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All rights reserved as an unpublished work.

This is an unapproved and unpublished IEEE Standards Draft, subject to change.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]

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.

by *we_wordv* . The *wordfree* () function shall free any memory associated with *pwordexp* from a previous call to *wordexp* ().

The argument *flags* is used to control the behavior of *wordexp* (). The value of *flags* is the bitwise inclusive OR of any of the constants in Table B-16, which are defined in <wordexp.h>.

Table B-16 – *wordexp* () *flags* Argument

Name	Description
WRDE_APPEND	Append words generated to the ones from a previous call to *wordexp* ().
WRDE_DOOFFS	Make use of *we_offs* . If this flag is set, *we_offs* is used to specify how many **NULL** pointers to add to the beginning of *we_wordv* . In other words, *we_wordv* shall point to *we_offs* **NULL** pointers, followed by *we_wordc* word pointers, followed by a **NULL** pointer.
WRDE_NOCMD	Fail if command substitution, as specified in 3.6.3, is requested.
WRDE_REUSE	The *pwordexp* argument was passed to a previous successful call to *wordexp* (), and has not been passed to *wordfree* (). The result shall be the same as if the application had called *wordfree* () and then called *wordexp* () without WRDE_REUSE.
WRDE_SHOWERR	Do not redirect standard error to /dev/null.
WRDE_UNDEF	Report error on an attempt to expand an undefined shell variable.

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

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

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

(3) After the second and each subsequent call, *we_wordv* shall point to a list containing the following:

  (a) Zero or more **NULL** s, as specified by WRDE _DOOFFS and *we_offs* .

  (b) Pointers to the words that were in the *we_wordv* list before the call, in the same order as before.

  (c) Pointers to the new words generated by the latest call, in the specified order.

(4) The count returned in *we_wordc* shall be the total number of words from all of the calls.

The application can change any of the fields in Table B-15 after a call to *wordexp* (), but if it does it shall reset them to the original value before a subsequent call, using the same *pwordexp* value, to *wordfree* () or *wordexp* () with the WRDE _APPEND or WRDE _REUSE flag.

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

If *words* contains an unquoted <newline>, |, &, ;, <, >, parenthesis, or brace in an inappropriate context, *wordexp* () shall fail, and the number of expanded words shall be zero.

Unless WRDE _SHOWERR is set in *flags* , *wordexp* () shall redirect standard error to /dev/null for any utilities executed as a result of command substitution while expanding *words* . If WRDE _SHOWERR is set, *wordexp* () may write messages to standard error if syntax errors are detected while expanding *words* .

If WRDE _DOOFFS is set, then *we_offs* shall have the same value for each *wordexp* () call and the *wordfree* () call using a given *pglob*.

B.9.3 Returns

If no errors are encountered while expanding *words* , *wordexp* () shall return zero. Otherwise it shall return a nonzero value.

B.9.4 Errors

Table B-17 – *wordexp* () Return Values

Name	Description
WRDE_BADCHAR	One of the unquoted characters |, &, ;, <, >, parentheses, or braces appears in *words* in an inappropriate context.
WRDE_BADVAL	Reference to undefined shell variable when WRDE_UNDEF is set in *flags* .
WRDE_CMDSUB	Command substitution requested when WRDE_NOCMD was set in flags.
WRDE_NOSPACE	Attempt to allocate memory failed
WRDE_SYNTAX	Shell syntax error, such as unbalanced parentheses or unterminated string.

If *wordexp* () terminates due to an error, it shall return one of the nonzero constants shown in Table B-17, which shall be defined in <wordexp.h>. The implementation may define additional error returns beginning with WRDE _.

If *wordexp* () returns the error value WRDE _NOSPACE , then *pwordexp->we_wordc* and *pwordexp->we_wordv* shall be updated to reflect any words that were successfully expanded. In other cases, they shall not be modified.

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

B.9.5 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

This function is intended to be used by an application that wants to do all of the shell's expansions on a word or words obtained from a user. For example, if the application prompts for a file name (or list of file names) and then used *wordexp* () to process the input, the user could respond with anything that would be valid as input to the shell.

The WRDE_NOCMD flag is provided for applications that, for security or other reasons, want to prevent a user from executing shell commands. Disallowing unquoted shell special characters also prevents unwanted side effects such as executing a command or writing a file.

History of Decisions Made

This function was added in Draft 9 as an alternative to *glob* (). There has been continuing controversy over exactly what features should be included in *glob* (). It is hoped that providing *wordexp* () (which provides all of the shell's word expansions, but will probably be slow to execute), and *glob* () (which is faster but does only expansion of pathnames, without tilde or parameter expansion), will satisfy the majority of reviewers.

While *wordexp* () could be implemented entirely as a library routine, it is expected that most implementations will run a shell in a subprocess to do the expansion.

Two different approaches have been proposed for how the required information might be presented to the shell and the results returned. They are presented here as examples.

One proposal is to extend the `echo` utility by adding a `-q` option. This option would cause `echo` to add a backslash before each backslash and each `<blank>` that occurs within an argument. The *wordexp* () function could then invoke the shell as follows:

```
(void) strcpy (buffer, "echo -q ");
(void) strcat (buffer, words );
if ((flags & WRDE_SHOWERR) == 0)
        (void) strcat (buffer, " 2>/dev/null");
f = popen (buffer, "r");
```

The *wordexp* () function would read the resulting output, remove unquoted backslashes, and break into words at unquoted `<blank>`s. If the WRDE_NOCMD flag was set, *wordexp* () would have to scan *words* before starting the subshell to make sure that there would be no command substitution. In any case, it would have to scan *words* for unquoted special characters.

Another proposal is to add the following options to `sh`:

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

1376 –w *wordlist* This option provides a wordlist expansion service to applications.
1377 The words in *wordlist* are expanded, and the following is written
1378 to standard output:

1379 (1) The count of the number of words after expansion, in
1380 decimal, followed by a null byte.

1381 (2) The number of bytes needed to represent the expanded
1382 words (not including null separators), in decimal, followed
1383 by a null byte.

1384 (3) The expanded words, each terminated by a null byte.

1385 If an error is encountered during word expansion, `sh` exits with a
1386 nonzero status after writing the above to report any words suc-
1387 cessfully expanded

1388 –P Run in ‘‘protected’’ mode. If specified with the –w option, no com-
1389 mand substitution is performed.

1390 With these options, *wordexp* () could be implemented fairly simply by creating a
1391 subprocess using *fork* (), and executing `sh` using the line:

1392 `execl(<`*shell path*`>, "sh", "-P", "-w",` *words* `, (char *)0);`

1393 after directing standard error to `/dev/null`.

1394 It seemed objectionable for a library routine to write messages to standard error,
1395 unless explicitly requested, so *wordexp* () is required to redirect standard error to
1396 `/dev/null` to ensure that no messages are generated, even for commands exe-
1397 cuted for command substitution. The new WRDE_SHOWERR flag can be specified
1398 to request that error messages be written.

1399 The WRDE_REUSE flag allows the implementation to avoid the expense of freeing
1400 and reallocating memory, if that is possible. A minimal implementation can just
1401 call *wordfree* () when WRDE_REUSE is set.

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

B.10 C Binding for Get POSIX Configurable Variables

B.10.1 C Binding for Get String-Valued Configurable Variables

Function: *confstr* ()

B.10.1.1 Synopsis

```
#include <unistd.h>
size_t confstr(int name , char *buf, size_t len );
```

B.10.1.2 Description

The *confstr* () function provides a method for applications to get configuration-defined string values. Its use and purpose are similar to the *sysconf* () function defined in POSIX .1 {8}, but it is used where string values rather than numeric values are returned.

The *name* argument represents the system variable to be queried. The implementation shall support all of the *name* values shown in Table B-18, which are defined in `<unistd.h>`. It may support others.

Table B-18 – `confstr()` *name* Values

***name* Value**	**String returned by `confstr()`**
_CS_PATH	A value for the **PATH** environment variable that finds all standard utilities.

If *len* is not zero, and if *name* has a configuration-defined value, *confstr* () shall copy that value into the *len*-byte buffer pointed to by *buf*. If the string to be returned is longer than *len* bytes, including the terminating null, then *confstr* () shall truncate the string to *len*−1 bytes and null-terminate the result. The application can detect that the string was truncated by comparing the value returned by *confstr* () with *len*.

If *len* is zero and *buf* is **NULL** , then *confstr* () still shall return the integer value as defined below, but shall not return a string. If *len* is zero but *buf* is not **NULL** , the result is unspecified.

B.10.1.3 Returns

If *name* does not have a configuration-defined value, *confstr* () shall return zero and leave *errno* unchanged.

If *name* has a configuration-defined value, the *confstr* () function shall return the size of buffer that would be needed to hold the entire configuration-defined value.

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