(int fd, const struct timespec times[2]);
int utimensat(int fd, const char *path,
    const struct timespec times[2], int flag);
```

XSI

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

int utimes(const char *path, const struct timeval times[2]);
```

The *futimens*() and *utimensat*() functions set the access and modification times of a file to the values of the *times* argument. The *futimens*() function changes the times of the file associated with the file descriptor *fd*. The *utimensat*() function changes the times of the file pointed to by the *path* argument, relative to the directory associated with the file descriptor *fd*.

The *times* argument is an array of two **timespec** structures. The first array member represents the date and time of last access, and the second member represents the date and time of last modification. The times in the **timespec** structure are measured in seconds and nanoseconds since the Epoch. The file's relevant timestamp is set to the greatest value supported by the file system that is not greater than the specified time.

If the *tv_nsec* field of a **timespec** structure has the special value UTIME_NOW, the file's relevant timestamp is set to the greatest value supported by the file system that is not greater than the current time. If the *tv_nsec* field has the special value UTIME_OMIT, the file's relevant timestamp is not changed. In either case, the *tv_sec* field is ignored.

If *utimensat*() is passed a relative path in the *path* argument, the file to be used is relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the file descriptor was opened without O_SEARCH, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with O_SEARCH, the function does not perform the check.

The AT_SYMLINK_NOFOLLOW flag can be used to specify that if *path* names a symbolic link, then the access and modification times of the symbolic link are

changed.

The purpose of the *utimensat*() function is to set the access and modification time of files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *utimes*(), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *utimensat*() function it can be guaranteed that the changed file is located relative to the desired directory.

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The LEGACY marking is removed from *utimes*().

The *utimensat*() function (renamed from *futimesat*()) is added from The Open Group Technical Standard, 2006, Extended API Set Part 2, and changed to allow modifying a symbolic link by adding a *flag* argument.

The *futimens*() function is added.

Changes are made related to support for finegrained timestamps.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

**fwide**

Purpose: Set stream orientation.

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

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

Derivation: First released in Issue 5. Included for alignment with .

Issue 7: No functional changes are made in this issue.

**fwprintf, swprintf, wprintf**

Purpose: Print formatted wide-character output.

Synopsis:
```
#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 .

Issue 7: Austin Group Interpretation 1003.1-2001 #161 is applied, updating the DESCRIPTION of the 0 flag.

Austin Group Interpretation 1003.1-2001 #170 is applied, changing the [EILSEQ] error condition from a "may fail" to a "shall fail".

ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #68 (SD5-XSH-ERN-70) is

applied, revising the description of `g` and `G`.

Functionality relating to the `"%n$"` form of conversion specification and the <apostrophe> flag is moved from the XSI option to the Base.

The [EOVERFLOW] error is added for *swprintf*( ).

Changes are made related to support for finegrained timestamps.

**fwrite**

Purpose: Binary output.

Synopsis:
```
#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 7: Changes are made related to support for finegrained timestamps.

**fwscanf, swscanf, wscanf**

Purpose: Convert formatted wide-character input.

Synopsis:
```
#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 .

Issue 7: Austin Group Interpretation 1003.1-2001 #170 is applied, changing the [EILSEQ] error condition from a ‘‘may fail’’ to a ‘‘shall fail’’.

SD5-XSH-ERN-132 is applied, adding the assignment-allocation character `'m'`.

Functionality relating to the `"%n$"` form of conversion specification is moved from the XSI option to the Base.

Changes are made related to support for finegrained timestamps.

**gai_strerror**

Purpose: Address and name information error description.

Synopsis:
```
#include <netdb.h>

const char *gai_strerror(int ecode);
```

Derivation: First released in Issue 6. Derived from the Commands and Utilities, Issue 5 (XCU5).

Issue 7: No functional changes are made in this issue.

**getc**

Purpose: Get a byte from a stream.

Synopsis:
```
#include <stdio.h>

int getc(FILE *stream);
```

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

Issue 7: No functional changes are made in this issue.

**getc_unlocked, getchar_unlocked, putc_unlocked, putchar_unlocked**

Purpose: Stdio with explicit client locking.

CX Synopsis:
```
#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 7: The *getc_unlocked*(), *getchar_unlocked*(), *putc_unlocked*(), and *putchar_unlocked*() functions are moved from the Thread-Safe Functions option to the Base.

**getchar**

Purpose: Get a byte from a *stdin* stream.

Synopsis:
```
#include <stdio.h>

int getchar(void);
```

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

Issue 7: No functional changes are made in this issue.

**getcwd**

Purpose: Get the pathname of the current working directory.

Synopsis:
```
#include <unistd.h>

char *getcwd(char *buf, size_t size);
```

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

Issue 7: Austin Group Interpretation 1003.1-2001 #140 is applied, changing the text for consistency with the *pwd* utility, adding text to address the case where the current directory is deeper in the file hierarchy than {PATH_MAX} bytes, and adding the requirements relating to pathnames beginning with two <slash> characters.

**getdate**

Purpose: Convert user format date and time.

XSI Synopsis:

```
#include <time.h>

struct tm *getdate(const char *string);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: The description of the *getdate_err* value is expanded.

**getdelim, getline**

Purpose: Read a delimited record from *stream*.

CX Synopsis:

```
#include <stdio.h>

ssize_t getdelim(char **restrict lineptr, size_t *restrict n,
    int delimiter, FILE *restrict stream);
ssize_t getline(char **restrict lineptr, size_t *restrict n,
    FILE *restrict stream);
```

The *getdelim*() function reads from *stream* until it encounters a character matching the *delimiter* character.

The *getline*() function is equivalent to the *getdelim*() function with the *delimiter* character equal to the <newline> character.

These functions are widely used to solve the problem that the *fgets*() function has with long lines. The functions automatically enlarge the target buffers if needed. These are especially useful since they reduce code needed for applications.

Application writers should note that setting **lineptr* to a null pointer and **n* to zero are allowed and a recommended way to start parsing a file.

The *ferror*() or *feof*() functions should be used to distinguish between an error condition and an end-of-file condition.

Although a NUL terminator is always supplied after the line, note that *strlen*(**lineptr*) will be smaller than the return value if the line contains embedded NUL characters.

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Issue 7: First released in Issue 7.

**getegid**

Purpose: Get the effective group ID.

Synopsis:

```
#include <unistd.h>

gid_t getegid(void);
```

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

Issue 7: No functional changes are made in this issue.

**getenv**

Purpose: Get value of an environment variable.

Synopsis:
```
#include <stdlib.h>

char *getenv(const char *name);
```

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

Issue 7: Austin Group Interpretation 1003.1-2001 #062 is applied, clarifying that a call to *putenv*( ) may also cause the string to be overwritten.

Austin Group Interpretation 1003.1-2001 #148 is applied, adding the FUTURE DIRECTIONS.

**geteuid**

Purpose: Get the effective user ID.

Synopsis:
```
#include <unistd.h>

uid_t geteuid(void);
```

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

Issue 7: No functional changes are made in this issue.

**getgid**

Purpose: Get the real group ID.

Synopsis:
```
#include <unistd.h>

gid_t getgid(void);
```

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

Issue 7: No functional changes are made in this issue.

**getgrgid, getgrgid_r**

Purpose: Get group database entry for a group ID.

Synopsis:
```
#include <grp.h>

struct group *getgrgid(gid_t gid);
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 7: SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XSH-ERN-166 is applied, changing *sysconf*(*_SC_GETGR_R_SIZE_MAX*) from the maximum size to an initial value suggested for the size, and adding an example of its use to the EXAMPLES section.

The *getgrgid_r*( ) function is moved from the Thread-Safe Functions option to the Base.

A minor addition is made to the EXAMPLES section, reminding the application developer to free memory allocated as if by *malloc*( ).

**getgrnam, getgrnam_r**

Purpose: Search group database for a name.

Synopsis:
```
#include <grp.h>

struct group *getgrnam(const char *name);
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 7: Austin Group Interpretation 1003.1-2001 #081 is applied, clarifying the RETURN VALUE section.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XSH-ERN-166 is applied, changing *sysconf*(*_SC_GETGR_R_SIZE_MAX*) from the maximum size to an initial value suggested for the size, and adding an example of its use to the EXAMPLES section.

The *getgrnam_r*() function is moved from the Thread-Safe Functions option to the Base.

A minor addition is made to the EXAMPLES section, reminding the application developer to free memory allocated as if by *malloc*().

**getgroups**

Purpose: Get supplementary group IDs.

Synopsis:
```
#include <unistd.h>

int getgroups(int gidsetsize, gid_t grouplist[]);
```

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

Issue 7: No functional changes are made in this issue.

**gethostid**

Purpose: Get an identifier for the current host.

XSI Synopsis:
```
#include <unistd.h>

long gethostid(void);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

**gethostname**

Purpose: Get name of current host.

Synopsis:
```
#include <unistd.h>

int gethostname(char *name, size_t namelen);
```

Derivation: First released in Issue 6. Derived from the Commands and Utilities, Issue 5 (XCU5).

Issue 7: No functional changes are made in this issue.

**getitimer, setitimer**

Purpose: Get and set value of interval timer.

OB XSI Synopsis:
```
#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 7: The *getitimer*() and *setitimer*() functions are marked obsolescent. Applications should use the *timer_gettime*() and *timer_settime*() functions, respectively.

**getlogin, getlogin_r**

Purpose: Get login name.

Synopsis:
```
#include <unistd.h>

char *getlogin(void);
int getlogin_r(char *name, size_t namesize);
```

Derivation: First released in Issue 1. Derived from System V Release 2.0.

Issue 7: SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

The *getlogin_r*() function is moved from the Thread-Safe Functions option to the Base.

**getmsg, getpmsg**

Purpose: Receive next message from a STREAMS file (**STREAMS**).

OB XSR Synopsis:
```
#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 7: The *getmsg*( ) and *getpmsg*( ) functions are marked obsolescent.

**getnameinfo**

Purpose: Get name information.

Synopsis:
```
#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, int flags);
```

Derivation: First released in Issue 6. Derived from the Commands and Utilities, Issue 5 (XCU5).

Issue 7: SD5-XSH-ERN-127 is applied, clarifying the behavior if the address is the IPv6 unspecified address.

**getopt, optarg, opterr, optind, optopt**

Purpose: Command option parsing.

Synopsis:
```
#include <unistd.h>

int getopt(int argc, char * const argv[], const char *optstring);
extern char *optarg;
extern int opterr, optind, optopt;
```

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

Issue 7: No functional changes are made in this issue.

**getpeername**

Purpose: Get the name of the peer socket.

Synopsis:
```
#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 Commands and Utilities, Issue 5 (XCU5).

Issue 7: No functional changes are made in this issue.

**getpgid**

Purpose: Get the process group ID for a process.

Synopsis:
```
#include <unistd.h>

pid_t getpgid(pid_t pid);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: The *getpgid*( ) function is moved from the XSI option to the Base.

**getpgrp**

Purpose: Get the process group ID of the calling process.

Synopsis:
```
#include <unistd.h>

pid_t getpgrp(void);
```

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

Issue 7: No functional changes are made in this issue.

**getpid**

Purpose: Get the process ID.

Synopsis:
```
#include <unistd.h>

pid_t getpid(void);
```

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

Issue 7: No functional changes are made in this issue.

**getppid**

Purpose: Get the parent process ID.

Synopsis:
```
#include <unistd.h>

pid_t getppid(void);
```

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

Issue 7: No functional changes are made in this issue.

**getpriority, setpriority**

Purpose: Get and set the nice value.

XSI Synopsis:
```
#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 7: No functional changes are made in this issue.

**getpwnam, getpwnam_r**

Purpose: Search user database for a name.

Synopsis:
```
#include <pwd.h>

struct passwd *getpwnam(const char *name);
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 7: SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XSH-ERN-166 is applied, changing *sysconf*(*_SC_GETPW_R_SIZE_MAX*) from the maximum size to an initial value suggested for the size, and adding an example of its use to the EXAMPLES section.

The *getpwnam_r*() function is moved from the Thread-Safe Functions option to the Base.

A minor addition is made to the EXAMPLES section, reminding the application developer to free memory allocated as if by *malloc*().

**getpwuid, getpwuid_r**

Purpose: Search user database for a user ID.

Synopsis:
```
#include <pwd.h>

struct passwd *getpwuid(uid_t uid);
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 7: SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XSH-ERN-166 is applied, changing *sysconf*(*_SC_GETPW_R_SIZE_MAX*) from the maximum size to an initial value suggested for the size, and adding an example of its use to the EXAMPLES section.

The *getpwuid_r*() function is moved from the Thread-Safe Functions option to the Base.

A minor addition is made to the EXAMPLES section, reminding the application developer to free memory allocated as if by *malloc*().

**getrlimit, setrlimit**

Purpose: Control maximum resource consumption.

XSI Synopsis:
```
#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 7: No functional changes are made in this issue.

**getrusage**

Purpose: Get information about resource utilization.

XSI Synopsis:
```
#include <sys/resource.h>

int getrusage(int who, struct rusage *r_usage);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

**gets**

Purpose: Get a string from a *stdin* stream.

OB Synopsis:
```
#include <stdio.h>
char *gets(char *s);
```

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

Issue 7: Austin Group Interpretation 1003.1-2001 #051 is applied, clarifying the RETURN VALUE section.

The *gets*() function is marked obsolescent. Applications should use the *fgets*() function instead.

Changes are made related to support for finegrained timestamps.

**getsid**

Purpose: Get the process group ID of a session leader.

Synopsis:
```
#include <unistd.h>
pid_t getsid(pid_t pid);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: The *getsid*() function is moved from the XSI option to the Base.

**getsockname**

Purpose: Get the socket name.

Synopsis:
```
#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 Commands and Utilities, Issue 5 (XCU5).

Issue 7: No functional changes are made in this issue.

**getsockopt**

Purpose: Get the socket options.

Synopsis:
```
#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 Commands and Utilities, Issue 5 (XCU5).

Issue 7: Austin Group Interpretation 1003.1-2001 #158 is applied, removing text relating to socket options that is now in XSH Section 2.10.16 .

**getsubopt**

Purpose: Parse suboption arguments from a string.

Synopsis:
```
#include <stdlib.h>

int getsubopt(char **optionp, char * const *keylistp,
    char **valuep);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: The *getsubopt*( ) function is moved from the XSI option to the Base.

**gettimeofday**

Purpose: Get the date and time.

OB XSI Synopsis:
```
#include <sys/time.h>

int gettimeofday(struct timeval *restrict tp,
    void *restrict tzp);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: The *gettimeofday*( ) function is marked obsolescent. Applications should use the *clock_gettime*( ) function instead.

**getuid**

Purpose: Get a real user ID.

Synopsis:
```
#include <unistd.h>

uid_t getuid(void);
```

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

Issue 7: No functional changes are made in this issue.

**getwc**

Purpose: Get a wide character from a stream.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**getwchar**

Purpose: Get a wide character from a *stdin* stream.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**glob, globfree**

Purpose: Generate pathnames matching a pattern.

Synopsis:

```
#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 the .

Issue 7: No functional changes are made in this issue.

**gmtime, gmtime_r**

Purpose: Convert a time value to a broken-down UTC time.

Synopsis:

```
#include <time.h>

struct tm *gmtime(const time_t *timer);
```

CX

```
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 7: The *gmtime_r*() function is moved from the Thread-Safe Functions option to the Base.

**grantpt**

Purpose: Grant access to the slave pseudo-terminal device.

XSI Synopsis:

```
#include <stdlib.h>

int grantpt(int fildes);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

**hcreate, hdestroy, hsearch**

Purpose: Manage hash search table.

XSI Synopsis:

```
#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 7: No functional changes are made in this issue.

**htonl, htons, ntohl, ntohs**

Purpose: Convert values between host and network byte order.

Synopsis:
```
#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 Commands and Utilities, Issue 5 (XCU5).

Issue 7: No functional changes are made in this issue.

**hypot, hypotf, hypotl**

Purpose: Euclidean distance function.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**iconv**

Purpose: Codeset conversion function.

Synopsis:
```
#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 7: The *iconv*( ) function is moved from the XSI option to the Base.

**iconv_close**

Purpose: Codeset conversion deallocation function.

Synopsis:
```
#include <iconv.h>

int iconv_close(iconv_t cd);
```

Derivation: First released in Issue 4. Derived from the HP-UX Manual.

Issue 7: The *iconv_close*( ) function is moved from the XSI option to the Base.

**iconv_open**

Purpose: Codeset conversion allocation function.

Synopsis:
```
#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 7: SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

The *iconv_open*( ) function is moved from the XSI option to the Base.

**if_freenameindex**

Purpose: Free memory allocated by if_nameindex.

Synopsis:
```
#include <net/if.h>

void if_freenameindex(struct if_nameindex *ptr);
```

Derivation: First released in Issue 6. Derived from the Commands and Utilities, Issue 5 (XCU5).

Issue 7: No functional changes are made in this issue.

**if_indextoname**

Purpose: Map a network interface index to its corresponding name.

Synopsis:
```
#include <net/if.h>

char *if_indextoname(unsigned ifindex, char *ifname);
```

Derivation: First released in Issue 6. Derived from the Commands and Utilities, Issue 5 (XCU5).

Issue 7: No functional changes are made in this issue.

**if_nameindex**

Purpose: Return all network interface names and indexes.

Synopsis:
```
#include <net/if.h>

struct if_nameindex *if_nameindex(void);
```

Derivation: First released in Issue 6. Derived from the Commands and Utilities, Issue 5 (XCU5).

Issue 7: No functional changes are made in this issue.

**if_nametoindex**

Purpose: Map a network interface name to its corresponding index.

Synopsis:
```
#include <net/if.h>

unsigned if_nametoindex(const char *ifname);
```

Derivation: First released in Issue 6. Derived from the Commands and Utilities, Issue 5 (XCU5).

Issue 7: No functional changes are made in this issue.

**ilogb, ilogbf, ilogbl**

Purpose: Return an unbiased exponent.

Synopsis:
```
#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 7: ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #48 (SD5-XSH-ERN-71), #49, and #79 (SD5-XSH-ERN-72) are applied.

**imaxabs**

Purpose: Return absolute value.

Synopsis:
```
#include <inttypes.h>

intmax_t imaxabs(intmax_t j);
```

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

Issue 7: No functional changes are made in this issue.

**imaxdiv**

Purpose: Return quotient and remainder.

Synopsis:
```
#include <inttypes.h>

imaxdiv_t imaxdiv(intmax_t numer, intmax_t denom);
```

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

Issue 7: No functional changes are made in this issue.

**inet_addr, inet_ntoa**

Purpose: IPv4 address manipulation.

Synopsis:
```
#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 Commands and Utilities, Issue 5 (XCU5).

Issue 7: No functional changes are made in this issue.

**inet_ntop, inet_pton**

Purpose: Convert IPv4 and IPv6 addresses between binary and text form.

Synopsis:
```
#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 Commands and Utilities, Issue 5 (XCU5).

Issue 7: No functional changes are made in this issue.

**initstate, random, setstate, srandom**

Purpose: Pseudo-random number functions.

XSI Synopsis:
```
#include <stdlib.h>

char *initstate(unsigned seed, char *state, size_t size);
long random(void);
char *setstate(char *state);
void srandom(unsigned seed);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: The type of the first argument to *setstate*( ) is changed from **const char** * to **char** *.

**insque, remque**

Purpose: Insert or remove an element in a queue.

XSI Synopsis:
```
#include <search.h>

void insque(void *element, void *pred);
void remque(void *element);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

**ioctl**

Purpose: Control a STREAMS device (**STREAMS**).

OB XSR Synopsis:
```
#include <stropts.h>

int ioctl(int fildes, int request, ... /* arg */);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #155 is applied, adding a ''may fail'' [EINVAL] error condition for the I_SENDFD command.

SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

The *ioctl*( ) function is marked obsolescent.

**isalnum, isalnum_l**

Purpose: Test for an alphanumeric character.

Synopsis:
```
#include <ctype.h>

int isalnum(int c);
CX int isalnum_l(int c, locale_t locale);
```

The *isalnum_l*( ) function tests whether *c* is a character of class **alpha** or **digit** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*( ) or *duplocale*( ).

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

Issue 7: The *isalnum_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**isalpha, isalpha_l**

Purpose: Test for an alphabetic character.

Synopsis:
```
#include <ctype.h>

int isalpha(int c);
CX int isalpha_l(int c, locale_t locale);
```

The *isalpha_l*( ) function tests whether *c* is a character of class **alpha** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*( ) or *duplocale*( ).

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

Issue 7: The *isalpha_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**isascii**

Purpose: Test for a 7-bit US-ASCII character.

OB XSI Synopsis:
```
#include <ctype.h>

int isascii(int c);
```

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

Issue 7: The *isascii*( ) function is marked obsolescent.

**isastream**

Purpose: Test a file descriptor (**STREAMS**).

OB XSR Synopsis:
```
#include <stropts.h>

int isastream(int fildes);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: The *isastream*() function is marked obsolescent.

**isatty**

Purpose: Test for a terminal device.

Synopsis:
```
#include <unistd.h>

int isatty(int fildes);
```

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

Issue 7: SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

**isblank, isblank_l**

Purpose: Test for a blank character.

Synopsis:
```
#include <ctype.h>

int isblank(int c);
CX int isblank_l(int c, locale_t locale);
```

The *isblank_l*() function tests whether *c* is a character of class **blank** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*() or *duplocale*().

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

Issue 7: The *isblank_l*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**iscntrl, iscntrl_l**

Purpose: Test for a control character.

Synopsis:
```
#include <ctype.h>

int iscntrl(int c);
CX int iscntrl_l(int c, locale_t locale);
```

The *iscntrl_l*() function tests whether *c* is a character of class **cntrl** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*() or *duplocale*().

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

Issue 7: The *iscntrl_l*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**isdigit, isdigit_l**

Purpose: Test for a decimal digit.

Synopsis:
```
#include <ctype.h>

int isdigit(int c);
CX int isdigit_l(int c, locale_t locale);
```

The *isdigit_l*() function tests whether *c* is a character of class **digit** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*() or *duplocale*().

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

Issue 7: The *isdigit_l*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**isfinite**

Purpose: Test for finite value.

Synopsis:
```
#include <math.h>

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

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

Issue 7: No functional changes are made in this issue.

**isgraph, isgraph_l**

Purpose: Test for a visible character.

Synopsis:
```
#include <ctype.h>

int isgraph(int c);
CX int isgraph_l(int c, locale_t locale);
```

The *isgraph_l*() function tests whether *c* is a character of class **graph** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*() or *duplocale*().

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

Issue 7: The *isgraph_l*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**isgreater**

Purpose: Test if x greater than y.

Synopsis:
```
#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.

Issue 7: No functional changes are made in this issue.

**isgreaterequal**

Purpose: Test if *x* is greater than or equal to *y*.

Synopsis:
```
#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.

Issue 7: No functional changes are made in this issue.

**isinf**

Purpose: Test for infinity.

Synopsis:
```
#include <math.h>
int isinf(real-floating x);
```

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

Issue 7: No functional changes are made in this issue.

**isless**

Purpose: Test if *x* is less than *y*.

Synopsis:
```
#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.

Issue 7: No functional changes are made in this issue.

**islessequal**

Purpose: Test if *x* is less than or equal to *y*.

Synopsis:
```
#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.

Issue 7: No functional changes are made in this issue.

**islessgreater**

Purpose: Test if *x* is less than or greater than *y*.

Synopsis:
```
#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.

Issue 7: No functional changes are made in this issue.

**islower, islower_l**

Purpose: Test for a lowercase letter.

Synopsis:
```
#include <ctype.h>

int islower(int c);
CX int islower_l(int c, locale_t locale);
```

The *islower_l*() function tests whether *c* is a character of class **lower** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*() or *duplocale*().

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

Issue 7: The *islower_l*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**isnan**

Purpose: Test for a NaN.

Synopsis:
```
#include <math.h>

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

Derivation: First released in Issue 3.

Issue 7: No functional changes are made in this issue.

**isnormal**

Purpose: Test for a normal value.

Synopsis:
```
#include <math.h>

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

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

Issue 7: No functional changes are made in this issue.

**isprint, isprint_l**

Purpose: Test for a printable character.

Synopsis:
```
#include <ctype.h>

int isprint(int c);
CX int isprint_l(int c, locale_t locale);
```

The *isprint_l*() function tests whether *c* is a character of class **print** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*() or *duplocale*().

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

Issue 7: The *isprint_l*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**ispunct, ispunct_l**

Purpose: Test for a punctuation character.

Synopsis:
```
#include <ctype.h>

int ispunct(int c);
CX int ispunct_l(int c, locale_t locale);
```

The *ispunct_l*() function tests whether *c* is a character of class **punct** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*() or *duplocale*().

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

Issue 7: The *ispunct_l*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**isspace, isspace_l**

Purpose: Test for a white-space character.

Synopsis:
```
#include <ctype.h>

int isspace(int c);
CX int isspace_l(int c, locale_t locale);
```

The *isspace_l*() function tests whether *c* is a character of class **space** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*() or *duplocale*().

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

Issue 7: The *isspace_l*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**isunordered**

Purpose: Test if arguments are unordered.

Synopsis:
```
#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.

Issue 7: No functional changes are made in this issue.

**isupper, isupper_l**

Purpose: Test for an uppercase letter.

Synopsis:
```
#include <ctype.h>

int isupper(int c);
CX int isupper_l(int c, locale_t locale);
```

The *isupper_l*() function tests whether *c* is a character of class **upper** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*() or *duplocale*().

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

Issue 7: The *isupper_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**iswalnum, iswalnum_l**

Purpose: Test for an alphanumeric wide-character code.

Synopsis:

```
#include <wctype.h>

int iswalnum(wint_t wc);
CX int iswalnum_l(wint_t wc, locale_t locale);
```

The *iswalnum_l*( ) function tests whether *wc* is a wide-character code representing a character of class **alpha** or **digit** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*( ) or *duplocale*( ).

Derivation: First released as a World-wide Portability Interface in Issue 4.

Issue 7: The *iswalnum_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**iswalpha, iswalpha_l**

Purpose: Test for an alphabetic wide-character code.

Synopsis:

```
#include <wctype.h>

int iswalpha(wint_t wc);
CX int iswalpha_l(wint_t wc, locale_t locale);
```

The *iswalpha_l*( ) function tests whether *wc* is a wide-character code representing a character of class **alpha** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*( ) or *duplocale*( ).

Derivation: First released in Issue 4.

Issue 7: The *iswalpha_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**iswblank, iswblank_l**

Purpose: Test for a blank wide-character code.

Synopsis:

```
#include <wctype.h>

int iswblank(wint_t wc);
CX int iswblank_l(wint_t wc, locale_t locale);
```

The *iswblank_l*( ) function tests whether *wc* is a wide-character code representing a character of class **blank** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*( ) or *duplocale*( ).

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

Issue 7: The *iswblank_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**iswcntrl, iswcntrl_l**

Purpose: Test for a control wide-character code.

Synopsis:
```
#include <wctype.h>

int iswcntrl(wint_t wc);
CX int iswcntrl_l(wint_t wc, locale_t locale);
```

The *iswcntrl_l*( ) function tests whether *wc* is a wide-character code representing a character of class **cntrl** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*( ) or *duplocale*( ).

Derivation: First released in Issue 4.

Issue 7: The *iswcntrl_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**iswctype, iswctype_l**

Purpose: Test character for a specified class.

Synopsis:
```
#include <wctype.h>

int iswctype(wint_t wc, wctype_t charclass);
CX int iswctype_l(wint_t wc, wctype_t charclass,
    locale_t locale);
```

The *iswctype_l*( ) function determines whether the wide-character code *wc* has the character class *charclass* in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*( ) or *duplocale*( ).

Derivation: First released as World-wide Portability Interfaces in Issue 4.

Issue 7: The *iswctype_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**iswdigit, iswdigit_l**

Purpose: Test for a decimal digit wide-character code.

Synopsis:
```
#include <wctype.h>

int iswdigit(wint_t wc);
CX int iswdigit_l(wint_t wc, locale_t locale);
```

The *iswdigit_l*( ) function tests whether *wc* is a wide-character code representing a character of class **digit** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*( ) or *duplocale*( ).

Derivation: First released in Issue 4.

Issue 7: The *iswdigit_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**iswgraph, iswgraph_l**

Purpose: Test for a visible wide-character code.

Synopsis:
```
#include <wctype.h>

int iswgraph(wint_t wc);
CX int iswgraph_l(wint_t wc, locale_t locale);
```

The *iswgraph_l*( ) function tests whether *wc* is a wide-character code representing a character of class **graph** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*( ) or *duplocale*( ).

Derivation: First released in Issue 4.

Issue 7: The *iswgraph_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**iswlower, iswlower_l**

Purpose: Test for a lowercase letter wide-character code.

Synopsis:
```
#include <wctype.h>

int iswlower(wint_t wc);
CX int iswlower_l(wint_t wc, locale_t locale);
```

The *iswlower_l*( ) function tests whether *wc* is a wide-character code representing a character of class **lower** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*( ) or *duplocale*( ).

Derivation: First released in Issue 4.

Issue 7: The *iswlower_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**iswprint, iswprint_l**

Purpose: Test for a printable wide-character code.

Synopsis:
```
#include <wctype.h>

int iswprint(wint_t wc);
CX int iswprint_l(wint_t wc, locale_t locale);
```

The *iswprint_l*( ) function tests whether *wc* is a wide-character code representing a character of class **print** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*( ) or *duplocale*( ).

Derivation: First released in Issue 4.

Issue 7: The *iswprint_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**iswpunct, iswpunct_l**

Purpose: Test for a punctuation wide-character code.

Synopsis:
```
#include <wctype.h>

int iswpunct(wint_t wc);
CX int iswpunct_l(wint_t wc, locale_t locale);
```

The *iswpunct_l*( ) function tests whether *wc* is a wide-character code representing a character of class **punct** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*( ) or *duplocale*( ).

Derivation: First released in Issue 4.

Issue 7: The *iswpunct_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**iswspace, iswspace_l**

Purpose: Test for a white-space wide-character code.

Synopsis:
```
#include <wctype.h>

int iswspace(wint_t wc);
CX int iswspace_l(wint_t wc, locale_t locale);
```

The *iswspace_l*( ) function tests whether *wc* is a wide-character code representing a character of class **space** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*( ) or *duplocale*( ).

Derivation: First released in Issue 4.

Issue 7: The *iswspace_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**iswupper, iswupper_l**

Purpose: Test for an uppercase letter wide-character code.

Synopsis:
```
#include <wctype.h>

int iswupper(wint_t wc);
CX int iswupper_l(wint_t wc, locale_t locale);
```

The *iswupper_l*( ) function tests whether *wc* is a wide-character code representing a character of class **upper** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*( ) or *duplocale*( ).

Derivation: First released in Issue 4.

Issue 7: The *iswupper_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**iswxdigit, iswxdigit_l**

Purpose: Test for a hexadecimal digit wide-character code.

Synopsis:
```
#include <wctype.h>

int iswxdigit(wint_t wc);
CX int iswxdigit_l(wint_t wc, locale_t locale);
```

The *iswxdigit_l*( ) function tests whether *wc* is a wide-character code representing a character of class **xdigit** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*( ) or *duplocale*( ).

Derivation: First released in Issue 4.

Issue 7: The *iswxdigit_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**isxdigit, isxdigit_l**

Purpose: Test for a hexadecimal digit.

Synopsis:
```
#include <ctype.h>

int isxdigit(int c);
CX int isxdigit_l(int c, locale_t locale);
```

The *isxdigit_l*( ) function tests whether *c* is a character of class **xdigit** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*( ) or *duplocale*( ).

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

Issue 7: The *isxdigit_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**j0, j1, jn**

Purpose: Bessel functions of the first kind.

XSI Synopsis:
```
#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 7: No functional changes are made in this issue.

**kill**

Purpose: Send a signal to a process or a group of processes.

CX Synopsis:

```
#include <signal.h>
int kill(pid_t pid, int sig);
```

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

Issue 7: No functional changes are made in this issue.

**killpg**

Purpose: Send a signal to a process group.

XSI Synopsis:

```
#include <signal.h>
int killpg(pid_t pgrp, int sig);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

**labs, llabs**

Purpose: Return a long integer absolute value.

Synopsis:

```
#include <stdlib.h>
long labs(long i);
long long llabs(long long i);
```

Derivation: First released in Issue 4. Derived from the IEEE Std 1003.1i-1995.

Issue 7: No functional changes are made in this issue.

**lchown**

Purpose: Change the owner and group of a symbolic link.

Synopsis:

```
#include <unistd.h>
int lchown(const char *path, uid_t owner, gid_t group);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The *lchown*( ) function is moved from the XSI option to the Base.

The [EOPNOTSUPP] error is removed.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

**ldexp, ldexpf, ldexpl**

Purpose: Load exponent of a floating-point number.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**ldiv, lldiv**

Purpose: Compute quotient and remainder of a long division.

Synopsis:
```
#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 IEEE Std 1003.1i-1995.

Issue 7: No functional changes are made in this issue.

**lgamma, lgammaf, lgammal, signgam**

Purpose: Log gamma function.

Synopsis:
```
#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 7: The DESCRIPTION is clarified regarding the value of *signgam* when *x* is Nan, –Inf, or a negative integer.

**link, linkat**

Purpose: Link one file to another file relative to two directory file descriptors.

Synopsis:
```
#include <unistd.h>

int link(const char *path1, const char *path2);
int linkat(int fd1, const char *path1, int fd2,
    const char *path2, int flag);
```

The *linkat*() function is equivalent to the *link*() function except in the case where either *path1* or *path2* or both are relative paths. In this case a relative path *path1* is interpreted relative to the directory associated with the file descriptor *fd1* instead of the current working directory and similarly for *path2* and the file descriptor *fd2*. If the file descriptor was opened without O_SEARCH, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with O_SEARCH, the function does not perform the check.

The AT_SYMLINK_FOLLOW flag can be used to specify that if *path1* names a symbolic link, a new link for the target of the symbolic link is created. By default a new link for the symbolic link itself is created.

The purpose of the *linkat*() function is to link files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *link*(), resulting in unspecified behavior. By opening a file descriptor for the directory of both the existing file and the target location and using the *linkat*() function it can be guaranteed that the both filenames are in the desired directories.

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

Issue 7: If *path1* names a symbolic link, the *link*() function is no longer required to follow the link: it is implementation-defined whether *link*() follows the link, or creates a new link to the symbolic link itself. Applications which need control over whether the link is followed can use the new *linkat*() function, setting the *flag* argument appropriately.

Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The *linkat*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Functionality relating to XSI STREAMS is marked obsolescent.

Changes are made related to support for finegrained timestamps.

**lio_listio**

Purpose: List directed I/O.

Synopsis:
```
#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.

Issue 7: The *lio_listio*() function is moved from the Asynchronous Input and Output option to the Base.

**listen**

Purpose: Listen for socket connections and limit the queue of incoming connections.

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

int listen(int socket, int backlog);
```

Derivation: First released in Issue 6. Derived from the Commands and Utilities, Issue 5 (XCU5).

Issue 7: No functional changes are made in this issue.

**llrint, llrintf, llrintl**

Purpose: Round to the nearest integer value using current rounding direction.

Synopsis:
```
#include <math.h>

long long llrint(double x);
long long llrintf(float x);
long long llrintl(long double x);
```

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

Issue 7: ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #53 is applied.

**llround, llroundf, llroundl**

Purpose: Round to nearest integer value.

Synopsis:
```
#include <math.h>

long long llround(double x);
long long llroundf(float x);
long long llroundl(long double x);
```

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

Issue 7: ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #54 (SD5-XSH-ERN-75) is applied.

**localeconv**

Purpose: Return locale-specific information.

Synopsis:
```
#include <locale.h>

struct lconv *localeconv(void);
```

Derivation: First released in Issue 4. Derived from the IEEE Std 1003.1b-1993.

Issue 7: The definitions of **int_curr_symbol** and **currency_symbol** are updated.

The examples in the APPLICATION USAGE section are updated.

**localtime, localtime_r**

Purpose: Convert a time value to a broken-down local time.

Synopsis:
```
#include <time.h>

struct tm *localtime(const time_t *timer);
```
CX
```
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 7: The *localtime_r*() function is moved from the Thread-Safe Functions option to the Base.

Changes are made to the EXAMPLES section related to support for finegrained timestamps.

**lockf**

Purpose: Record locking on files.

XSI Synopsis:

```
#include <unistd.h>

int lockf(int fildes, int function, off_t size);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #054 is applied, updating the DESCRIPTION to change "other threads" to "threads in other processes".

**log, logf, logl**

Purpose: Natural logarithm function.

Synopsis:

```
#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 7: No functional changes are made in this issue.

**log10, log10f, log10l**

Purpose: Base 10 logarithm function.

Synopsis:

```
#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 7: No functional changes are made in this issue.

**log1p, log1pf, log1pl**

Purpose: Compute a natural logarithm.

Synopsis:

```
#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 7: No functional changes are made in this issue.

**log2, log2f, log2l**

Purpose: Compute base 2 logarithm functions.

Synopsis:
```
#include <math.h>

double log2(double x);
float log2f(float x);
long double log2l(long double x);
```

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

Issue 7: No functional changes are made in this issue.

**logb, logbf, logbl**

Purpose: Radix-independent exponent.

Synopsis:
```
#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 7: ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #50 (SD5-XSH-ERN-76) is applied.

**longjmp**

Purpose: Non-local goto.

Synopsis:
```
#include <setjmp.h>

void longjmp(jmp_buf env, int val);
```

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

Issue 7: No functional changes are made in this issue.

**lrint, lrintf, lrintl**

Purpose: Round to nearest integer value using current rounding direction.

Synopsis:
```
#include <math.h>

long lrint(double x);
long lrintf(float x);
long lrintl(long double x);
```

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

Issue 7: ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #53 (SD5-XSH-ERN-77) is applied.

**lround, lroundf, lroundl**

Purpose: Round to nearest integer value.

Synopsis:
```
#include <math.h>

long lround(double x);
long lroundf(float x);
long lroundl(long double x);
```

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

Issue 7: ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #54 (SD5-XSH-ERN-78) is applied.

**lsearch, lfind**

Purpose: Linear search and update.

XSI Synopsis:
```
#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 7: No functional changes are made in this issue.

**lseek**

Purpose: Move the read/write file offset.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**malloc**

Purpose: A memory allocator.

Synopsis:
```
#include <stdlib.h>

void *malloc(size_t size);
```

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

Issue 7: No functional changes are made in this issue.

**mblen**

Purpose: Get number of bytes in a character.

Synopsis:
```
#include <stdlib.h>

int mblen(const char *s, size_t n);
```

Derivation: First released in Issue 4. Aligned with the IEEE Std 1003.1i-1995.

Issue 7: No functional changes are made in this issue.

**mbrlen**

Purpose: Get number of bytes in a character (restartable).

Synopsis:
```
#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 .

Issue 7: Austin Group Interpretation 1003.1-2001 #170 is applied, changing the [EILSEQ] error condition from a "may fail" to a "shall fail".

**mbrtowc**

Purpose: Convert a character to a wide-character code (restartable).

Synopsis:
```
#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 .

Issue 7: Austin Group Interpretation 1003.1-2001 #170 is applied, changing the [EILSEQ] error condition from a "may fail" to a "shall fail".

**mbsinit**

Purpose: Determine conversion object status.

Synopsis:
```
#include <wchar.h>

int mbsinit(const mbstate_t *ps);
```

Derivation: First released in Issue 5. Included for alignment with .

Issue 7: No functional changes are made in this issue.

**mbsnrtowcs, mbsrtowcs**

Purpose: Convert a character string to a wide-character string (restartable).

Synopsis:
```
#include <wchar.h>

CX  size_t mbsnrtowcs(wchar_t *restrict dst,
        const char **restrict src, size_t nmc,
        size_t len, mbstate_t *restrict ps);
    size_t mbsrtowcs(wchar_t *restrict dst,
        const char **restrict src, size_t len,
        mbstate_t *restrict ps);
```

The *mbsnrtowcs*() function is equivalent to the *mbsrtowcs*() function, except that the conversion of characters pointed to by *src* is limited to at most *nmc* bytes (the size of the input buffer).

Derivation: First released in Issue 5. Included for alignment with .

Issue 7: Austin Group Interpretation 1003.1-2001 #170 is applied, changing the [EILSEQ] error condition from a ''may fail'' to a ''shall fail''.

The *mbsnrtowcs*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

**mbstowcs**

Purpose: Convert a character string to a wide-character string.

Synopsis:
```
#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 IEEE Std 1003.1i-1995.

Issue 7: Austin Group Interpretation 1003.1-2001 #170 is applied, changing the [EILSEQ] error condition from a ''may fail'' to a ''shall fail''.

**mbtowc**

Purpose: Convert a character to a wide-character code.

Synopsis:
```
#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 IEEE Std 1003.1i-1995.

Issue 7: Austin Group Interpretation 1003.1-2001 #170 is applied, changing the [EILSEQ] error condition from a ''may fail'' to a ''shall fail''.

**memccpy**

Purpose: Copy bytes in memory.

XSI Synopsis:
```
#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 7: No functional changes are made in this issue.

**memchr**

Purpose: Find byte in memory.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**memcmp**

Purpose: Compare bytes in memory.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**memcpy**

Purpose: Copy bytes in memory.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**memmove**

Purpose: Copy bytes in memory with overlapping areas.

Synopsis:
```
#include <string.h>
void *memmove(void *s1, const void *s2, size_t n);
```

Derivation: First released in Issue 4. Derived from the IEEE Std 1003.1b-1993.

Issue 7: No functional changes are made in this issue.

**memset**

Purpose: Set bytes in memory.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**mkdir, mkdirat**

Purpose: Make a directory relative to directory file descriptor.

Synopsis:

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

int mkdir(const char *path, mode_t mode);
int mkdirat(int fd, const char *path, mode_t mode);
```

The *mkdirat*() function is equivalent to the *mkdir*() function except in the case where *path* specifies a relative path. In this case the newly created directory is created relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the file descriptor was opened without O_SEARCH, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with O_SEARCH, the function does not perform the check.

The purpose of the *mkdirat*() function is to create a directory in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to the call to *mkdir*(), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *mkdirat*() function it can be guaranteed that the newly created directory is located relative to the desired directory.

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

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The *mkdirat*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

**mkdtemp, mkstemp**

Purpose: Create a unique directory or file.

CX Synopsis:

```
#include <stdlib.h>

char *mkdtemp(char *template);
int mkstemp(char *template);
```

The *mkdtemp*() function uses the contents of *template* to construct a unique directory name. The string provided in *template* is a filename ending with six trailing 'X's. The *mkdtemp*() function replaces each 'X' with a character from the portable filename character set. The characters are chosen such that the resulting name does not duplicate the name of an existing file at the time of a call to *mkdtemp*(). The unique directory name is used to attempt to create the directory using mode 0700 as modified by the file creation mask.

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

SD5-XSH-ERN-168 is applied, clarifying file permissions upon creation.

The *mkstemp*() function is moved from the XSI option to the Base.

The *mkdtemp*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

**mkfifo, mkfifoat**

Purpose: Make a FIFO special file relative to directory file descriptor.

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

int mkfifo(const char *path, mode_t mode);
int mkfifoat(int fd, const char *path, mode_t mode);
```

The *mkfifoat*() function is equivalent to the *mkfifo*() function except in the case where *path* specifies a relative path. In this case the newly created FIFO is created relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the file descriptor was opened without O_SEARCH, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with O_SEARCH, the function does not perform the check.

The purpose of the *mkfifoat*() function is to create a FIFO special file in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *mkfifo*(), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *mkfifoat*() function it can be guaranteed that the newly created FIFO is located relative to the desired directory.

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

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The *mkfifoat*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

**mknod, mknodat**

Purpose: Make directory, special file, or regular file.

XSI Synopsis:
```
#include <sys/stat.h>

int mknod(const char *path, mode_t mode, dev_t dev);
int mknodat(int fd, const char *path, mode_t mode, dev_t dev);
```

The *mknodat*() function is equivalent to the *mknod*() function except in the case where *path* specifies a relative path. In this case the newly created directory, special file, or regular file is located relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the file descriptor was opened without O_SEARCH, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with O_SEARCH, the function does not perform the check.

The purpose of the *mknodat*() function is to create directories, special files, or regular files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in

parallel to a call to *mknod*(), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *mknodat*() function it can be guaranteed that the newly created directory, special file, or regular file is located relative to the desired directory.

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The *mknodat*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

**mktime**

Purpose: Convert broken-down time into time since the Epoch.

Synopsis:
```
#include <time.h>

time_t mktime(struct tm *timeptr);
```

Derivation: First released in Issue 3. Included for alignment with the IEEE Std 1003.1-1988 (POSIX.1) and the IEEE Std 1003.1b-1993.

Issue 7: No functional changes are made in this issue.

**mlock, munlock**

Purpose: Lock or unlock a range of process address space (**REALTIME**).

MLR Synopsis:
```
#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.

Issue 7: No functional changes are made in this issue.

**mlockall, munlockall**

Purpose: Lock/unlock the address space of a process (**REALTIME**).

ML Synopsis:
```
#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.

Issue 7: No functional changes are made in this issue.

**mmap**

Purpose: Map pages of memory.

Synopsis:
```
#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 7: Austin Group Interpretations 1003.1-2001 #078 and #079 are applied, clarifying page alignment requirements and adding a note about the state of synchronization objects becoming undefined when a shared region is unmapped.

Functionality relating to the Memory Protection and Memory Mapped Files options is moved to the Base.

Changes are made related to support for finegrained timestamps.

**modf, modff, modfl**

Purpose: Decompose a floating-point number.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**mprotect**

Purpose: Set protection of memory mapping.

Synopsis:
```
#include <sys/mman.h>

int mprotect(void *addr, size_t len, int prot);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #078 is applied, clarifying page alignment requirements.

The *mprotect*( ) function is moved from the Memory Protection option to the Base.

**mq_close**

Purpose: Close a message queue (**REALTIME**).

MSG Synopsis:
```
#include <mqueue.h>

int mq_close(mqd_t mqdes);
```

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

Issue 7: No functional changes are made in this issue.

**mq_getattr**

Purpose: Get message queue attributes (**REALTIME**).

MSG Synopsis:

```
#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.

Issue 7: No functional changes are made in this issue.

**mq_notify**

Purpose: Notify process that a message is available (**REALTIME**).

MSG Synopsis:

```
#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.

Issue 7: SD5-XSH-ERN-38 is applied, adding the *mq_timedreceive*() function to the DESCRIPTION.

Austin Group Interpretation 1003.1-2001 #032 is applied, adding the [EINVAL] error.

An example is added.

**mq_open**

Purpose: Open a message queue (**REALTIME**).

MSG Synopsis:

```
#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.

Issue 7: Austin Group Interpretation 1003.1-2001 #077 is applied, clarifying the *name* argument and changing [ENAMETOOLONG] from a "shall fail" to a "may fail" error.

Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

SD5-XSH-ERN-170 is applied, updating the DESCRIPTION to clarify the wording for setting the user ID and group ID of the message queue.

**mq_receive, mq_timedreceive**

Purpose: Receive a message from a message queue (**REALTIME**).

MSG Synopsis:

```
#include <mqueue.h>

ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len,
    unsigned *msg_prio);

#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 abstime);
```

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

Issue 7: The *mq_timedreceive*( ) function is moved from the Timeouts option to the Base.

Functionality relating to the Timers option is moved to the Base.

**mq_send, mq_timedsend**

Purpose: Send a message to a message queue (**REALTIME**).

MSG Synopsis:

```
#include <mqueue.h>

int mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
    unsigned msg_prio);

#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 *abstime);
```

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

Issue 7: The *mq_timedsend*( ) function is moved from the Timeouts option to the Base.

Functionality relating to the Timers option is moved to the Base.

**mq_setattr**

Purpose: Set message queue attributes (**REALTIME**).

MSG Synopsis:

```
#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.

Issue 7: No functional changes are made in this issue.

**mq_unlink**

Purpose: Remove a message queue (**REALTIME**).

MSG Synopsis:

```
#include <mqueue.h>

int mq_unlink(const char *name);
```

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

Issue 7: Austin Group Interpretation 1003.1-2001 #077 is applied, changing [ENAMETOOLONG] from a "shall fail" to a "may fail" error .

Austin Group Interpretation 1003.1-2001 #141 is applied, requiring that after a successful call to *mq_unlink*(), reuse of the name shall subsequently cause *mq_open*() to behave as if no message queue of that name exists (that is, *mq_open*() will fail if O_CREAT is not set, or will create a new message queue if O_CREAT is set). Previously, attempts to recreate the message queue were allowed to fail.

**msgctl**

Purpose: XSI message control operations.

XSI Synopsis:

```
#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 7: No functional changes are made in this issue.

**msgget**

Purpose: Get the XSI message queue identifier.

XSI Synopsis:

```
#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 7: No functional changes are made in this issue.

**msgrcv**

Purpose: XSI message receive operation.

XSI Synopsis:

```
#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 7: No functional changes are made in this issue.

**msgsnd**

Purpose: XSI message send operation.

XSI Synopsis:

```
#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 7: No functional changes are made in this issue.

**msync**

Purpose: Synchronize memory with physical storage.

XSI | SIO Synopsis:

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

int msync(void *addr, size_t len, int flags);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #078 is applied, clarifying page alignment requirements.

The *msync*() function is marked as part of the Synchronized Input and Output option or XSI option as the Memory Mapped Files is moved to the Base.

Changes are made related to support for finegrained timestamps.

**munmap**

Purpose: Unmap pages of memory.

Synopsis:

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

int munmap(void *addr, size_t len);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #078 is applied, clarifying page alignment requirements.

The *munmap*() function is moved from the Memory Mapped Files option to the Base.

**nan, nanf, nanl**

Purpose: Return quiet NaN.

Synopsis:

```
#include <math.h>

double nan(const char *tagp);
float nanf(const char *tagp);
long double nanl(const char *tagp);
```

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

Issue 7: No functional changes are made in this issue.

**nanosleep**

Purpose: High resolution sleep.

CX Synopsis:

```
#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.

Issue 7: SD5-XBD-ERN-33 is applied, clarifying that the *rqtp* and *rmtp* arguments may point to the same object.

The *nanosleep*( ) function is moved from the Timers option to the Base.

**nearbyint, nearbyintf, nearbyintl**

Purpose: Floating-point rounding functions.

Synopsis:

```
#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.

Issue 7: No functional changes are made in this issue.

**newlocale**

Purpose: Create or modify a locale object.

CX Synopsis:

```
#include <locale.h>

locale_t newlocale(int category_mask, const char *locale,
     locale_t base);
```

The *newlocale*( ) function creates a new locale object or modifies an existing one.

Application writers should note that handles for locale objects created by the *newlocale*( ) function should be released by a corresponding call to *freelocale*( ). Also, the special locale object *LC_GLOBAL_LOCALE* must not be passed for the *base* argument, even when returned by the *uselocale*( ) function.

The following example shows the construction of a locale where the *LC_CTYPE* category data comes from a locale *loc1*, and the *LC_TIME* category data from a locale *tok2*:

```
#include <locale.h>
...
locale_t loc, new_loc;

/* Get the "loc1" data. */

loc = newlocale (LC_CTYPE_MASK, "loc1", NULL);
if (loc == (locale_t) 0)
```

```
    abort ();
/* Get the "loc2" data. */
new_loc = newlocale (LC_TIME_MASK, "loc2", loc);
if (new_loc != (locale_t) 0)
    /* We don t abort if this fails. In this case this
       simply used to unchanged locale object. */
    loc = new_loc;
...
```

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 4.

Issue 7: First released in Issue 7.

**nextafter, nextafterf, nextafterl, nexttoward, nexttowardf, nexttowardl**

Purpose: Next representable floating-point number.

Synopsis:

```
#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 7: No functional changes are made in this issue.

**nftw**

Purpose: Walk a file tree.

XSI Synopsis:

```
#include <ftw.h>

int nftw(const char *path, int (*fn)(const char *,
    const struct stat *, int, struct FTW *), int fd_limit,
    int flags);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

APPLICATION USAGE is added and the EXAMPLES section is replaced with a new example.

**nice**

Purpose: Change the nice value of a process.

XSI Synopsis:

```
#include <unistd.h>

int nice(int incr);
```

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

Issue 7: No functional changes are made in this issue.

**nl_langinfo, nl_langinfo_l**

Purpose: Language information.

Synopsis:

```
#include <langinfo.h>

char *nl_langinfo(nl_item item);
char *nl_langinfo_l(nl_item item, locale_t locale);
```

Derivation: First released in Issue 2.

Issue 7: The *nl_langinfo*() function is moved from the XSI option to the Base.

The *nl_langinfo_l*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**open, openat**

Purpose: Open file relative to directory file descriptor.

OH Synopsis:

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

int open(const char *path, int oflag, ...);
int openat(int fd, const char *path, int oflag, ...);
```

The *openat*() function is equivalent to the *open*() function except in the case where *path* specifies a relative path. In this case the file to be opened is determined relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the file descriptor was opened without O_SEARCH, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with O_SEARCH, the function does not perform the check.

The purpose of the *openat*() function is to enable opening files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *open*(), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *openat*() function it can be guaranteed that the opened file is located relative to the desired directory.

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

Issue 7: Austin Group Interpretation 1003.1-2001 #113 is applied, requiring the O_SYNC flag to be supported for regular files, even if the Synchronized Input and Output option is not supported.

Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

Austin Group Interpretation 1003.1-2001 #144 is applied, adding the O_TTY_INIT flag.

Austin Group Interpretation 1003.1-2001 #171 is applied, adding support to set the FD_CLOEXEC flag atomically at *open*(), and adding the F_DUPFD_CLOEXEC flag.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

This page is revised and the *openat*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Functionality relating to the XSI STREAMS option is marked obsolescent.

Changes are made related to support for finegrained timestamps.

**open_memstream, open_wmemstream**

Purpose: Open a dynamic memory buffer stream.

CX Synopsis:

```
#include <stdio.h>

FILE *open_memstream(char **bufp, size_t *sizep);

#include <wchar.h>

FILE *open_wmemstream(wchar_t **bufp, size_t *sizep);
```

The *open_memstream*() and *open_wmemstream*() functions create an I/O stream associated with a dynamically allocated memory buffer.

These functions are similar to *fmemopen*(), except that the memory is always allocated dynamically by the function, and the stream is opened only for output.

Application writers should note that the buffer created by these functions should be freed by the application after closing the stream, by means of a call to *free*().

An example program using the *open_memstream*() interface follows:

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

int
main (void)
{
    FILE *stream;
    char *buf;
    size_t len;
    off_t eob;

    stream = open_memstream (&buf, &len);
    if (stream == NULL)
        /* handle error */ ;
    fprintf (stream, "hello my world");
    fflush (stream);
    printf ("buf=%s, len=%zu\n", buf, len);
    eob = ftello(stream);
    fseeko (stream, 0, SEEK_SET);
    fprintf (stream, "good-bye");
    fseeko (stream, eob, SEEK_SET);
```

```
    fclose (stream);
    printf ("buf=%s, len=%zu\n", buf, len);
    free (buf);
    return 0;
}
```

This program produces the following output:

```
buf=hello my world, len=14
buf=good-bye world, len=14
```

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Issue 7: First released in Issue 7.

**pause**

Purpose: Suspend the thread until a signal is received.

Synopsis:

```
#include <unistd.h>

int pause(void);
```

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

Issue 7: No functional changes are made in this issue.

**pclose**

Purpose: Close a pipe stream to or from a process.

CX Synopsis:

```
#include <stdio.h>

int pclose(FILE *stream);
```

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

Issue 7: No functional changes are made in this issue.

**perror**

Purpose: Write error messages to standard error.

Synopsis:

```
#include <stdio.h>

void perror(const char *s);
```

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

Issue 7: Changes are made related to support for finegrained timestamps.

**pipe**

Purpose: Create an interprocess channel.

Synopsis:

```
#include <unistd.h>

int pipe(int fildes[2]);
```

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

Issue 7: SD5-XSH-ERN-156 is applied, updating the DESCRIPTION to state the setting of the pipe's user ID and group ID.

Changes are made related to support for finegrained timestamps.

**poll**

Purpose: Input/output multiplexing.

Synopsis:
```
#include <poll.h>

int poll(struct pollfd fds[], nfds_t nfds, int timeout);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #209 is applied, clarifying the POLLHUP event.

The *poll*( ) function is moved from the XSI option to the Base.

Functionality relating to the XSI STREAMS option is marked obsolescent.

**popen**

Purpose: Initiate pipe streams to or from a process.

CX Synopsis:
```
#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 7: Austin Group Interpretation 1003.1-2001 #029 is applied, clarifying the values for *mode* in the DESCRIPTION.

SD5-XSH-ERN-149 is applied, changing the {STREAM_MAX} [EMFILE] error condition from a "may fail" to a "shall fail".

**posix_fadvise**

Purpose: File advisory information (**ADVANCED REALTIME**).

ADV Synopsis:
```
#include <fcntl.h>

int posix_fadvise(int fd, off_t offset, off_t len,
    int advice);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: Austin Group Interpretation 1003.1-2001 #024 is applied, changing the definition of the [EINVAL] error.

**posix_fallocate**

Purpose: File space control (**ADVANCED REALTIME**).

ADV Synopsis:

```
#include <fcntl.h>

int posix_fallocate(int fd, off_t offset, off_t len);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: Austin Group Interpretations 1003.1-2001 #022, #024, and #162 are applied, changing the definition of the [EINVAL] error.

**posix_madvise**

Purpose: Memory advisory information and alignment control (**ADVANCED REALTIME**).

ADV Synopsis:

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

int posix_madvise(void *addr, size_t len, int advice);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: No functional changes are made in this issue.

**posix_mem_offset**

Purpose: Find offset and length of a mapped typed memory block (**ADVANCED REALTIME**).

TYM Synopsis:

```
#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);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: No functional changes are made in this issue.

**posix_memalign**

Purpose: Aligned memory allocation (**ADVANCED REALTIME**).

ADV Synopsis:

```
#include <stdlib.h>

int posix_memalign(void **memptr, size_t alignment,
    size_t size);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: Austin Group Interpretation 1003.1-2001 #058 is applied, clarifying the value of the *alignment* argument in the DESCRIPTION.

Austin Group Interpretation 1003.1-2001 #152 is applied, clarifying the behavior when the size of the space requested is 0.

**posix_openpt**

Purpose: Open a pseudo-terminal device.

XSI Synopsis:

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

int posix_openpt(int oflag);
```

Derivation: First released in Issue 6.

Issue 7: SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XSH-ERN-51 is applied, correcting an error in the EXAMPLES section.

**posix_spawn, posix_spawnp**

Purpose: Spawn a process (**ADVANCED REALTIME**).

SPN Synopsis:

```
#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]);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: Functionality relating to the Threads option is moved to the Base.

**posix_spawn_file_actions_addclose, posix_spawn_file_actions_addopen**

Purpose: Add close or open action to spawn file actions object (**ADVANCED REALTIME**).

SPN Synopsis:

```
#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);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: No functional changes are made in this issue.

**posix_spawn_file_actions_adddup2**

Purpose: Add dup2 action to spawn file actions object (**ADVANCED REALTIME**).

SPN Synopsis:

```
#include <spawn.h>

int posix_spawn_file_actions_adddup2(
    posix_spawn_file_actions_t
    *file_actions, int fildes, int newfildes);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: No functional changes are made in this issue.

**posix_spawn_file_actions_destroy, posix_spawn_file_actions_init**

Purpose: Destroy and initialize spawn file actions object (**ADVANCED REALTIME**).

SPN Synopsis:

```
#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);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: No functional changes are made in this issue.

**posix_spawnattr_destroy, posix_spawnattr_init**

Purpose: Destroy and initialize spawn attributes object (**ADVANCED REALTIME**).

SPN Synopsis:

```
#include <spawn.h>

int posix_spawnattr_destroy(posix_spawnattr_t *attr);
int posix_spawnattr_init(posix_spawnattr_t *attr);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: No functional changes are made in this issue.

**posix_spawnattr_getflags, posix_spawnattr_setflags**

Purpose: Get and set the spawn-flags attribute of a spawn attributes object (**ADVANCED REALTIME**).

SPN Synopsis:

```
#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);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: No functional changes are made in this issue.

**posix_spawnattr_getpgroup, posix_spawnattr_setpgroup**

Purpose: Get and set the spawn-pgroup attribute of a spawn attributes object (**ADVANCED REALTIME**).

SPN Synopsis:

```
#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);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: No functional changes are made in this issue.

**posix_spawnattr_getschedparam, posix_spawnattr_setschedparam**

Purpose: Get and set the spawn-schedparam attribute of a spawn attributes object (**ADVANCED REALTIME**).

SPN PS Synopsis:

```
#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);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: No functional changes are made in this issue.

**posix_spawnattr_getschedpolicy, posix_spawnattr_setschedpolicy**

Purpose: Get and set the spawn-schedpolicy attribute of a spawn attributes object (**ADVANCED REALTIME**).

SPN PS Synopsis:

```
#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);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: No functional changes are made in this issue.

**posix_spawnattr_getsigdefault, posix_spawnattr_setsigdefault**

Purpose: Get and set the spawn-sigdefault attribute of a spawn attributes object (**ADVANCED REALTIME**).

SPN Synopsis:

```
#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);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: No functional changes are made in this issue.

**posix_spawnattr_getsigmask, posix_spawnattr_setsigmask**

Purpose: Get and set the spawn-sigmask attribute of a spawn attributes object (**ADVANCED REALTIME**).

SPN Synopsis:

```
#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);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: No functional changes are made in this issue.

**posix_trace_attr_destroy, posix_trace_attr_init**

Purpose: Destroy and initialize the trace stream attributes object (**TRACING**).

OB TRC Synopsis:

```
#include <trace.h>

int posix_trace_attr_destroy(trace_attr_t *attr);
int posix_trace_attr_init(trace_attr_t *attr);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: The *posix_trace_attr_destroy*() and *posix_trace_attr_init*() functions are marked obsolescent.

**posix_trace_attr_getclockres, posix_trace_attr_getcreatetime, posix_trace_attr_getgenversion, posix_trace_attr_getname, posix_trace_attr_setname**

Purpose: Retrieve and set information about a trace stream (**TRACING**).

OB TRC Synopsis:

```
#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);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: The *posix_trace_attr_getclockres*( ), *posix_trace_attr_getcreatetime*( ), *posix_trace_attr_getgenversion*( ), *posix_trace_attr_getname*( ), and *posix_trace_attr_setname*( ) functions are marked obsolescent.

**posix_trace_attr_getinherited, posix_trace_attr_getlogfullpolicy, posix_trace_attr_getstreamfullpolicy, posix_trace_attr_setinherited, posix_trace_attr_setlogfullpolicy, posix_trace_attr_setstreamfullpolicy**

Purpose: Retrieve and set the behavior of a trace stream (**TRACING**).

OB TRC Synopsis:

```
#include <trace.h>

TRI int posix_trace_attr_getinherited(
        const trace_attr_t *restrict attr,
        int *restrict inheritancepolicy);
TRL int posix_trace_attr_getlogfullpolicy(
        const trace_attr_t *restrict attr,
        int *restrict logpolicy);
int posix_trace_attr_getstreamfullpolicy(
        const trace_attr_t *restrict
        attr, int *restrict streampolicy);
TRI int posix_trace_attr_setinherited(trace_attr_t *attr,
        int inheritancepolicy);
TRL int posix_trace_attr_setlogfullpolicy(trace_attr_t *attr,
        int logpolicy);
int posix_trace_attr_setstreamfullpolicy(trace_attr_t *attr,
        int streampolicy);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: SD5-XSH-ERN-116 is applied, adding the missing **restrict** keyword to the *posix_trace_attr_getstreamfullpolicy*( ) function declaration.

These functions are marked obsolescent.

**posix_trace_attr_getlogsize, posix_trace_attr_getmaxdatasize, posix_trace_attr_getmaxsystemeventsize, posix_trace_attr_getmaxusereventsize, posix_trace_attr_getstreamsize, posix_trace_attr_setlogsize, posix_trace_attr_setmaxdatasize, posix_trace_attr_setstreamsize**

Purpose: Retrieve and set trace stream size attributes (**TRACING**).

OB TRC Synopsis:

```
#include <sys/types.h>
#include <trace.h>

int posix_trace_attr_getlogsize(
    const trace_attr_t *restrict attr,
    size_t *restrict logsize);
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);
int posix_trace_attr_setlogsize(trace_attr_t *attr,
    size_t logsize);
int posix_trace_attr_setmaxdatasize(trace_attr_t *attr,
    size_t maxdatasize);
int posix_trace_attr_setstreamsize(trace_attr_t *attr,
    size_t streamsize);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: These functions are marked obsolescent.

**posix_trace_clear**

Purpose: Clear trace stream and trace log (**TRACING**).

OB TRC Synopsis:

```
#include <sys/types.h>
#include <trace.h>

int posix_trace_clear(trace_id_t trid);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: The *posix_trace_clear*( ) function is marked obsolescent.

**posix_trace_close, posix_trace_open, posix_trace_rewind**

Purpose: Trace log management (**TRACING**).

OB TRC Synopsis:

```
#include <trace.h>

int posix_trace_close(trace_id_t trid);
TRL int posix_trace_open(int file_desc, trace_id_t *trid);
int posix_trace_rewind(trace_id_t trid);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: The *posix_trace_close*(), *posix_trace_open*(), and *posix_trace_rewind*() functions are marked obsolescent.

**posix_trace_create, posix_trace_create_withlog, posix_trace_flush, posix_trace_shutdown**

Purpose: Trace stream initialization, flush, and shutdown from a process (**TRACING**).

OB TRC Synopsis:

```
#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);
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);
int posix_trace_shutdown(trace_id_t trid);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: These functions are marked obsolescent.

SD5-XSH-ERN-154 is applied, updating the DESCRIPTION to remove the *posix_trace_trygetnext_event*() function from the list of functions that use the *trid* argument.

**posix_trace_event, posix_trace_eventid_open**

Purpose: Trace functions for instrumenting application code (**TRACING**).

OB TRC Synopsis:

```
#include <sys/types.h>
#include <trace.h>

void posix_trace_event(trace_event_id_t event_id,
    const void *restrict data_ptr, size_t data_len);
int posix_trace_eventid_open(const char *restrict event_name,
    trace_event_id_t *restrict event_id);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: The *posix_trace_event*() and *posix_trace_eventid_open*() functions are marked obsolescent.

**posix_trace_eventid_equal, posix_trace_eventid_get_name, posix_trace_trid_eventid_open**

Purpose: Manipulate the trace event type identifier (**TRACING**).

OB TRC Synopsis:

```
#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);
TEF int posix_trace_trid_eventid_open(trace_id_t trid,
    const char *restrict event_name,
    trace_event_id_t *restrict event);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: These functions are marked obsolescent.

**posix_trace_eventset_add, posix_trace_eventset_del, posix_trace_eventset_empty, posix_trace_eventset_fill, posix_trace_eventset_ismember**

Purpose: Manipulate trace event type sets (**TRACING**).

OB TRC Synopsis:

```
#include <trace.h>

TEF 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);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: The *posix_trace_eventset_add*( ), *posix_trace_eventset_del*( ), *posix_trace_eventset_empty*( ), *posix_trace_eventset_fill*( ), and *posix_trace_eventset_ismember*( ) functions are marked obsolescent.

**posix_trace_eventtypelist_getnext_id, posix_trace_eventtypelist_rewind**

Purpose: Iterate over a mapping of trace event types (**TRACING**).

OB TRC Synopsis:

```
#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);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: The *posix_trace_eventtypelist_getnext_id*() and *posix_trace_eventtypelist_rewind*() functions are marked obsolescent.

**posix_trace_get_attr, posix_trace_get_status**

Purpose: Retrieve the trace attributes or trace status (**TRACING**).

OB TRC Synopsis:

```
#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);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: The *posix_trace_get_attr*() and *posix_trace_get_status*() functions are marked obsolescent.

**posix_trace_get_filter, posix_trace_set_filter**

Purpose: Retrieve and set the filter of an initialized trace stream (**TRACING**).

OB TRC Synopsis:

```
#include <trace.h>
```

TEF

```
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);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: The *posix_trace_get_filter*() and *posix_trace_set_filter*() functions are marked obsolescent.

**posix_trace_getnext_event, posix_trace_timedgetnext_event, posix_trace_trygetnext_event**

Purpose: Retrieve a trace event (**TRACING**).

OB TRC Synopsis:

```
#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);
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 abstime);
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);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: The *posix_trace_getnext_event*( ), *posix_trace_timedgetnext_event*( ), and *posix_trace_trygetnext_event*( ) functions are marked obsolescent.

Functionality relating to the Timers option is moved to the Base.

**posix_trace_start, posix_trace_stop**

Purpose: Trace start and stop (**TRACING**).

OB TRC Synopsis:

```
#include <trace.h>

int posix_trace_start(trace_id_t trid);
int posix_trace_stop (trace_id_t trid);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: The *posix_trace_start*( ) and *posix_trace_stop*( ) functions are marked obsolescent.

**posix_typed_mem_get_info**

Purpose: Query typed memory information (**ADVANCED REALTIME**).

TYM Synopsis:

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

int posix_typed_mem_get_info(int fildes,
    struct posix_typed_mem_info *info);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: No functional changes are made in this issue.

**posix_typed_mem_open**

Purpose: Open a typed memory object (**ADVANCED REALTIME**).

TYM Synopsis:

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

int posix_typed_mem_open(const char *name, int oflag,
    int tflag);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

**pow, powf, powl**

Purpose: Power function.

Synopsis:

```
#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 7: ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #51 (SD5-XSH-ERN-81) is applied.

**pselect, select**

Purpose: Synchronous I/O multiplexing.

Synopsis:

```
#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);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: SD5-XSH-ERN-122 is applied, adding text to the DESCRIPTION for when a thread is canceled during a call to *pselect*(), and adding example code to the RATIONALE.

Functionality relating to the XSI STREAMS option is marked obsolescent.

Functionality relating to the Threads option is moved to the Base.

**psiginfo, psignal**

Purpose: Print signal information to standard error.

CX Synopsis:

```
#include <signal.h>

void psiginfo(const siginfo_t *pinfo, const char *message);
void psignal(int signum, const char *message);
```

The *psiginfo*() and *psignal*() functions print a message out on *stderr* associated with a signal number.

Application writers should note that System V historically has *psignal*() and *psiginfo*() in **<siginfo.h>**. However, the **<siginfo.h>** header is not specified in the Base Definitions volume of IEEE Std 1003.1-2001, and the type **siginfo_t** is defined in **<signal.h>**.

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Issue 7: First released in Issue 7.

**pthread_atfork**

Purpose: Register fork handlers.

Synopsis:
```
#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 7: The *pthread_atfork*() function is moved from the Threads option to the Base.

SD5-XSH-ERN-145 is applied, updating the RATIONALE to confirm the requirement that a child of a multi-threaded process may only execute async-signal-safe operations until such time as one of the *exec* functions is called.

**pthread_attr_destroy, pthread_attr_init**

Purpose: Destroy and initialize the thread attributes object.

Synopsis:
```
#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 7: The *pthread_attr_destroy*() and *pthread_attr_init*() functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

The [EBUSY] error for an already initialized thread attributes object is removed; this condition results in undefined behavior.

**pthread_attr_getdetachstate, pthread_attr_setdetachstate**

Purpose: Get and set the detachstate attribute.

Synopsis:
```
#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 7: The *pthread_attr_setdetachstate*() and *pthread_attr_getdetachstate*() functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

**pthread_attr_getguardsize, pthread_attr_setguardsize**

Purpose: Get and set the thread guardsize attribute.

Synopsis:
```
#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 7: SD5-XSH-ERN-111 is applied, removing the reference to the *stack* attribute in the DESCRIPTION.

SD5-XSH-ERN-175 is applied, updating the DESCRIPTION to note that the default size of the guard area is implementation-defined.

The *pthread_attr_getguardsize*() and *pthread_attr_setguardsize*() functions are moved from the XSI option to the Base.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

**pthread_attr_getinheritsched, pthread_attr_setinheritsched**

Purpose: Get and set the inheritsched attribute (**REALTIME THREADS**).

TPS Synopsis:
```
#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.

Issue 7: The *pthread_attr_getinheritsched*() and *pthread_attr_setinheritsched*() functions are moved from the Threads option.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

**pthread_attr_getschedparam, pthread_attr_setschedparam**

Purpose: Get and set the schedparam attribute.

Synopsis:
```
#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 7: The *pthread_attr_getschedparam*() and *pthread_attr_setschedparam*() functions are moved from the Threads option to the Base.

Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements for the *sched_ss_repl_period* and *sched_ss_init_budget* values.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

**pthread_attr_getschedpolicy, pthread_attr_setschedpolicy**

Purpose: Get and set the schedpolicy attribute (**REALTIME THREADS**).

TPS Synopsis:

```
#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.

Issue 7: The *pthread_attr_getschedpolicy*() and *pthread_attr_setschedpolicy*() functions are moved from the Threads option.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

**pthread_attr_getscope, pthread_attr_setscope**

Purpose: Get and set the contentionscope attribute (**REALTIME THREADS**).

TPS Synopsis:

```
#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.

Issue 7: The *pthread_attr_getscope*() and *pthread_attr_setscope*() functions are moved from the Threads option.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

**pthread_attr_getstack, pthread_attr_setstack**

Purpose: Get and set stack attributes.

TSA TSS Synopsis:

```
#include <pthread.h>

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);
```

Derivation: First released in Issue 6.

Issue 7: SD5-XSH-ERN-66 is applied, correcting the use of *attr* in the [EINVAL] error condition.

Austin Group Interpretation 1003.1-2001 #057 is applied, clarifying the behavior if the function is called before the *stackaddr* attribute is set.

SD5-XSH-ERN-157 is applied, updating the APPLICATION USAGE section.

The description of the *stackaddr* attribute is updated in the DESCRIPTION and APPLICATION USAGE sections.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

**pthread_attr_getstacksize, pthread_attr_setstacksize**

Purpose: Get and set the stacksize attribute.

TSS Synopsis:

```
#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 7: The *pthread_attr_getstacksize*() and *pthread_attr_setstacksize*() functions are moved from the Threads option.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

**pthread_barrier_destroy, pthread_barrier_init**

Purpose: Destroy and initialize a barrier object.

Synopsis:

```
#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);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: The *pthread_barrier_destroy*() and *pthread_barrier_init*() functions are moved from the Barriers option to the Base.

The [EINVAL] error for an uninitialized barrier object and an uninitialized barrier attributes object is removed; this condition results in undefined behavior.

The [EBUSY] error for a barrier that is in use or an already initialized barrier object is removed; this condition results in undefined behavior.

**pthread_barrier_wait**

Purpose: Synchronize at a barrier.

Synopsis:
```
#include <pthread.h>

int pthread_barrier_wait(pthread_barrier_t *barrier);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: The *pthread_barrier_wait*() function is moved from the Barriers option to the Base.

The [EINVAL] error for an uninitialized barrier object is removed; this condition results in undefined behavior.

**pthread_barrierattr_destroy, pthread_barrierattr_init**

Purpose: Destroy and initialize the barrier attributes object.

Synopsis:
```
#include <pthread.h>

int pthread_barrierattr_destroy(pthread_barrierattr_t *attr);
int pthread_barrierattr_init(pthread_barrierattr_t *attr);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: The *pthread_barrierattr_destroy*() and *pthread_barrierattr_init*() functions are moved from the Barriers option to the Base.

The [EINVAL] error for an uninitialized barrier attributes object is removed; this condition results in undefined behavior.

**pthread_barrierattr_getpshared, pthread_barrierattr_setpshared**

Purpose: Get and set the process-shared attribute of the barrier attributes object.

TSH Synopsis:
```
#include <pthread.h>

int pthread_barrierattr_getpshared(
    const pthread_barrierattr_t *restrict attr,
    int *restrict pshared);
int pthread_barrierattr_setpshared(
    pthread_barrierattr_t *attr, int pshared);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000

Issue 7: The *pthread_barrierattr_getpshared*() and *pthread_barrierattr_setpshared*() functions are moved from the Barriers option.

The [EINVAL] error for an uninitialized barrier attributes object is removed; this condition results in undefined behavior.

**pthread_cancel**

Purpose: Cancel execution of a thread.

Synopsis:
```
#include <pthread.h>

int pthread_cancel(pthread_t thread);
```

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

Issue 7: The *pthread_cancel*() function is moved from the Threads option to the Base.

Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

**pthread_cleanup_pop, pthread_cleanup_push**

Purpose: Establish cancellation handlers.

Synopsis:
```
#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 7: The *pthread_cleanup_pop*() and *pthread_cleanup_push*() functions are moved from the Threads option to the Base.

**pthread_cond_broadcast, pthread_cond_signal**

Purpose: Broadcast or signal a condition.

Synopsis:
```
#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 7: The *pthread_cond_broadcast*() and *pthread_cond_signal*() functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized condition variable is removed; this condition results in undefined behavior.

**pthread_cond_destroy, pthread_cond_init**

Purpose: Destroy and initialize condition variables.

Synopsis:
```
#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 7: The *pthread_cond_destroy*() and *pthread_cond_init*() functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized condition variable and an uninitialized condition variable attributes object is removed; this condition results in undefined behavior.

The [EBUSY] error for a condition variable already in use or an already initialized condition variable is removed; this condition results in undefined behavior.

**pthread_cond_timedwait, pthread_cond_wait**

Purpose: Wait on a condition.

Synopsis:
```
#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 7: SD5-XSH-ERN-44 is applied, changing the definition of the "shall fail" case of the [EINVAL] error.

Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

The *pthread_cond_timedwait*() and *pthread_cond_wait*() functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized condition variable or uninitialized mutex object is removed; this condition results in undefined behavior"

The [EPERM] error is revised and moved to the "shall fail" list of error conditions for the *pthread_cond_timedwait*() function.

The DESCRIPTION is updated to clarify the behavior when *mutex* is a robust mutex.

The ERRORS section is updated to include "shall fail" cases for PTHREAD_MUTEX_ERRORCHECK mutexes.

The DESCRIPTION is rewritten to clarify that undefined behavior occurs only for mutexes where the [EPERM] error is not mandated.

**pthread_condattr_destroy, pthread_condattr_init**

Purpose: Destroy and initialize the condition variable attributes object.

Synopsis:
```
#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 7: The *pthread_condattr_destroy*() and *pthread_condattr_init*() functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized condition variable attributes object is removed; this condition results in undefined behavior.

**pthread_condattr_getclock, pthread_condattr_setclock**

Purpose: Get and set the clock selection condition variable attribute.

Synopsis:
```
#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);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: The *pthread_condattr_getclock*() and *pthread_condattr_setclock*() functions are moved from the Clock Selection option to the Base.

The [EINVAL] error for an uninitialized condition variable attributes object is removed; this condition results in undefined behavior.

**pthread_condattr_getpshared, pthread_condattr_setpshared**

Purpose: Get and set the process-shared condition variable attributes.

TSH Synopsis:
```
#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 7: The *pthread_condattr_getpshared*() and *pthread_condattr_setpshared*() functions are moved from the Threads option.

The [EINVAL] error for an uninitialized condition variable attributes object is removed; this condition results in undefined behavior.

**pthread_create**

Purpose: Thread creation.

Synopsis:
```
#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 7: The *pthread_create*() function is moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

**pthread_detach**

Purpose: Detach a thread.

Synopsis:
```
#include <pthread.h>

int pthread_detach(pthread_t thread);
```

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

Issue 7: The *pthread_detach*() function is moved from the Threads option to the Base.

Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

The [EINVAL] error for a non-joinable thread is removed; this condition results in undefined behavior.

**pthread_equal**

Purpose: Compare thread IDs.

Synopsis:
```
#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 7: The *pthread_equal*() function is moved from the Threads option to the Base.

**pthread_exit**

Purpose: Thread termination.

Synopsis:
```
#include <pthread.h>

void pthread_exit(void *value_ptr);
```

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

Issue 7: The *pthread_exit*() function is moved from the Threads option to the Base.

**pthread_getconcurrency, pthread_setconcurrency**

Purpose: Get and set the level of concurrency.

OB XSI Synopsis:
```
#include <pthread.h>

int pthread_getconcurrency(void);
int pthread_setconcurrency(int new_level);
```

Derivation: First released in Issue 5.

Issue 7: The *pthread_getconcurrency*() and *pthread_setconcurrency*() functions are marked obsolescent.

**pthread_getcpuclockid**

Purpose: Access a thread CPU-time clock (**ADVANCED REALTIME THREADS**).

TCT Synopsis:

```
#include <pthread.h>
#include <time.h>

int pthread_getcpuclockid(pthread_t thread_id,
    clockid_t *clock_id);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: The *pthread_getcpuclockid*() function is moved from the Threads option.

Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

**pthread_getschedparam, pthread_setschedparam**

Purpose: Dynamic thread scheduling parameters access (**REALTIME THREADS**).

TPS Synopsis:

```
#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 7: The *pthread_getschedparam*() and *pthread_setschedparam*() functions are moved from the Threads option.

Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements for the *sched_ss_repl_period* and *sched_ss_init_budget* values.

Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

**pthread_getspecific, pthread_setspecific**

Purpose: Thread-specific data management.

Synopsis:

```
#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 7: The *pthread_getspecific*() and *pthread_setspecific*() functions are moved from the Threads option to the Base.

The [EINVAL] error for a key value not obtained from *pthread_key_create*() or a key

deleted with *pthread_key_delete*() is removed; this condition results in undefined behavior.

**pthread_join**

Purpose: Wait for thread termination.

Synopsis:
```
#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 7: The *pthread_join*() function is moved from the Threads option to the Base.

Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

The [EINVAL] error for a non-joinable thread is removed; this condition results in undefined behavior.

The [EDEADLK] error for the calling thread is removed; this condition results in undefined behavior.

**pthread_key_create**

Purpose: Thread-specific data key creation.

Synopsis:
```
#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 7: The *pthread_key_create*() function is moved from the Threads option to the Base.

**pthread_key_delete**

Purpose: Thread-specific data key deletion.

Synopsis:
```
#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 7: The *pthread_key_delete*() function is moved from the Threads option to the Base.

The [EINVAL] error for a key value not obtained from *pthread_key_create*() or a key deleted with *pthread_key_delete*() is removed; this condition results in undefined behavior.

**pthread_kill**

Purpose: Send a signal to a thread.

CX Synopsis:

```
#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 7: The *pthread_kill*() function is moved from the Threads option to the Base.

Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

**pthread_mutex_consistent**

Purpose: Mark state protected by robust mutex as consistent.

Synopsis:

```
#include <pthread.h>

int pthread_mutex_consistent(pthread_mutex_t *mutex);
```

If *mutex* is a robust mutex in an inconsistent state, the *pthread_mutex_consistent*() function can be used to mark the state protected by the mutex referenced by *mutex* as consistent again.

Application writers should note that the *pthread_mutex_consistent*() function is only responsible for notifying the implementation that the state protected by the mutex has been recovered and that normal operations with the mutex can be resumed. It is the responsibility of the application to recover the state so it can be reused. If the application is not able to perform the recovery, it can notify the implementation that the situation is unrecoverable by a call to *pthread_mutex_unlock*() without a prior call to *pthread_mutex_consistent*(), in which case subsequent threads that attempt to lock the mutex will fail to acquire the lock and be returned [ENOTRECOVERABLE].

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 3.

Issue 7: First released in Issue 7.

**pthread_mutex_destroy, pthread_mutex_init**

Purpose: Destroy and initialize a mutex.

Synopsis:

```
#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 7: Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

The *pthread_mutex_destroy*() and *pthread_mutex_init*() functions are moved from

the Threads option to the Base.

The [EINVAL] error for an uninitialized mutex or an uninitialized mutex attributes object is removed; this condition results in undefined behavior.

The [EBUSY] error for a locked mutex, a mutex that is referenced, or an already initialized mutex is removed; this condition results in undefined behavior.

**pthread_mutex_getprioceiling, pthread_mutex_setprioceiling**

Purpose: Get and set the priority ceiling of a mutex (**REALTIME THREADS**).

RPP | TPP Synopsis:

```
#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.

Issue 7: Austin Group Interpretation 1003.1-2001 #052 is applied, adding [EDEADLK] as a "may fail" error.

SD5-XSH-ERN-158 is applied, updating the ERRORS section to include a "shall fail" error case for when the protocol attribute of *mutex* is PTHREAD_PRIO_NONE.

The *pthread_mutex_getprioceiling*() and *pthread_mutex_setprioceiling*() functions are moved from the Threads option to require support of either the Robust Mutex Priority Protection option or the Non-Robust Mutex Priority Protection option.

The DESCRIPTION and ERRORS sections are updated to account properly for all of the various mutex types.

**pthread_mutex_lock, pthread_mutex_trylock, pthread_mutex_unlock**

Purpose: Lock and unlock a mutex.

Synopsis:

```
#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 7: SD5-XSH-ERN-43 is applied, marking the "shall fail" case of the [EINVAL] error as dependent on the Thread Priority Protection option.

Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

The *pthread_mutex_lock*(), *pthread_mutex_trylock*(), and *pthread_mutex_unlock*() functions are moved from the Threads option to the Base.

The PTHREAD_MUTEX_NORMAL, PTHREAD_MUTEX_ERRORCHECK,

PTHREAD_MUTEX_RECURSIVE, and PTHREAD_MUTEX_DEFAULT extended mutex types are moved from the XSI option to the Base.

The DESCRIPTION is updated to clarify the behavior when *mutex* does not refer to an initialized mutex.

The ERRORS section is updated to account properly for all of the various mutex types.

**pthread_mutex_timedlock**

Purpose: Lock a mutex.

Synopsis:
```
#include <pthread.h>
#include <time.h>

int pthread_mutex_timedlock(pthread_mutex_t *restrict mutex,
    const struct timespec *restrict abstime);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

The *pthread_mutex_timedlock*() function is moved from the Timeouts option to the Base.

Functionality relating to the Timers option is moved to the Base.

The DESCRIPTION is updated to clarify the behavior when *mutex* does not refer to an initialized mutex.

The ERRORS section is updated to account properly for all of the various mutex types.

**pthread_mutexattr_destroy, pthread_mutexattr_init**

Purpose: Destroy and initialize the mutex attributes object.

Synopsis:
```
#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 7: The *pthread_mutexattr_destroy*() and *pthread_mutexattr_init*() functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition results in undefined behavior.

**pthread_mutexattr_getprioceiling, pthread_mutexattr_setprioceiling**

Purpose: Get and set the prioceiling attribute of the mutex attributes object (**REALTIME THREADS**).

RPP|TPP Synopsis:

```
#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.

Issue 7: The *pthread_mutexattr_getprioceiling*() and *pthread_mutexattr_setprioceiling*() functions are moved from the Threads option to require support of either the Robust Mutex Priority Protection option or the Non-Robust Mutex Priority Protection option.

The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition results in undefined behavior.

**pthread_mutexattr_getprotocol, pthread_mutexattr_setprotocol**

Purpose: Get and set the protocol attribute of the mutex attributes object (**REALTIME THREADS**).

MC1 Synopsis:

```
#include <pthread.h>

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.

Issue 7: SD5-XSH-ERN-135 is applied, updating the DESCRIPTION to define a default value for the *protocol* attribute.

SD5-XSH-ERN-188 is applied, clarifying that propagation of the priority inheritance effect only applies if the other mutex has the protocol attribute PTHREAD_PRIO_INHERIT.

The *pthread_mutexattr_getprotocol*() and *pthread_mutexattr_setprotocol*() functions are moved from the Threads option to require support of either the Non-Robust Mutex Priority Protection option or the Non-Robust Mutex Priority Inheritance option or the Robust Mutex Priority Protection option or the Robust Mutex Priority Inheritance option.

The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition results in undefined behavior.

**pthread_mutexattr_getpshared, pthread_mutexattr_setpshared**

Purpose: Get and set the process-shared attribute.

TSH Synopsis:

```
#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 7: The *pthread_mutexattr_getpshared*( ) and *pthread_mutexattr_setpshared*( ) functions are moved from the Threads option.

The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition results in undefined behavior.

**pthread_mutexattr_getrobust, pthread_mutexattr_setrobust**

Purpose: Get and set the mutex robust attribute.

Synopsis:

```
#include <pthread.h>

int pthread_mutexattr_getrobust(
    const pthread_mutexattr_t *restrict attr,
    int *restrict robust);
int pthread_mutexattr_setrobust(pthread_mutexattr_t *attr,
    int robust);
```

The *pthread_mutexattr_getrobust*( ) and *pthread_mutexattr_setrobust*( ) functions, respectively, get and set the mutex *robust* attribute.

Valid values for *robust* include:

PTHREAD_MUTEX_STALLED
: No special actions are taken if the owner of the mutex is terminated while holding the mutex lock. This can lead to deadlocks if no other thread can unlock the mutex.
This is the default value.

PTHREAD_MUTEX_ROBUST
: If the process containing the owning thread of a robust mutex terminates while holding the mutex lock, the next thread that acquires the mutex is notified about the termination by the return value [EOWNERDEAD] from the locking function. If the owning thread of a robust mutex terminates while holding the mutex lock, the next thread that acquires the mutex may be notified about the termination by the return value [EOWNERDEAD]. The notified thread can then attempt to mark the state protected by the mutex as consistent again by a call to *pthread_mutex_consistent*( ). After a subsequent successful call to *pthread_mutex_unlock*( ), the mutex lock is released and can be used normally by other threads. If the mutex is unlocked without a call to *pthread_mutex_consistent*( ), it is placed in a permanently unusable state and all attempts to lock the mutex fail with the error [ENOTRECOVERABLE]. The only permissible operation on such a mutex is *pthread_mutex_destroy*( ).

Application writers should note that the actions required to make the state

protected by the mutex consistent again are solely dependent on the application. If it is not possible to make the state of a mutex consistent, robust mutexes can be used to notify this situation by calling *pthread_mutex_unlock*() without a prior call to *pthread_mutex_consistent*().

If the state is declared inconsistent by calling *pthread_mutex_unlock*() without a prior call to *pthread_mutex_consistent*(), a possible approach could be to destroy the mutex and then reinitialize it. However, it should be noted that this is possible only in certain situations where the state protected by the mutex has to be reinitialized and coordination achieved with other threads blocked on the mutex, because otherwise a call to a locking function with a reference to a mutex object invalidated by a call to *pthread_mutex_destroy*() results in undefined behavior.

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 3.

Issue 7: First released in Issue 7.

**pthread_mutexattr_gettype, pthread_mutexattr_settype**

Purpose: Get and set the mutex type attribute.

Synopsis:
```
#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 7: The *pthread_mutexattr_gettype*() and *pthread_mutexattr_settype*() functions are moved from the XSI option to the Base.

The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition results in undefined behavior.

**pthread_once**

Purpose: Dynamic package initialization.

Synopsis:
```
#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 7: The *pthread_once*() function is moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized **pthread_once_t** object is removed; this condition results in undefined behavior.

**pthread_rwlock_destroy, pthread_rwlock_init**

Purpose: Destroy and initialize a read-write lock object.

Synopsis:
```
#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);
```
XSI
```
pthread_rwlock_t rwlock = PTHREAD_RWLOCK_INITIALIZER;
```

Derivation: First released in Issue 5.

Issue 7: Austin Group Interpretation 1003.1-2001 #048 is applied, adding the PTHREAD_RWLOCK_INITIALIZER macro.

The *pthread_rwlock_destroy*() and *pthread_rwlock_init*() functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized read-write lock object or read-write lock attributes object is removed; this condition results in undefined behavior.

The [EBUSY] error for a locked read-write lock object or an already initialized read-write lock object is removed; this condition results in undefined behavior.

**pthread_rwlock_rdlock, pthread_rwlock_tryrdlock**

Purpose: Lock a read-write lock object for reading.

Synopsis:
```
#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 7: The *pthread_rwlock_rdlock*() and *pthread_rwlock_tryrdlock*() functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results in undefined behavior.

**pthread_rwlock_timedrdlock**

Purpose: Lock a read-write lock for reading.

Synopsis:
```
#include <pthread.h>
#include <time.h>

int pthread_rwlock_timedrdlock(
    pthread_rwlock_t *restrict rwlock,
    const struct timespec *restrict abstime);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: The *pthread_rwlock_timedrdlock*() function is moved from the Timeouts option to the Base.

The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results in undefined behavior.

**pthread_rwlock_timedwrlock**

Purpose: Lock a read-write lock for writing.

Synopsis:
```
#include <pthread.h>
#include <time.h>

int pthread_rwlock_timedwrlock(
    pthread_rwlock_t *restrict rwlock,
    const struct timespec *restrict abstime);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: The *pthread_rwlock_timedwrlock*() function is moved from the Timeouts option to the Base.

The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results in undefined behavior.

**pthread_rwlock_trywrlock, pthread_rwlock_wrlock**

Purpose: Lock a read-write lock object for writing.

Synopsis:
```
#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 7: The *pthread_rwlock_trywrlock*() and *pthread_rwlock_wrlock*() functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results in undefined behavior.

**pthread_rwlock_unlock**

Purpose: Unlock a read-write lock object.

Synopsis:
```
#include <pthread.h>

int pthread_rwlock_unlock(pthread_rwlock_t *rwlock);
```

Derivation: First released in Issue 5.

Issue 7: The *pthread_rwlock_unlock*() function is moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results in undefined behavior.

The [EPERM] error for a read-write lock object for which the current thread does not hold a lock is removed; this condition results in undefined behavior.

**pthread_rwlockattr_destroy, pthread_rwlockattr_init**

Purpose: Destroy and initialize the read-write lock attributes object.

Synopsis:
```
#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 7: The *pthread_rwlockattr_destroy*() and *pthread_rwlockattr_init*() functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized read-write lock attributes object is removed; this condition results in undefined behavior.

**pthread_rwlockattr_getpshared, pthread_rwlockattr_setpshared**

Purpose: Get and set the process-shared attribute of the read-write lock attributes object.

TSH Synopsis:
```
#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 7: The *pthread_rwlockattr_getpshared*() and *pthread_rwlockattr_setpshared*() functions are moved from the Threads option.

The [EINVAL] error for an uninitialized read-write lock attributes object is removed; this condition results in undefined behavior.

**pthread_self**

Purpose: Get the calling thread ID.

Synopsis:
```
#include <pthread.h>

pthread_t pthread_self(void);
```

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

Issue 7: Austin Group Interpretation 1003.1-2001 #063 is applied, updating the RETURN VALUE section to indicate that the *pthread_self*() function is always successful and no return value is reserved to indicate an error.

The *pthread_self*() function is moved from the Threads option to the Base.

**pthread_setcancelstate, pthread_setcanceltype, pthread_testcancel**

Purpose: Set cancelability state.

Synopsis:
```
#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 7: The *pthread_setcancelstate*( ), *pthread_setcanceltype*( ), and *pthread_testcancel*( ) functions are moved from the Threads option to the Base.

**pthread_setschedprio**

Purpose: Dynamic thread scheduling parameters access (**REALTIME THREADS**).

TPS Synopsis:
```
#include <pthread.h>

int pthread_setschedprio(pthread_t thread, int prio);
```

Derivation: First released in Issue 6. Included as a response to IEEE PASC Interpretation 1003.1 #96.

Issue 7: The *pthread_setschedprio*( ) function is moved from the Threads option.

Austin Group Interpretation 1003.1-2001 #069 is applied, updating the [EPERM] error.

Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

**pthread_sigmask, sigprocmask**

Purpose: Examine and change blocked signals.

CX Synopsis:
```
#include <signal.h>

int pthread_sigmask(int how, const sigset_t *restrict set,
    sigset_t *restrict oset);
int sigprocmask(int how, const sigset_t *restrict set,
    sigset_t *restrict oset);
```

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

Issue 7: The *pthread_sigmask*( ) function is moved from the Threads option to the Base.

**pthread_spin_destroy, pthread_spin_init**

Purpose: Destroy or initialize a spin lock object.

Synopsis:
```
#include <pthread.h>

int pthread_spin_destroy(pthread_spinlock_t *lock);
int pthread_spin_init(pthread_spinlock_t *lock, int pshared);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: The *pthread_spin_destroy*() and *pthread_spin_init*() functions are moved from the Spin Locks option to the Base.

The [EINVAL] error for an uninitialized spin lock object is removed; this condition results in undefined behavior.

The [EBUSY] error for a locked spin lock object or an already initialized spin lock object is removed; this condition results in undefined behavior.

**pthread_spin_lock, pthread_spin_trylock**

Purpose: Lock a spin lock object.

Synopsis:
```
#include <pthread.h>

int pthread_spin_lock(pthread_spinlock_t *lock);
int pthread_spin_trylock(pthread_spinlock_t *lock);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: The *pthread_spin_lock*() and *pthread_spin_trylock*() functions are moved from the Spin Locks option to the Base.

The [EINVAL] error for an uninitialized spin lock object is removed; this condition results in undefined behavior.

The [EDEADLK] error for a spin lock object for which the calling thread already holds the lock is removed; this condition results in undefined behavior.

**pthread_spin_unlock**

Purpose: Unlock a spin lock object.

Synopsis:
```
#include <pthread.h>

int pthread_spin_unlock(pthread_spinlock_t *lock);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: The *pthread_spin_unlock*() function is moved from the Spin Locks option to the Base.

The [EINVAL] error for an uninitialized spin lock object is removed; this condition results in undefined behavior.

The [EPERM] error for a spin lock object for which the current thread does not hold the lock is removed; this condition results in undefined behavior.

**ptsname**

Purpose: Get name of the slave pseudo-terminal device.

XSI Synopsis:

```
#include <stdlib.h>
char *ptsname(int fildes);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

**putc**

Purpose: Put a byte on a stream.

Synopsis:

```
#include <stdio.h>
int putc(int c, FILE *stream);
```

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

Issue 7: No functional changes are made in this issue.

**putchar**

Purpose: Put a byte on a stdout stream.

Synopsis:

```
#include <stdio.h>
int putchar(int c);
```

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

Issue 7: No functional changes are made in this issue.

**putenv**

Purpose: Change or add a value to an environment.

XSI Synopsis:

```
#include <stdlib.h>
int putenv(char *string);
```

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

Issue 7: No functional changes are made in this issue.

**putmsg, putpmsg**

Purpose: Send a message on a STREAM (**STREAMS**).

OB XSR Synopsis:

```
#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 7: The *putmsg*() and *putpmsg*() functions are marked obsolescent.

**puts**

Purpose: Put a string on standard output.

Synopsis:
```
#include <stdio.h>

int puts(const char *s);
```

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

Issue 7: Changes are made related to support for finegrained timestamps.

**putwc**

Purpose: Put a wide character on a stream.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**putwchar**

Purpose: Put a wide character on a stdout stream.

Synopsis:
```
#include <wchar.h>

wint_t putwchar(wchar_t wc);
```

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

**qsort**

Purpose: Sort a table of data.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**raise**

Purpose: Send a signal to the executing process.

Synopsis:
```
#include <signal.h>

int raise(int sig);
```

Derivation: First released in Issue 4. Derived from the IEEE Std 1003.1b-1993.

Issue 7: Functionality relating to the Threads option is moved to the Base.

**rand, rand_r, srand**

Purpose: Pseudo-random number generator.

Synopsis:
```
#include <stdlib.h>

int rand(void);
OB CX int rand_r(unsigned *seed);
void srand(unsigned seed);
```

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

Issue 7: The *rand_r*() function is marked obsolescent. Applications should use *random*() instead, or *erand48*(), *nrand48*(), or *jrand48*() when an independent random number sequence in multiple threads is required.

**pread, read**

Purpose: Read from a file.

Synopsis:
```
#include <unistd.h>

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 7: The *pread*() function is moved from the XSI option to the Base.

Functionality relating to the XSI STREAMS option is marked obsolescent.

Changes are made related to support for finegrained timestamps.

**readdir, readdir_r**

Purpose: Read a directory.

Synopsis:
```
#include <dirent.h>

struct dirent *readdir(DIR *dirp);
int readdir_r(DIR *restrict dirp,
    struct dirent *restrict entry,
    struct dirent **restrict result);
```

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #059 is applied, updating the ERRORS section.

The *readdir_r*() function is moved from the Thread-Safe Functions option to the Base.

Changes are made related to support for finegrained timestamps.

The value of the *d_ino* member is no longer unspecified for symbolic links.

**readlink, readlinkat**

Purpose: Read the contents of a symbolic link relative to a directory file descriptor.

Synopsis:
```
#include <unistd.h>

ssize_t readlink(const char *restrict path,
    char *restrict buf, size_t bufsize);
ssize_t readlinkat(int fd, const char *restrict path,
    char *restrict buf, size_t bufsize);
```

The *readlinkat*() function is equivalent to the *readlink*() function except in the case where *path* specifies a relative path. In this case the symbolic link whose content is read is relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the file descriptor was opened without O_SEARCH, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with O_SEARCH, the function does not perform the check.

The purpose of the *readlinkat*() function is to read the content of symbolic links in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *readlink*(), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *readlinkat*() function it can be guaranteed that the symbolic link read is located relative to the desired directory.

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The *readlinkat*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

The [EACCES] error is removed from the ''may fail'' error conditions.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

**readv**

Purpose: Read a vector.

XSI Synopsis:
```
#include <sys/uio.h>

ssize_t readv(int fildes, const struct iovec *iov,
    int iovcnt);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: Changes are made related to support for finegrained timestamps.

**realloc**

Purpose: Memory reallocator.

Synopsis:
```
#include <stdlib.h>

void *realloc(void *ptr, size_t size);
```

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

Issue 7: No functional changes are made in this issue.

**realpath**

Purpose: Resolve a pathname.

XSI Synopsis:
```
#include <stdlib.h>

char *realpath(const char *restrict file_name,
    char *restrict resolved_name);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

This function is updated for passing a null pointer for the *resolved_name* argument, to request that it allocate memory for the generated pathname, as if by *malloc*( ). If *resolved_name* is not a null pointer and {PATH_MAX} is not defined as a constant in the **<limits.h>** header, the behavior is undefined.

**recv**

Purpose: Receive a message from a connected socket.

Synopsis:
```
#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 Commands and Utilities, Issue 5 (XCU5).

Issue 7: No functional changes are made in this issue.

**recvfrom**

Purpose: Receive a message from a socket.

Synopsis:
```
#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 Commands and Utilities, Issue 5 (XCU5).

Issue 7: No functional changes are made in this issue.

**recvmsg**

Purpose: Receive a message from a socket.

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

ssize_t recvmsg(int socket, struct msghdr *message, int flags);
```

Derivation: First released in Issue 6. Derived from the Commands and Utilities, Issue 5 (XCU5).

Issue 7: No functional changes are made in this issue.

**regcomp, regerror, regexec, regfree**

Purpose: Regular expression matching.

Synopsis:
```
#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 the .

Issue 7: Austin Group Interpretation 1003.1-2001 #134 is applied, clarifying that if more than one error occurs in processing a function call, any one of the possible constants may be returned.

SD5-XBD-ERN-60 is applied, removing the requirement that the type **regoff_t** can hold the largest value that can be stored in type **off_t**, and adding the requirement that the type **regoff_t** can hold the largest value that can be stored in type **ptrdiff_t**.

**remainder, remainderf, remainderl**

Purpose: Remainder function.

Synopsis:
```
#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 7: ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #55 (SD5-XSH-ERN-82) is applied.

**remove**

Purpose: Remove a file.

Synopsis:
```
#include <stdio.h>

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

Derivation: First released in Issue 3. Included for alignment with the IEEE Std 1003.1-1988 (POSIX.1) and the IEEE Std 1003.1i-1995.

Issue 7: No functional changes are made in this issue.

**remquo, remquof, remquol**

Purpose: Remainder functions.

Synopsis:
```
#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.

Issue 7: ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #56 (SD5-XSH-ERN-83) is applied.

**rename, renameat**

Purpose: Rename file relative to directory file descriptor.

Synopsis:
```
#include <stdio.h>

int rename(const char *old, const char *new);
```
CX
```
int renameat(int oldfd, const char *old, int newfd,
    const char *new);
```

The *renameat*() function is equivalent to the *rename*() function except in the case where either *old* or *new* specifies a relative path. If *old* is a relative path, the file to be renamed is located relative to the directory associated with the file descriptor *oldfd* instead of the current working directory. If *new* is a relative path, the same happens only relative to the directory associated with *newfd*. If the file descriptor was opened without O_SEARCH, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with O_SEARCH, the function does not perform the check.

The purpose of the *renameat*() function is to rename files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *rename*(), resulting in unspecified behavior. By opening file descriptors for the source and target directories and using the *renameat*() function it can be guaranteed that that renamed file is located correctly and the resulting file is in the desired directory.

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

Issue 7: Austin Group Interpretation 1003.1-2001 #016 is applied, changing the definition of the [ENOTDIR] error.

Austin Group Interpretation 1003.1-2001 #076 is applied, clarifying the behavior if the final component of a path is either dot or dot-dot, and adding the associated [EINVAL] error case.

Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

Austin Group Interpretation 1003.1-2001 #145 is applied, clarifying that the [ENOENT] error condition also applies to the case in which a component of *new* does not exist.

Austin Group Interpretation 1003.1-2001 #181 is applied, updating the requirements for operations when the S_ISVTX bit is set on a directory.

The *renameat*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

**rewind**

Purpose: Reset the file position indicator in a stream.

Synopsis:
```
#include <stdio.h>

void rewind(FILE *stream);
```

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

Issue 7: No functional changes are made in this issue.

**rewinddir**

Purpose: Reset the position of a directory stream to the beginning of a directory.

Synopsis:
```
#include <dirent.h>

void rewinddir(DIR *dirp);
```

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**rint, rintf, rintl**

Purpose: Round-to-nearest integral value.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**rmdir**

Purpose: Remove a directory.

Synopsis:
```
#include <unistd.h>

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

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

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

Austin Group Interpretation 1003.1-2001 #181 is applied, updating the requirements for operations when the S_ISVTX bit is set.

Changes are made related to support for finegrained timestamps.

**round, roundf, roundl**

Purpose: Round to the nearest integer value in a floating-point format.

Synopsis:
```
#include <math.h>

double round(double x);
float roundf(float x);
long double roundl(long double x);
```

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

Issue 7: No functional changes are made in this issue.

**scalbln, scalblnf, scalblnl, scalbn, scalbnf, scalbnl**

Purpose: Compute exponent using FLT_RADIX.

Synopsis:
```
#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.

Issue 7: No functional changes are made in this issue.

**sched_get_priority_max, sched_get_priority_min**

Purpose: Get priority limits (**REALTIME**).

PS | TPS Synopsis:
```
#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.

Issue 7: No functional changes are made in this issue.

**sched_getparam**

Purpose: Get scheduling parameters (**REALTIME**).

PS Synopsis:

```
#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.

Issue 7: No functional changes are made in this issue.

**sched_getscheduler**

Purpose: Get scheduling policy (**REALTIME**).

PS Synopsis:

```
#include <sched.h>

int sched_getscheduler(pid_t pid);
```

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

Issue 7: No functional changes are made in this issue.

**sched_rr_get_interval**

Purpose: Get execution time limits (**REALTIME**).

PS | TPS Synopsis:

```
#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.

Issue 7: No functional changes are made in this issue.

**sched_setparam**

Purpose: Set scheduling parameters (**REALTIME**).

PS Synopsis:

```
#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.

Issue 7: Austin Group Interpretation 1003.1-2001 #061 is applied, clarifying the effect of process scheduling on the scheduling of threads within the process.

Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements for the *sched_ss_repl_period* and *sched_ss_init_budget* values.

**sched_setscheduler**

Purpose: Set scheduling policy and parameters (**REALTIME**).

PS Synopsis:

```
#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.

Issue 7: Austin Group Interpretation 1003.1-2001 #061 is applied, clarifying the effect of process scheduling on the scheduling of threads within the process.

Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements for the *sched_ss_repl_period* and *sched_ss_init_budget* values.

**sched_yield**

Purpose: Yield the processor.

Synopsis:

```
#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 7: SD5-XSH-ERN-120 is applied, adding APPLICATION USAGE.

The *sched_yield*() function is moved to the Base.

**seekdir**

Purpose: Set the position of a directory stream.

XSI Synopsis:

```
#include <dirent.h>

void seekdir(DIR *dirp, long loc);
```

Derivation: First released in Issue 2.

Issue 7: SD5-XSH-ERN-200 is applied, updating the DESCRIPTION to note that the value of *loc* should have been returned from an earlier call to *telldir*() using the same directory stream.

**sem_close**

Purpose: Close a named semaphore.

Synopsis:

```
#include <semaphore.h>

int sem_close(sem_t *sem);
```

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

Issue 7: The *sem_close*() function is moved from the Semaphores option to the Base.

**sem_destroy**

Purpose: Destroy an unnamed semaphore.

Synopsis:
```
#include <semaphore.h>

int sem_destroy(sem_t *sem);
```

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

Issue 7: The *sem_destroy*() function is moved from the Semaphores option to the Base.

**sem_getvalue**

Purpose: Get the value of a semaphore.

Synopsis:
```
#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.

Issue 7: The *sem_getvalue*() function is moved from the Semaphores option to the Base.

**sem_init**

Purpose: Initialize an unnamed semaphore.

Synopsis:
```
#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.

Issue 7: SD5-XSH-ERN-176 is applied.

The *sem_init*() function is moved from the Semaphores option to the Base.

**sem_open**

Purpose: Initialize and open a named semaphore.

Synopsis:
```
#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.

Issue 7: Austin Group Interpretation 1003.1-2001 #066 is applied, updating the [ENOSPC] error case and adding the [ENOMEM] error case.

Austin Group Interpretation 1003.1-2001 #077 is applied, clarifying the *name* argument and adding [ENAMETOOLONG] as a "may fail" error.

Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

SD5-XSH-ERN-170 is applied, updating the DESCRIPTION to clarify the wording for setting the user ID and group ID of the semaphore.

The *sem_open*() function is moved from the Semaphores option to the Base.

**sem_post**

Purpose: Unlock a semaphore.

Synopsis:
```
#include <semaphore.h>

int sem_post(sem_t *sem);
```

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

Issue 7: The *sem_post*() function is moved from the Semaphores option to the Base.

**sem_timedwait**

Purpose: Lock a semaphore.

Synopsis:
```
#include <semaphore.h>
#include <time.h>

int sem_timedwait(sem_t *restrict sem,
    const struct timespec *restrict abstime);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: The *sem_timedwait*() function is moved from the Semaphores option to the Base.

Functionality relating to the Timers option is moved to the Base.

An example is added.

**sem_trywait, sem_wait**

Purpose: Lock a semaphore.

Synopsis:
```
#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.

Issue 7: SD5-XSH-ERN-54 is applied, removing the *sem_wait*() function from the "shall fail" error cases.

The *sem_trywait*() and *sem_wait*() functions are moved from the Semaphores option to the Base.

**sem_unlink**

Purpose: Remove a named semaphore.

Synopsis:
```
#include <semaphore.h>

int sem_unlink(const char *name);
```

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

Issue 7: Austin Group Interpretation 1003.1-2001 #077 is applied, changing [ENAMETOOLONG] from a "shall fail" to a "may fail" error.

Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

The *sem_unlink*( ) function is moved from the Semaphores option to the Base.

**semctl**

Purpose: XSI semaphore control operations.

XSI Synopsis:

```
#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 7: No functional changes are made in this issue.

**semget**

Purpose: Get set of XSI semaphores.

XSI Synopsis:

```
#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 7: No functional changes are made in this issue.

**semop**

Purpose: XSI semaphore operations.

XSI Synopsis:

```
#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 7: SD5-XSH-ERN-171 is applied, updating the DESCRIPTION to clarify the order in which the operations in *sops* will be performed when there are multiple operations.

**send**

Purpose: Send a message on a socket.

Synopsis:

```
#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 Commands and Utilities, Issue 5 (XCU5).

Issue 7: Austin Group Interpretation 1003.1-2001 #035 is applied, updating the DESCRIPTION to clarify the behavior when the socket is a connectionless-mode socket.

The MSG_NOSIGNAL flag is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

The [EPIPE] error is modified.

**sendmsg**

Purpose: Send a message on a socket using a message structure.

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

ssize_t sendmsg(int socket, const struct msghdr *message,
    int flags);
```

Derivation: First released in Issue 6. Derived from the Commands and Utilities, Issue 5 (XCU5).

Issue 7: Austin Group Interpretation 1003.1-2001 #073 is applied, describing the allowed behaviors when a peer address has been pre-specified.

Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The MSG_NOSIGNAL flag is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

The [EPIPE] error is modified.

**sendto**

Purpose: Send a message on a socket.

Synopsis:
```
#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 Commands and Utilities, Issue 5 (XCU5).

Issue 7: Austin Group Interpretations 1003.1-2001 #035 and #073 are applied, describing the allowed behaviors when a peer address has been pre-specified.

Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The MSG_NOSIGNAL flag is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

The [EPIPE] error is modified.

**setbuf**

Purpose: Assign buffering to a stream.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**setegid**

Purpose: Set the effective group ID.

Synopsis:
```
#include <unistd.h>
int setegid(gid_t gid);
```

Derivation: First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

Issue 7: No functional changes are made in this issue.

**setenv**

Purpose: Add or change environment variable.

CX Synopsis:
```
#include <stdlib.h>
int setenv(const char *envname, const char *envval,
    int overwrite);
```

Derivation: First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

Issue 7: No functional changes are made in this issue.

**seteuid**

Purpose: Set effective user ID.

Synopsis:
```
#include <unistd.h>
int seteuid(uid_t uid);
```

Derivation: First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

Issue 7: No functional changes are made in this issue.

**setgid**

Purpose: Set group ID.

Synopsis:
```
#include <unistd.h>
int setgid(gid_t gid);
```

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

Issue 7: No functional changes are made in this issue.

**setjmp**

Purpose: Set jump point for a non-local goto.

Synopsis:
```
#include <setjmp.h>
int setjmp(jmp_buf env);
```

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

Issue 7: No functional changes are made in this issue.

**setkey**

Purpose: Set encoding key (**CRYPT**).

XSI Synopsis:

```
#include <stdlib.h>
void setkey(const char *key);
```

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

Issue 7: No functional changes are made in this issue.

**setlocale**

Purpose: Set program locale.

Synopsis:

```
#include <locale.h>
char *setlocale(int category, const char *locale);
```

Derivation: First released in Issue 3.

Issue 7: Functionality relating to the Threads option is moved to the Base.

**setpgid**

Purpose: Set process group ID for job control.

Synopsis:

```
#include <unistd.h>
int setpgid(pid_t pid, pid_t pgid);
```

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

Issue 7: No functional changes are made in this issue.

**setpgrp**

Purpose: Set the process group ID.

OB XSI Synopsis:

```
#include <unistd.h>
pid_t setpgrp(void);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: The *setpgrp*() function is marked obsolescent. Applications should use *setpgid*() or *setsid*() as appropriate.

**setregid**

Purpose: Set real and effective group IDs.

XSI Synopsis:

```
#include <unistd.h>
int setregid(gid_t rgid, gid_t egid);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: SD5-XSH-ERN-177 is applied, adding the ability to set both the effective group ID and saved set-group-ID to be the same as the real group ID.

**setreuid**

Purpose: Set real and effective user IDs.

XSI Synopsis:

```
#include <unistd.h>

int setreuid(uid_t ruid, uid_t euid);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: SD5-XSH-ERN-177 is applied, adding the ability to set both the effective user ID and the saved set-user-ID to be the same as the real user ID.

**setsid**

Purpose: Create session and set process group ID.

Synopsis:

```
#include <unistd.h>

pid_t setsid(void);
```

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

Issue 7: No functional changes are made in this issue.

**setsockopt**

Purpose: Set the socket options.

Synopsis:

```
#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 Commands and Utilities, Issue 5 (XCU5).

Issue 7: Austin Group Interpretation 1003.1-2001 #158 is applied, removing text relating to socket options that is now in XSH Section 2.10.16 .

**setuid**

Purpose: Set user ID.

Synopsis:

```
#include <unistd.h>

int setuid(uid_t uid);
```

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

Issue 7: No functional changes are made in this issue.

**setvbuf**

Purpose: Assign buffering to a stream.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**shm_open**

Purpose: Open a shared memory object (**REALTIME**).

SHM Synopsis:
```
#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.

Issue 7: Austin Group Interpretation 1003.1-2001 #077 is applied, clarifying the *name* argument and changing [ENAMETOOLONG] from a "shall fail" to a "may fail" error.

Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

SD5-XSH-ERN-170 is applied, updating the DESCRIPTION to clarify the wording for setting the user ID and group ID of the shared memory object.

**shm_unlink**

Purpose: Remove a shared memory object (**REALTIME**).

SHM Synopsis:
```
#include <sys/mman.h>

int shm_unlink(const char *name);
```

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

Issue 7: Austin Group Interpretation 1003.1-2001 #077 is applied, changing [ENAMETOOLONG] from a "shall fail" to a "may fail" error.

Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

**shmat**

Purpose: XSI shared memory attach operation.

XSI Synopsis:
```
#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 7: No functional changes are made in this issue.

**shmctl**

Purpose: XSI shared memory control operations.

XSI Synopsis:
```
#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 7: No functional changes are made in this issue.

**shmdt**

Purpose: XSI shared memory detach operation.

XSI Synopsis:
```
#include <sys/shm.h>
int shmdt(const void *shmaddr);
```

Derivation: First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 7: No functional changes are made in this issue.

**shmget**

Purpose: Get an XSI shared memory segment.

XSI Synopsis:
```
#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 7: No functional changes are made in this issue.

**shutdown**

Purpose: Shut down socket send and receive operations.

Synopsis:
```
#include <sys/socket.h>
int shutdown(int socket, int how);
```

Derivation: First released in Issue 6. Derived from the Commands and Utilities, Issue 5 (XCU5).

Issue 7: No functional changes are made in this issue.

**sigaction**

Purpose: Examine and change a signal action.

CX Synopsis:

```
#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 the IEEE Std 1003.1-1988 (POSIX.1).

Issue 7: Austin Group Interpretation 1003.1-2001 #004 is applied, clarifying that the *sigaction*( ) function may fail if the SA_SIGINFO flag is set in the *sa_flags* field of the **sigaction** structure for a signal not in the range SIGRTMIN to SIGRTMAX.

Austin Group Interpretations 1003.1-2001 #065 and #084 are applied, clarifying the role of the SA_NODEFER flag with respect to the signal mask, and clarifying the SA_RESTART flag for interrupted functions which use timeouts.

SD5-XSH-ERN-167 is applied, updating the APPLICATION USAGE section to explain that unless all signal handlers have *errno* set on return as it was on entry, the value of *errno* is unspecified.

SD5-XSH-ERN-172 is applied, updating the DESCRIPTION to make optional the requirement that when the SA_RESETHAND flag is set, *sigaction*( ) shall behave as if the SA_NODEFER flag were also set.

Functionality relating to the Realtime Signals Extension option is moved to the Base.

The description of the *si_code* member is replaced with a reference to XSH Section 2.4.3 .

**sigaddset**

Purpose: Add a signal to a signal set.

CX Synopsis:

```
#include <signal.h>

int sigaddset(sigset_t *set, int signo);
```

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

Issue 7: No functional changes are made in this issue.

**sigaltstack**

Purpose: Set and get signal alternate stack context.

XSI Synopsis:

```
#include <signal.h>

int sigaltstack(const stack_t *restrict ss,
    stack_t *restrict oss);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

**sigdelset**

Purpose: Delete a signal from a signal set.

CX Synopsis:

```
#include <signal.h>

int sigdelset(sigset_t *set, int signo);
```

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

Issue 7: No functional changes are made in this issue.

**sigemptyset**

Purpose: Initialize and empty a signal set.

CX Synopsis:

```
#include <signal.h>

int sigemptyset(sigset_t *set);
```

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

Issue 7: No functional changes are made in this issue.

**sigfillset**

Purpose: Initialize and fill a signal set.

CX Synopsis:

```
#include <signal.h>

int sigfillset(sigset_t *set);
```

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

Issue 7: No functional changes are made in this issue.

**sighold, sigignore, sigpause, sigrelse, sigset**

Purpose: Signal management.

OB XSI Synopsis:

```
#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 7: These functions are marked obsolescent. Applications should use the *sigaction*() function instead of the *sigset*() function, the *pthread_sigmask*() or *sigprocmask*() functions instead of the *sighold*() and *sigrelse*() functions, and the *sigsuspend*() function instead of the *sigpause*() function.

**siginterrupt**

Purpose: Allow signals to interrupt functions.

OB XSI Synopsis:

```
#include <signal.h>
int siginterrupt(int sig, int flag);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: The *siginterrupt*() function is marked obsolescent. Applications should use *sigaction*() with the SA_RESTART flag instead.

**sigismember**

Purpose: Test for a signal in a signal set.

CX Synopsis:

```
#include <signal.h>
int sigismember(const sigset_t *set, int signo);
```

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

Issue 7: No functional changes are made in this issue.

**siglongjmp**

Purpose: Non-local goto with signal handling.

CX Synopsis:

```
#include <setjmp.h>
void siglongjmp(sigjmp_buf env, int val);
```

Derivation: First released in Issue 3. Included for alignment with the .

Issue 7: No functional changes are made in this issue.

**signal**

Purpose: Signal management.

Synopsis:

```
#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 7: No functional changes are made in this issue.

**signbit**

Purpose: Test sign.

Synopsis:

```
#include <math.h>
int signbit(real-floating x);
```

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

Issue 7: No functional changes are made in this issue.

**sigpending**

Purpose: Examine pending signals.

CX Synopsis:
```
#include <signal.h>
int sigpending(sigset_t *set);
```

Derivation: First released in Issue 3.

Issue 7: No functional changes are made in this issue.

**sigqueue**

Purpose: Queue a signal to a process.

CX Synopsis:
```
#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 7: The *sigqueue*( ) function is moved from the Realtime Signals Extension option to the Base.

**sigsetjmp**

Purpose: Set jump point for a non-local goto.

CX Synopsis:
```
#include <setjmp.h>
int sigsetjmp(sigjmp_buf env, int savemask);
```

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

Issue 7: No functional changes are made in this issue.

**sigsuspend**

Purpose: Wait for a signal.

CX Synopsis:
```
#include <signal.h>
int sigsuspend(const sigset_t *sigmask);
```

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

Issue 7: SD5-XSH-ERN-122 is applied, adding the example code in the RATIONALE.

**sigtimedwait, sigwaitinfo**

Purpose: Wait for queued signals.

CX Synopsis:

```
#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 7: The *sigtimedwait*( ) and *sigwaitinfo*( ) functions are moved from the Realtime Signals Extension option to the Base.

**sigwait**

Purpose: Wait for queued signals.

CX Synopsis:

```
#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 7: Functionality relating to the Realtime Signals Extension option is moved to the Base.

**sin, sinf, sinl**

Purpose: Sine function.

Synopsis:

```
#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 7: No functional changes are made in this issue.

**sinh, sinhf, sinhl**

Purpose: Hyperbolic sine functions.

Synopsis:

```
#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 7: No functional changes are made in this issue.

**sleep**

Purpose: Suspend execution for an interval of time.

Synopsis:
```
#include <unistd.h>

unsigned sleep(unsigned seconds);
```

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

Issue 7: No functional changes are made in this issue.

**sockatmark**

Purpose: Determine whether a socket is at the out-of-band mark.

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

int sockatmark(int s);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1g-2000.

Issue 7: SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

**socket**

Purpose: Create an endpoint for communication.

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

int socket(int domain, int type, int protocol);
```

Derivation: First released in Issue 6. Derived from the Commands and Utilities, Issue 5 (XCU5).

Issue 7: No functional changes are made in this issue.

**socketpair**

Purpose: Create a pair of connected sockets.

Synopsis:
```
#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 Commands and Utilities, Issue 5 (XCU5).

Issue 7: The description of the [EMFILE] error condition is aligned with the *pipe*( ) function.

**sqrt, sqrtf, sqrtl**

Purpose: Square root function.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**stderr, stdin, stdout**

Purpose: Standard I/O streams.

Synopsis:
```
#include <stdio.h>

extern FILE *stderr, *stdin, *stdout;
```

Derivation: First released in Issue 1.

Issue 7: No functional changes are made in this issue.

**strcasecmp, strcasecmp_l, strncasecmp, strncasecmp_l**

Purpose: Case-insensitive string comparisons.

Synopsis:
```
#include <strings.h>

int strcasecmp(const char *s1, const char *s2);
int strcasecmp_l(const char *s1, const char *s2,
    locale_t locale);
int strncasecmp(const char *s1, const char *s2, size_t n);
int strncasecmp_l(const char *s1, const char *s2,
    size_t n, locale_t locale);
```

The *strcasecmp_l*() function compares, while ignoring differences in case, the string pointed to by *s1* to the string pointed to by *s2*. The *strncasecmp_l*() function compares, while ignoring differences in case, not more than *n* bytes from the string pointed to by *s1* to the string pointed to by *s2*.

These functions use the locale represented by *locale* to determine the case of the characters. A handle for use as *locale* can be obtained using *newlocale*() or *duplocale*().

Derivation: First released in Issue 4, Version 2.

Issue 7: The *strcasecmp*() and *strncasecmp*() functions are moved from the XSI option to the Base.

The *strcasecmp_l*() and *strncasecmp_l*() functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**strcat**

Purpose: Concatenate two strings.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**strchr**

Purpose: String scanning operation.

Synopsis:
```
#include <string.h>

char *strchr(const char *s, int c);
```

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

Issue 7: No functional changes are made in this issue.

**strcmp**

Purpose: Compare two strings.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**strcoll, strcoll_l**

Purpose: String comparison using collating information.

Synopsis:
```
#include <string.h>

int strcoll(const char *s1, const char *s2);
```
CX
```
int strcoll_l(const char *s1, const char *s2,
    locale_t locale);
```

The *strcoll_l*( ) function compares the string pointed to by *s1* to the string pointed to by *s2*, both interpreted as appropriate to the *LC_COLLATE* category of the locale represented by *locale*.

A handle for use as *locale* can be obtained using *newlocale*( ) or *duplocale*( ).

Derivation: First released in Issue 3.

Issue 7: The *strcoll_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**stpcpy, strcpy**

Purpose: Copy a string and return a pointer to the end of the result.

Synopsis:
```
#include <string.h>
```
CX
```
char *stpcpy(char *restrict s1, const char *restrict s2);
char *strcpy(char *restrict s1, const char *restrict s2);
```

The *stpcpy*( ) function is equivalent to the *strcpy*( ) function, except that it returns a pointer to the terminating NUL character copied into the *s1* buffer.

The following example constructs a multi-part message in a single buffer:

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

int
main (void)
```

```
{
    char buffer [10];
    char *name = buffer;

    name = stpcpy (stpcpy (stpcpy (name, "ice"),"-"), "cream");
    puts (buffer);
    return 0;
}
```

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

Issue 7: The *stpcpy*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

**strcspn**

Purpose: Get the length of a complementary substring.

Synopsis:

```
#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 7: No functional changes are made in this issue.

**strdup, strndup**

Purpose: Duplicate a specific number of bytes from a string.

CX Synopsis:

```
#include <string.h>

char *strdup(const char *s);
char *strndup(const char *s, size_t size);
```

The *strndup*() function is equivalent to the *strdup*() function, duplicating the provided *s* in a new block of memory allocated as if by using *malloc*(), with the exception being that *strndup*() copies at most *size* plus one bytes into the newly allocated memory, terminating the new string with a NUL character. If the length of *s* is larger than *size*, only *size* bytes are duplicated. If *size* is larger than the length of *s*, all bytes in *s* are copied into the new memory buffer, including the terminating NUL character. The newly created string is always properly terminated.

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #044 is applied, changing the "may fail" [ENOMEM] error to become a "shall fail" error.

The *strdup*() function is moved from the XSI option to the Base.

The *strndup*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

The APPLICATION USAGE section is updated to clarify that memory is allocated as if by *malloc*().

**strerror, strerror_l, strerror_r**

Purpose: Get error message string.

Synopsis:
```
#include <string.h>

char *strerror(int errnum);
CX char *strerror_l(int errnum, locale_t locale);
int strerror_r(int errnum, char *strerrbuf, size_t buflen);
```

Derivation: First released in Issue 3.

Issue 7: Austin Group Interpretation 1003.1-2001 #187 is applied, clarifying the behavior when the generated error message is an empty string.

SD5-XSH-ERN-191 is applied, disallowing *perror*() from overwriting the string returned by *strerror*(), for alignment with the C Standard.

The *strerror_l*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

The *strerror_r*() function is moved from the Thread-Safe Functions option to the Base.

**strfmon, strfmon_l**

Purpose: Convert monetary value to a string.

Synopsis:
```
#include <monetary.h>

ssize_t strfmon(char *restrict s, size_t maxsize,
    const char *restrict format, ...);
ssize_t strfmon_l(char *restrict s, size_t maxsize,
    locale_t locale, const char *restrict format, ...);
```

The *strfmon_l*() function is equivalent to the *strfmon*() function, except that the locale data used is from the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*() or *duplocale*().

Derivation: First released in Issue 4.

Issue 7: SD5-XSH-ERN-29 is applied, updating the examples for `%(#5n` and `%!(#5n`.

SD5-XSH-ERN-233 is applied, changing the definition of the `'+'` or `'('` flags to refer to multiple locales.

The *strfmon*() function is moved from the XSI option to the Base.

The *strfmon_l*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**strftime, strftime_l**

Purpose: Convert date and time to a string.

Synopsis:
```
#include <time.h>

size_t strftime(char *restrict s, size_t maxsize,
    const char *restrict format,
    const struct tm *restrict timeptr);
CX size_t strftime_l(char *restrict s, size_t maxsize,
    const char *restrict format,
```

```
    const struct tm *restrict timeptr, locale_t locale);
```

Derivation: First released in Issue 3.

Issue 7: Austin Group Interpretation 1003.1-2001 #163 is applied, making extensive changes to the required behavior of the *strftime*() function, including the addition of flags and field widths in conversion specifications.

The *strftime_l*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**strlen, strnlen**

Purpose: Get length of fixed size string.

Synopsis:

```
#include <string.h>

size_t strlen(const char *s);
size_t strnlen(const char *s, size_t maxlen);
```

CX

The *strnlen*() function computes the smaller of the number of bytes in the array to which *s* points, not including the terminating NUL character, or the value of the *maxlen* argument. The *strnlen*() function never examines more than *maxlen* bytes of the array pointed to by *s*.

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

Issue 7: The *strnlen*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

**strncat**

Purpose: Concatenate a string with part of another.

Synopsis:

```
#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 7: No functional changes are made in this issue.

**strncmp**

Purpose: Compare part of two strings.

Synopsis:

```
#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 7: No functional changes are made in this issue.

**stpncpy, strncpy**

Purpose: Copy fixed length string, returning a pointer to the array end.

Synopsis:

```
#include <string.h>
```

CX

```
char *stpncpy(char *restrict s1, const char *restrict s2,
    size_t n);
char *strncpy(char *restrict s1, const char *restrict s2,
    size_t n);
```

The *stpncpy*() function is equivalent to the *strncpy*() function, except for the return value. If a NUL character is written to the destination, the *stpncpy*() function returns the address of the first such NUL character. Otherwise, it returns *&s1*[*n*].

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

Issue 7: The *stpncpy*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

**strpbrk**

Purpose: Scan a string for a byte.

Synopsis:

```
#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 7: No functional changes are made in this issue.

**strptime**

Purpose: Date and time conversion.

XSI Synopsis:

```
#include <time.h>

char *strptime(const char *restrict buf,
    const char *restrict format,
    struct tm *restrict tm);
```

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretations 1003.1-2001 #041 and #163 are applied, making extensive changes to the required behavior of the *strptime*() function, including the addition of flags and field widths in conversion specifications.

SD5-XSH-ERN-67 is applied, correcting the APPLICATION USAGE to remove the impression that `%Y` is 4-digit years.

**strrchr**

Purpose: String scanning operation.

Synopsis:

```
#include <string.h>

char *strrchr(const char *s, int c);
```

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

Issue 7: No functional changes are made in this issue.

**strsignal**

Purpose: Get name of signal.

CX Synopsis:

```
#include <string.h>

char *strsignal(int signum);
```

The *strsignal*() function maps the signal number in *signum* to an implementation-defined string and returns a pointer to it. It uses the same set of messages as the *psignal*() function.

Application writers should note that if *signum* is not a valid signal number, some implementations return NULL, while for others the *strsignal*() function returns a pointer to a string containing an unspecified message denoting an unknown signal. IEEE Std 1003.1-2001 leaves this return value unspecified.

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Issue 7: First released in Issue 7.

**strspn**

Purpose: Get length of a substring.

Synopsis:

```
#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 7: No functional changes are made in this issue.

**strstr**

Purpose: Find a substring.

Synopsis:

```
#include <string.h>

char *strstr(const char *s1, const char *s2);
```

Derivation: First released in Issue 3. Included for alignment with the IEEE Std 1003.1b-1993.

Issue 7: No functional changes are made in this issue.

**strtod, strtof, strtold**

Purpose: Convert a string to a double-precision number.

Synopsis:

```
#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 7: No functional changes are made in this issue.

**strtoimax, strtoumax**

Purpose: Convert string to integer type.

Synopsis:

```
#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);
```

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

Issue 7: No functional changes are made in this issue.

**strtok, strtok_r**

Purpose: Split string into tokens.

Synopsis:

```
#include <string.h>

char *strtok(char *restrict s1, const char *restrict s2);
```

CX

```
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 7: SD5-XSH-ERN-235 is applied, correcting an example.

The *strtok_r*() function is moved from the Thread-Safe Functions option to the Base.

**strtol, strtoll**

Purpose: Convert a string to a long integer.

Synopsis:

```
#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 7: No functional changes are made in this issue.

**strtoul, strtoull**

Purpose: Convert a string to an unsigned long.

Synopsis:

```
#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 the IEEE Std 1003.1b-1993.

Issue 7: No functional changes are made in this issue.

**strxfrm, strxfrm_l**

Purpose: String transformation.

Synopsis:
```
#include <string.h>

size_t strxfrm(char *restrict s1, const char *restrict s2,
    size_t n);
CX size_t strxfrm_l(char *restrict s1, const char *restrict s2,
    size_t n, locale_t locale);
```

The *strxfrm_l*() function is equivalent to the *strxfrm*() function, except that the locale data used is from the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*() or *duplocale*().

Derivation: First released in Issue 3. Included for alignment with the IEEE Std 1003.1i-1995.

Issue 7: The *strxfrm_l*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**swab**

Purpose: Swap bytes.

XSI Synopsis:
```
#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 7: No functional changes are made in this issue.

**symlink, symlinkat**

Purpose: Make a symbolic link relative to directory file descriptor.

Synopsis:
```
#include <unistd.h>

int symlink(const char *path1, const char *path2);
int symlinkat(const char *path1, int fd, const char *path2);
```

The *symlinkat*() function is equivalent to the *symlink*() function except in the case where *path2* specifies a relative path. In this case the symbolic link is created relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the file descriptor was opened without O_SEARCH, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with O_SEARCH, the function does not perform the check.

The purpose of the *symlinkat*() function is to create symbolic links in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *symlink*(), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *symlinkat*() function it can be guaranteed that the created symbolic link is

located relative to the desired directory.

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The *symlinkat*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Additions have been made describing how *symlink*() sets the user and group IDs and file mode of the symbolic link, and its effect on timestamps.

**sync**

Purpose: Schedule file system updates.

XSI Synopsis:

```
#include <unistd.h>

void sync(void);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

**sysconf**

Purpose: Get configurable system variables.

Synopsis:

```
#include <unistd.h>

long sysconf(int name);
```

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

Issue 7: Austin Group Interpretation 1003.1-2001 #160 is applied, clarifying the requirements related to variables that have no limit.

SD5-XSH-ERN-166 is applied, changing ''Maximum size'' to ''Initial size'' for the ''Maximum size of ...'' entries in the table in the DESCRIPTION.

The variables for the supported programming environments are updated to be V7 and the LEGACY variables are removed.

The following constants are added:

_POSIX_THREAD_ROBUST_PRIO_INHERIT
_POSIX_THREAD_ROBUST_PRIO_PROTECT

The _XOPEN_UUCP variable and its associated _SC_XOPEN_UUCP value is added to the table of system variables.

**system**

Purpose: Issue a command.

Synopsis:
```
#include <stdlib.h>

int system(const char *command);
```

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

Issue 7: Austin Group Interpretation 1003.1-2001 #055 is applied, clarifying the thread-safety of this function and treatment of *pthread_atfork*( ) handlers.

**tan, tanf, tanl**

Purpose: Tangent function.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**tanh, tanhf, tanhl**

Purpose: Hyperbolic tangent functions.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**tcdrain**

Purpose: Wait for transmission of output.

Synopsis:
```
#include <termios.h>

int tcdrain(int fildes);
```

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

Issue 7: No functional changes are made in this issue.

**tcflow**

Purpose: Suspend or restart the transmission or reception of data.

Synopsis:
```
#include <termios.h>

int tcflow(int fildes, int action);
```

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

Issue 7: SD5-XSH-ERN-190 is applied, clarifying in the DESCRIPTION the transmission of START and STOP characters.

**tcflush**

Purpose: Flush non-transmitted output data, non-read input data, or both.

Synopsis:
```
#include <termios.h>

int tcflush(int fildes, int queue_selector);
```

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

Issue 7: No functional changes are made in this issue.

**tcgetattr**

Purpose: Get the parameters associated with the terminal.

Synopsis:
```
#include <termios.h>

int tcgetattr(int fildes, struct termios *termios_p);
```

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

Issue 7: No functional changes are made in this issue.

**tcgetpgrp**

Purpose: Get the foreground process group ID.

Synopsis:
```
#include <unistd.h>

pid_t tcgetpgrp(int fildes);
```

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

Issue 7: No functional changes are made in this issue.

**tcgetsid**

Purpose: Get the process group ID for the session leader for the controlling terminal.

Synopsis:
```
#include <termios.h>

pid_t tcgetsid(int fildes);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: The *tcgetsid*( ) function is moved from the XSI option to the Base.

**tcsendbreak**

Purpose: Send a break for a specific duration.

Synopsis:
```
#include <termios.h>

int tcsendbreak(int fildes, int duration);
```

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

Issue 7: No functional changes are made in this issue.

**tcsetattr**

Purpose: Set the parameters associated with the terminal.

Synopsis:
```
#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 the IEEE Std 1003.1-1988 (POSIX.1).

Issue 7: Austin Group Interpretation 1003.1-2001 #144 is applied, adding requirements related to the new O_TTY_INIT flag.

**tcsetpgrp**

Purpose: Set the foreground process group ID.

Synopsis:
```
#include <unistd.h>

int tcsetpgrp(int fildes, pid_t pgid_id);
```

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

Issue 7: No functional changes are made in this issue.

**tdelete, tfind, tsearch, twalk**

Purpose: Manage a binary search tree.

XSI Synopsis:
```
#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 7: Austin Group Interpretation 1003.1-2001 #149 is applied, clarifying concurrent use of the tree in another thread.

Austin Group Interpretation 1003.1-2001 #151 is applied, clarifying behavior for *tdelete*( ) when the deleted node is the root node.

Austin Group Interpretation 1003.1-2001 #153 is applied, clarifying that if the functions pointed to by *action* or *compar* change the tree, the results are undefined.

**telldir**

Purpose: Current location of a named directory stream.

XSI Synopsis:
```
#include <dirent.h>

long telldir(DIR *dirp);
```

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**tempnam**

Purpose: Create a name for a temporary file.

OB XSI Synopsis:
```
#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 7: The *tempnam*( ) function is marked obsolescent. Applications should use the *tmpfile*( ), *mkdtemp*( ), or *mkstemp*( ) functions instead.

**tgamma, tgammaf, tgammal**

Purpose: Compute *gamma*( ) function.

Synopsis:
```
#include <math.h>

double tgamma(double x);
float tgammaf(float x);
long double tgammal(long double x);
```

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

Issue 7: ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #52 (SD5-XSH-ERN-85) is applied.

**time**

Purpose: Get time.

Synopsis:
```
#include <time.h>

time_t time(time_t *tloc);
```

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

Issue 7: No functional changes are made in this issue.

**timer_create**

Purpose: Create a per-process timer.

CX Synopsis:
```
#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.

Issue 7: The *timer_create*() function is moved from the Timers option to the Base.

**timer_delete**

Purpose: Delete a per-process timer.

CX Synopsis:
```
#include <time.h>

int timer_delete(timer_t timerid);
```

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

Issue 7: The *timer_delete*() function is moved from the Timers option to the Base.

**timer_getoverrun, timer_gettime, timer_settime**

Purpose: Per-process timers.

CX Synopsis:
```
#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.

Issue 7: The *timer_getoverrun*(), *timer_gettime*(), and *timer_settime*() functions are moved from the Timers option to the Base.

Functionality relating to the Realtime Signals Extension option is moved to the Base.

**times**

Purpose: Get process and waited-for child process times.

Synopsis:
```
#include <sys/times.h>

clock_t times(struct tms *buffer);
```

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

Issue 7: No functional changes are made in this issue.

**tmpfile**

Purpose: Create a temporary file.

Synopsis:
```
#include <stdio.h>

FILE *tmpfile(void);
```

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

Issue 7: Austin Group Interpretation 1003.1-2001 #025 is applied, clarifying that implementations may restrict the permissions of the file created.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XSH-ERN-149 is applied, adding the mandatory [EMFILE] error condition for {STREAM_MAX} streams open.

**tmpnam**

Purpose: Create a name for a temporary file.

OB Synopsis:
```
#include <stdio.h>

char *tmpnam(char *s);
```

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

Issue 7: Austin Group Interpretation 1003.1-2001 #148 is applied, clarifying that the *tmpnam*( ) function need not be thread-safe if called with a NULL parameter.

The *tmpnam*( ) function is marked obsolescent. Applications should use the *tmpfile*( ), *mkdtemp*( ), or *mkstemp*( ) functions instead.

**toascii**

Purpose: Translate an integer to a 7-bit ASCII character.

OB XSI Synopsis:
```
#include <ctype.h>

int toascii(int c);
```

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

Issue 7: The *toascii*( ) function is marked obsolescent.

**tolower, tolower_l**

Purpose: Transliterate uppercase characters to lowercase.

Synopsis:
```
#include <ctype.h>

int tolower(int c);
```
CX
```
int tolower_l(int c, locale_t locale);
```

The *tolower_l*( ) function is equivalent to the *tolower*( ) function, except that the locale data used is from the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*( ) or *duplocale*( ).

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

Issue 7: The *tolower_l*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**toupper, toupper_l**

Purpose: Transliterate lowercase characters to uppercase.

Synopsis:

```
#include <ctype.h>

int toupper(int c);
CX int toupper_l(int c, locale_t locale);
```

The *toupper_l*() function is equivalent to the *toupper*() function, except that the locale data used is from the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*() or *duplocale*().

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

Issue 7: The *toupper_l*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**towctrans, towctrans_l**

Purpose: Wide-character transliteration.

Synopsis:

```
#include <wctype.h>

wint_t towctrans(wint_t wc, wctrans_t desc);
CX wint_t towctrans_l(wint_t wc, wctrans_t desc,
       locale_t locale);
```

The *towctrans_l*() function is equivalent to the *towctrans*() function, except that the locale data used is from the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*() or *duplocale*().

Derivation: First released in Issue 5. Derived from .

Issue 7: The *towctrans_l*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**towlower, towlower_l**

Purpose: Transliterate uppercase wide-character code to lowercase.

Synopsis:

```
#include <wctype.h>

wint_t towlower(wint_t wc);
CX wint_t towlower_l(wint_t wc, locale_t locale);
```

The *towlower_l*() function is equivalent to the *towlower*() function, except that the locale data used is from the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*() or *duplocale*().

Derivation: First released in Issue 4.

Issue 7: The *towlower_l*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**towupper, towupper_l**

Purpose: Transliterate lowercase wide-character code to uppercase.

Synopsis:
```
#include <wctype.h>

wint_t towupper(wint_t wc);
CX wint_t towupper_l(wint_t wc, locale_t locale);
```

The *towupper_l*() function is equivalent to the *towupper*() function, except that the locale data used is from the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*() or *duplocale*().

Derivation: First released in Issue 4.

Issue 7: The *towupper_l*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**trunc, truncf, truncl**

Purpose: Round to truncated integer value.

Synopsis:
```
#include <math.h>

double trunc(double x);
float truncf(float x);
long double truncl(long double x);
```

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

Issue 7: No functional changes are made in this issue.

**truncate**

Purpose: Truncate a file to a specified length.

Synopsis:
```
#include <unistd.h>

int truncate(const char *path, off_t length);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The *truncate*() function is moved from the XSI option to the Base.

Changes are made related to support for finegrained timestamps.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

**ttyname, ttyname_r**

Purpose: Find the pathname of a terminal.

Synopsis:
```
#include <unistd.h>

char *ttyname(int fildes);
int ttyname_r(int fildes, char *name, size_t namesize);
```

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

Issue 7: SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

The *ttyname_r*() function is moved from the Thread-Safe Functions option to the Base.

**daylight, timezone, tzname, tzset**

Purpose: Set timezone conversion information.

Synopsis:
```
#include <time.h>

extern int daylight;
extern long timezone;
extern char *tzname[2];
void tzset(void);
```

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

Issue 7: No functional changes are made in this issue.

**ulimit**

Purpose: Get and set process limits.

Synopsis:
```
#include <ulimit.h>

long ulimit(int cmd, ...);
```

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

Issue 7: The *ulimit*() function is marked obsolescent. Applications should use the *getrlimit*() or *setrlimit*() functions instead.

**umask**

Purpose: Set and get the file mode creation mask.

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

mode_t umask(mode_t cmask);
```

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

Issue 7: No functional changes are made in this issue.

**uname**

Purpose: Get the name of the current system.

Synopsis:
```
#include <sys/utsname.h>

int uname(struct utsname *name);
```

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

Issue 7: No functional changes are made in this issue.

**ungetc**

Purpose: Push byte back into input stream.

Synopsis:
```
#include <stdio.h>

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

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

Issue 7: No functional changes are made in this issue.

**ungetwc**

Purpose: Push wide-character code back into the input stream.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**unlink, unlinkat**

Purpose: Remove a directory entry relative to directory file descriptor.

Synopsis:
```
#include <unistd.h>

int unlink(const char *path);
int unlinkat(int fd, const char *path, int flag);
```

The *unlinkat*() function is equivalent to the *unlink*() or *rmdir*() function except in the case where *path* specifies a relative path. In this case the directory entry to be removed is determined relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the file descriptor was opened without O_SEARCH, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with O_SEARCH, the function does not perform the check.

The AT_REMOVEDIR flag controls whether *unlinkat*() behaves like *unlink*() or *rmdir*(): if AT_REMOVEDIR is set, the directory entry specified by *fd* and *path* is removed as a directory.

The purpose of the *unlinkat*() function is to remove directory entries in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *unlink*(), resulting in unspecified behavior. By opening a file descriptor for the target directory and

using the *unlinkat*() function it can be guaranteed that the removed directory entry is located relative to the desired directory.

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

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

Austin Group Interpretation 1003.1-2001 #181 is applied, updating the requirements for operations when the S_ISVTX bit is set on a directory.

Text arising from the LSB Conflicts TR is added to the RATIONALE about the use of [EPERM] and [EISDIR].

The *unlinkat*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

**unlockpt**

Purpose: Unlock a pseudo-terminal master/slave pair.

XSI Synopsis:

```
#include <stdlib.h>
int unlockpt(int fildes);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

**unsetenv**

Purpose: Remove an environment variable.

CX Synopsis:

```
#include <stdlib.h>
int unsetenv(const char *name);
```

Derivation: First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

Issue 7: No functional changes are made in this issue.

**uselocale**

Purpose: Use locale in current thread.

CX Synopsis:

```
#include <locale.h>
locale_t uselocale(locale_t newloc);
```

The *uselocale*() function sets the current locale for the current thread to the locale represented by *newloc*.

Application writers should note that unlike the *setlocale*() function, the *uselocale*() function does not allow replacing some locale categories only. Applications that need to install a locale which differs only in a few categories must use *newlocale*()

to change a locale object equivalent to the currently used locale and install it.

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 4.

Issue 7: First released in Issue 7.

**utime**

Purpose: Set file access and modification times.

OB Synopsis:

```
#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 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The *utime*() function is marked obsolescent. Applications should use the *utimensat*() function instead.

Changes are made related to support for finegrained timestamps.

**vdprintf, vfprintf, vprintf, vsnprintf, vsprintf**

Purpose: Format output of a stdarg argument list.

Synopsis:

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

CX

```
int vdprintf(int fildes, const char *restrict format,
    va_list ap);
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);
```

The *vdprintf*() function is equivalent to the *vfprintf*() function, except that *vdprintf*() writes output to the file associated with the file descriptor specified by the *fildes* argument rather than placing output on a stream.

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

Issue 7: The *vdprintf*() function is added to complement the *dprintf*() function from The Open Group Technical Standard, 2006, Extended API Set Part 1.

**vfscanf, vscanf, vsscanf**

Purpose: Format input of a stdarg argument list.

Synopsis:
```
#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);
```

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

Issue 7: No functional changes are made in this issue.

**vfwprintf, vswprintf, vwprintf**

Purpose: Wide-character formatted output of a stdarg argument list.

Synopsis:
```
#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 .

Issue 7: No functional changes are made in this issue.

**vfwscanf, vswscanf, vwscanf**

Purpose: Wide-character formatted input of a stdarg argument list.

Synopsis:
```
#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);
```

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

Issue 7: No functional changes are made in this issue.

**wait, waitpid**

Purpose: Wait for a child process to stop or terminate.

Synopsis:
```
#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 7: APPLICATION USAGE is added, recommending that the *wait*() function not be used and that the *waitpid*() function not be used with a *pid* argument of –1.

An additional example for *waitpid*() is added.

**waitid**

Purpose: Wait for a child process to change state.

Synopsis:
```
#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 7: Austin Group Interpretation 1003.1-2001 #060 is applied, updating the DESCRIPTION to require that applications set at least one of the flags WEXITED, WSTOPPED or WCONTINUED in the *options* argument.

The *waitid*() function is moved from the XSI option to the Base.

APPLICATION USAGE is added, recommending that the *waitid*() function not be used with *idtype* equal to P_ALL.

The description of the WNOHANG flag is updated to match the one on the **<sys/wait.h>** page.

**wcrtomb**

Purpose: Convert a wide-character code to a character (restartable).

Synopsis:
```
#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 .

Issue 7: Austin Group Interpretation 1003.1-2001 #148 is applied, clarifying that the *wcrtomb*() function need not be thread-safe if called with a NULL *ps* argument.

Austin Group Interpretation 1003.1-2001 #170 is applied, changing the [EILSEQ] error condition from a "may fail" to a "shall fail".

**wcscasecmp, wcscasecmp_l, wcsncasecmp, wcsncasecmp_l**

Purpose: Case-insensitive wide-character string comparison.

CX Synopsis:

```
#include <wchar.h>

int wcscasecmp(const wchar_t *ws1, const wchar_t *ws2);
int wcscasecmp_l(const wchar_t *ws1, const wchar_t *ws2,
    locale_t locale);
int wcsncasecmp(const wchar_t *ws1, const wchar_t *ws2,
    size_t n);
int wcsncasecmp_l(const wchar_t *ws1, const wchar_t *ws2,
    size_t n, locale_t locale);
```

The *wcscasecmp*( ) and *wcsncasecmp*( ) functions are the wide-character equivalent of the *strcasecmp*( ) and *strncasecmp*( ) functions, respectively.

The *wcscasecmp*( ) and *wcscasecmp_l*( ) functions compare, while ignoring differences in case, the wide-character string pointed to by *ws1* to the wide-character string pointed to by *ws2*.

The *wcsncasecmp*( ) and *wcsncasecmp_l*( ) functions compare, while ignoring differences in case, not more than *n* wide-characters from the wide-character string pointed to by *ws1* to the wide-character string pointed to by *ws2*.

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Issue 7: First released in Issue 7.

**wcscat**

Purpose: Concatenate two wide-character strings.

Synopsis:

```
#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 7: No functional changes are made in this issue.

**wcschr**

Purpose: Wide-character string scanning operation.

Synopsis:

```
#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 7: No functional changes are made in this issue.

**wcscmp**

Purpose: Compare two wide-character strings.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**wcscoll, wcscoll_l**

Purpose: Wide-character string comparison using collating information.

Synopsis:
```
#include <wchar.h>

int wcscoll(const wchar_t *ws1, const wchar_t *ws2);
CX int wcscoll_l(const wchar_t *ws1, const wchar_t *ws2,
        locale_t locale);
```

The *wcscoll_l*() function compares the wide-character string pointed to by *ws1* to the wide-character string pointed to by *ws2,* both interpreted as appropriate to the *LC_COLLATE* category of the locale represented by *locale*.

A handle for use as *locale* can be obtained using *newlocale*() or *duplocale*().

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

Issue 7: The *wcscoll_l*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**wcpcpy, wcscpy**

Purpose: Copy a wide-character string, returning a pointer to its end.

Synopsis:
```
#include <wchar.h>

CX wchar_t *wcpcpy(wchar_t *restrict ws1,
        const wchar_t *restrict ws2);
wchar_t *wcscpy(wchar_t *restrict ws1,
        const wchar_t *restrict ws2);
```

The *wcpcpy*() function is equivalent to the *wcscpy*() function, except that it returns a pointer to the terminating null wide-character code copied into the *ws1* buffer.

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

Issue 7: The *wcpcpy*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

**wcscspn**

Purpose: Get the length of a complementary wide substring.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**wcsdup**

Purpose: Duplicate a wide-character string.

CX Synopsis:
```
#include <wchar.h>

wchar_t *wcsdup(const wchar_t *string);
```

The *wcsdup*( ) function is the wide-character equivalent of the *strdup*( ) function.

Application writers should note that for functions that allocate memory as if by *malloc*( ), (such as *wcsdup*( )) the application should release such memory when it is no longer required by a call to *free*( ). For *wcsdup*( ), this is the return value.

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Issue 7: First released in Issue 7.

**wcsftime**

Purpose: Convert date and time to a wide-character string.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**wcslen, wcsnlen**

Purpose: Get length of a fixed-sized wide-character string.

Synopsis:
```
#include <wchar.h>

size_t wcslen(const wchar_t *ws);
```
CX
```
size_t wcsnlen(const wchar_t *ws, size_t maxlen);
```

The *wcsnlen*( ) function computes the smaller of the number of wide characters in the string to which *ws* points, not including the terminating null wide-character code, and the value of *maxlen*. The *wcsnlen*( ) function never examines more than the first *maxlen* characters of the wide-character string pointed to by *ws*.

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

Issue 7: The *wcsnlen*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

**wcsncat**

Purpose: Concatenate a wide-character string with part of another.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**wcsncmp**

Purpose: Compare part of two wide-character strings.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**wcpncpy, wcsncpy**

Purpose: Copy a fixed-size wide-character string, returning a pointer to its end.

Synopsis:
```
#include <wchar.h>
```

CX
```
wchar_t *wcpncpy(wchar_t restrict *ws1,
    const wchar_t *restrict ws2, size_t n);
wchar_t *wcsncpy(wchar_t *restrict ws1,
    const wchar_t *restrict ws2, size_t n);
```

The *wcpncpy*() function is equivalent to the *wcsncpy*() function, except for the return value. If any null wide-character codes were written into the destination, the *wcpncpy*() function returns the address of the first such null wide-character code. Otherwise, it returns *&ws1*[*n*].

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

Issue 7: The *wcpncpy*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

**wcspbrk**

Purpose: Scan a wide-character string for a wide-character code.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**wcsrchr**

Purpose: Wide-character string scanning operation.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**wcsnrtombs, wcsrtombs**

Purpose: Convert a wide-character string to a character string (restartable).

Synopsis:
```
#include <wchar.h>
```

CX
```
size_t wcsnrtombs(char *restrict dst,
    const wchar_t **restrict src, size_t nwc,
    size_t len, mbstate_t *restrict ps);
size_t wcsrtombs(char *restrict dst,
    const wchar_t **restrict src, size_t len,
    mbstate_t *restrict ps);
```

The *wcsnrtombs*( ) function is equivalent to the *wcsrtombs*( ) function, except that the conversion is limited to the first *nwc* wide characters.

Derivation: First released in Issue 5. Included for alignment with .

Issue 7: Austin Group Interpretation 1003.1-2001 #148 is applied, clarifying that the *wcsrtombs*( ) function need not be thread-safe if called with a NULL *ps* argument.

Austin Group Interpretation 1003.1-2001 #170 is applied, changing the [EILSEQ] error condition from a "may fail" to a "shall fail".

The *wcnsrtombs*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

**wcsspn**

Purpose: Get the length of a wide substring.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**wcsstr**

Purpose: Find a wide-character substring.

Synopsis:
```
#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 .

Issue 7: No functional changes are made in this issue.

**wcstod, wcstof, wcstold**

Purpose: Convert a wide-character string to a double-precision number.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**wcstoimax, wcstoumax**

Purpose: Convert a wide-character string to an integer type.

Synopsis:
```
#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.

Issue 7: No functional changes are made in this issue.

**wcstok**

Purpose: Split a wide-character string into tokens.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**wcstol, wcstoll**

Purpose: Convert a wide-character string to a long integer.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**wcstombs**

Purpose: Convert a wide-character string to a character string.

Synopsis:
```
#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 IEEE Std 1003.1i-1995.

Issue 7: Austin Group Interpretation 1003.1-2001 #170 is applied, changing the [EILSEQ] error condition from a "may fail" to a "shall fail".

**wcstoul, wcstoull**

Purpose: Convert a wide-character string to an unsigned long.

Synopsis:
```
#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 7: No functional changes are made in this issue.

**wcswidth**

Purpose: Number of column positions of a wide-character string.

XSI Synopsis:
```
#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 7: No functional changes are made in this issue.

**wcsxfrm, wcsxfrm_l**

Purpose: Wide-character string transformation.

Synopsis:
```
#include <wchar.h>

size_t wcsxfrm(wchar_t *restrict ws1,
    const wchar_t *restrict ws2, size_t n);
```
CX
```
size_t wcsxfrm_l(wchar_t *restrict ws1,
    const wchar_t *restrict ws2, size_t n,
    locale_t locale);
```

The *wcsxfrm_l*() function is equivalent to the *wcsxfrm*() function, except that the locale data used is from the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*() or *duplocale*().

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

Issue 7: The *wcsxfrm_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**wctob**

Purpose: Wide-character to single-byte conversion.

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

int wctob(wint_t c);
```

Derivation: First released in Issue 5. Included for alignment with .

Issue 7: No functional changes are made in this issue.

**wctomb**

Purpose: Convert a wide-character code to a character.

Synopsis:
```
#include <stdlib.h>

int wctomb(char *s, wchar_t wchar);
```

Derivation: First released in Issue 4. Derived from the IEEE Std 1003.1b-1993.

Issue 7: Austin Group Interpretation 1003.1-2001 #170 is applied, adding the [EILSEQ] error condition.

**wctrans, wctrans_l**

Purpose: Define character mapping.

Synopsis:
```
#include <wctype.h>

wctrans_t wctrans(const char *charclass);
```
CX
```
wctrans_t wctrans_l(const char *charclass, locale_t locale);
```

The *wctrans_l*( ) function is equivalent to the *wctrans*( ) function, except that the locale data used is from the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*( ) or *duplocale*( ).

Derivation: First released in Issue 5. Derived from .

Issue 7: The *wctrans_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**wctype, wctype_l**

Purpose: Define character class.

Synopsis:
```
#include <wctype.h>

wctype_t wctype(const char *property);
```
CX
```
wctype_t wctype_l(const char *property, locale_t locale);
```

The *wctype_l*( ) function is equivalent to the *wctype*( ) function, except that the locale data used is from the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale*( ) or *duplocale*( ).

Derivation: First released in Issue 4.

Issue 7: The *wctype_l*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**wcwidth**

Purpose: Number of column positions of a wide-character code.

XSI Synopsis:

```
#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 7: No functional changes are made in this issue.

**wmemchr**

Purpose: Find a wide character in memory.

Synopsis:

```
#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 .

Issue 7: No functional changes are made in this issue.

**wmemcmp**

Purpose: Compare wide characters in memory.

Synopsis:

```
#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 .

Issue 7: No functional changes are made in this issue.

**wmemcpy**

Purpose: Copy wide characters in memory.

Synopsis:

```
#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 .

Issue 7: No functional changes are made in this issue.

**wmemmove**

Purpose: Copy wide characters in memory with overlapping areas.

Synopsis:
```
#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 .

Issue 7: No functional changes are made in this issue.

**wmemset**

Purpose: Set wide characters in memory.

Synopsis:
```
#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 .

Issue 7: No functional changes are made in this issue.

**wordexp, wordfree**

Purpose: Perform word expansions.

Synopsis:
```
#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 the .

Issue 7: Austin Group Interpretation 1003.1-2001 #148 is applied, adding APPLICATION USAGE explaining that the *wordexp*( ) function need not be thread safe if passed an expression referencing an environment variable while any other thread is concurrently modifying any environment variable.

**pwrite, write**

Purpose: Write on a file.

Synopsis:
```
#include <unistd.h>

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 7: The *pwrite*( ) function is moved from the XSI option to the Base.

Functionality relating to the XSI STREAMS option is marked obsolescent.

SD5-XSH-ERN-160 is applied, updating the DESCRIPTION to clarify the requirements for the *pwrite*( ) function, and to change the use of the phrase "file pointer" to "file offset".

**writev**

Purpose: Write a vector.

XSI Synopsis:

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

ssize_t writev(int fildes, const struct iovec *iov,
    int iovcnt);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

**y0, y1, yn**

Purpose: Bessel functions of the second kind.

XSI Synopsis:

```
#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 7: No functional changes are made in this issue.

Chapter 12

# Utilities Migration

## 12.1 Introduction

This chapter contains a section for each utility interface defined in XCU, Issue 7. Each section contains the SYNOPSIS and gives the derivation of the interface. Where new option letters have been added in Issue 7, a brief description is included, complete with examples where appropriate. For interfaces carried forward from Issue 6, syntax and semantic changes made to the interface in Issue 7 are identified (if any). Only changes that might affect an application programmer are included.

## 12.2 Utilities

**admin**

Purpose: Create and administer SCCS files (**DEVELOPMENT**).

XSI Synopsis:

```
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 7: No functional changes are made in this issue.

**alias**

Purpose: Define or display aliases.

Synopsis: `alias [alias-name[=string]...]`

Derivation: First released in Issue 4.

Issue 7: The *alias* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

The first example is changed to remove the creation of an alias for a standard utility that alters its behavior to be non-conforming.

**ar**

Purpose: Create and maintain library archives.

SD Synopsis:
```
ar –d [–v] archive file...
```

XSI
```
ar –m [–v] archive file...
ar –m –a [–v] posname archive file...
ar –m –b [–v] posname archive file...
ar –m –i [–v] posname archive file...
```

XSI
```
ar –p [–v] [–s] archive [file...]
```

XSI
```
ar –q [–cv] archive file...
```

```
ar –r [–cuv] archive file...
```

XSI
```
ar –r –a [–cuv] posname archive file...
ar –r –b [–cuv] posname archive file...
ar –r –i [–cuv] posname archive file...
```

XSI
```
ar –t [–v] [–s] archive [file...]
```

XSI
```
ar –x [–v] [–sCT] archive [file...]
```

Derivation: First released in Issue 2.

Issue 7: SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply (options can be interspersed with operands).

**asa**

Purpose: Interpret carriage-control characters.

FR Synopsis:
```
asa [file...]
```

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDIN section to reflect that standard input is also used if a *file* operand is '–' and the implementation treats the '–' as meaning standard input.

**at**

Purpose: Execute commands at a later time.

Synopsis:
```
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 7: The *at* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-95 is applied, removing the references to fixed locations for the files referenced by the *at* utility.

**awk**

Purpose: Pattern scanning and processing language.

Synopsis: `awk [–F ERE] [–v assignment]... program [argument...]`

`awk [–F ERE] –f progfile [–f progfile]... [–v assignment]... [argument...]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #189 is applied, changing the EXTENDED DESCRIPTION to make the support of hexadecimal integer and floating constants optional.

Austin Group Interpretation 1003.1-2001 #201 is applied, permitting implementations to support infinities and NaNs.

SD5-XCU-ERN-79 is applied, restoring the horizontal lines to XCU Table 4-1, Expressions in Decreasing Precedence in *awk*, and SD5-XCU-ERN-80 is applied, changing the order of some table entries.

**basename**

Purpose: Return non-directory portion of a pathname.

Synopsis: `basename string [suffix]`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**batch**

Purpose: Schedule commands to be executed in a batch queue.

Synopsis: `batch`

Derivation: First released in Issue 2.

Issue 7: The *batch* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-95 is applied, removing the references to fixed locations for the files referenced by the *batch* utility.

**bc**

Purpose: Arbitrary-precision arithmetic language.

Synopsis: `bc [–l] [file...]`

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

**bg**

Purpose: Run jobs in the background.

UP Synopsis:

```
bg [job_id...]
```

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

**c99**

Purpose: Compile standard C programs.

CD Synopsis:

```
c99 [options...] pathname [[pathname] [–I directory]
    [–L directory] [–l library]]...
```

Derivation: First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

Issue 7: Austin Group Interpretation 1003.1-2001 #020 (SD5-XCU-ERN-10) is applied, adding a statement to the OUTPUT FILES section about unspecified behavior when the pathname of an object file or executable file to be created by *c99* resolves to an existing directory entry for a file that is not a regular file.

Austin Group Interpretation 1003.1-2001 #166 is applied, adding information about the use of *getconf* to obtain *c99* arguments used for the threaded programming environment.

Austin Group Interpretation 1003.1-2001 #190 is applied, clarifying the handling of trailing white-space characters.

Austin Group Interpretation 1003.1-2001 #191 is applied, adding APPLICATION USAGE and RATIONALE regarding C-language trigraphs.

SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply (options can be interspersed with operands).

SD5-XCU-ERN-11 is applied, adding the **<net/if.h>** header to the descriptions of **–l c** and **–l xnet**.

SD5-XCU-ERN-65 is applied, updating the EXAMPLES section.

The *getconf* variables for the supported programming environments are updated to be V7.

The **–l trace** library is marked obsolescent.

The *c99* reference page is rewritten to describe **–l** as an option rather than an operand.

**cal**

Purpose: Print a calendar.

XSI Synopsis:
```
cal [[month] year]
```

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**cat**

Purpose: Concatenate and print files.

Synopsis:
```
cat [−u] [file...]
```

Derivation: First released in Issue 2.

Issue 7: SD5-XCU-ERN-174 is applied, changing the RATIONALE concerning an alternative to the historical *cat* −**etv**.

**cd**

Purpose: Change the working directory.

Synopsis:
```
cd [−L|−P] [directory]
cd −
```

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #037 is applied, updating steps 6 through 10 of the processing performed by *cd* to correct a number of defects.

Austin Group Interpretation 1003.1-2001 #199 is applied, clarifying how the *cd* utility handles concatenation of two pathnames when the first pathname ends in a slash character.

Step 7 of the processing performed by *cd* is revised to refer to **curpath** instead of "the operand".

The description of how the *cd* utility sets the *PWD* environment variable has been changed to refer to the output of the *pwd* utility.

**cflow**

Purpose: Generate a C-language flowgraph (**DEVELOPMENT**).

XSI Synopsis:
```
cflow [−r] [−d num] [−D name[=def]]... [−i incl] [−I dir]...
    [−U dir]... file...
```

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**chgrp**

Purpose: Change the file group ownership.

Synopsis: `chgrp [–h] group file...`

`chgrp –R [–H|–L|–P] group file...`

Derivation: First released in Issue 2.

Issue 7: SD5-XCU-ERN-8 is applied, removing the –**R** from the first line of the SYNOPSIS.

**chmod**

Purpose: Change the file modes.

Synopsis: `chmod [–R] mode file...`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #130 is applied, adding text to the DESCRIPTION about marking for update the last file status change timestamp of the file.

**chown**

Purpose: Change the file ownership.

Synopsis: `chown [–h] owner[:group] file...`

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

Derivation: First released in Issue 2.

Issue 7: SD5-XCU-ERN-9 is applied, removing the –**R** from the first line of the SYNOPSIS.

The description of the –**h** and –**P** options is revised.

**cksum**

Purpose: Write file checksums and sizes.

Synopsis: `cksum [file...]`

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDIN section to reflect that standard input is also used if a *file* operand is `'–'` and the implementation treats the `'–'` as meaning standard input.

**cmp**

Purpose: Compare two files.

Synopsis: `cmp [–l|–s] file1 file2`

Derivation: First released in Issue 2.

Issue 7: SD5-XCU-ERN-96 is applied, updating the STDERR section to specify the output when the –**l** option is used.

**comm**

Purpose: Select or reject lines common to two files.

Synopsis: `comm [-123] file1 file2`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**command**

Purpose: Execute a simple command.

Synopsis: `command [-p] command_name [argument...]`

`command [-p][-v|-V] command_name`

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #196 is applied, changing the SYNOPSIS to allow **-p** to be used with **-v** (or **-V**).

The *command* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

The APPLICATION USAGE and EXAMPLES are revised to replace the non-standard *getconf* _CS_PATH with *getconf* PATH.

**compress**

Purpose: Compress data.

XSI Synopsis: `compress [-fv] [-b bits] [file...]`

`compress [-cfv] [-b bits] [file]`

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #125 is applied, revising the ENVIRONMENT VARIABLES section in relation to locale usage.

**cp**

Purpose: Copy files.

Synopsis: `cp [-Pfip] source_file target_file`

`cp [-Pfip] source_file... target`

`cp -R [-H|-L|-P] [-fip] source_file... target`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #092 is applied, specifying that a *source_file* or *target_file* operand of `'-'` shall refer to a file named `'-'`; implementations shall not treat them as meaning standard input or standard output.

Austin Group Interpretation 1003.1-2001 #164 is applied, making the behavior unspecified when *cp* encounters an existing *dest_file* that was written by a previous step.

Austin Group Interpretation 1003.1-2001 #165 is applied, correcting the description

of the **–i** option to reflect that prompts are not written for existing directory files (only non-directory files), as per the detailed steps in the DESCRIPTION.

Austin Group Interpretation 1003.1-2001 #168 is applied, updating the description of how two pathnames are concatenated so that a slash character is only inserted if the first pathname does not end in a slash.

The obsolescent **–r** option is removed.

The **–P** option is added to the SYNOPSIS and to the DESCRIPTION with respect to its use without the **–R** option.

**crontab**

Purpose: Schedule periodic background work.

Synopsis: `crontab [`*`file`*`]`

UP `crontab [–e|–l|–r]`

Derivation: First released in Issue 2.

Issue 7: The *crontab* utility (except for the **–e** option) is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-95 is applied, removing the references to fixed locations for the files referenced by the *crontab* utility.

The first example is changed to remove the unreliable use of *find* | *xargs*.

**csplit**

Purpose: Split files based on context.

Synopsis: `csplit [–ks] [–f` *`prefix`*`] [–n` *`number`*`]` *`file arg...`*

Derivation: First released in Issue 2.

Issue 7: The *csplit* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

The SYNOPSIS and OPERANDS sections are revised to clarify that use of a single *arg* operand is permitted.

**ctags**

Purpose: Create a tags file (**DEVELOPMENT**, **FORTRAN**).

SD Synopsis:

```
ctags [–a] [–f tagsfile] pathname...
ctags –x pathname...
```

Derivation: First released in Issue 4.

Issue 7: The *ctags* utility is no longer dependent on support for the User Portability Utilities option.

**cut**

Purpose: Cut out selected fields of each line of a file.

Synopsis:
```
cut –b list [–n] [file...]
cut –c list [file...]
cut –f list [–d delim] [–s] [file...]
```

Derivation: First released in Issue 2.

Issue 7: SD5-XCU-ERN-171 is applied, adding APPLICATION USAGE regarding the use of the *cut* and *fold* utilities to create text files out of files with arbitrary line lengths.

**cxref**

Purpose: Generate a C-language program cross-reference table (**DEVELOPMENT**).

XSI Synopsis:
```
cxref [–cs] [–o file] [–w num] [–D name[=def]]... [–I dir]...
    [–U name]... file...
```

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**date**

Purpose: Write the date and time.

Synopsis:
```
date [–u] [+format]
```

XSI
```
date [–u] mmddhhmm[[cc]yy]
```

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**dd**

Purpose: Convert and copy a file.

Synopsis:
```
dd [operand...]
```

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #102 is applied, adding requirements for the output file to be extended when the input file is empty, **seek**=*expr* is specified but **conv**=**notrunc** is not, and either the size of the seek is greater than the previous size of the output file or the output file did not previously exist.

**delta**

Purpose: Make a delta (change) to an SCCS file (**DEVELOPMENT**).

XSI Synopsis:
```
delta [–nps] [–g list] [–m mrlist] [–r SID]
    [–y[comment]] file...
```

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**df**

Purpose: Report free disk space.

XSI Synopsis: `df [-k] [-P|-t] [file...]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #099 is applied, clarifying the XSI requirements for operands which name a special file containing a file system.

The *df* utility is removed from the User Portability Utilities option. User Portability Utilities is now an option for interactive utilities.

**diff**

Purpose: Compare two files.

Synopsis: `diff [-c|-e|-f|-u|-C n|-U n] [-br] file1 file2`

When the **-u** option is specified, *diff* produces output in a form that provides three lines of unified context.

When the **-U** *n* option is specified, *diff* produces output in a form that provides *n* lines of unified context.

The **-u** or **-U** options behave like the **-c** or **-C** options, except that the context lines are not repeated; instead, the context, deleted, and added lines are shown together, interleaved.

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #114 is applied, requiring *diff* to detect infinite loops in the file system when the **-r** option is specified.

Austin Group Interpretation 1003.1-2001 #115 is applied, updating requirements when block or character special files are encountered in directories being compared.

Austin Group Interpretation 1003.1-2001 #192 is applied, clarifying the behavior if one or both files are non-text files.

SD5-XCU-ERN-103 and SD5-XCU-ERN-120 are applied, adding the **-u** and **-U** options.

**dirname**

Purpose: Return the directory portion of a pathname.

Synopsis: `dirname string`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**du**

Purpose: Estimate file space usage.

Synopsis: `du [-a|-s] [-kx] [-H|-L] [file...]`

Derivation: First released in Issue 2.

Issue 7: The *du* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

**echo**

Purpose: Write arguments to standard output.

Synopsis: `echo [string...]`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**ed**

Purpose: Edit text.

Synopsis: `ed [-p string] [-s] [file]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if an operand is `'-'`.

Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BRE back-references when a subexpression does not participate in the match.

SD5-XCU-ERN-94 is applied, updating text in the EXTENDED DESCRIPTION where a terminal disconnect is detected (in Commands in *ed*).

**env**

Purpose: Set the environment for command invocation.

Synopsis: `env [-i] [name=value]... [utility [argument...]]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first argument is `'-'`.

Austin Group Interpretation 1003.1-2001 #047 is applied, providing RATIONALE on how to use the *env* utility to preserve a conforming environment.

The EXAMPLES section is revised to change the use of *env* **-i** so that it preserves a conforming environment.

**ex**

Purpose: Text editor.

UP Synopsis:
```
ex [–rR] [–s|–v] [–c command] [–t tagstring]
    [–w size] [file...]
```

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if an operand is '–'.

Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BRE back-references when a subexpression does not participate in the match.

**expand**

Purpose: Convert tabs to spaces.

Synopsis: `expand [–t tablist] [file...]`

Derivation: First released in Issue 4.

Issue 7: The *expand* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

**expr**

Purpose: Evaluate arguments as an expression.

Synopsis: `expr operand...`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BRE back-references when a subexpression does not participate in the match.

**false**

Purpose: Return false value.

Synopsis: `false`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**fc**

Purpose: Process the command history list.

UP Synopsis:
```
fc [–r] [–e editor] [first [last]]
fc –l [–nr] [first [last]]
fc –s [old=new] [first]
```

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

**fg**

Purpose: Run jobs in the foreground.

UP Synopsis: `fg [job_id]`

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

**file**

Purpose: Determine file type.

Synopsis: `file [–dh] [–M file] [–m file] file...`

`file –i [–h] file...`

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDIN section to reflect that standard input is used if a *file* operand is `'–'` and the implementation treats the `'–'` as meaning standard input.

SD5-XCU-ERN-4 is applied, adding further entries in the Notes column in XCU Table 4-9, File Utility Output Strings .

The *file* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

The EXAMPLES section is revised to make use of the `"––"` delimiter.

**find**

Purpose: Find files.

Synopsis: `find [–H|–L] path... [operand_expression...]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #127 is applied, rephrasing the description of the –**exec** primary to be "immediately follows".

Austin Group Interpretation 1003.1-2001 #185 is applied, clarifying the requirements for the –**H** and –**L** options.

Austin Group Interpretation 1003.1-2001 #186 is applied, clarifying the requirements for the evaluation of *path* operands with trailing slashes.

Austin Group Interpretation 1003.1-2001 #195 is applied, clarifying the interpretation of the first operand.

SD5-XCU-ERN-48 is applied, clarifying the –**L** option in the case that the file referenced by a symbolic link does not exist.

SD5-XCU-ERN-117 is applied, clarifying the –**perm** primary.

SD5-XCU-ERN-122 is applied, adding a new EXAMPLE showing the useful technique:

```
-exec sh -c '... "$@" ...' sh {} +
```

The description of the –**name** primary is revised and the –**path** primary is added (with a new example).

**fold**

Purpose: Filter for folding lines.

Synopsis: `fold [-bs] [-w width] [file...]`

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDIN section to reflect that standard input is also used if a *file* operand is '-' and the implementation treats the '-' as meaning standard input.

Austin Group Interpretation 1003.1-2001 #204 is applied, updating the DESCRIPTION to clarify when a <newline> can be inserted before or after a <backspace>.

**fort77**

Purpose: FORTRAN compiler (**FORTRAN**).

FD Synopsis:
```
fort77 [-c] [-g] [-L directory]... [-O optlevel] [-o outfile]
    [-s] [-w] operand...
```

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

**fuser**

Purpose: List process IDs of all processes that have one or more files open.

XSI Synopsis:
```
fuser [-cfu] file...
```

Derivation: First released in Issue 5.

Issue 7: SD5-XCU-ERN-90 is applied, updating the EXAMPLES section.

**gencat**

Purpose: Generate a formatted message catalog.

Synopsis: `gencat catfile msgfile...`

Derivation: First released in Issue 3.

Issue 7: The *gencat* utility is moved from the XSI option to the Base.

**get**

Purpose: Get a version of an SCCS file (**DEVELOPMENT**).

XSI Synopsis:
```
get [-begkmnlLpst] [-c cutoff] [-i list] [-r SID]
    [-x list] file...
```

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**getconf**

Purpose: Get configuration values.

Synopsis:
```
getconf [–v specification] system_var
getconf [–v specification] path_var pathname
```

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

**getopts**

Purpose: Parse utility options.

Synopsis:
```
getopts optstring name [arg...]
```

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

**grep**

Purpose: Search a file for a pattern.

Synopsis:
```
grep [–E|–F] [–c|–l|–q] [–insvx] –e pattern_list
    [–e pattern_list]... [–f pattern_file]... [file...]
grep [–E|–F] [–c|–l|–q] [–insvx] [–e pattern_list]...
    –f pattern_file [–f pattern_file]... [file...]
grep [–E|–F] [–c|–l|–q] [–insvx] pattern_list [file...]
```

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDIN section to reflect that standard input is also used if a *file* operand is `'–'` and the implementation treats the `'–'` as meaning standard input.

SD5-XCU-ERN-98 is applied, updating the STDOUT section to reflect the fact that the **–l** and **–q** options are shown in the SYNOPSIS as mutually exclusive.

**hash**

Purpose: Remember or report utility locations.

Synopsis:
```
hash [utility...]
hash –r
```

Derivation: First released in Issue 2.

Issue 7: The *hash* utility is moved from the XSI option to the Base.

**head**

Purpose: Copy the first part of files.

Synopsis: `head [–n number] [file...]`

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDIN section to reflect that standard input is also used if a *file* operand is `'–'` and the implementation treats the `'–'` as meaning standard input.

The EXAMPLES section is revised to make use of the "––" delimiter.

**iconv**

Purpose: Codeset conversion.

Synopsis:
```
iconv [–cs] –f frommap –t tomap [file...]
iconv –f fromcode [–cs] [–t tocode] [file...]
iconv –t tocode [–cs] [–f fromcode] [file...]
iconv –l
```

Derivation: First released in Issue 3.

Issue 7: No functional changes are made in this issue.

**id**

Purpose: Return user identity.

Synopsis:
```
id [user]
id –G [–n] [user]
id –g [–nr] [user]
id –u [–nr] [user]
```

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**ipcrm**

Purpose: Remove an XSI message queue, semaphore set, or shared memory segment identifier.

XSI Synopsis:
```
ipcrm [–q msgid|–Q msgkey|–s semid|–S semkey|
   –m shmid|–M shmkey]...
```

Derivation: First released in Issue 5.

Issue 7: No functional changes are made in this issue.

**ipcs**

Purpose: Report XSI interprocess communication facilities status.

XSI Synopsis:
```
ipcs [-qms] [-a|-bcopt]
```

Derivation: First released in Issue 5.

Issue 7: No functional changes are made in this issue.

**jobs**

Purpose: Display status of jobs in the current session.

UP Synopsis:
```
jobs [-l|-p] [job_id...]
```

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

**join**

Purpose: Relational database operator.

Synopsis:
```
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 7: No functional changes are made in this issue.

**kill**

Purpose: Terminate or signal processes.

Synopsis:
```
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 7: No functional changes are made in this issue.

**lex**

Purpose: Generate programs for lexical tasks (**DEVELOPMENT**).

CD Synopsis:
```
lex [-t] [-n|-v] [file...]
```

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #190 is applied, clarifying the requirements for generated code to conform to the IEEE Std 1003.1i-1995.

Austin Group Interpretation 1003.1-2001 #191 is applied, clarifying the handling of C-language trigraphs and curly brace preprocessing tokens.

SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply (options can be interspersed with operands).

**link**

Purpose: Call *link*( ) function.

XSI Synopsis:

```
link file1 file2
```

Derivation: First released in Issue 5.

Issue 7: No functional changes are made in this issue.

**ln**

Purpose: Link files.

Synopsis:

```
ln [-fs] [-L|-P] source_file target_file
ln [-fs] [-L|-P] source_file... target_dir
```

When the **−L** option is specified (and the **−s** option is not specified), for each *source_file* operand that names a file of type symbolic link, *ln* creates a (hard) link to the file referenced by the symbolic link.

When the **−P** option is specified (and the **−s** option is not specified), for each *source_file* operand that names a file of type symbolic link, *ln* creates a (hard) link to the symbolic link itself.

If the **−s** option is not specified and neither a **−L** nor a **−P** option is specified, it is implementation-defined which of the **−L** and **−P** options will be used as the default.

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #164 is applied, allowing *ln* to report an error when it encounters an existing destination path that was written by a previous step.

Austin Group Interpretation 1003.1-2001 #168 is applied, updating the description of how two pathnames are concatenated so that a slash character is only inserted if the first pathname does not end in a slash.

Austin Group Interpretation 1003.1-2001 #169 is applied, updating the requirements when *destination* names the same directory entry as the current *source_file*.

The **−L** and **−P** options are added to provide control over how the *ln* utility creates hard links to symbolic links.

**locale**

Purpose: Get locale-specific information.

Synopsis:

```
locale [-a|-m]
locale [-ck] name...
```

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #017 is applied, clarifying the standard output for the –**k** option for non-numeric compound keyword values.

Austin Group Interpretations 1003.1-2001 #021 and #088 are applied, clarifying the standard output for the –**k** option when *LANG* is not set or is an empty string.

**localedef**

Purpose: Define locale environment.

Synopsis: `localedef [–c] [–f charmap] [–i sourcefile] [–u code_set_name] name`

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

**logger**

Purpose: Log messages.

Synopsis: `logger string...`

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

**logname**

Purpose: Return the user's login name.

Synopsis: `logname`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**lp**

Purpose: Send files to a printer.

Synopsis: `lp [–c] [–d dest] [–n copies] [–msw] [–o option]... [–t title] [file...]`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**ls**

Purpose: List directory contents.

XSI Synopsis: `ls [–ACFRSacdfiklmnpqrstux1] [–H|–L] [–go] [file...]`

When the –**A** option is specified, *ls* writes out all directory entries, including those whose names begin with a <period> (`'.'`) but excluding the entries dot and dot-dot (if they exist).

When the –**S** option is specified, *ls* sorts with the primary key being file size (in decreasing order) and the secondary key being filename in the collating sequence (in increasing order).

When the –**k** option is specified, *ls* sets the block size for the –**s** option and the per-directory block count written for the –**l**, –**n**, –**s**, –**g**, and –**o** options to 1 024 bytes.

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #101 is applied, clarifying the optional alternate access method flag in the STDOUT section.

Austin Group Interpretation 1003.1-2001 #128 is applied, clarifying the DESCRIPTION and the definition of the −**R** option with regard to symbolic links.

Austin Group Interpretation 1003.1-2001 #198 is applied, clarifying the requirements for the −**H** option for symbolic links specified on the command line.

SD5-XCU-ERN-50 is applied, adding the −**A** option.

The −**S** option is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

The −**f**, −**m**, −**n**, −**p**, −**s**, and −**x** options are moved from the XSI option to the Base.

The description of the −**f**, −**s**, and −**t** options are revised and the −**k** option is added.

**m4**

Purpose: Macro processor.

Synopsis: `m4 [−s] [−D name[=val]]... [−U name]... file...`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #117 is applied, marking the **maketemp** macro obsolescent and adding a new **mkstemp** macro.

Austin Group Interpretation 1003.1-2001 #207 is applied, clarifying the handling of white-space characters that precede or trail any macro arguments.

SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply (options can be interspersed with operands).

SD5-XCU-ERN-99 is applied, clarifying the definition of the **divert** macro in the EXTENDED DESCRIPTION.

SD5-XCU-ERN-100 is applied, clarifying the definition of the **syscmd** macro in the EXTENDED DESCRIPTION.

SD5-XCU-ERN-101 is applied, clarifying the definition of the **undivert** macro in the EXTENDED DESCRIPTION.

SD5-XCU-ERN-111 is applied to the EXTENDED DESCRIPTION, clarifying that the string `"${"` produces unspecified behavior.

SD5-XCU-ERN-112 is applied, updating the **changequote** macro.

SD5-XCU-ERN-118 is applied, clarifying the definition of the **define** macro in the EXTENDED DESCRIPTION and APPLICATION USAGE sections.

SD5-XCU-ERN-119 is applied, clarifying the definition of the **translit** macro in the EXTENDED DESCRIPTION and RATIONALE sections.

SD5-XCU-ERN-130 is applied, making the behavior unspecified when macro names are used without arguments.

SD5-XCU-ERN-131 is applied, making the behavior unspecified when either argument to the **changecom** macro is provided but null.

SD5-XCU-ERN-137 is applied, updating the description of the **eval** macro in the

EXTENDED DESCRIPTION and APPLICATION USAGE sections.

The *m4* utility is moved from the XSI option to the Base.

**mailx**

Purpose: Process messages.

Synopsis: **Send Mode**

```
mailx [–s subject] address...
```

**Receive Mode**

UP

```
mailx –e
mailx [–HiNn] [–F] [–u user]
mailx –f [–HiNn] [–F] [file]
```

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #089 is applied, clarifying the effect of the *LC_TIME* environment variable.

Austin Group Interpretation 1003.1-2001 #090 is applied, updating the description of the **next** command.

**make**

Purpose: Maintain, update, and regenerate groups of programs (**DEVELOPMENT**).

SD Synopsis:

```
make [–einpqrst] [–f makefile]... [–k|–S] [macro=value...]
    [target_name...]
```

Derivation: First released in Issue 2.

Issue 7: SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply (options can be interspersed with operands).

Include lines in makefiles are introduced.

Austin Group Interpretation 1003.1-2001 #131 is applied, changing the **Makefile Execution** section.

**man**

Purpose: Display system documentation.

Synopsis:

```
man [–k] name...
```

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #108 is applied, clarifying that informational messages may appear on standard error.

**mesg**

Purpose: Permit or deny messages.

Synopsis: `mesg [y|n]`

Derivation: First released in Issue 2.

Issue 7: The *mesg* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

**mkdir**

Purpose: Make directories.

Synopsis: `mkdir [-p] [-m mode] dir...`

Derivation: First released in Issue 2.

Issue 7: SD5-XCU-ERN-56 is applied, aligning the **-m** option with the IEEE P1003.2b draft standard to clarify an ambiguity.

**mkfifo**

Purpose: Make FIFO special files.

Synopsis: `mkfifo [-m mode] file...`

Derivation: First released in Issue 3.

Issue 7: No functional changes are made in this issue.

**more**

Purpose: Display files on a page-by-page basis.

UP Synopsis:
```
more [-ceisu] [-n number] [-p command] [-t tagstring]
    [file...]
```

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that '+' may be recognized as an option delimiter in the OPTIONS section.

**mv**

Purpose: Move files.

Synopsis:
```
mv [-if] source_file target_file
mv [-if] source_file... target_dir
```

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #016 is applied, updating requirements relating to a *target_file* operand with a trailing slash .

Austin Group Interpretation 1003.1-2001 #164 is applied, allowing *mv* to report an error when it encounters an existing destination path that was written by a previous step.

Austin Group Interpretation 1003.1-2001 #168 is applied, updating the description of how two pathnames are concatenated so that a slash character is only inserted if

the first pathname does not end in a slash.

Austin Group Interpretation 1003.1-2001 #169 is applied, updating the requirements when the *source_file* operand and destination path name the same existing file.

SD5-XCU-ERN-51 is applied to the DESCRIPTION, clarifying that it is unspecified whether hard links to other files are preserved when files are being duplicated to another file system.

Changes are made related to support for finegrained timestamps.

**newgrp**

Purpose: Change to a new group.

Synopsis: `newgrp [-l] [group]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first argument is `'-'`.

The *newgrp* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

**nice**

Purpose: Invoke a utility with an altered nice value.

Synopsis: `nice [-n increment] utility [argument...]`

Derivation: First released in Issue 4.

Issue 7: The *nice* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

**nl**

Purpose: Line numbering filter.

XSI Synopsis:
```
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 7: Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDIN section to reflect that standard input is also used if a *file* operand is `'-'` and the implementation treats the `'-'` as meaning standard input.

**nm**

Purpose: Write the name list of an object file (**DEVELOPMENT**).

SD Synopsis:
```
nm [-APv] [-g|-u] [-t format] file...
```
XSI
```
nm [-APv] [-efox] [-g|-u] [-t format] file...
```

Derivation: First released in Issue 2.

Issue 7: The *nm* utility is removed from the User Portability Utilities option. User Portability Utilities is now an option for interactive utilities.

**nohup**

Purpose: Invoke a utility immune to hangups.

Synopsis: `nohup` *utility* **[***argument...***]**

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #104 is applied, allowing *nohup* to redirect standard input from an unspecified file if it is associated with a terminal.

Austin Group Interpretations 1003.1-2001 #105 and #106 are applied, updating requirements related to redirection of standard output and standard error.

**od**

Purpose: Dump files in various formats.

Synopsis: `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 7: Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDIN section to reflect that standard input is also used if a *file* operand is '–' and the implementation treats the '–' as meaning standard input.

**paste**

Purpose: Merge corresponding or subsequent lines of files.

Synopsis: `paste [`–s`] [`–d *list*`] *file*...`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**patch**

Purpose: Apply changes to files.

Synopsis: `patch [`–blNR`] [`–c|–e|–n|–u`] [`–d *dir*`] [`–D *define*`] [`–i *patchfile*`]`
`    [`–o *outfile*`] [`–p *num*`] [`–r *rejectfile*`] [`*file*`]`

When the –**u** option is specified, *patch* interprets the patch file as a unified context difference (the output of the *diff* utility when the –**u** or –**U** options are specified).

Derivation: First released in Issue 4.

Issue 7: The *patch* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-103 and SD5-XCU-ERN-120 are applied, adding the –**u** option.

Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the *LC_CTYPE* environment variable and adding the *LC_COLLATE*

environment variable.

**pathchk**

Purpose: Check pathnames.

Synopsis: `pathchk [-p] [-P] pathname...`

When the **-P** option is specified, *pathchk* writes a diagnostic for each *pathname* operand that:

- Contains a component whose first character is the <hyphen> character
- Is empty

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretations 1003.1-2001 #039 and #040 are applied, adding the **-P** option.

SD5-XCU-ERN-121 is applied, updating the way *xargs* is used in the EXAMPLES section.

**pax**

Purpose: Portable archive interchange.

Synopsis:
```
pax [-dv] [-c|-n] [-H|-L] [-o options] [-f archive]
    [-s replstr]... [pattern...]

pax -r[-c|-n] [-dikuv] [-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] [-o options]... [-p string]...
    [-s replstr]... [file...] directory
```

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #011 is applied, clarifying how symbolic links are archived in **cpio** format.

Austin Group Interpretation 1003.1-2001 #086 is applied, clarifying that when a list of files to copy is read from the standard input, each entry in the list is processed as if it had been a *file* operand on the command line.

Austin Group Interpretation 1003.1-2001 #109 is applied, adding the **hdrcharset** keyword to the *pax* extended headers, and related requirements.

SD5-XCU-ERN-2 is applied, making the **-c** and **-n** options mutually-exclusive in the SYNOPSIS.

SD5-XCU-ERN-60 is applied, revising text which incorrectly implied that the **-x** option could be used in copy mode.

The *pax* utility is no longer allowed to create separate identical symbolic links when extracting linked symbolic links from an archive, because the standard now requires implementations to support (hard) linking of symbolic links.

**pr**

Purpose: Print files.

Synopsis:
```
pr [+page] [–column] [–adFmrt] [–e[char][gap]] [–h header]
    [–i[char][gap]] [–l lines] [–n[char][width]]
XSI     [–o offset] [–s[char]] [–w width] [–fp] [file...]
```

Derivation: First released in Issue 2.

Issue 7: PASC Interpretation 1003.2-92 #151 (SD5-XCU-ERN-44) is applied, replacing ‘‘two or more’’ in the description of the **–i** option with ‘‘one or more’’.

Austin Group Interpretation 1003.1-2001 #093 is applied, adding APPLICATION USAGE warning that a first operand that starts with a <plus-sign> needs to be preceded with the `"--"` argument that denotes the end of the options.

**printf**

Purpose: Write formatted output.

Synopsis:
```
printf format [argument...]
```

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #175 is applied, updating requirements related to floating-point conversions to align with the *printf*( ) function.

Austin Group Interpretation 1003.1-2001 #177 is applied, clarifying the behavior of the %c conversion.

**prs**

Purpose: Print an SCCS file (**DEVELOPMENT**).

XSI Synopsis:
```
prs [–a] [–d dataspec] [–r[SID]] file...
prs [–e|–l] –c cutoff [–d dataspec] file...
prs [–e|–l] –r[SID] [–d dataspec] file...
```

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**ps**

Purpose: Report process status.

XSI Synopsis:
```
ps [–aA] [–defl] [–g grouplist] [–G grouplist]
    [–n namelist] [–o format]... [–p proclist] [–t termlist]
    [–u userlist] [–U userlist]
```

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**pwd**

Purpose: Return working directory name.

Synopsis: `pwd [-L|-P]`

Derivation: First released in Issue 2.

Issue 7: Changes have been made to match the changes to the *getcwd*() function, adding text to address the case where the current directory is deeper in the file hierarchy than {PATH_MAX} bytes, and adding the requirements relating to pathnames beginning with two slash characters.

**qalter**

Purpose: Alter batch job.

OB BE Synopsis:

```
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...
```

Derivation: Derived from IEEE Std 1003.2d-1994.

Issue 7: The *qalter* utility is marked obsolescent.

**qdel**

Purpose: Delete batch jobs.

OB BE Synopsis:

```
qdel job_identifier...
```

Derivation: Derived from IEEE Std 1003.2d-1994.

Issue 7: The *qdel* utility is marked obsolescent.

**qhold**

Purpose: Hold batch jobs.

OB BE Synopsis:

```
qhold [-h hold_list] job_identifier...
```

Derivation: Derived from IEEE Std 1003.2d-1994.

Issue 7: The *qhold* utility is marked obsolescent.

**qmove**

Purpose: Move batch jobs.

OB BE Synopsis:

```
qmove destination job_identifier...
```

Derivation: Derived from IEEE Std 1003.2d-1994.

Issue 7: The *qmove* utility is marked obsolescent.

**qmsg**

Purpose: Send message to batch jobs.

OB BE Synopsis:

```
qmsg [–EO] message_string job_identifier...
```

Derivation: Derived from IEEE Std 1003.2d-1994.

Issue 7: The *qmsg* utility is marked obsolescent.

**qrerun**

Purpose: Rerun batch jobs.

OB BE Synopsis:

```
qrerun job_identifier...
```

Derivation: Derived from IEEE Std 1003.2d-1994.

Issue 7: The *qrerun* utility is marked obsolescent.

**qrls**

Purpose: Release batch jobs.

OB BE Synopsis:

```
qrls [–h hold_list] job_identifier...
```

Derivation: Derived from IEEE Std 1003.2d-1994.

Issue 7: The *qrls* utility is marked obsolescent.

**qselect**

Purpose: Select batch jobs.

OB BE Synopsis:

```
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]
```

Derivation: Derived from IEEE Std 1003.2d-1994.

Issue 7: The *qselect* utility is marked obsolescent.

**qsig**

Purpose: Signal batch jobs.

OB BE Synopsis:

```
qsig [–s signal] job_identifier...
```

Derivation: Derived from IEEE Std 1003.2d-1994.

Issue 7: The *qsig* utility is marked obsolescent.

**qstat**

Purpose: Show status of batch jobs.

OB BE Synopsis:

```
qstat [–f] job_identifier...
qstat –Q [–f] destination...
qstat –B [–f] server_name...
```

Derivation: Derived from IEEE Std 1003.2d-1994.

Issue 7: The *qstat* utility is marked obsolescent.

**qsub**

Purpose: Submit a script.

OB BE Synopsis:

```
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]
```

Derivation: Derived from IEEE Std 1003.2d-1994.

Issue 7: The *qsub* utility is marked obsolescent.

**read**

Purpose: Read a line from standard input.

Synopsis: `read [–r] var...`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #194 is applied, clarifying the handling of the <backslash> escape character.

**renice**

Purpose: Set nice values of running processes.

Synopsis: `renice [–g|–p|–u] –n increment ID...`

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply (options can be interspersed with operands).

The *renice* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

**rm**

Purpose: Remove directory entries.

Synopsis: `rm [-fiRr] file...`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #019 is applied, requiring *rm* to report an error if an operand resolves to the root directory.

Austin Group Interpretation 1003.1-2001 #091 is applied, updating the description of exit status 0 in the EXIT STATUS section.

**rmdel**

Purpose: Remove a delta from an SCCS file (**DEVELOPMENT**).

XSI Synopsis: `rmdel -r SID file...`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**rmdir**

Purpose: Remove directories.

Synopsis: `rmdir [-p] dir...`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**sact**

Purpose: Print current SCCS file-editing activity (**DEVELOPMENT**).

XSI Synopsis: `sact file...`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**sccs**

Purpose: Front end for the SCCS subsystem (**DEVELOPMENT**).

XSI Synopsis:
```
sccs [-r] [-d path] [-p path] command [options...]
    [operands...]
```

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

**sed**

Purpose: Stream editor.

Synopsis:
```
sed [–n] script [file...]
sed [–n] –e script [–e script]... [–f script_file]...
    [file...]
sed [–n] [–e script]... –f script_file [–f script_file]...
    [file...]
```

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BRE back-references when a subexpression does not participate in the match.

Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDIN section to reflect that standard input is also used if a *file* operand is '–' and the implementation treats the '–' as meaning standard input.

SD5-XCU-ERN-123 is applied, updating the SYNOPSIS so that it correctly reflects the relationship between the *script* operand and the **–e** and **–f** options.

A second example is added, giving a simpler method of squeezing empty lines.

**sh**

Purpose: Shell, the standard command language interpreter.

Synopsis:
```
sh [–abCefhimnuvx] [–o option]... [+abCefhimnuvx]
    [+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 7: Austin Group Interpretation 1003.1-2001 #098 is applied, changing the description of *IFS* to match the one in section 2.5.3.

The description of the *PWD* environment variable is updated to reflect that assignments to the variable may always be ignored.

Minor changes are made to the install script example in the APPLICATION USAGE section.

**sleep**

Purpose: Suspend execution for an interval.

Synopsis:
```
sleep time
```

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**sort**

Purpose: Sort, merge, or sequence check text files.

Synopsis:
```
sort [–m] [–o output] [–bdfinru] [–t char] [–k keydef]...
    [file...]

sort [–c|–C] [–bdfinru] [–t char] [–k keydef] [file]
```

The **–C** option is the same as **–c**, except that a warning message is not sent to standard error if disorder or, with **–u**, a duplicate key is detected.

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply (options can be interspersed with operands), and noting that `'+'` may be recognized as an option delimiter.

Austin Group Interpretation 1003.1-2001 #120 is applied, updating the **–c** option to require that the warning message sent to standard error indicates where the disorder or duplicate key was found, and introducing the **–C** option.

XCU-ERN-81 is applied, modifying the description of the **–i** option to state that the behavior is undefined for a sort key for which **–n** also applies.

**split**

Purpose: Split files into pieces.

Synopsis:
```
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 7: The *split* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

**strings**

Purpose: Find printable strings in files.

Synopsis:
```
strings [–a] [–t format] [–n number] [file...]
```

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first argument is `'–'`.

The *strings* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

**strip**

Purpose: Remove unnecessary information from strippable files (**DEVELOPMENT**).

SD Synopsis:
```
strip file...
```

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #103 is applied, clarifying that XSI-conformant systems support use of *strip* on archive files containing object files or relocatable files.

**stty**

Purpose: Set the options for a terminal.

Synopsis: `stty [-a|-g]`

`stty operand...`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #144 is applied, moving functionality relating to the IXANY symbol from the XSI option to the Base.

**tabs**

Purpose: Set terminal tabs.

XSI Synopsis: `tabs [-n|-a|-a2|-c|-c2|-c3|-f|-p|-s|-u] [-T type]`

`tabs [-T type] n[[sep[+]n]...]`

Derivation: First released in Issue 2.

Issue 7: The *tabs* utility is removed from the User Portability Utilities option. User Portability Utilities is now an option for interactive utilities.

**tail**

Purpose: Copy the last part of a file.

Synopsis: `tail [-f] [-c number|-n number] [file]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that '+' may be recognized as an option delimiter in the OPTIONS section.

Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDIN section to reflect that standard input is also used if the *file* operand is '-' and the implementation treats the '-' as meaning standard input.

Austin Group Interpretation 1003.1-2001 #100 is applied, adding the requirement on applications that if the sign of the option-argument *number* is '+', the *number* option-argument is non-zero.

SD5-XCU-ERN-114 is applied, updating the **-f** option so that a FIFO on standard input is treated the same as a pipe.

**talk**

Purpose: Talk to another user.

UP Synopsis: `talk address [terminal]`

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

**tee**

Purpose: Duplicate standard input.

Synopsis: `tee [-ai] [file...]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #092 is applied, specifying that a *file* operand of `'-'` shall refer to a file named `'-'`; implementations shall not treat it as meaning standard output.

**test**

Purpose: Evaluate expression.

Synopsis: `test [expression]`

`[ [expression] ]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #107 is applied, marking the XSI extensions specifying the **-a** and **-o** primaries and the `'('` and `')'` operators as obsolescent. Applications should combine separate *test* commands instead. For example, using:

```
test expr1 && test expr2
```

instead of:

```
test expr1 -a expr2
```

**time**

Purpose: Time a simple command.

Synopsis: `time [-p] utility [argument...]`

Derivation: First released in Issue 2.

Issue 7: The *time* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

**touch**

Purpose: Change file access and modification times.

Synopsis: `touch [-acm] [-r ref_file|-t time|-d date_time] file...`

When the **-d** *date_time* option is specified, *touch* uses the specified *date_time* instead of the current time. The option-argument is a string of the form:

*YYYY*`-`*MM*`-`*DD*`Thh:`*mm*`:`*SS*`[.`*frac*`][`*tz*`]`

or:

*YYYY*`-`*MM*`-`*DD*`Thh:`*mm*`:`*SS*`[,`*frac*`][`*tz*`]`

where:

- *YYYY* are at least four decimal digits giving the year.

- *MM*, *DD*, *hh*, *mm*, and *SS* are as with –**t** *time*.
- `T` is the time designator, and can be replaced by a single <space>.
- `[.`*`frac`*`]` and `[,`*`frac`*`]` are either empty, or a <period> (`'.'`) or a <comma> (`','`) respectively, followed by one or more decimal digits, specifying a fractional second.
- `[`*`tz`*`]` is either empty, signifying local time, or the letter `'Z'`, signifying UTC.

The following examples demonstrate the use of the –**d** option.

Create or update a file called **dwc**; the resulting file has both the last data modification and last data access timestamps set to November 12, 2007 at 10:15:30 local time:

```
touch –d 2007-11-12T10:15:30 dwc
```

Create or update a file called **nick**; the resulting file has both the last data modification and last data access timestamps set to November 12, 2007 at 10:15:30 UTC:

```
touch –d 2007-11-12T10:15:30Z nick
```

Create or update a file called **gwc**; the resulting file has both the last data modification and last data access timestamps set to November 12, 2007 at 10:15:30 local time with a fractional second timestamp of .002 seconds:

```
touch –d 2007-11-12T10:15:30,002 gwc
```

Create or update a file called **ajosey**; the resulting file has both the last data modification and last data access timestamps set to November 12, 2007 at 10:15:30 UTC with a fractional second timestamp of .002 seconds:

```
touch –d "2007-11-12 10:15:30.002Z" ajosey
```

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #118 is applied, allowing *touch* to support times that precede the Epoch.

Austin Group Interpretation 1003.1-2001 #193 is applied, adding the –**d** option with support for subsecond timestamps.

SD5-XCU-ERN-45 is applied, adding a new paragraph to the RATIONALE noting that if at least two operands are specified and the first operand is an eight or ten-digit decimal integer, the first operand will be taken to be a *file* operand, whereas in previous versions of the standard it would have been taken to be an (obsolescent) *date_time* operand.

Changes are made related to support for finegrained timestamps.

**tput**

Purpose: Change terminal characteristics.

Synopsis: `tput [–T `*`type`*`] `*`operand`*`...`

Derivation: First released in Issue 4.

Issue 7: The *tput* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

**tr**

Purpose: Translate characters.

Synopsis:
```
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 7: Austin Group Interpretation 1003.1-2001 #132 is applied, clarifying that the behavior is unspecified if an unescaped trailing <backslash> is present in *string1* or *string2*.

**true**

Purpose: Return true value.

Synopsis:
```
true
```

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**tsort**

Purpose: Topological sort.

Synopsis:
```
tsort [file]
```

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDIN section to reflect that standard input is also used if a *file* operand is `'-'` and the implementation treats the `'-'` as meaning standard input.

The *tsort* utility is moved from the XSI option to the Base.

**tty**

Purpose: Return user's terminal name.

Synopsis:
```
tty
```

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**type**

Purpose: Write a description of command type.

XSI Synopsis:
```
type name...
```

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**ulimit**

Purpose: Set or report file size limit.

XSI Synopsis: `ulimit [-f] [blocks]`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**umask**

Purpose: Get or set the file mode creation mask.

Synopsis: `umask [-S] [mask]`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**unalias**

Purpose: Remove alias definitions.

Synopsis: `unalias alias-name...`

`unalias -a`

Derivation: First released in Issue 4.

Issue 7: The *unalias* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

**uname**

Purpose: Return system name.

Synopsis: `uname [-amnrsv]`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**uncompress**

Purpose: Expand compressed data.

XSI Synopsis: `uncompress [-cfv] [file...]`

Derivation: First released in Issue 4.

Issue 7: SD5-XCU-ERN-26 is applied, clarifying that this utility is allowed to break the Utility Syntax Guidelines by having ten letters in its name.

**unexpand**

Purpose: Convert spaces to tabs.

Synopsis: `unexpand [-a|-t tablist] [file...]`

Derivation: First released in Issue 4.

Issue 7: The *unexpand* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

**unget**

Purpose: Undo a previous get of an SCCS file (**DEVELOPMENT**).

XSI Synopsis: `unget [-ns] [-r SID] file...`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**uniq**

Purpose: Report or filter out repeated lines in a file.

Synopsis: `uniq [-c|-d|-u] [-f fields] [-s char] [input_file [output_file]]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that '+' may be recognized as an option delimiter in the OPTIONS section.

Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDOUT section to reflect that standard output is also used if an *output_file* operand is '-' and the implementation treats the '-' as meaning standard output.

Austin Group Interpretation 1003.1-2001 #133 is applied, clarifying that the trailing <newline> of each line in the input is ignored when doing comparisons.

**unlink**

Purpose: Call the *unlink*() function.

XSI Synopsis: `unlink file`

Derivation: First released in Issue 5.

Issue 7: No functional changes are made in this issue.

**uucp**

Purpose: System-to-system copy.

UU Synopsis: `uucp [-cCdfjmr] [-n user] source-file... destination-file`

Derivation: First released in Issue 2.

Issue 7: SD5-XCU-ERN-46 is applied, moving this utility to the UUCP Utilities Option Group.

**uudecode**

Purpose: Decode a binary file.

Synopsis: `uudecode [–o outfile] [file]`

Derivation: First released in Issue 4.

Issue 7: The *uudecode* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

**uuencode**

Purpose: Encode a binary file.

Synopsis: `uuencode [–m] [file] decode_pathname`

Derivation: First released in Issue 4.

Issue 7: The *uuencode* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

**uustat**

Purpose: uucp status enquiry and job control.

UU Synopsis:

```
uustat [–q|–k jobid|–r jobid]
uustat [–s system] [–u user]
```

Derivation: First released in Issue 2.

Issue 7: SD5-XCU-ERN-46 is applied, moving this utility to the UUCP Utilities Option Group.

**uux**

Purpose: Remote command execution.

UU Synopsis:

```
uux [–jnp] command–string
```

Derivation: First released in Issue 2.

Issue 7: SD5-XCU-ERN-46 is applied, moving this utility to the UUCP Utilities Option Group.

**val**

Purpose: Validate SCCS files (**DEVELOPMENT**).

XSI Synopsis:

```
val –
val [–s] [–m name] [–r SID] [–y type] file...
```

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**vi**

Purpose: Screen-oriented (visual) display editor.

UP Synopsis: `vi [–rR] [–c command] [–t tagstring] [–w size] [file...]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that '+' may be recognized as an option delimiter in the OPTIONS section.

Austin Group Interpretation 1003.1-2001 #087 is applied, updating the Put from Buffer Before (**P**) command description to address multi-line requirements.

**wait**

Purpose: Await process completion.

Synopsis: `wait [pid...]`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**wc**

Purpose: Word, line, and byte or character count.

Synopsis: `wc [–c|–m] [–lw] [file...]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDIN section to reflect that standard input is also used if a *file* operand is '–' and the implementation treats the '–' as meaning standard input.

**what**

Purpose: Identify SCCS files (**DEVELOPMENT**).

XSI Synopsis: `what [–s] file...`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

**who**

Purpose: Display who is on the system.

XSI Synopsis: `who [–mTu] [–abdHlprt] [file]`

XSI
```
who [–mu] –s [–bHlprt] [file]
who –q [file]
who am i
who am I
```

Derivation: First released in Issue 2.

Issue 7: SD5-XCU-ERN-58 is applied, clarifying the **–b** option.

The *who* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

**write**

Purpose: Write to another user.

Synopsis: `write` *user_name* **[***terminal***]**

Derivation: First released in Issue 2.

Issue 7: The *write* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

**xargs**

Purpose: Construct argument lists and invoke utility.

XSI Synopsis: `xargs [–ptx] [–E` *eofstr*`] [–I` *replstr*`|–L` *number*`|–n` *number*`] [–s` *size*`] [`*utility* **[***argument*`...]]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #123 is applied, changing the description of the *xargs* **–I** option.

SD5-XCU-ERN-68 is applied, changing requirements related to the **–x** option and changing the SYNOPSIS to show that the **–I**, **–L** and **–n** options are mutually exclusive.

SD5-XCU-ERN-128 is applied, clarifying the DESCRIPTION of the logical end-of-file string.

SD5-XCU-ERN-132 is applied, updating the EXAMPLES section to demonstrate how to quote *xargs* input appropriately, and the use of **–E** "" to prevent accidental logical end-of-file processing.

**yacc**

Purpose: Yet another compiler compiler (**DEVELOPMENT**).

CD Synopsis: `yacc [–dltv] [–b` *file_prefix*`] [–p` *sym_prefix*`]` *grammar*

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #190 is applied, clarifying the requirements for generated code to conform to the IEEE Std 1003.1i-1995.

Austin Group Interpretation 1003.1-2001 #191 is applied, clarifying the handling of C-language trigraphs and curly brace preprocessing tokens.

SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply (options can be interspersed with operands).

**zcat**

Purpose: Expand and concatenate data.

XSI Synopsis:

```
zcat [file...]
```

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

Chapter 13

# Headers Migration

## 13.1 Introduction

This chapter contains a section for each header defined in XBD, Issue 7. Each section contains the SYNOPSIS, gives the derivation of the header, and identifies syntax and semantic changes made to the header in Issue 7 (if any). Only changes that might affect an application programmer are identified.

## 13.2 Headers

**<aio.h>**

Purpose: Asynchronous input and output.

Synopsis: `#include <aio.h>`

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

Issue 7: The **<aio.h>** header is moved from the Asynchronous Input and Output option to the Base.

This reference page is clarified with respect to macros and symbolic constants, and type and structure definitions from other headers are added.

**<arpa/inet.h>**

Purpose: Definitions for Internet operations.

Synopsis: `#include <arpa/inet.h>`

Derivation: First released in Issue 6. Derived from the Commands and Utilities, Issue 5 (XCU5).

Issue 7: No functional changes are made in this issue.

**<assert.h>**

Purpose: Verify program assertion.

Synopsis: `#include <assert.h>`

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

Issue 7: No functional changes are made in this issue.

**<complex.h>**

Purpose: Complex arithmetic.

Synopsis: `#include <complex.h>`

Derivation: First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

Issue 7: No functional changes are made in this issue.

**<cpio.h>**

Purpose: Cpio archive values.

Synopsis: `#include <cpio.h>`

Derivation: First released in the . Derived from the IEEE Std 1003.1-1988 (POSIX.1).

Issue 7: The **<cpio.h>** header is moved from the XSI option to the Base.

This reference page is clarified with respect to macros and symbolic constants.

**<ctype.h>**

Purpose: Character types.

Synopsis: `#include <ctype.h>`

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

Issue 7: The **_l*() functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

**<dirent.h>**

Purpose: Format of directory entries.

Synopsis: `#include <dirent.h>`

Derivation: First released in Issue 2.

Issue 7: The *alphasort*(), *dirfd*(), and *scandir*() functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

The *fdopendir*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Austin Group Interpretation 1003.1-2001 #110 is applied, clarifying that the **DIR** type may be defined as an incomplete type.

**<dlfcn.h>**

Purpose: Dynamic linking.

Synopsis: `#include <dlfcn.h>`

Derivation: First released in Issue 5.

Issue 7: The **<dlfcn.h>** header is moved from the XSI option to the Base.

This reference page is clarified with respect to macros and symbolic constants.

**<errno.h>**

Purpose: System error numbers.

Synopsis: `#include <errno.h>`

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

Issue 7: Austin Group Interpretation 1003.1-2001 #050 is applied, allowing [ENOTSUP] and [EOPNOTSUPP] to be the same values.

The [ENOTRECOVERABLE] and [EOWNERDEAD] errors are added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Functionality relating to the XSI STREAMS option is marked obsolescent.

Functionality relating to the Threads option is moved to the Base.

This reference page is clarified with respect to macros and symbolic constants.

**<fcntl.h>**

Purpose: File control options.

Synopsis: `#include <fcntl.h>`

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

Issue 7: Austin Group Interpretation 1003.1-2001 #144 is applied, adding the O_TTY_INIT flag.

Austin Group Interpretation 1003.1-2001 #171 is applied, adding the F_DUPFD_CLOEXEC and O_CLOEXEC flags.

The *openat*( ) function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

The O_EXEC and O_SEARCH flags are added.

Additional flags are added to support *faccessat*( ), *fchmodat*( ), *fchownat*( ), *fstatat*( ), *linkat*( ), *openat*( ), and *unlinkat*( ).

This reference page is clarified with respect to macros and symbolic constants.

Changes are made related to support for finegrained timestamps.

**<fenv.h>**

Purpose: Floating-point environment.

Synopsis: `#include <fenv.h>`

Derivation: First released in Issue 6. Included for alignment with the ISO/IEC 9899: 1999 standard.

Issue 7: ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #37 (SD5-XBD-ERN-49) is applied, clarifying that if no floating-point exception macros are defined by the implementation, FE_ALL_EXCEPT shall be defined as zero.

ISO/IEC 9899: 1999 standard, Technical Corrigendum 3 #36 is applied, requiring that the floating-point exception macros expand to integer constant expressions with values that are bitwise-distinct.

SD5-XBD-ERN-48 and SD5-XBD-ERN-69 are applied, clarifying that implementations which support the IEC 60559 Floating-Point option are required

to define all five floating-point exception macros and all four rounding direction macros.

This reference page is clarified with respect to macros and symbolic constants.

**<float.h>**

Purpose: Floating types.

Synopsis: `#include <float.h>`

Derivation: First released in Issue 4. Derived from the IEEE Std 1003.1i-1995.

Issue 7: ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #4 (SD5-XBD-ERN-50) is applied, clarifying that an implementation may give zero and non-numeric values, such as infinities and NaNs, a sign, or may leave them unsigned.

ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #5 (SD5-XBD-ERN-51) is applied, extending the text concerning floating-point accuracy to cover conversion between floating-point internal representations and string representations performed by the functions in **<stdio.h>**, **<stdlib.h>**, and **<wchar.h>**.

**<fmtmsg.h>**

Purpose: Message display structures.

XSI Synopsis: `#include <fmtmsg.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: This reference page is clarified with respect to macros and symbolic constants.

**<fnmatch.h>**

Purpose: Filename-matching types.

Synopsis: `#include <fnmatch.h>`

Derivation: First released in Issue 4. Derived from the .

Issue 7: The obsolescent FNM_NOSYS constant is removed.

This reference page is clarified with respect to macros and symbolic constants.

**<ftw.h>**

Purpose: File tree traversal.

XSI Synopsis: `#include <ftw.h>`

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

Issue 7: The *ftw*( ) function is marked obsolescent.

This reference page is clarified with respect to macros and symbolic constants.

**<glob.h>**

Purpose: Pathname pattern-matching types.

Synopsis: `#include <glob.h>`

Derivation: First released in Issue 4. Derived from the .

Issue 7: SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t** type.

The obsolescent GLOB_NOSYS constant is removed.

This reference page is clarified with respect to macros and symbolic constants.

**<grp.h>**

Purpose: Group structure.

Synopsis: `#include <grp.h>`

Derivation: First released in Issue 1.

Issue 7: SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t** type.

**<iconv.h>**

Purpose: Codeset conversion facility.

Synopsis: `#include <iconv.h>`

Derivation: First released in Issue 4.

Issue 7: SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t** type.

The **<iconv.h>** header is moved from the XSI option to the Base.

**<inttypes.h>**

Purpose: Fixed size integer types.

Synopsis: `#include <inttypes.h>`

Derivation: First released in Issue 5.

Issue 7: No functional changes are made in this issue.

**<iso646.h>**

Purpose: Alternative spellings.

Synopsis: `#include <iso646.h>`

Derivation: First released in Issue 5. Derived from .

Issue 7: No functional changes are made in this issue.

**<langinfo.h>**

Purpose: Language information constants.

Synopsis: `#include <langinfo.h>`

Derivation: First released in Issue 2.

Issue 7: The **<langinfo.h>** header is moved from the XSI option to the Base.

The *nl_langinfo_l*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

This reference page is clarified with respect to macros and symbolic constants, and a reference to **<locale.h>** for the **locale_t** type is added.

**<libgen.h>**

Purpose: Definitions for pattern matching functions.

XSI Synopsis: `#include <libgen.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

**<limits.h>**

Purpose: Implementation-defined constants.

Synopsis: `#include <limits.h>`

Derivation: First released in Issue 1.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

Austin Group Interpretation 1003.1-2001 #173 is applied, updating the descriptions of {TRACE_EVENT_NAME_MAX} and {TRACE_NAME_MAX} to not include the terminating null.

SD5-XBD-ERN-36 is applied, changing the description of {RE_DUP_MAX} to clarify that it applies to both BREs and EREs.

{NL_NMAX} is removed; it should have been removed in Issue 6.

The Trace option values are marked obsolescent.

The {ATEXIT_MAX}, {LONG_BIT}, {NL_MSGMAX}, {NL_SETMAX}, {NL_TEXTMAX}, and {WORD_BIT} values are moved from the XSI option to the Base.

Functionality relating to the Asynchronous Input and Output, Realtime Signals Extension, Threads, and Timers options is moved to the Base.

This reference page is clarified with respect to macros and symbolic constants.

**<locale.h>**

Purpose: Category macros.

Synopsis: `#include <locale.h>`

Derivation: First released in Issue 3.

Included for alignment with the IEEE Std 1003.1i-1995.

Issue 7: The *duplocale*(), *freelocale*(), *newlocale*(), and *uselocale*() functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

This reference page is clarified with respect to macros and symbolic constants.

**<math.h>**

Purpose: Mathematical declarations.

Synopsis: `#include <math.h>`

Derivation: First released in Issue 1.

Issue 7: ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #47 (SD5-XBD-ERN-52) is applied, updating the wording of the FP_FAST_FMA macro to require that it expands to the integer constant 1 if it is defined.

The MAXFLOAT constant is marked obsolescent. Applications should use FLT_MAX as described in the **<float.h>** header instead.

This reference page is clarified with respect to macros and symbolic constants.

**<monetary.h>**

Purpose: Monetary types.

Synopsis: `#include <monetary.h>`

Derivation: First released in Issue 4.

Issue 7: The **<monetary.h>** header is moved from the XSI option to the Base.

The *strmon_l*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

A reference to **<locale.h>** for the **locale_t** type is added.

**<mqueue.h>**

Purpose: Message queues (**REALTIME**).

MSG Synopsis: `#include <mqueue.h>`

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

Issue 7: Type and structure definitions from other headers are added.

**<ndbm.h>**

Purpose: Definitions for ndbm database operations.

XSI Synopsis: `#include <ndbm.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: This reference page is clarified with respect to macros and symbolic constants.

**<netdb.h>**

Purpose: Definitions for network database operations.

Synopsis: `#include <netdb.h>`

Derivation: First released in Issue 6. Derived from the Commands and Utilities, Issue 5 (XCU5).

Issue 7: SD5-XBD-ERN-14 is applied, changing the description of the *s_port* member of the **servent** structure to clarify the way in which port numbers are converted to and from network byte order.

The obsolescent *h_errno* external integer, and the obsolescent *gethostbyaddr*() and *gethostbyname*() functions are removed, along with the HOST_NOT_FOUND, NO_DATA, NO_RECOVERY, and TRY_AGAIN macros.

This reference page is clarified with respect to macros and symbolic constants.

**<net/if.h>**

Purpose: Sockets local interfaces.

Synopsis: `#include <net/if.h>`

Derivation: First released in Issue 6. Derived from the Commands and Utilities, Issue 5 (XCU5).

Issue 7: This reference page is clarified with respect to macros and symbolic constants.

**<netinet/in.h>**

Purpose: Internet address family.

Synopsis: `#include <netinet/in.h>`

Derivation: First released in Issue 6. Derived from the Commands and Utilities, Issue 5 (XCU5).

Issue 7: This reference page is clarified with respect to macros and symbolic constants.

**<netinet/tcp.h>**

Purpose: Definitions for the Internet Transmission Control Protocol (TCP).

Synopsis: `#include <netinet/tcp.h>`

Derivation: First released in Issue 6. Derived from the Commands and Utilities, Issue 5 (XCU5).

Issue 7: This reference page is clarified with respect to macros and symbolic constants.

**<nl_types.h>**

Purpose: Data types.

Synopsis: `#include <nl_types.h>`

Derivation: First released in Issue 2.

Issue 7: The **<nl_types.h>** header is moved from the XSI option to the Base.

This reference page is clarified with respect to macros and symbolic constants.

**<poll.h>**

Purpose: Definitions for the *poll*( ) function.

Synopsis: `#include <poll.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: The **<poll.h>** header is moved from the XSI option to the Base.

**<pthread.h>**

Purpose: Threads.

Synopsis: `#include <pthread.h>`

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

Issue 7: SD5-XBD-ERN-55 is applied, adding the **restrict** keyword to the *pthread_mutex_timedlock*( ) function prototype so that it matches the definition in XSH.

Austin Group Interpretation 1003.1-2001 #048 is applied, reinstating the PTHREAD_RWLOCK_INITIALIZER symbol.

The **<pthread.h>** header is moved from the Threads option to the Base.

The PTHREAD_MUTEX_NORMAL, PTHREAD_MUTEX_ERRORCHECK, PTHREAD_MUTEX_RECURSIVE, and PTHREAD_MUTEX_DEFAULT extended mutex types are moved from the XSI option to the Base.

The PTHREAD_MUTEX_ROBUST and PTHREAD_MUTEX_STALLED symbols and the *pthread_mutex_consistent*( ), *pthread_mutexattr_getrobust*( ), and *pthread_mutexattr_setrobust*( ) functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Functionality relating to the Thread Priority Protection and Thread Priority Inheritance options is changed to be Non-Robust Mutex or Robust Mutex Priority Protection and Non-Robust Mutex or Robust Mutex Priority Inheritance, respectively.

This reference page is clarified with respect to macros and symbolic constants.

**<pwd.h>**

Purpose: Password structure.

Synopsis: `#include <pwd.h>`

Derivation: First released in Issue 1.

Issue 7: SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t** type.

**<regex.h>**

Purpose: Regular expression matching types.

Synopsis: `#include <regex.h>`

Derivation: First released in Issue 4.

Originally derived from the .

Issue 7: SD5-XBD-ERN-60 is applied, removing the requirement that the type **regoff_t** can hold the largest value that can be stored in type **off_t**, and adding the requirement that the type **regoff_t** can hold the largest value that can be stored in type **ptrdiff_t**.

The obsolescent REG_ENOSYS constant is removed.

This reference page is clarified with respect to macros and symbolic constants.

**<sched.h>**

Purpose: Execution scheduling.

Synopsis: `#include <sched.h>`

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

Issue 7: Austin Group Interpretation 1003.1-2001 #064 is applied, correcting the option markings.

The **<sched.h>** header is moved from the Threads option to the Base.

Definitions for the **pid_t** and **time_t** types and the **timespec** structure are added.

**<search.h>**

Purpose: Search tables.

XSI Synopsis: `#include <search.h>`

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

Issue 7: No functional changes are made in this issue.

**<semaphore.h>**

Purpose: Semaphores.

Synopsis: `#include <semaphore.h>`

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

Issue 7: SD5-XBD-ERN-57 is applied, allowing the header to make visible symbols from the **<time.h>** header.

The **<semaphore.h>** header is moved from the Semaphores option to the Base.

This reference page is clarified with respect to macros and symbolic constants.

**<setjmp.h>**

Purpose: Stack environment declarations.

Synopsis: `#include <setjmp.h>`

Derivation: First released in Issue 1.

Issue 7: No functional changes are made in this issue.

**<signal.h>**

Purpose: Signals.

Synopsis: `#include <signal.h>`

Derivation: First released in Issue 1.

Issue 7: SD5-XBD-ERN-39 is applied, removing the **sigstack** structure which should have been removed at the same time as the LEGACY *sigstack*( ) function.

SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t** type.

Austin Group Interpretation 1003.1-2001 #034 is applied, moving SIGPOLL from the XSI option to the XSI STREAMS option.

The *psiginfo*( ) and *psignal*( ) functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Functionality relating to the XSI STREAMS option is marked obsolescent.

The SA_RESETHAND, SA_RESTART, SA_NOCLDWAIT, and SA_NODEFER constants are moved from the XSI option to the Base.

Functionality relating to the Realtime Signals Extension option is moved to the Base.

This reference page is clarified with respect to macros and symbolic constants, and type and structure definitions from other headers are added.

The descriptions of SIGRTMIN and SIGRTMAX are updated to clarify that they expand to positive integer expressions with type **int**, but which need not be constant expressions.

The APPLICATION USAGE section is updated to describe the conditions under which the *si_pid* and *si_uid* members of **siginfo_t** are required to be valid.

**<spawn.h>**

Purpose: Spawn (**ADVANCED REALTIME**).

SPN Synopsis: `#include <spawn.h>`

Derivation: First released in Issue 6. Included for alignment with IEEE Std 1003.1d-1999.

Issue 7: This reference page is clarified with respect to macros and symbolic constants, and type and structure definitions from other headers are added.

**<stdarg.h>**

Purpose: Handle variable argument list.

Synopsis: `#include <stdarg.h>`

Derivation: First released in Issue 4. Derived from the IEEE Std 1003.1b-1993.

Issue 7: No functional changes are made in this issue.

**<stdbool.h>**

Purpose: Boolean type and values.

Synopsis: `#include <stdbool.h>`

Derivation: First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

Issue 7: No functional changes are made in this issue.

**<stddef.h>**

Purpose: Standard type definitions.

Synopsis: `#include <stddef.h>`

Derivation: First released in Issue 4. Derived from the IEEE Std 1003.1b-1993.

Issue 7: This reference page is clarified with respect to macros and symbolic constants.

SD5-XBD-ERN-53 is applied, updating the definition of **wchar_t** to align with ISO/IEC 9899:1999 standard, Technical Corrigendum 3 in relation to the __STDC_MB_MIGHT_NEQ_WC__ indicator macro.

**<stdint.h>**

Purpose: Integer types.

Synopsis: `#include <stdint.h>`

Derivation: First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

Issue 7: ISO/IEC 9899:1999 standard, Technical Corrigendum 3 #40 is applied, requiring the argument to the *INT*_C*( ) macros to be an unsuffixed integer constant.

SD5-XBD-ERN-67 is applied, updating the RATIONALE to clarify that {SCHAR_MIN} has the value −128.

**<stdio.h>**

Purpose: Standard buffered input/output.

Synopsis: `#include <stdio.h>`

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

Issue 7: Austin Group Interpretation 1003.1-2001 #172 is applied, adding rationale about a conflict for the definition of {TMP_MAX} with the IEEE Std 1003.1i-1995 and the related ISO C defect report.

SD5-XBD-ERN-99 is applied, adding APPLICATION USAGE about {FOPEN_MAX} and the use of file descriptors not associated with streams.

The *dprintf*(), *fmemopen*(), *getdelim*(), *getline*(), *open_memstream*(), and *vdprintf*() functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

The *renameat*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

The *gets*(), *tmpnam*(), and *tempnam*() functions and the L_tmpnam macro are marked obsolescent.

This reference page is clarified with respect to macros and symbolic constants, and a reference to **<sys/types.h>** for the **off_t** type is added.

**<stdlib.h>**

Purpose: Standard library definitions.

Synopsis: `#include <stdlib.h>`

Derivation: First released in Issue 3.

Issue 7: The LEGACY functions are removed.

The *mkdtemp*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

The *rand_r*() function is marked obsolescent.

This reference page is clarified with respect to macros and symbolic constants.

The type of the first argument to *setstate*() is changed from **const char** * to **char** *.

**<string.h>**

Purpose: String operations.

Synopsis: `#include <string.h>`

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

Issue 7: SD5-XBD-ERN-15 is applied, correcting the prototype for the *strerror_r*() function.

The *stpcpy*(), *stpncpy*(), *strndup*(), *strnlen*(), and *strsignal*() functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

The *strcoll_l*(), *strerror_l*(), and *strxfrm_l*() functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

This reference page is clarified with respect to macros and symbolic constants, and a reference to **<locale.h>** for the **locale_t** type is added.

**<strings.h>**

Purpose: String operations.

Synopsis: `#include <strings.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t** type.

The LEGACY functions are removed.

The **<strings.h>** header is moved from the XSI option to the Base.

The *strcasecmp_l*() and *strncasecmp_l*() functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

A reference to **<locale.h>** for the **locale_t** type is added.

**<stropts.h>**

Purpose: STREAMS interface (**STREAMS**).

OB XSR Synopsis: `#include <stropts.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: SD5-XBD-ERN-87 is applied, correcting an error in the **strrecvfd** structure.

The **<stropts.h>** header is marked obsolescent.

This reference page is clarified with respect to macros and symbolic constants.

**<sys/ipc.h>**

Purpose: XSI interprocess communication access structure.

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

Derivation: First released in Issue 2. Derived from System V Release 2.0.

Issue 7: This reference page is clarified with respect to macros and symbolic constants.

**<syslog.h>**

Purpose: Definitions for system error logging.

XSI Synopsis: `#include <syslog.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: This reference page is clarified with respect to macros and symbolic constants.

**<sys/mman.h>**

Purpose: Memory management declarations.

Synopsis: `#include <sys/mman.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: Functionality relating to the Memory Protection and Memory Mapped Files options is moved to the Base.

This reference page is clarified with respect to macros and symbolic constants.

**<sys/msg.h>**

Purpose: XSI message queue structures.

XSI Synopsis: `#include <sys/msg.h>`

Derivation: First released in Issue 2. Derived from System V Release 2.0.

Issue 7: Austin Group Interpretation 1003.1-2001 #179 is applied, clarifying that everything from **<sys/ipc.h>** is made visible by **<sys/msg.h>**.

This reference page is clarified with respect to macros and symbolic constants.

**<sys/resource.h>**

Purpose: Definitions for XSI resource operations.

XSI Synopsis: `#include <sys/resource.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: This reference page is clarified with respect to macros and symbolic constants.

**<sys/select.h>**

Purpose: Select types.

Synopsis: `#include <sys/select.h>`

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1g-2000.

Issue 7: This reference page is clarified with respect to macros and symbolic constants.

**<sys/sem.h>**

Purpose: XSI semaphore facility.

XSI Synopsis: `#include <sys/sem.h>`

Derivation: First released in Issue 2. Derived from System V Release 2.0.

Issue 7: Austin Group Interpretation 1003.1-2001 #179 is applied, clarifying that everything from **<sys/ipc.h>** is made visible by **<sys/sem.h>**.

This reference page is clarified with respect to macros and symbolic constants.

**<sys/shm.h>**

Purpose: XSI shared memory facility.

XSI Synopsis: `#include <sys/shm.h>`

Derivation: First released in Issue 2. Derived from System V Release 2.0.

Issue 7: Austin Group Interpretation 1003.1-2001 #179 is applied, clarifying that everything from **<sys/ipc.h>** is made visible by **<sys/shm.h>**.

This reference page is clarified with respect to macros and symbolic constants.

**<sys/socket.h>**

Purpose: Main sockets header.

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

Derivation: First released in Issue 6. Derived from the Commands and Utilities, Issue 5 (XCU5).

Issue 7: SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **ssize_t** type.

The MSG_NOSIGNAL symbolic constant is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

This reference page is clarified with respect to macros and symbolic constants, and a reference to **<sys/types.h>** for the **size_t** type is added.

**<sys/stat.h>**

Purpose: Data returned by the *stat*( ) function.

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

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

Issue 7: SD5-XSH-ERN-161 is applied, updating the DESCRIPTION to clarify that the descriptions of the interfaces should be consulted in order to determine which structure members have meaningful values.

The *fchmodat*( ), *fstatat*( ), *mkdirat*( ), *mkfifoat*( ), *mknodat*( ), and *utimensat*( ) functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

This reference page is clarified with respect to macros and symbolic constants.

Changes are made related to support for finegrained timestamps, and the *futimens*( ) function and the UTIME_NOW and UTIME_OMIT symbolic constants are added.

**<sys/statvfs.h>**

Purpose: VFS File System information structure.

Synopsis: `#include <sys/statvfs.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: The **<sys/statvfs.h>** header is moved from the XSI option to the Base.

This reference page is clarified with respect to macros and symbolic constants.

**<sys/time.h>**

Purpose: Time types.

XSI Synopsis: `#include <sys/time.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: This reference page is clarified with respect to macros and symbolic constants.

**<sys/times.h>**

Purpose: File access and modification times structure.

Synopsis: `#include <sys/times.h>`

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

Issue 7: No functional changes are made in this issue.

**<sys/types.h>**

Purpose: Data types.

Synopsis: `#include <sys/types.h>`

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

Issue 7: Austin Group Interpretation 1003.1-2001 #033 is applied, requiring **key_t** to be an arithmetic type.

The Trace option types are marked obsolescent.

The **clock_t** and **id_t** types are moved from the XSI option to the Base.

Functionality relating to the Barriers, Spin Locks, Timers, and Threads options is moved to the Base.

**<sys/uio.h>**

Purpose: Definitions for vector I/O operations.

XSI Synopsis: `#include <sys/uio.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

**<sys/un.h>**

Purpose: Definitions for UNIX domain sockets.

Synopsis: `#include <sys/un.h>`

Derivation: First released in Issue 6. Derived from the Commands and Utilities, Issue 5 (XCU5).

Issue 7: The value for {_POSIX_PATH_MAX} stated in APPLICATION USAGE is updated to 256.

**<sys/utsname.h>**

Purpose: System name structure.

Synopsis: `#include <sys/utsname.h>`

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

Issue 7: No functional changes are made in this issue.

**<sys/wait.h>**

Purpose: Declarations for waiting.

Synopsis: `#include <sys/wait.h>`

Derivation: First released in Issue 3.

Included for alignment with the IEEE Std 1003.1-1988 (POSIX.1).

Issue 7: The *waitid*() function and symbolic constants for its *options* argument are moved to the Base.

The description of the WNOHANG constant is clarified.

**<tar.h>**

Purpose: Extended tar definitions.

Synopsis: `#include <tar.h>`

Derivation: First released in Issue 3. Derived from the IEEE Std 1003.1-1988 (POSIX.1).

Issue 7: This reference page is clarified with respect to macros and symbolic constants.

**<termios.h>**

Purpose: Define values for termios.

Synopsis: `#include <termios.h>`

Derivation: First released in Issue 3.

Included for alignment with the .

Issue 7: Austin Group Interpretation 1003.1-2001 #144 is applied, moving functionality relating to the IXANY symbol from the XSI option to the Base.

This reference page is clarified with respect to macros and symbolic constants, and a reference to **<sys/types.h>** for the **pid_t** type is added.

**<tgmath.h>**

Purpose: Type-generic macros.

Synopsis: `#include <tgmath.h>`

Derivation: First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

Issue 7: Austin Group Interpretation 1003.1-2001 #184 is applied, clarifying the functions for which a corresponding type-generic macro exists with the same name as the function.

**<time.h>**

Purpose: Time types.

Synopsis: `#include <time.h>`

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

Issue 7: The *strftime_l*() function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

Functionality relating to the Timers option is moved to the Base.

This reference page is clarified with respect to macros and symbolic constants, and type and structure definitions from other headers are added.

The description of *getdate_err* is expanded to state that it is unspecified whether *getdate_err* is a macro or an identifier declared with external linkage, and whether or not it is a modifiable lvalue.

**<trace.h>**

Purpose: Tracing.

OB TRC Synopsis: `#include <trace.h>`

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t** type.

The **<trace.h>** header is marked obsolescent.

This reference page is clarified with respect to macros and symbolic constants.

**<ulimit.h>**

Purpose: Ulimit commands.

OB XSI Synopsis: `#include <ulimit.h>`

Derivation: First released in Issue 3.

Issue 7: The **<ulimit.h>** header is marked obsolescent.

**<unistd.h>**

Purpose: Standard symbolic constants and types.

Synopsis: `#include <unistd.h>`

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

Issue 7: Austin Group Interpretation 1003.1-2001 #026 is applied, clarifying the meanings of the values −1, 0, and greater than 0 for constants for Options and Option Groups, and making an undefined constant mean the same as the value −1.

Austin Group Interpretation 1003.1-2001 #047 is applied, adding the _CS_V7_ENV constant.

Austin Group Interpretation 1003.1-2001 #166 is applied to permit an additional compiler flag to enable threads.

Austin Group Interpretation 1003.1-2001 #178 is applied, clarifying the values allowed for _POSIX2_CHAR_TERM.

SD5-XBD-ERN-41 is applied, adding the _POSIX2_SYMLINKS constant.

SD5-XBD-ERN-77 is applied, moving _POSIX_VDISABLE out of Constants for Options and Option Groups, since its value does not follow the convention for those constants.

Symbols to support the UUCP Utilities option are added.

The variables for the supported programming environments are updated to be V7.

The LEGACY and obsolescent symbols are removed.

The *faccessat*(), *fchownat*(), *fexecve*(), *linkat*(), *readlinkat*(), *symlinkat*(), and *unlinkat*() functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

The _POSIX_TRACE* constants from the Trace option are marked obsolescent.

The _POSIX2_PBS* constants from the Batch Environment Services and Utilities option are marked obsolescent.

Functionality relating to the Asynchronous Input and Output, Barriers, Clock Selection, Memory Mapped Files, Memory Protection, Realtime Signals Extension, Semaphores, Spin Locks, Threads, Timeouts, and Timers options is moved to the Base.

Functionality relating to the Thread Priority Protection and Thread Priority Inheritance options is changed to be Non-Robust Mutex or Robust Mutex Priority Protection and Non-Robust Mutex or Robust Mutex Priority Inheritance, respectively.

The following symbolic constants are added:

_SC_THREAD_ROBUST_PRIO_INHERIT
_SC_THREAD_ROBUST_PRIO_PROTECT

This reference page is clarified with respect to macros and symbolic constants.

Changes are made related to support for finegrained timestamps.

**<utime.h>**

Purpose: Access and modification times structure.

OB Synopsis: `#include <utime.h>`

Derivation: First released in Issue 3.

Issue 7: The **<utime.h>** header is marked obsolescent.

**<utmpx.h>**

Purpose: User accounting database definitions.

XSI Synopsis: `#include <utmpx.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

**<wchar.h>**

Purpose: Wide-character handling.

Synopsis: `#include <wchar.h>`

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

**<wctype.h>**

Purpose: Wide-character classification and mapping utilities.

Synopsis: `#include <wctype.h>`

Derivation: First released in Issue 5. Derived from the .

Issue 7: The **_l*() functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

This reference page is clarified with respect to macros and symbolic constants.

**<wordexp.h>**

Purpose: Word-expansion types.

Synopsis: `#include <wordexp.h>`

Derivation: First released in Issue 4. Derived from the .

Issue 7: The obsolescent WRDE_NOSYS constant is removed.

This reference page is clarified with respect to macros and symbolic constants.

Chapter 14

# ISO C Migration

*This chapter is by Finnbarr P. Murphy. At the time of writing, Finnbarr was a software engineer in the Business Critical Systems Group (BCSG) at Compaq Computer Corporation in Nashua, New Hampshire.*

## 14.1 Introduction

The original ISO/IEC C language programming standard (the ) 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 , Multibyte Support Extension.[1]

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.

## 14.2 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.

### 14.2.1 New Keywords

The following new keywords were defined:

- **inline**
- **restrict**
- **_Bool**
- **_Complex**
- **_Imaginary**
- **long long**

1. Information about the ISO C Working Group (JTC1/SC22/WG14) can be found at: wwwold.dkuug.dk/JTC1/SC22/WG14/.

### 14.2.2 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**.

### 14.2.3 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*( ) |

### 14.2.4 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.

### 14.2.5 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.

### 14.2.6 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.

### 14.2.7 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**.

### 14.2.8 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.

### 14.2.9 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.

## 14.3 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**.

### 14.3.1 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.

## 14.4 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.

### 14.4.1 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]).

### 14.4.2 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.

### 14.4.3 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.

## 14.5 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.

### 14.5.1 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.

### 14.5.2 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.

### 14.5.3 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.

### 14.5.4 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.

### 14.5.5 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);
```

### 14.5.6 errno

For underflow, *errno* is no longer required to be set to [EDOM] or [ERANGE].

## 14.6 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;
```

### 14.6.1 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.

### 14.6.2 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. |

### 14.6.3 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.

### 14.6.4 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.

### 14.6.5 Minimum Maximum Line Length

The minimum maximum line length was increased from 254 to 4095.

### 14.6.6 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.

### 14.6.7 #line Directive

This directive now allows the specification of a line number up to 2**31–1. Previously the limit was 32 767.

### 14.6.8 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.

### 14.6.9 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.

### 14.6.10 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.

### 14.6.11 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.

### 14.6.12 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.

### 14.6.13 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.

## 14.7 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.

## 14.8 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

### 14.8.1 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.

### 14.8.2 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**

### 14.8.3 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**

### 14.8.4 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.

### 14.8.5 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);
```

### 14.8.6 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.

### 14.8.7 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**.

## 14.9 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.

### 14.9.1 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);
```

### 14.9.2 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);
```

### 14.9.3 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);
```

### 14.9.4 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);
```

### 14.9.5 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.

## 14.10 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.

### 14.10.1 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);
```

### 14.10.2 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);
```

### 14.10.3 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);
```

### 14.10.4 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);
```

### 14.10.5 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);
```

### 14.10.6 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);
```

### 14.10.7 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);
```

### 14.10.8 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.

## 14.11 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
```

### 14.11.1 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.

### 14.11.2 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.

### 14.11.3 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**.

## 14.12 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.

### 14.12.1 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.

### 14.12.2 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.

## 14.13 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 the IEEE Std 1003.1c-1995, 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);
```

### 14.13.1 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);
```

## 14.14 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.

# Index