THE Open GROUP

The Open Group Base Working Group

New API Extensions (Extended Interfaces Strawman Draft 7.3)

Work Item # 1.2.3

November 2004

The Open Group

Document Number: This draft document is work item 1.2.3 of The Open Group Base Working Group Work Plan dated 9 June 2004.

Comments relating to the material contained in this document may be submitted in aardvark format to:

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

http://www.opengroup.org/platform/bugreport.html

See http://www.opengroup.org/austin/aardvark/format.html for information on the aardvark comment format.

Chapter 1

New API Extensions (Extended Interfaces Strawman draft 7.3)

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 version 3 of the Single UNIX Specification .

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

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 namespace, or whether related interfaces should be merged with existing pages.

1.1 Change History

Draft 7.3

- Remove *endusershell*( ), *getusershell*( ), *memmem*( ), *on_exit*( ), and *setusershell*( ).
- Add additional reviewers comments to further pages.

Draft 7.2

- Remove *alloca*( ), *strdupa*( ) and *strndupa*( ).
- Add additional reviewers comments to pages initially reviewed by the Base Working Group
- Fixup *on_exit*( ) prototype in frontmatter.

Draft 7.1

- Reorder the *scandir*( ) manual page to list the *alphasort*( ) function first.
- Update **<string.h>** so that *strdupa*( ) and *strndupa*( ) are listed separately.
- Update the example in *open_memstream*( ).
- Update descriptions of *stpcpy*( ) and *stpncpy*( ) to be closer to *strcpy*( ).
- Update *strdupa*( ) and *strndupa*( ) so that it can either be implemented as a function or a macro. This is for consistency with *alloca*( ).

Draft 7

Minor updates for proposal to have this set as part of an Extended Interfaces Option Group.

Draft 6

- Numerous updates after comments on draft 5.
- Added new functions *fmemopen*( ) and *open_memstream*( )

Draft 5

Key changes in draft 5 are as follows

- Removed *hcreate_r*( ), *hdestroy_r*( ), and *hsearch_r*( ).
- Added EINVAL error to *dirfd*( ) and error return of −1.
- Added EBADF error to *dprintf*( ).
- Removed *fgetgrent*( ), *fgetgrent_r*( ), *fgetpwent*( ), and *fgetpwent_r*( ).
- Merged *getdelim*( ) and *getline*( ) pages, corrected return types to *ssize_t* and tidy up error cases.
- Corrected DESCRIPTION of *mbsnrtowcs*( ) since *nmc* is the input buffer size in bytes and a general rewrite more in line with *mbsrtowcs*( ).
- *mkdtemp*( ) should have 6 Xs not 7, plus tidy up of the RETURN VALUE section.
- Updates to *on_exit*( ) to make it clearer the interworking with *atexit*( ).
- A rewrite of *scandir*( ).

- Tidy up the RETURN VALUE in *stpncpy*( ) and *wcpncpy*( ).
- Remove extraneous ENOMEM errors for *strdupa*( ) and *strndupa*( ).
- Some tidy up to the DESCRIPTION of *strnlen*( ) to make it clearer that only *maxlen* bytes are examined.
- Correct the RETURN VALUE in *wcpncpy*( ) as per *stpncpy*( ).
- A tidy up of *strsignal*( ).
- Make it clear that its the first *nwc* wide characters for *wcsnrtombs*( ).

1.2 XBD Changes

It is proposed that these additions comprise a new option group, called the Extended Interfaces option group.

1.2.1 1.5.1 Codes

Add a new margin marker code "EX Extended Interfaces", with the text

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

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

1.2.2 13. Headers

The following header file man pages will need the following additions, margin marked and shaded as part of the EX 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 *);
```

<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 *, va_list);
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 *);
```

1.3 XSH Manual Pages

The man pages follow.

NAME

alphasort,scandir — 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 **));
```

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 *sel*( ) function on each directory entry. Entries for which *sel*( ) returns non-zero shall be stored in strings allocated via *malloc*( ), and sorted using *qsort*( ) with the comparison function *compar*( ), and collected in array *namelist* which shall be allocated via *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 directory pointed to by *d1* is 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*. The *scandir*( ) function shall return −1 if the directory cannot be opened for reading or if *malloc*( ) cannot allocate enough memory to hold all the data structures.

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 that 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 and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

the Base Definitions volume of IEEE Std 1003.1-2001, **<dirent.h>**

CHANGE HISTORY

First released in Issue X

NAME

dirfd — extracts the file descriptor used by a DIR stream

SYNOPSIS

```
#include <dirent.h>

int dirfd(DIR *dirp);
```

DESCRIPTION

Notes to Reviewers

This section with side shading will not appear in the final copy. - Ed.

Commentary on this function:

This interface was introduced because glibc 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 working group has some concern that a file descriptor is not required for the **DIR** data structure in the present standard, so there would be a need either to prefix *dirfd*( ), with text along the lines of "If a file descriptor is used to implement...," or to require an underlying file descriptor. The former would require applications to know about the implementation, and hence applications would not be able to make portable use of this function.

Thus the implication would be that to introduce this we would have to mandate an underlying file descriptor for a DIR object for it to be useful for portable applications.

So if we take this change it would need a number of other changes to the existing directory related functions.

The *dirfd*( ) function shall return the file descriptor used by the *dirp* argument.

RETURN VALUE

Upon successful completion, the *dirfd*( ) function shall return an integer which contains the 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.

EXAMPLES

None.

APPLICATION USAGE

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

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

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

CHANGE HISTORY

First released in Issue X

NAME

dprintf — formated output conversion to a file descriptor

SYNOPSIS

```
#include <stdio.h>

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

DESCRIPTION

Notes to Reviewers

This section with side shading will not appear in the final copy. - Ed.

Commentary on this function:

It is unclear as to what the required buffering behavior is for this function. More information is needed. On the surface this function would appear to be a convenience function more than a necessity. Is this really done frequently enough to justify adding a new function when *snprintf*( ) and *write*( ) are sufficient to do the job? It was also suggested that *fdprintf*( ) would be a better name.

The *dprintf*( ) function shall be equivalent to the *printf*( ) function, producing output according to the contents of *format*, with the exception that instead of the output going to *stdout*, the output of *dprintf*( ) is directed to the file descriptor *fildes*.

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 and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

printf( ) 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

```
#include <stdio.h>

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

DESCRIPTION

Notes to Reviewers

This section with side shading will not appear in the final copy. - Ed.

Commentary on this function:

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. A wide character version has not yet been proposed. It was proposed that *fmemopen*( ) should leave the results unoriented.

There appears to be a need to modify other related stdio pages that talk about handling **FILE** objects; how would they behave if a memory stream is underlying the stream? If writes on a stream with an underlying memory buffer, would overflow the memory buffer, the behavior is as the same as filesystem full, that is [ENOSPC].

Further work would be needed to cleanup this page, and other pages.

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, but is allowed for ISO C standard conformance.

If a null pointer is specified as the *buf* argument, *memopen*( ) shall use *malloc*( ) to allocate a buffer that is *size* bytes long. 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 begin 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.

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 write 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 end of the buffer if it fits. 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)
```

Unapproved Draft, Subject to Change

```
            /* 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 and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

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

CHANGE HISTORY

First released in Issue X

NAME

getdelim — reads a delimited record from *stream*.

SYNOPSIS

```
#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

Notes to Reviewers

This section with side shading will not appear in the final copy. - Ed.

Commentary on this function:

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.

More words needed on the description, need to clean up bytes vs characters to be compatible with the standard

The *getdelim*( ) function shall read from *stream* until it encounters a byte 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 nonzero, the application shall ensure that **lineptr* points to an object containing 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 bytes read shall be stored in the string pointed to by the argument *lineptr*.

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 bytes 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

The *getdelim*( ) and *getline*( ) functions shall fail if:

[EINVAL]	When *lineptr* or *n* are a null pointer.

The *getdelim*( ) and *getline*( ) functions may fail if:

[EINVAL]	*stream* is not a valid file descriptor.

[EOVERFLOW] More than SSIZE_MAX bytes were read without encountering the *delimiter* character.

EXAMPLES

APPLICATION USAGE

The *getdelim*( ) and *getline*( ) functions are part of the Extended Interfaces option 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 have been explicitly designed to enlarge the buffer if necessary.

FUTURE DIRECTIONS

None.

SEE ALSO

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

CHANGE HISTORY

First released in Issue X

NAME

mbsnrtowcs — converts a multi-byte string to a wide character string.

SYNOPSIS

```
#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*.

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 and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mbsrtowcs( ), *iconv*( ), 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

```
#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 set *errno*.

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 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 a dynamic memory buffer stream

SYNOPSIS

```
#include <stdio.h>

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

DESCRIPTION

Notes to Reviewers

This section with side shading will not appear in the final copy. - Ed.

Commentary on this function:

This function is similar to *fmem_open*( ) except that the memory is allocated dynamically by the function. This would need a wide version *open_wmemstream*( ). This interface does not have a *mode* parameter since it can only be written to.

It was agreed that further cleanup is needed on the wording.

The *open_memstream*( ) function shall create a stream that is associated with a dynamically allocated buffer. The buffer is obtained by calls to *malloc*( ) and *realloc*( ) and expanded as necessary. It must be freed by the caller after closing the stream. The stream is opened for writing and shall be seekable.

The stream maintains a current position in the allocated buffer and a current buffer length. The position is initially set to zero (the begin of the buffer). Each write starts at the current position and moves this position by the number of successfully written bytes. The length is initially set to zero. If a write moves the position to a value larger than the current length, the current length is set to this position. In this case a null byte is appended to the current buffer (but not accounted for in the buffer length).

The maximum value of the buffer length and position is given by the smaller of {SIZE_MAX} and the maximum allowed file offset {OFF_MAX}.

After a successful *fflush*( ) or *fclose*( ) the locations pointed to by *bufp* and *sizep* contain the address of the buffer and the current buffer length and the buffer is guaranteed to be terminated by a null byte (which is not accounted for in the length).

An attempt to seek a dynamic buffer stream to a negative position or to a position larger than the minimum of {SIZE_MAX} and {OFF_MAX} shall return an error.

RETURN VALUE

Upon successful completion, *open_memstream*( ) 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 *open_memstream*( ) function 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 = fmemopen(&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

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

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

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

CHANGE HISTORY

First released in Issue X

NAME

psignal — print signal information to standard error

SYNOPSIS

```
#include <signal.h>

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

DESCRIPTION

Notes to Reviewers

This section with side shading will not appear in the final copy. - Ed.

Commentary on this function:

System V historically has *psignal*( ) and *psiginfo*( ) in **<siginfo.h>**.

It was agreed during the preliminary review that there should be an additional *psiginfo*( ) function added since we have the type **siginfo_t** within the standard.

The issue of which header the function(s) occur in needs to be resolved.

The *psignal*( ) function 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*. If the *message* argument is null or points to an empty string, then only the signal description shall be printed. If *signum* is not a valid signal number, the behavior is implementation-defined.

RETURN VALUE

The *psignal*( ) function shall not return a value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

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

RATIONALE

None.

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

```
#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];
chsr *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 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

```
#include <string.h>

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

DESCRIPTION

Notes to Reviewers

This section with side shading will not appear in the final copy. - Ed.

Commentary on this function:

The 2nd paragraph of the DESCRIPTION is ambiguous (length of the string is usually equivalent to strlen(string), but it is off by 1 in this case) and needs to be fixed up.

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 smaller than 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 larger 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 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* array do not overlap.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strncpy( ), *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

```
#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 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

APPLICATION USAGE

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

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

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

```
#include <string.h>

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

DESCRIPTION

Notes to Reviewers

This section with side shading will not appear in the final copy. - Ed.

Commentary on this function:

The RETURN VALUE section wording is ambiguous. (How is "size of the string" related to string length?) Do we assume that the return value is strlen (s) or maxlen whichever is smaller?

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 size of the string pointed to by *s* or *maxlen*.

ERRORS

No errors are defined.

EXAMPLES

APPLICATION USAGE

The *strnlen*( ) function is part of the Extended Interfaces option 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

```
#include <string.h>

char *strsignal(int signum);
```

DESCRIPTION

Notes to Reviewers

This section with side shading will not appear in the final copy. - Ed.

Commentary on this function:

Some implementations return NULL rather than unknown, so need to decide whether its worth picking one, perhaps unspecified is the best we can do for this interface.

The *strsignal*( ) function shall map the signal number in *signum* to a 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 *strsignal*( ) function shall return a pointer to a string containing an unspecified message denoting an unknown signal.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

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

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

perror( ), *psignal*( ), 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

```
#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 and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

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

CHANGE HISTORY

NAME

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

SYNOPSIS

```
#include <wchar.h>

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

DESCRIPTION

Notes to Reviewers

This section with side shading will not appear in the final copy. - Ed.

Commentary on this function:

This page needs further work to improve the language to match the standard and also to tidy up some points (the current description makes it impossible to implement this function if n == 0).

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

The *wcpncpy*( ) function shall return a pointer to the first null wide character that was written into the *dst* array, whatever the relation between *size* and the length of *src*. 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 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

```
#include <wchar.h>

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

DESCRIPTION

Notes to Reviewers

This section with side shading will not appear in the final copy. - Ed.

Commentary on this function:

Some issues with this man page. The return value section doesn't match the description. Language referring to st1 and st2, needs to be clear that its the the first wide character pointed to by st1 and st2

The *wcscasecmp*( ) function is the wide character equivalent of the *strcasecmp*( ) function. It shall compare the wide character string in *st1* with that found in *st2*. This comparison shall ignore case differences between the two strings.

RETURN VALUE

The *wcscasecmp*( ) function shall return an integer containing the value 0 when the two strings are equal (ignoring case differences). The returned integer shall be positive when *st1* is greater than *st2*, ignoring case. The returned integer shall be negative when *st1* is smaller than *st2*, ignoring case. 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 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

```
#include <wchar.h>

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

DESCRIPTION

Notes to Reviewers

This section with side shading will not appear in the final copy. - Ed.

Commentary on this function:

Some issues with this man page. The description and return value sections do not actually state that the wide characters in the string argument are actually copied into the memory pointed to by the return value.

The *wcsdup*( ) function is the wide character equivalent of the *strdup*( ) function. It shall allocate memory for a wide character string duplicate of that passed in *string*.

The memory is allocated using *malloc*( ), and should be freed using *free*( ).

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 and need not be available on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

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

```
#include <wchar.h>

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

DESCRIPTION

Notes to Reviewers

This section with side shading will not appear in the final copy. - Ed.

Commentary on this function:

Some issues with this man page. The return value section doesn't match the description. Phrases such as "truncated st1" need to be "the wide character string pointed to by st1".

The *wcsncasecmp*( ) function is the wide character equivalent of the *strncasecmp*( ) function. It shall compare at most *n* wide characters in *st1* to their counterparts in *st2*, ignoring case differences.

RETURN VALUE

The *wcsncasecmp*( ) function shall return an integer containing the value 0 when at most *n* wide characters compare equal, ignoring case. This integer shall be a positive value when the truncated *st1* is greater than the truncated *st2*, ignoring case. It shall be a negative value when the truncated *st1* is less than the truncated *st2*, ignoring case. 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 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

```
#include <wchar.h>

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

DESCRIPTION

Notes to Reviewers

This section with side shading will not appear in the final copy. - Ed.

Commentary on this function:

Some issues with this man page. The description and return value sections use non-standard terms ("termination character", "size" of a wide character string). Uses of phrases such as "size of" need to be updated to "length of a string"

The *wcsnlen*( ) function is the wide character equivalent of the *strnlen*( ) function. It shall scan the wide character string pointed to by the *wcs* argument up to at most *maxlen* wide characters, looking for a termination character.

RETURN VALUE

The *wcsnlen*( ) function shall return an integer containing the smaller of either the size 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 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

```
#include <wchar.h>

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

DESCRIPTION

Notes to Reviewers

This section with side shading will not appear in the final copy. - Ed.

Commentary on this function:

The man page for this interface is incomplete and the references to wcsrtombs() are not sufficient to understand how this is supposed to work in the general case. Need a much better description

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

RETURN VALUE

Refer to *wcsrtombs*( )

ERRORS

Refer to *wcsrtombs*( )

EXAMPLES

None.

APPLICATION USAGE

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

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

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

CHANGE HISTORY

First released in Issue X