Technical Standard

Extended API Set Part 1

The Open Group

Technical Standard

Extended API Set Part 1

ISBN: 1-931624-66-6
Document Number: C062

Published in the U.K. by The Open Group, October 2006.

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

The Open Group
Thames Tower
37-45 Station Road
Reading
Berkshire, RG1 1LX
United Kingdom

or by Electronic Mail to:

OGSpecs@opengroup.org

Contents

Preface

The Open Group

The Open Group is a vendor-neutral and technology-neutral consortium, whose vision of Boundaryless Information Flow will enable access to integrated information within and between enterprises based on open standards and global interoperability. The Open Group works with customers, suppliers, consortia, and other standards bodies. Its role is to capture, understand, and address current and emerging requirements, establish policies, and share best practices; to facilitate interoperability, develop consensus, and evolve and integrate specifications and Open Source technologies; to offer a comprehensive set of services to enhance the operational efficiency of consortia; and to operate the industry's premier certification service, including UNIX certification.

Further information on The Open Group is available at *www.opengroup.org*.

The Open Group has over 15 years' experience in developing and operating certification programs and has extensive experience developing and facilitating industry adoption of test suites used to validate conformance to an open standard or specification.

More information is available at *www.opengroup.org/certification*.

The Open Group publishes a wide range of technical documentation, the main part of which is focused on development of Technical and Product Standards and Guides, but which also includes white papers, technical studies, branding and testing documentation, and business titles. Full details and a catalog are available at *www.opengroup.org/bookstore*.

As with all *live* documents, Technical Standards and Specifications require revision to align with new developments and associated international standards. To distinguish between revised specifications which are fully backwards-compatible and those which are not:

- A new *Version* indicates there is no change to the definitive information contained in the previous publication of that title, but additions/extensions are included. As such, it *replaces* the previous publication.
- A new *Issue* indicates there is substantive change to the definitive information contained in the previous publication of that title, and there may also be additions/extensions. As such, both previous and new documents are maintained as current publications.

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published at *www.opengroup.org/corrigenda*.

This Document

This document has been prepared by The Open Group Base Working Group. The Open Group Base Working Group is considering submitting a number of API sets to the Austin Group as input to the revision of the Base Specifications, Issue 6.

This is the first document in that set.

Trademarks

Acknowledgements

The contributions of the following to the development of this document are gratefully acknowledged:

- The Open Group Base Working Group

Chapter 1

Introduction

1.1 Scope

The purpose of this document is to define a set of new API extensions to further increase application capture and hence portability for systems built upon the Single UNIX Specification, Version 3.

The scope of this set of extensions has been to consider interfaces from existing open source implementations, such as the GNU C library.

1.2 Relationship to Other Formal Standards

No decision has been made on whether these interfaces will be added to a future Technical Standard of The Open Group, how these interfaces would announce themselves in the name space, or whether related interfaces should be merged with existing reference pages. This Technical Standard is being forwarded to the Austin Group for consideration as input to the revision of the Base Specifications, Issue 6.

Chapter 2

Changes to the Base Definitions Volume

It is proposed that these additions comprise a new Option Group called Extended Interfaces.

2.1 Section 1.5.1, Codes

Add a new margin code as follows:

UX Extended Interfaces

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

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

Notes:

1. This section is repeated in XBD, XSH, and XCU and therefore will appear in XBD (Section 1.5.1), XSH (Section 1.8.1), and XCU (Section 1.8.1).
2. The use of UX as a margin code is a placeholder and may change in the final publication.

2.2 Section 3.362, Stream

Add *fmemopen*( ) and *open_memstream*( ) to the list of functions that can create a stream, marked with the UX margin legend and shaded.

2.3 Chapter 13, Headers

The following header file reference pages will need the following additions, marked with the UX margin legend and shaded as part of the Extended Interfaces Option Group.

<dirent.h>

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int alphasort(const struct dirent **, const struct dirent **);
int dirfd (DIR *);
int scandir (const char *, struct dirent ***,
    int (*) (const struct dirent *),
    int (*) (const struct dirent **, const struct dirent **));
```

<signal.h>

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
void psignal (int, const char *);
void psiginfo (siginfo_t *, const char *);
```

<stdio.h>

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int dprintf (int, const char *, ...);
FILE *fmemopen(void *,size_t, const char *);
ssize_t getdelim (char **, size_t *, int, FILE *);
ssize_t getline (char **, size_t *, FILE *);
FILE *open_memstream(char **, size_t *);
```

<stdlib.h>

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
char *mkdtemp(char *);
```

<string.h>

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
char *stpcpy (char *, const char *);
char *stpncpy (char *, const char *, size_t);
char *strndup (const char *, size_t);
size_t strnlen (const char *, size_t);
char *strsignal(int signum);
```

<wchar.h>

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
size_t mbsnrtowcs (wchar_t *, const char **, size_t, size_t, mbstate_t *);
wchar_t *wcpcpy (wchar_t *, const wchar_t *);
wchar_t *wcpncpy (wchar_t *, const wchar_t *, size_t);
int wcscasecmp (const wchar_t *, const wchar_t *);
wchar_t *wcsdup (const wchar_t *);
int wcsncasecmp (const wchar_t *, const wchar_t *, size_t);
size_t wcsnlen (const wchar_t *, size_t);
size_t wcsnrtombs (char *, const wchar_t **, size_t, size_t, mbstate_t *);
```

Chapter 3

Changes to the Shell and Utilities Volume

It is proposed that the following changes are made to Chapter 4, Utilities, the *ls* command.

Note: All page and line numbers in this proposal refer to the Shell and Utilities volume of IEEE Std 1003.1-2001, 2004 Edition.

SYNOPSIS

In the SYNOPSIS section on Page 571, Line 22014 add the −**S** option by changing the SYNOPSIS from:

UX
```
ls [−CFRacdilqrtu1] [−H | −L ] [−fgmnopsx] [file...]
```

to:

UX
```
ls [−CFRSacdilqrtu1] [−H | −L ] [−fgmnopsx] [file...]
```

OPTIONS

In the OPTIONS section after Page 571, Line 22054 add a description of the new −**S** option as follows:

−**S** Sort with the primary key being file size (in decreasing order) and the secondary key being filename in the collating sequence (in increasing order).

On Page 572, Lines 22065-22067 specify the interaction between the −**f** and −**S** options by changing the description of the −**f** option from:

UX −**f** Force each argument to be interpreted as a directory and list the name found in each slot. This option shall turn off −**l**, −**t**, −**s**, and −**r**, and shall turn on −**a**; the order is the order in which entries appear in the directory.

to:

UX −**f** Force each argument to be interpreted as a directory and list the name found in each slot. This option shall turn off −**l**, −**t**, −**S**, −**s**, and −**r**, and shall turn on −**a**; the order is the order in which entries appear in the directory.

On Page 572, Line 22082 note the interaction between −**S** and −**r** by changing the description of the −**r** option from:

−**r** Reverse the order of the sort to get reverse collating sequence or oldest first.

to:

−**r** Reverse the order of the sort to get reverse collating sequence oldest first, or smallest file size first depending on the other options given.

On Page 572, Lines 22092-22094 add −**t** and −**S** to the list of mutually-exclusive options by changing from:

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

to:

UX Specifying more than one of the options in the following mutually-exclusive pairs shall not be considered an error: −**C** and −**l** (ell), −**m** and −**l** (ell), −**x** and −**l** (ell), −**C** and −**1** (one), −**H** and −**L**, −**c** and −**u**, −**t** and −**S**. The last option specified in each pair shall determine the output format.

RATIONALE

Add a new paragraph after Page 577, Line 22291:

The −**S** option was added to the standard in Issue 7, but had been provided by several implementations for many years. The description given in the standard documents historic practice, but does not match much of the documentation that described its behavior. Historical documentation typically described it as something like:

−**S**	Sort by size (largest size first) instead of by name. Special character devices (listed last) are sorted by name.

even though the file type was never considered when sorting the output. Character special files do typically sort close to the end of the list because their file size on most implementations is zero. But they are sorted alphabetically with any other files that happen to have the same file size (zero), not sorted separately and added to the end.

Chapter 4

Changes to the System Interfaces Volume

It is proposed that the following changes are made to Section 2.5, Standard I/O Streams.

Note: The text described in this proposal refers to the System Interfaces volume of IEEE Std 1003.1, 2004 Edition.

4.1 Section 2.5, Standard I/O Streams

Change the first sentence to:

UX A stream is associated with an external file (which may be a physical device) or memory buffer
UX by ''opening'' a file or buffer. This may involve ''creating'' a new file.

Add the following to the end:

UX A stream associated with a memory buffer shall have the same operations for text files that a stream associated with an external file would have. In addition, the stream orientation shall be determined in exactly the same fashion.

Input and output operations on a stream associated with a memory buffer by a call to *fmemopen*( ) shall be constrained by the implementation to take place within the bounds of the memory buffer. In the case of a stream opened by *open_memstream*( ) or *open_wmemstream*( ), the memory area shall grow dynamically to accommodate write operations as necessary. For output, data is moved from the buffer provided by *setvbuf*( ) to the memory stream during a flush or close operation.

4.2 fclose( ) and fflush( )

Add the following to the ''shall fail'' section within the ERRORS section:

[ENOMEM] The underlying stream was created by *open_memstream*( ) or *open_wmemstream*( ) and insufficient memory is available.

Update the [ENOSPC] error condition to:

[ENOSPC] There was no free space remaining on the device containing the file or in the buffer used by the *fmemopen*( ) function.

4.3 Reference Pages

Add the following new system interface descriptions in alphabetical order with the existing system interface descriptions in Chapter 3, System Interfaces.

NAME

alphasort, scandir — scan a directory

SYNOPSIS

UX
```
#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 **));
```

DESCRIPTION

The *alphasort*( ) function can be used as the comparison function for the *scandir*( ) function to sort the directory entries into alphabetical order, as if by the *strcoll*( ) function. Its parameters are the two directory entries, *d1* and *d2*, to compare.

The *scandir*( ) function shall scan the directory *dir*, calling the function referenced by *sel* on each directory entry. Entries for which the function referenced by *sel* returns non-zero shall be stored in strings allocated as if by a call to *malloc*( ), and sorted using *qsort*( ) with the comparison function *compar*( ), and collected in array *namelist* which shall be allocated as if by a call to *malloc*( ). If *sel* is a null pointer, all entries shall be selected.

RETURN VALUE

Upon successful completion, *alphasort*( ) shall return an integer greater than, equal to, or less than 0, according to whether the name of the directory entry pointed to by *d1* is lexically greater than, equal to, or less than the directory pointed to by *d2* when both are interpreted as appropriate to the current locale. There is no return value reserved to indicate an error.

Upon successful completion, the *scandir*( ) function shall return the number of entries in the array and a pointer to the array through the parameter *namelist*. Otherwise, the *scandir*( ) function shall return –1.

ERRORS

The *scandir*( ) function shall fail if:

[EACCES]	Search permission is denied for the component of the path prefix of *dir* or read permission is denied for *dir*.
[ELOOP]	A loop exists in symbolic links encountered during resolution of the *dir* argument.
[ENAMETOOLONG]	The length of the *dir* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.
[ENOENT]	A component of *dir* does not name an existing directory or *dir* is an empty string.
[ENOMEM]	Insufficient storage space is available.
[ENOTDIR]	A component of *dir* is not a directory.

The *scandir*( ) function may fail if:

[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *dir* argument.
[EMFILE]	{OPEN_MAX} file descriptors are currently open in the calling process.

[ENAMETOOLONG]
As a result of encountering a symbolic link in resolution of the *dir* argument, the length of the substituted pathname string exceeded {PATH_MAX}.

[ENFILE] Too many files are currently open in the system.

EXAMPLES

An example to print the files in the current directory:

```
#include <dirent.h>
#include <stdio.h>
...
struct dirent **namelist;
int i,n;

    n = scandir(".", &namelist, 0, alphasort);
    if (n < 0)
        perror("scandir");
    else {
        for (i = 0; i < n; i++) {
            printf("%s\n", namelist[i]->d_name);
            free(namelist[i]);
            }
        }
    free(namelist);
...
```

APPLICATION USAGE

These functions are part of the Extended Interfaces Option Group and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

compar( ), *malloc*( ), *qsort*( ), *strcoll*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<dirent.h>**

CHANGE HISTORY

First released in Issue X.

NAME

dirfd — extract the file descriptor used by a DIR stream

SYNOPSIS

UX
```
#include <dirent.h>

int dirfd(DIR *dirp);
```

DESCRIPTION

The *dirfd*( ) function shall return a file descriptor referring to the same directory as the *dirp* argument. This file descriptor shall be closed by a call to *closedir*( ). The behavior of future calls to *readdir*( ) and *readdir_r*( ) is undefined if the application attempts to alter the file position indicator using the returned file descriptor. The behavior of future calls to *closedir*( ), *readdir*( ), and *readdir_r*( ) is undefined if the application attempts to close the file descriptor.

RETURN VALUE

Upon successful completion, the *dirfd*( ) function shall return an integer which contains a file descriptor for the stream pointed to by *dirp*. Otherwise, it shall return –1 and may set *errno* to indicate the error.

ERRORS

The *dirfd*( ) function may fail if:

[EINVAL]	The *dirp* argument does not refer to a valid directory stream.
[ENOTSUP]	The implementation does not support the association of a file descriptor with a directory.

EXAMPLES

None.

APPLICATION USAGE

The *dirfd*( ) function is part of the Extended Interfaces Option Group and need not be available on all implementations.

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

RATIONALE

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*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

closedir( ), *diropen*( ), *fchdir*( ), *fileno*( ), *open*( ), *opendir*( ), *readdir*( ), *readdir_r*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<dirent.h>**, **<stdio.h>**

CHANGE HISTORY

First released in Issue X.

NAME

dprintf — formatted output conversion to a file descriptor

SYNOPSIS

UX
```
#include <stdio.h>

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

DESCRIPTION

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

RETURN VALUE

Upon successful completion, the *dprintf*( ) function shall return the number of bytes transmitted. If an output error was encountered, it shall return a negative value.

ERRORS

Refer to *fprintf*( ).

In addition, the *dprintf*( ) function may fail if:

[EBADF]	The *fildes* argument is not a valid file descriptor.

EXAMPLES

None.

APPLICATION USAGE

The *dprintf*( ) function is part of the Extended Interfaces Option Group and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fprintf( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

CHANGE HISTORY

First released in Issue X.

NAME

fmemopen — open a memory buffer stream

SYNOPSIS

UX
```
#include <stdio.h>

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

DESCRIPTION

The *fmemopen*( ) function shall associate the buffer given by the *buf* and *size* arguments with a stream. The *buf* argument shall be either a null pointer or point to a buffer that is at least *size* bytes long.

The *mode* argument is a character string having one of the following values:

r or *rb*	Open the stream for reading.
w or *wb*	Open the stream for writing.
a or *ab*	Append; open the stream for writing at the first null byte.
r+ or *rb*+ or *r*+*b*	Open the stream for update (reading and writing).
w+ or *wb*+ or *w*+*b*	Open the stream for update (reading and writing). Truncate the buffer contents.
a+ or *ab*+ or *a*+*b*	Append; open the stream for update (reading and writing); the initial position is at the first null byte.

The character `'b'` shall have no effect.

If a null pointer is specified as the *buf* argument, *fmemopen*( ) shall allocate *size* bytes of memory as if by a call to *malloc*( ). This buffer shall be automatically freed when the stream is closed. Because this feature is only useful when the stream is opened for updating (because there is no way to get a pointer to the buffer) the *fmemopen*( ) call may fail if the *mode* argument does not include a `'+'`.

The stream maintains a current position in the buffer. This position is initially set to either the beginning of the buffer (for *r* and *w* modes) or to the first null byte in the buffer (for *a* modes). If no null byte is found in append mode, the initial position is set to one byte after the end of the buffer.

If *buf* is a null pointer, the initial position shall always be set to the beginning of the buffer.

The stream also maintains the size of the current buffer contents. For modes *r* and *r*+ the size is set to the value given by the *size* argument. For modes *w* and *w*+ the initial size is zero and for modes *a* and *a*+ the initial size is either the position of the first null byte in the buffer or the value of the *size* argument if no null byte is found.

A read operation on the stream cannot advance the current buffer position behind the current buffer size. Reaching the buffer size in a read operation counts as ‘‘end-of-file’’. Null bytes in the buffer have no special meaning for reads. The read operation starts at the current buffer position of the stream.

A write operation starts either at the current position of the stream (if mode has not specified `'a'` as the first character) or at the current size of the stream (if mode had `'a'` as the first character). If the current position at the end of the write is larger than the current buffer size, the current buffer size is set to the current position. A write operation on the stream cannot advance the current buffer size behind the size given in the *size* argument.

When a stream open for writing is flushed or closed, a null byte is written at the current position or at the end of the buffer, depending on the size of the contents. If a stream open for update is flushed or closed and the last write has advanced the current buffer size, a null byte is written at the end of the buffer if it fits.

An attempt to seek a memory buffer stream to a negative position or to a position larger than the buffer size given in the *size* argument shall fail.

RETURN VALUE

Upon successful completion, *fmemopen*( ) shall return a pointer to the object controlling the stream. Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

ERRORS

The *fmemopen*( ) function shall fail if:

[EINVAL]	The *size* argument specifies a buffer size of zero.

The *fmemopen*( ) function may fail if:

[EINVAL]	The value of the *mode* argument is not valid.
[EINVAL]	The *buf* argument is a null pointer and the *mode* argument does not include a '+' character.
[ENOMEM]	The *buf* argument is a null pointer and the allocation of a buffer of length *size* has failed.
[EMFILE]	{FOPEN_MAX} streams are currently open in the calling process.

EXAMPLES

```
#include <stdio.h>

static char buffer[] = "foobar";

int
main (void)
{
    int ch;
    FILE *stream;

    stream = fmemopen(buffer, strlen (buffer), "r");
    if (stream == NULL)
        /* handle error */;

    while ((ch = fgetc(stream)) != EOF)
        printf("Got %c\n", ch);

    fclose(stream);
    return (0);
}
```

This program produces the following output:

```
Got f
Got o
Got o
Got b
Got a
Got r
```

APPLICATION USAGE

The *fmemopen*( ) function is part of the Extended Interfaces Option Group and need not be available on all implementations.

RATIONALE

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.

FUTURE DIRECTIONS

None.

SEE ALSO

fdopen( ), *fopen*( ), *freopen*( ), *malloc*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

CHANGE HISTORY

First released in Issue X.

NAME

getdelim, getline — read a delimited record from *stream*

SYNOPSIS

UX
```
#include <stdio.h>

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

ssize_t getline(char **lineptr, size_t *n, FILE *stream);
```

DESCRIPTION

The *getdelim*( ) function shall read from *stream* until it encounters a character matching the *delimiter* character. The argument *delimiter* (when converted to a **char**) shall specify the character that terminates the read process.

The *delimiter* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal value to the macro EOF. If the *delimiter* argument has any other value, the behavior is undefined.

The application shall ensure that **lineptr* is a valid argument that could be passed to the *free*( ) function. If **n* is non-zero, the application shall ensure that **lineptr* points to an object of size at least **n* bytes.

The size of the object pointed to by **lineptr* shall be increased to fit the incoming line, if it isn't already large enough. The characters read shall be stored in the string pointed to by the *lineptr* argument.

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

RETURN VALUE

Upon successful completion, the *getdelim*( ) function shall return the number of characters written into the buffer, including the delimiter character if one was encountered before EOF. Otherwise, it shall return –1 and set *errno* to indicate the error.

ERRORS

These functions shall fail if:

[EINVAL]	When *lineptr* or *n* are a null pointer.
[ENOMEM]	Insufficient memory is available.

These functions may fail if:

[EINVAL]	*stream* is not a valid file descriptor.
[EOVERFLOW]	More than {SSIZE_MAX} characters were read without encountering the *delimiter* character.

EXAMPLES

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

int main(void)
{
    FILE * fp;
    char * line = NULL;
    size_t len = 0;
    ssize_t read;
    fp = fopen("/etc/motd", "r");
    if (fp == NULL)
        exit(1);
    while ((read = getline(&line, &len, fp)) != -1) {
        printf("Retrieved line of length %zu :\n", read);
        printf("%s", line);
    }
    if (line)
        free(line);
    fclose(fp);
    return 0;
}
```

APPLICATION USAGE

These functions are part of the Extended Interfaces Option Group and need not be available on all implementations.

Setting **lineptr* to a null pointer and **n* to zero are allowed and a recommended way to start parsing a file.

RATIONALE

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.

FUTURE DIRECTIONS

None.

SEE ALSO

fgets( ), *free*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

CHANGE HISTORY

First released in Issue X.

NAME

mbsnrtowcs — convert a multi-byte string to a wide-character string

SYNOPSIS

UX
```
#include <wchar.h>

size_t mbsnrtowcs(wchar_t *restrict dst, const char **restrict src,
    size_t nmc, size_t len, mbstate_t *restrict ps);
```

DESCRIPTION

The *mbsnrtowcs*( ) function works like 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).

If *dst* is not a null pointer, then *mbsnrtowcs*( ) shall attempt to convert *nmc* bytes from the multi-byte string pointed to by *src* into a wide-character string starting at *dst*. No more than *len* wide characters shall be written to *dst*. The shift state, pointed at by *ps*, is updated by the conversion. Each conversion shall take place, as if by repeated calls to *mbrtowc*(*dest*, **src*, *n*, ps), where *n* is a positive number. As long as this call succeeds, it is repeated, each time incrementing *dst* by one and **src* by the number of bytes converted.

Conversion shall stop early if any of the following cases occurs:

1. An invalid sequence of bytes was encountered in the *src* buffer. Under these conditions **src* is left pointing to the bytes which caused the conversion to halt. –1 is returned, and *errno* is set to [EILSEQ].

2. Either the *nmc* limit has been reached, or *len* non-null wide characters have already been stored in *dst*. Here, **src* is left to point to the next multi-byte sequence that has not been converted, and the total number of wide characters written to *dst* is returned.

3. The conversion of the multi-byte buffer pointed to by *src* has been completed by encountering a null byte. In this case **src* is set to a null pointer, **ps* is returned to its initial state, and the number of wide characters written to *dst*, excluding the terminating null character, is returned.

When *dst* is a null pointer, the conversion proceeds as above, except that no wide characters are written to memory, and the *len* argument is ignored, so no destination length limit is imposed.

In either case, if *ps* is a null pointer, *mbsnrtowcs*( ) shall use its own internal **mbstate_t** object, which is initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object pointed to by *ps* shall be used to completely describe the current conversion state of the associated character sequence.

It is the responsibility of the calling program to ensure that *dst* is large enough to hold at least *len* wide characters.

RETURN VALUE

The *mbsnrtowcs*( ) function shall return the number of characters successfully converted, not including the terminating null (if any). If an error occurs, *mbsnrtowcs*( ) shall return –1 and may set *errno* to indicate the error.

ERRORS

The *mbsnrtowcs*( ) function may fail if:

[EILSEQ]	An invalid multi-byte sequence was encountered.

EXAMPLES

None.

APPLICATION USAGE

The *mbsnrtowcs*( ) function is part of the Extended Interfaces Option Group and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iconv( ), *mbsrtowcs*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue X.

NAME

mkdtemp — create a unique directory

SYNOPSIS

UX
```
#include <stdlib.h>

char *mkdtemp(char *template);
```

DESCRIPTION

The *mkdtemp*( ) function uses the contents of *template* to construct a unique directory name. The string provided in *template* shall be a filename ending with six trailing `'X'`s. The *mkdtemp*( ) function shall replace 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.

RETURN VALUE

Upon successful completion, the *mkdtemp*( ) function shall return a pointer to the string containing the directory name if it was created. Otherwise, it shall return a null pointer and shall set *errno* to indicate the error.

ERRORS

The *mkdtemp*( ) function shall fail if:

[EACCES]	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the directory to be created.
[EINVAL]	The string pointed to by *template* does not end in `"XXXXXX"`.
[ELOOP]	A loop exists in symbolic links encountered during resolution of the path of the directory to be created.
[EMLINK]	The link count of the parent directory would exceed {LINK_MAX}.
[ENAMETOOLONG]	The length of the *template* argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.
[ENOENT]	A component of the path prefix specified by the *template* argument does not name an existing directory or path is an empty string.
[ENOSPC]	The file system does not contain enough space to hold the contents of the new directory or to extend the parent directory of the new directory.
[ENOTDIR]	A component of the path prefix is not a directory.
[EROFS]	The parent directory resides on a read-only file system.

The *mkdtemp*( ) function may fail if:

[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the path of the directory to be created.
[ENAMETOOLONG]	As a result of encountering a symbolic link in resolution of the path of the directory to be created, the length of the substituted pathname string exceeded {PATH_MAX}.

EXAMPLES

None.

APPLICATION USAGE

The *mkdtemp*( ) function is part of the Extended Interfaces Option Group and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mkdir( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdlib.h>**

CHANGE HISTORY

First released in Issue X.

NAME

open_memstream, open_wmemstream — open a dynamic memory buffer stream

SYNOPSIS

UX

```
#include <stdio.h>

FILE *open_memstream(char **bufp, size_t *sizep);

#include <wchar.h>

FILE *open_wmemstream(wchar_t **bufp, size_t *sizep);
```

DESCRIPTION

The *open_memstream*( ) and *open_wmemstream*( ) functions shall create an I/O stream associated with a dynamically allocated memory buffer. The stream shall be opened for writing and shall be seekable.

The stream associated with a call to *open_memstream*( ) shall be byte-oriented.

The stream associated with a call to *open_wmemstream*( ) shall be wide-oriented.

The stream shall maintain a current position in the allocated buffer and a current buffer length. The position shall be initially set to zero (the start of the buffer). Each write to the stream shall start at the current position and move this position by the number of successfully written bytes for *open_memstream*( ) or the number of successfully written wide characters for *open_wmemstream*( ). The length shall be initially set to zero. If a write moves the position to a value larger than the current length, the current length shall be set to this position. In this case a null character for *open_memstream*( ) or a null wide character for *open_wmemstream*( ) shall be appended to the current buffer. For both functions the terminating null is not included in the calculation of the buffer length.

After a successful *fflush*( ) or *fclose*( ), the pointer referenced by *bufp* shall contain the address of the buffer, and the variable pointed to by *sizep* shall contain the number of successfully written bytes for *open_memstream*( ) or the number of successfully written wide characters for *open_wmemstream*( ). The buffer shall be terminated by a null character for *open_memstream*( ) or a null wide character for *open_wmemstream*( ).

After a successful *fflush*( ) the pointer referenced by *bufp* and the variable referenced by *sizep* remain valid only until the next write operation on the stream or a call to *fclose*( ).

RETURN VALUE

Upon successful completion, these functions shall return a pointer to the object controlling the stream. Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

ERRORS

These functions may fail if:

[EINVAL]	*bufp* or *sizep* are NULL.
[EMFILE]	{FOPEN_MAX} streams are currently open in the calling process.
[ENOMEM]	Memory for the stream or the buffer could not be allocated.

EXAMPLES

```
#include <stdio.h>
int main (void)
{
    FILE *stream;
    char *buf;
    size_t len;

    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);
    fseeko(stream, 0, SEEK_SET);
    fprintf(stream, "good-bye");
    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
```

APPLICATION USAGE

These functions are part of the Extended Interfaces Option Group and need not be available on all implementations.

The buffer created by these functions should be freed by the application after closing the stream, by means of a call to *free*( ).

RATIONALE

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.

FUTURE DIRECTIONS

None.

SEE ALSO

fclose( ), *fdopen*( ), *fflush*( ), *fopen*( ), *fmemopen*( ), *free*( ), *freopen*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<stdio.h>**

CHANGE HISTORY

First released in Issue X.

NAME

psiginfo, psignal — print signal information to standard error

SYNOPSIS

UX
```
#include <signal.h>

void psiginfo(siginfo_t *pinfo, const char *message);

void psignal(int signum, const char *message);
```

DESCRIPTION

The *psiginfo*( ) and *psignal*( ) functions shall print a message out on *stderr* associated with a signal number. If *message* is not null and is not the empty string, then the string pointed to by the *message* argument shall be printed first, followed by a colon, a space, and the signal description string indicated by *signum*, or by the signal associated with *pinfo*. If the *message* argument is null or points to an empty string, then only the signal description shall be printed. For *psiginfo*( ), the argument *pinfo* references a valid **siginfo_t** structure. For *psignal*( ), if *signum* is not a valid signal number, the behavior is implementation-defined.

RETURN VALUE

These functions shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Extended Interfaces Option Group and need not be available on all implementations.

RATIONALE

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>**.

FUTURE DIRECTIONS

None.

SEE ALSO

perror( ), *strsignal*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<signal.h>**

CHANGE HISTORY

First released in Issue X.

NAME

stpcpy — copy a string and return a pointer to the end of the result

SYNOPSIS

UX
```
#include <string.h>

char *stpcpy(char *restrict dst, const char *restrict src);
```

DESCRIPTION

The *stpcpy*( ) function shall be equivalent to *strcpy*( ), copying the string pointed to by *src* into the array pointed to by *dst*, with the exception that *stpcpy*( ) shall return a pointer to the terminating null byte in *dst*, rather than the beginning of this array, allowing succeeding calls to add additional text to the *dst* array.

If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

The *stpcpy*( ) function shall return a pointer to the terminating null byte at the end of the *dst* buffer. No return values are reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

The following example demonstrates the construction of 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;
}
```

APPLICATION USAGE

The *stpcpy*( ) function is part of the Extended Interfaces Option Group and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strcpy( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue X.

NAME

stpncpy — copy fixed length string, returning a pointer to the array end

SYNOPSIS

UX
```
#include <string.h>

char *stpncpy(char *restrict dst, const char *restrict src, size_t size);
```

DESCRIPTION

The *stpncpy*( ) function shall be equivalent to the *stpcpy*( ) function, with the added restriction that it shall copy at most *size* bytes from *src* into *dst*.

If *size* is less than or equal to the length of the string pointed to by *src* then no termination null byte shall be inserted into the *dst* array after the *size* bytes have been copied.

If *size* is greater than the length of the string pointed to by *src* then all of the bytes in *src* are copied into the *dst* array. As many terminating null bytes are inserted as are needed to bring the total bytes transferred equal to *size*.

If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

If a null byte is written to the destination, the *stpncpy*( ) function shall return the address of the first such null byte. Otherwise, it shall return &*src*[*size*]. No return values are reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

APPLICATION USAGE

The *stpncpy*( ) function is part of the Extended Interfaces Option Group and need not be available on all implementations.

Applications must provide the space in *dst* for the *size* bytes to be transferred, as well as ensure that the *src* and *dst* arrays do not overlap.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

stpcpy( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue X.

NAME

strndup — duplicate a specific number of bytes from a string

SYNOPSIS

UX
```
#include <string.h>

char *strndup(const char *string, size_t size);
```

DESCRIPTION

The *strndup*( ) function shall be equivalent to the *strdup*( ) function, duplicating the provided *string* 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 null byte.

If the length of *string* is larger than *size*, only *size* bytes shall be duplicated. If *size* is larger than the length of *string*, all bytes in *string* shall be copied into the new memory buffer, including the terminating null byte. The newly created string shall always be properly terminated.

RETURN VALUE

Upon successful completion, the *strndup*( ) function shall return a pointer to the newly allocated memory containing the duplicated string. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

ERRORS

The *strndup*( ) function shall fail if:

[ENOMEM]	Insufficient memory available for the target string.

EXAMPLES

None.

APPLICATION USAGE

The *strndup*( ) function is part of the Extended Interfaces Option Group and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

malloc( ), *strdup*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue X.

NAME

strnlen — determine length of fixed size string

SYNOPSIS

UX

```
#include <string.h>

size_t strnlen(const char *s, size_t maxlen);
```

DESCRIPTION

The *strnlen*( ) function shall compute the smaller of the number of bytes in the string to which *s* points, not including the terminating null byte, or the value of the *maxlen* argument. The *strnlen*( ) function shall never examine more than *maxlen* bytes of the string pointed to by *s*.

RETURN VALUE

The *strnlen*( ) function shall return an integer containing the smaller of either the length of the string pointed to by *s* or *maxlen*.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The *strnlen*( ) function is part of the Extended Interfaces Option Group and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strlen( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue X.

NAME

strsignal — get name of signal

SYNOPSIS

UX
```
#include <string.h>

char *strsignal(int signum);
```

DESCRIPTION

The *strsignal*( ) function shall map the signal number in *signum* to an implementation-defined string and shall return a pointer to it. It shall use the same set of messages as the *psignal*( ) function.

The string pointed to shall not be modified by the application, but may be overwritten by a subsequent call to *strsignal*( ) or *setlocale*( ).

The contents of the message strings returned by *strsignal*( ) should be determined by the setting of the *LC_MESSAGES* category in the current locale.

The implementation shall behave as if no function defined in this standard calls *strsignal*( ).

Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *strsignal*( ), then check *errno*.

The *strsignal*( ) function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

RETURN VALUE

Upon successful completion, *strsignal*( ) shall return a pointer to a string. Otherwise, if *signum* is not a valid signal number, the return value is unspecified.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The *strsignal*( ) function is part of the Extended Interfaces Option Group and need not be available on all implementations.

RATIONALE

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. This standard leaves this return value unspecified.

FUTURE DIRECTIONS

None.

SEE ALSO

perror( ), *psignal*( ), *setlocale*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**

CHANGE HISTORY

First released in Issue X.

NAME

wcpcpy — copy a wide-character string, returning a pointer to its end

SYNOPSIS

UX
```
#include <wchar.h>

wchar_t *wcpcpy(wchar_t *restrict dst, const wchar_t *restrict src);
```

DESCRIPTION

The *wcpcpy*( ) function is the wide-character equivalent of the *stpcpy*( ) function. It shall copy the wide-character string pointed to by *src*, including the terminating null wide-character code, into the array pointed to by *dst*.

The application shall ensure that there is room for at least *wcslen*(*src*)+1 wide characters in the *dst* array, and that the *src* and *dst* arrays do not overlap.

RETURN VALUE

The *wcpcpy*( ) function shall return a pointer to the last wide character written into the *dst* array that is a pointer to the terminating null wide-character code. No return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The *wcpcpy*( ) function is part of the Extended Interfaces Option Group and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

stpcpy( ), *strcpy*( ), *wcscpy*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue X.

NAME

wcpncpy — copy a fixed-size wide-character string, returning a pointer to its end

SYNOPSIS

UX
```
#include <wchar.h>

wchar_t *wcpncpy(wchar_t restrict *dst, const wchar_t *restrict src,
    size_t n);
```

DESCRIPTION

The *wcpncpy*( ) function is the wide-character equivalent of the *stpncpy*( ) function. It shall copy at most *n* wide characters from the string pointed to by *src*, including the terminating null wide-character code, into the array pointed to by *dst*. Exactly *n* wide characters shall be written into *dst*. If the length of *src* is smaller than *n*, remaining characters for *dst* are filled in using the terminating null wide-character code. If the *src* array length is greater than or equal to *n*, then *n* characters from *src* shall be copied to *dst* with no terminating null wide-character code in the *dst* array.

The application shall ensure that there is room for at least *n* wide characters in the *dst* array, and that the *src* and *dst* arrays do not overlap.

RETURN VALUE

If any null wide-character codes were written into the *dst* array, the *wcpncpy*( ) function shall return the address of the first such null wide-character code. Otherwise, it shall return &*dst*[*n*]. No return values are reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The *wcpncpy*( ) function is part of the Extended Interfaces Option Group and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

stpncpy( ), *wcpcpy*( ), *wcsncpy*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue X.

NAME

wcscasecmp — compare two wide-character strings, ignoring case

SYNOPSIS

UX
```
#include <wchar.h>

int wcscasecmp(const wchar_t *st1, const wchar_t *st2);
```

DESCRIPTION

The *wcscasecmp*( ) function is the wide-character equivalent of the *strcasecmp*( ) function.

The *wcscasecmp*( ) function shall compare, while ignoring differences in case, the string pointed to by *st1* to the string pointed to by *st2*.

In the POSIX locale, *wcscasecmp*( ) shall behave as if the strings had been converted to lowercase and then a character comparison performed. The results are unspecified in other locales.

RETURN VALUE

Upon completion, the *wcscasecmp*( ) function shall return an integer greater than, equal to, or less than 0 if the wide-character string pointed to by *st1* is, ignoring case, greater than, equal to, or less than the wide-character string pointed to by *st2*, respectively. No return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The *wcscasecmp*( ) function is part of the Extended Interfaces Option Group and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strcasecmp( ), *wcscmp*( ), *wcsncasecmp*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue X.

NAME

wcsdup — duplicate a wide-character string

SYNOPSIS

UX
```
#include <wchar.h>

wchar_t *wcsdup(const wchar_t *string);
```

DESCRIPTION

The *wcsdup*( ) function is the wide-character equivalent of the *strdup*( ) function.

The *wcsdup*( ) function shall return a pointer to a new wide-character string, which is the duplicate of the wide-character string *string*. The returned pointer can be passed to *free*( ). A null pointer is returned if the new wide-character string cannot be created.

RETURN VALUE

Upon successful completion, the *wcsdup*( ) function shall return a pointer to the newly allocated wide-character string. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

ERRORS

The *wcsdup*( ) function shall fail if:

[ENOMEM]	Memory large enough for the duplicate string could not be allocated.

EXAMPLES

None.

APPLICATION USAGE

The *wcsdup*( ) function is part of the Extended Interfaces Option Group and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

free( ), *strdup*( ), *wcscpy*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue X.

NAME

wcsncasecmp — compare two fixed-size wide-character strings, ignoring case

SYNOPSIS

UX

```
#include <wchar.h>

int wcsncasecmp(const wchar_t *st2, const wchar_t *st2, size_t n);
```

DESCRIPTION

The *wcsncasecmp*( ) function is the wide-character equivalent of the *strncasecmp*( ) function.

The *wcsncasecmp*( ) function shall compare, while ignoring differences in case, not more than *n* characters from the wide-character string pointed to by *st1* to the wide-character string pointed to by *st2*.

In the POSIX locale, *wcsncasecmp*( ) shall behave as if the strings had been converted to lowercase and then a character comparison performed. The results are unspecified in other locales.

RETURN VALUE

Upon completion, the *wcsncasecmp*( ) function shall return an integer greater than, equal to, or less than 0 if the possibly null wide-character terminated string pointed to by *st1* is, ignoring case, greater than, equal to, or less than the possibly null wide-character terminated string pointed to by *st2*, respectively. No return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The *wcsncasecmp*( ) function is part of the Extended Interfaces Option Group and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strncasecmp( ), *wcscasecmp*( ), *wcsncmp*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue X.

NAME

wcsnlen — determine the length of a fixed-sized wide-character string

SYNOPSIS

UX
```
#include <wchar.h>

size_t wcsnlen(const wchar_t *wcs, size_t maxlen);
```

DESCRIPTION

The *wcsnlen*( ) function is the wide-character equivalent of the *strnlen*( ) function.

The *wcsnlen*( ) function shall compute the smaller of the number of wide characters in the string to which *wcs* points, not including the terminating null wide-character code, and the value of *maxlen*. The *wcsnlen*( ) function shall never examine more than the first *maxlen* characters of the wide-character string pointed to by *wcs*.

RETURN VALUE

The *wcsnlen*( ) function shall return an integer containing the smaller of either the length of the wide-character string pointed to by *wcs* or *maxlen*. No return value is reserved to indicate an error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

The *wcsnlen*( ) function is part of the Extended Interfaces Option Group and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strnlen( ), *wcslen*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue X.

NAME

wcsnrtombs — convert wide-character string to multi-byte string

SYNOPSIS

UX
```
#include <wchar.h>

size_t wcsnrtombs(char *dst, const wchar_t **src, size_t nwc,
    size_t len, mbstate_t *ps);
```

DESCRIPTION

The *wcsnrtombs*( ) function shall be equivalent to the *wcsrtombs*( ) function, except that the conversion is limited to the first *nwc* wide characters.

The *wcsnrtombs*( ) function shall convert a sequence of at most *nwc* wide characters from the array indirectly pointed to by *src* into a sequence of corresponding characters, beginning in the conversion state described by the object pointed to by *ps*. If *dst* is not a null pointer, the converted characters shall then be stored into the array pointed to by *dst*. Conversion continues up to and including a terminating null wide character, which shall also be stored. Conversion shall stop earlier in the following cases:

- When a code is reached that does not correspond to a valid character
- When the next character would exceed the limit of *len* total bytes to be stored in the array pointed to by *dst* (and *dst* is not a null pointer)
- When *nwc* wide characters from *src* have been converted

Each conversion shall take place as if by a call to the *wcrtomb*( ) function.

If *dst* is not a null pointer, the pointer object pointed to by *src* shall be assigned either a null pointer (if conversion stopped due to reaching a terminating null wide character) or the address just past the last wide character converted (if any). If conversion stopped due to reaching a terminating null wide character, the resulting state described shall be the initial conversion state.

If *ps* is a null pointer, the *wcsnrtombs*( ) function shall use its own internal **mbstate_t** object, which is initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object pointed to by *ps* shall be used to completely describe the current conversion state of the associated character sequence. The implementation shall behave as if no function defined in System Interfaces volume of IEEE Std 1003.1-2001 calls *wcsnrtombs*( ).

UX CX If the application uses any of the _POSIX_THREAD_SAFE_FUNCTIONS or _POSIX_THREADS functions, the application shall ensure that the *wcsnrtombs*( ) function is called with a non-NULL *ps* argument.

The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

RETURN VALUE

Refer to *wcsrtombs*( ).

ERRORS

Refer to *wcsrtombs*( ).

EXAMPLES

None.

APPLICATION USAGE

The *wcsnrtombs*( ) function is part of the Extended Interfaces Option Group and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcrtomb( ), *wcsrtombs*( ), the Base Definitions volume of IEEE Std 1003.1-2001, **<wchar.h>**

CHANGE HISTORY

First released in Issue X.

Index