IEEE P1003.2 Draft 11.2 – September 1991

Copyright © 1991 by the
Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street
New York, NY 10017, USA
All rights reserved as an unpublished work.

This is an unapproved and unpublished IEEE Standards Draft, subject to change. The publication, distribution, or copying of this draft, as well as all derivative works based on this draft, is expressly prohibited except as set forth below.

Permission is hereby granted for IEEE Standards Committee participants to reproduce this document for purposes of IEEE standardization activities only, and subject to the restrictions contained herein.

Permission is hereby also granted for member bodies and technical committees of ISO and IEC to reproduce this document for purposes of developing a national position, subject to the restrictions contained herein.

Permission is hereby also granted to the preceding entities to make limited copies of this document in an electronic form only for the stated activities.

The following restrictions apply to reproducing or transmitting the document in any form: 1) all copies or portions thereof must identify the document's IEEE project number and draft number, and must be accompanied by this entire notice in a prominent location; 2) no portion of this document may be redistributed in any modified or abridged form without the prior approval of the IEEE Standards Department.

Other entities seeking permission to reproduce this document, or any portion thereof, for standardization or other activities, must contact the IEEE Standards Department for the appropriate license.

Use of information contained in this unapproved draft is at your own risk.

IEEE Standards Department
Copyright and Permissions
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331, USA
+1 (908) 562-3800
+1 (908) 562-1571 [FAX]

The description was extensively rewritten in Draft 9 to be more explicit about how *optarg* and *optind* are set, and to recognize that this routine deals with a vector of string pointers, not directly with a shell command line.

The description was modified in Draft 9 to make it clear that *getopt* (), like the `getopts` utility, shall deal with option-arguments whether separated from the option by <blank>s or not. Note that the requirements on *getopt* () and `getopts` are more stringent than the Utility Syntax Guidelines.

The *getopt* () function has been changed to return `−1`, rather than `EOF`, so that *<stdio.h>* is not required.

The special significance of a colon as the first character of *optstring* was added in Draft 11 to make *getopt* () consistent with the `getopts` utility. It allows an application to make a distinction between a missing argument and an incorrect option letter without having to examine the option letter. It is true that a missing argument can only be detected in one case, but that is a case that has to be considered.

B.8 C Binding for Generate Pathnames Matching a Pattern

Functions: *glob* (), *globfree* ()

B.8.1 Synopsis

```
#include <glob.h>

int glob(const char *pattern , int flags ,
        int (*errfunc )(const char *epath , int eerrno ), glob_t *pglob);
void globfree(glob_t *pglob);
```

B.8.2 Description

The *glob* () function is a pathname generator that implements the rules defined in 3.13, with optional support for rule (3) in 3.13.3.

The header `<glob.h>` defines the structure type *glob_t* , which includes at least the members shown in Table B-12.

The argument *pattern* is a pointer to a pathname pattern to be expanded. The *glob* () function shall match all accessible pathnames against this pattern and develop a list of all pathnames that match. In order to have access to a pathname, *glob* () requires search permission on every component of a path except the last and read permission on each directory of any filename component of *pattern* that contains any of the special characters `*`, `?` or `[`. The *glob* () function stores the number of matched pathnames into *pglob->gl_pathc* and a pointer to a list of pointers to pathnames into *pglob->gl_pathv* . The pathnames are in sort order as defined by 2.2.2.30. The first pointer after the last pathname shall be **NULL** . If the pattern does not match any pathnames, the returned number of matched paths is set to zero.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table B-12 – Structure Type *glob_t*

Member Type	Member Name	Description
size_t	*gl_pathc*	Count of paths matched by *pattern* .
char **	*gl_pathv*	Pointer to a list of matched pathnames.
size_t	*gl_offs*	Slots to reserve at the beginning of *gl_pathv* .

It is the caller's responsibility to create the structure pointed to by *pglob*. The *glob* () function shall allocate other space as needed, including the memory pointed to by *gl_pathv* . The *globfree* () function shall free any space associated with *pglob* from a previous call to *glob* ().

The argument *flags* is used to control the behavior of *glob* (). The value of *flags* is the bitwise inclusive OR of any of the constants shown in Table B-13, which are defined in <glob.h>.

Table B-13 – *glob* () *flags* Argument

Name	Description
GLOB _APPEND	Append pathnames generated to the ones from a previous call to *glob* ().
GLOB _DOOFFS	Make use of *pglob->gl_offs* . If this flag is set, *pglob->gl_offs* is used to specify how many **NULL** pointers to add to the beginning of *pglob->gl_pathv* . In other words, *pglob->gl_pathv* shall point to *pglob->gl_offs* **NULL** pointers, followed by *pglob->gl_pathc* pathname pointers, followed by a **NULL** pointer.
GLOB _ERR	Causes *glob* () to return when it encounters a directory that it cannot open or read. Ordinarily, *glob* () continues to find matches.
GLOB _MARK	Each pathname that is a directory that matches *pattern* has a slash appended.
GLOB _NOCHECK	Support rule (3) in 3.13.3. If *pattern* does not match any pathname, then *glob* () shall return a list consisting of only *pattern* , and the number of matched pathnames is 1.
GLOB _NOESCAPE	Disable backslash escaping.
GLOB _NOSORT	Ordinarily, *glob* () sorts the matching pathnames according to the definition of *collation sequence* in 2.2.2.30. When this flag is used the order of pathnames returned is unspecified.

The GLOB _APPEND flag can be used to append a new set of words to those generated by a previous call to *glob* (). The following rules apply when two or more calls to *glob* () are made with the same value of *pglob* and without intervening calls to *globfree* ():

(1) The first such call shall not set GLOB _APPEND. All subsequent calls shall set it.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

(2) All of the calls shall set GLOB_DOOFFS, or all shall not set it.

(3) After the second call, *pglob->gl_pathv* shall point to a list containing the following:

(a) Zero or more **NULL**s, as specified by GLOB_DOOFFS and *pglob->gl_offs*.

(b) Pointers to the pathnames that were in the *pglob->gl_pathv* list before the call, in the same order as before.

(c) Pointers to the new pathnames generated by the second call, in the specified order.

(4) The count returned in *pglob->gl_pathc* shall be the total number of pathnames from the two calls.

The application can change any of the fields in Table B-12 after a call to *glob*(), but if it does it shall reset them to the original value before a subsequent call, using the same *pglob* value, to *globfree*() or *glob*() with the GLOB_APPEND flag.

If, during the search, a directory is encountered that cannot be opened or read and *errfunc* is not **NULL**, *glob*() shall call (**errfunc*)() with two arguments:

(1) The *epath* argument is a pointer to the path that failed.

(2) The *eerrno* argument is the value of *errno* from the failure, as set by the POSIX.1 {8} *opendir*(), *readdir*(), or *stat*() functions. (Other values may be used to report other errors not explicitly documented for those functions.)

If (**errfunc*)() is called and returns nonzero, or if the GLOB_ERR flag is set in *flags*, *glob*() shall stop the scan and return GLOB_ABORTED after setting *gl_pathc* and *gl_pathv* in *pglob* to reflect the paths already scanned. If GLOB_ERR is not set and either *errfunc* is **NULL** or (**errfunc*)() returns zero, the error shall be ignored.

B.8.3 Returns

On successful completion, *glob*() shall return zero. The argument *pglob->gl_pathc* shall return the number of matched pathnames and the argument *pglob->gl_pathv* shall contain a pointer to a null-terminated list of matched and sorted pathnames. However, if *pglob->gl_pathc* is zero, the content of *pglob->gl_pathv* is undefined.

B.8.4 Errors

If *glob*() terminates due to an error, it shall return one of the nonzero constants shown in Table B-14, which are defined in `<glob.h>`. The arguments *pglob->gl_pathc* and *pglob->gl_pathv* are still set as defined above in Returns.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table B-14 – *glob* () Error Return Values

Name	Description
GLOB _ABORTED	The scan was stopped because GLOB _ERR was set or (**errfunc*)() returned nonzero.
GLOB _NOMATCH	The *pattern* does not match any exiting pathname, and GLOB _NOCHECK was not set in *flags* .
GLOB _NOSPACE	An attempt to allocate memory failed.

B.8.5 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

This function is not provided for the purpose of enabling utilities to perform pathname expansion on their arguments, as this operation is performed by the shell, and utilities are explicitly not expected to redo this. Instead, it is provided for applications that need to do pathname expansion on strings obtained from other sources, such as a pattern typed by a user or read from a file.

If a utility needs to see if a pathname matches a given pattern, it can use *fnmatch* ().

Note that *gl_pathc* and *gl_pathv* have meaning even if *glob* () fails. This allows *glob* () to report partial results in the event of an error. However, if *gl_pathc* is zero, *gl_pathv* is unspecified even if *glob* () did not return an error.

The GLOB _NOCHECK option could be used when an application wants to expand a pathname if wildcards are specified, but wants to treat the pattern as just a string otherwise. The `sh` utility might use this for option-arguments, for example.

One use of the GLOB _DOOFFS flag is by applications that build an argument list for use with the POSIX .1 {8} *execv* (), *execve* (), or *execvp* () functions. Suppose, for example, that an application wants to do the equivalent of `ls –l *.c`, but for some reason `system("ls –l *.c")` is not acceptable. The application could obtain (*approximately*) the same result using the sequence:

```
globbuf.gl_offs = 2;
glob ("*.c", GLOB_DOOFFS, NULL, &globbuf);
globbuf.gl_pathv[0] = "ls";
globbuf.gl_pathv[1] = "-l";
execvp ("ls", &globbuf.gl_pathv[0]);
```

Using the same example, `ls –l *.c *.h` could be approximately simulated using GLOB _APPEND as follows:

```
globbuf.gl_offs = 2;
glob ("*.c", GLOB_DOOFFS, NULL, &globbuf);
glob ("*.h", GLOB_DOOFFS|GLOB_APPEND, NULL, &globbuf);
 ... etc. ...
```

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

The new pathnames generated by a subsequent call with GLOB_APPEND are not sorted together with the previous pathnames. This mirrors the way that the shell handles pathname expansion when multiple expansions are done on a command line.

History of Decisions Made

The interface was simplified to a useful, but less complex, subset. The *errfunc* argument was added to allow errors to be reported.

A reviewer claimed that the GLOB_DOOFFS flag is unnecessary because it could be simulated using:

```
new = (char **)malloc((n + pglob->gl_pathc + 1)
                * sizeof (char *));
(void) memcpy (new+n, pglob->gl_pathv,
                pglob->gl_pathc * sizeof(char *));
(void) memset (new, 0, n * sizeof (char *));
free (pglob->gl_pathv);
pglob->gl_pathv = new;
```

However, this assumes that the memory pointed to by *gl_pathv* is a block that was separately created using *malloc* (). This is not necessarily the case. An application should make no assumptions about how the memory referenced by fields in *pglob* was allocated. It might have been obtained from *malloc* () in a large chunk, and then carved up within *glob* (), or it might have been created using a different memory allocator. It is not the intent of this standard to specify or imply how the memory used by *glob* () is managed.

The structure elements *gl_pathc* and *gl_pathv* were renamed from *gl_argc* and *gl_argv* in Draft 9. The old names implied an association with the parameters to *main* () that does not necessarily exist.

The GLOB_APPEND flag was added in Draft 9 at the request of a reviewer. This flag would be used when an application wants to expand several different patterns into a single list.

Tilde and parameter expansion were removed from *glob* () in Draft 9. Applications that need these expansions should use the *wordexp* () function [see B.9].

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

B.9 C Binding for Perform Word Expansions

Functions: *wordexp* (), *wordfree* ()

B.9.1 Synopsis

```
#include <wordexp.h>
int wordexp(const char *words , wordexp_t *pwordexp , int flags );
void wordfree(wordexp_t *pwordexp );
```

B.9.2 Description

The *wordexp* () function shall perform word expansions as described in 3.6, subject to quoting as in 3.2, and place the list of expanded words into *pwordexp* . The expansions shall be the same as would be performed by the shell if *words* were the part of a command line representing the arguments to a utility. Therefore, *words* shall not contain an unquoted <newline> or any of the unquoted shell special characters |, &, ;, <, or >, except in the context of command substitution as specified in 3.6.3. It also shall not contain unquoted parentheses or braces, except in the context of command or variable substitution. If *words* contains an unquoted comment character (number sign) that is the beginning of a token, *wordexp* () may treat the comment character as a regular character, or may interpret it as a comment indicator and ignore the remainder of *words* .

The header <wordexp.h> defines the structure type *wordexp_t* , which includes at least the members shown in Table B-15.

Table B-15 – Structure Type *wordexp_t*

Member Type	Member Name	Description
size_t	*we_wordc*	Count of words matched by *words* .
char **	*we_wordv*	Pointer to list of expanded words.
size_t	*we_offs*	Slots to reserve at the beginning of *we_wordv* .

The argument *words* is a pointer to a string containing one or more words to be expanded. The *wordexp* () function shall store the number of generated words into *we_wordc* and a pointer to a list of pointers to words in *we_wordv* . Each individual field created during field splitting (see 3.6.5) or pathname expansion (see 3.6.6) is a separate word in the *we_wordv* list. The words are in order as described in 3.6. The first pointer after the last word pointer shall be **NULL** . The expansion of special parameters described in 3.5.2 is unspecified.

It is the caller's responsibility to create the structure pointed to by *pwordexp* . The *wordexp* () function allocates other space as needed, including memory pointed to

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.