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All rights reserved as an unpublished work.

This is an unapproved and unpublished IEEE Standards Draft, subject to change.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.4 C Binding for Access Environment Variables

Function: *getenv* ()

The C language binding to the service described in 7.2 shall be the POSIX .1 {8} *getenv* () function.

## B.5 C Binding for Regular Expression Matching

Functions: *regcomp* (), *regexec* (), *regfree* (), *regerror* ()

### B.5.1 Synopsis

```
#include <sys/types.h>
#include <regex.h>

int regcomp(regex_t *preg, const char *pattern, int cflags);

int regexec(const regex_t *preg, const char *string,
        size_t nmatch, regmatch_t pmatch[], int eflags);

size_t regerror(int errcode, const regex_t *preg,
        char *errbuf, size_t errbuf_size);

void regfree(regex_t *preg);
```

### B.5.2 Description

These functions shall interpret basic and extended regular expressions, as described in 2.8.

The header <regex.h> shall define the structure types *regex_t* and *regmatch_t* . The structure type *regex_t* shall include at least the member shown in Table B-6.

The structure type *regmatch_t* shall contain at least the members shown in Table B-7. The type *regoff_t* , which shall be defined in <regex.h>, shall be a signed arithmetic type that can hold the largest value that can be stored in either an *off_t* or a *ssize_t* .

The *regcomp* () function shall compile the regular expression contained in the string pointed to by the *pattern* argument and place the results in the structure pointed to by *preg*. The *cflags* argument shall be the bitwise inclusive OR of zero or more of the flags shown in Table B-8, which shall be defined in the header <regex.h>.

The default regular expression type for *pattern* shall be a Basic Regular Expression. The application can specify Extended Regular Expressions using the REG _EXTENDED *cflags* flag.

If the function *regcomp* () succeeds, it shall return zero; otherwise it shall return nonzero, and the content of *preg* shall be undefined.

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

**Table B-6 – Structure Type *regex_t***

| Member Type | Member Name | Description |
|---|---|---|
| *size_t* | *re_nsub* | Number of parenthesized subexpressions. |

**Table B-7 – Structure Type *regmatch_t***

| Member Type | Member Name | Description |
|---|---|---|
| *regoff_t* | *rm_so* | Byte offset from start of *string* to start of substring. |
| *regoff_t* | *rm_eo* | Byte offset from start of *string* of the first character after the end of substring. |

**Table B-8 – *regcomp* () *cflags* Argument**

| *flag* | Description |
|---|---|
| REG_EXTENDED | Use Extended Regular Expressions. |
| REG_ICASE | Ignore case in match. See 2.8.2. |
| REG_NOSUB | Report only success/fail in *regexec* (). |
| REG_NEWLINE | Change the handling of <newline>, as described in the text. |

**Table B-9 – *regexec* () *eflags* Argument**

| *flag* | Description |
|---|---|
| REG_NOTBOL | The first character of the string pointed to by *string* is not the beginning of the line. Therefore, the circumflex character (^), when taken as a special character, shall not match the beginning of *string* . |
| REG_NOTEOL | The last character of the string pointed to by *string* is not the end of the line. Therefore, the dollar sign ($), when taken as a special character, shall not match the end of *string* . |

If the REG_NOSUB flag was not set in *cflags* , then *regcomp* () shall set *re_nsub* to the number of parenthesized subexpressions [delimited by \( \) in basic regular expressions or ( ) in extended regular expressions] found in *pattern.*

The *regexec* () function shall compare the null-terminated string specified by *string* against the compiled regular expression *preg* initialized by a previous call to *regcomp* (). If it finds a match, *regexec* () shall return zero; otherwise it shall return nonzero indicating either no match or an error. The *eflags* argument shall be the bitwise inclusive OR of zero or more of the flags shown in Table B-9, which shall be defined in the header <regex.h>.

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

If *nmatch* is zero or REG_NOSUB was set in the *cflags* argument to *regcomp* (), then *regexec* () shall ignore the *pmatch* argument. Otherwise, the *pmatch* argument shall point to an array with at least *nmatch* elements, and *regexec* () shall fill in the elements of that array with offsets of the substrings of *string* that correspond to the parenthesized subexpressions of *pattern* : *pmatch* [*i*].*rm_so* shall be the byte offset of the beginning and *pmatch* [*i*].*rm_eo* shall be one greater than the byte offset of the end of substring *i*. (Subexpression *i* begins at the *i*th matched open parenthesis, counting from 1.) Offsets in *pmatch* [0] shall identify the substring that corresponds to the entire regular expression. Unused elements of *pmatch* up to *pmatch* [*nmatch* −1] shall be filled with −1. If there are more than *nmatch* subexpressions in *pattern* (*pattern* itself counts as a subexpression), then *regexec* () shall still do the match, but shall record only the first *nmatch* substrings.

When matching a basic or extended regular expression, any given parenthesized subexpression of *pattern* might participate in the match of several different substrings of *string* , or it might not match any substring even though the pattern as a whole did match. The following rules shall be used to determine which substrings to report in *pmatch* when matching regular expressions:

(1) If subexpression *i* in a regular expression is not contained within another subexpression, and it participated in the match several times, then the byte offsets in *pmatch* [*i*] shall delimit the last such match.

(2) If subexpression *i* is not contained within another subexpression, and it did not participate in an otherwise successful match, then the byte offsets in *pmatch* [*i*] shall be −1. A subexpression shall not participate in the match when:

   (a) * or \{ \} appears immediately after the subexpression in a basic regular expression, or *, ?, or { } appears immediately after the subexpression in an extended regular expression, and the subexpression did not match (matched zero times), or

   (b) | is used in an extended regular expression to select this subexpression or another, and the other subexpression matched.

(3) If subexpression *i* is contained within another subexpression *j*, and *i* is not contained within any other subexpression that is contained within *j*, and a match of subexpression *j* is reported in *pmatch* [*j*], then the match or nonmatch of subexpression *i* reported in *pmatch* [*i*] shall be as described in (1) and (2) above, but within the substring reported in *pmatch* [*j*] rather than the whole string.

(4) If subexpression *i* is contained in subexpression *j*, and the byte offsets in *pmatch* [*j*] are −1, then the byte offsets in *pmatch* [*i*] also shall be −1.

(5) If subexpression *i* matched a zero-length string, then both byte offsets in *pmatch* [*i*] shall be the byte offset of the character or null terminator immediately following the zero-length string.

If, when *regexec* () is called, the locale is different than when the regular expression was compiled, the result is undefined.

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

If REG_NEWLINE is not set in *cflags*, then a <newline> character in *pattern* or *string* shall be treated as an ordinary character. If REG_NEWLINE is set, then <newline> shall be treated as an ordinary character except as follows:

(1) A <newline> in *string* shall not be matched by a period outside of a bracket expression (see 2.8.3.1.3) or by any form of a nonmatching list (see 2.8.3.2).

(2) A circumflex (^) in *pattern*, when used to specify expression anchoring (see 2.8.4.4 and 2.8.4.6), shall match the zero-length string immediately after a <newline> in *string*, regardless of the setting of REG_NOTBOL.

(3) A dollar-sign ($) in *pattern*, when used to specify expression anchoring, shall match the zero-length string immediately before a <newline> in *string*, regardless of the setting of REG_NOTEOL.

The *regfree*() function shall free any memory allocated by *regcomp*() associated with *preg*.

The *regerror*() function provides a mapping from error codes returned by *regcomp*() and *regexec*() to unspecified printable strings. It shall generate a string corresponding to the value of the *errcode* argument, which shall be the last nonzero value returned by *regcomp*() or *regexec*() with the given value of *preg*. If *errcode* is not such a value, the content of the generated string is unspecified. If *preg* is (*regexec_t*)0, but *errcode* is a value returned by a previous call to *regexec*() or *regcomp*(), then *regerror*() still shall generate an error string corresponding to the value of *errcode*, but it might not be as detailed under some implementations.

If the *errbuf_size* argument is not zero, *regerror*() shall place the generated string into the *errbuf_size*-byte buffer pointed to by *errbuf*. If the string (including the terminating null) cannot fit in the buffer, *regerror*() shall truncate the string and null-terminate the result.

If *errbuf_size* is zero, *regerror*() shall ignore the *errbuf* argument, but shall return the integer value described below.

If the *preg* argument to *regexec*() or *regfree*() is not a compiled regular expression returned by *regcomp*(), the result is undefined. A *preg* shall no longer be treated as a compiled regular expression after it is given to *regfree*().

### B.5.3 Returns

On successful completion, the *regcomp*() function shall return zero. On successful completion, the *regexec*() function shall return zero to indicate that *string* matched *pattern*, or REG_NOMATCH (which shall be defined in <regex.h>) to indicate no match.

The *regerror*() function shall return the size of the buffer needed to hold the entire generated string, including the null termination. If the return value is greater than *errbuf_size*, the string returned in the buffer pointed to by *errbuf* has been truncated.

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

**Table B-10 – *regcomp* (), *regexec* () Return Values**

| Error Code | Description |
|---|---|
| REG_NOMATCH | *regexec* () failed to match |
| REG_BADPAT | Invalid regular expression |
| REG_ECOLLATE | Invalid collating element referenced |
| REG_ECTYPE | Invalid character class type referenced |
| REG_EESCAPE | Trailing \ in pattern |
| REG_ESUBREG | Number in \ *digit* invalid or in error |
| REG_EBRACK | [ ] imbalance |
| REG_EPAREN | \( \) or ( ) imbalance |
| REG_EBRACE | \{ \} imbalance |
| REG_BADBR | Content of \{ \} invalid: Not a number, number too large, more than two numbers, first larger than second |
| REG_ERANGE | Invalid endpoint in range expression |
| REG_ESPACE | Out of memory |
| REG_BADRPT | ?, *, or + not preceded by valid regular expression |

### B.5.4 Errors

If *regcomp* () or *regexec* () fails, it shall return a nonzero value indicating the type of failure. Table B-10 contains the names of macros for error codes that may be returned. If a code is returned, the interpretation shall be as given in the table. The implementation shall define the macros in Table B-10 in `<regex.h>`, and may define additional macros beginning with 'REG_'' for other error codes.

If *regcomp* () detects an illegal regular expression, it may return REG_BADPAT, or it may return one of the error codes that more precisely describes the error.

### B.5.5 Rationale. *(This subclause is not a part of P1003.2)*

### Examples, Usage

An example of using the functions is shown in Figure B-3

The following demonstrates how the REG_NOTBOL flag could be used with *regexec* () to find all substrings in a line that match a pattern supplied by a user. (For simplicity of the example, very little error checking is done.)

```
(void) regcomp (&re, pattern, 0);
/* this call to regexec() finds the first match on the line */
error = regexec (&re, &buffer[0], 1, &pm, 0);
while (error == 0) {    /* while matches found */
        <substring found between pm.rm_sp and pm.rm_ep>
        /* This call to regexec() finds the next match */
        error = regexec (&re, pm.rm_ep, 1, &pm, REG_NOTBOL);
}
```

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

```
#include <regex.h>

/*
 * Match string against the extended regular expression in
 * pattern, treating errors as no match.
 *
 * Return 1 for match, 0 for no match.
 */

int
match(const char *string, const char *pattern)
{
        int     status;
        regex_t re;

        if (regcomp(&re, pattern, REG_EXTENDED|REG_NOSUB) != 0) {
                return(0);      /* report error */
        }
        status = regexec(&re, string, (size_t) 0, NULL, 0);
        regfree(&re);
        if (status != 0) {
                return(0);      /* report error */
        }
        return status == 0;
}
```

**Figure B-3 – Example Regular Expression Matching**

An application could use `regerror(code,preg,NULL,(size_t)0)` to find out how big a buffer is needed for the generated string, *malloc* () a buffer to hold the string, and then call *regerror* () again to get the string. Alternately, it could allocate a fixed, static buffer that is big enough to hold most strings (perhaps 128 bytes), and then *malloc* () a larger buffer if it finds that this is too small.

The *regmatch* () function must fill in all *nmatch* elements of *pmatch* , where *nmatch* and *pmatch* are supplied by the application, even if some elements of *pmatch* do not correspond to subexpressions in *pattern* . The application writer should note that there is probably no reason for using a value of *nmatch* that is larger than *preg->re_nsub* .

### History of Decisions Made

The REG_ICASE flag supports the operations taken by the `grep -i` option and the historical implementations of `ex` and `vi`. Including this flag will make it easier for application code to be written that does the same thing as these utilities.

The substrings reported in *pmatch* [] are defined using offsets from the start of the string rather than pointers. Since this is a new interface, there should be no impact on historical implementations or applications, and offsets should be just as

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

easy to use as pointers. The change to offsets was made to facilitate future extensions in which the string to be searched is presented to *regexec* () in blocks, allowing a string to be searched that is not all in memory at once.

A new type *regoff_t* is used for the elements of *pmatch* [] to ensure that the application can represent either the largest possible array in memory (important for a POSIX. 2-conforming application) or the largest possible file (important for an application using the extension where a file is searched in chunks).

The working group has rejected, at least for now, the inclusion of a *regsub* () function that would be used to do substitutions for a matched regular expression. While such a routine would be useful to some applications, its utility would be much more limited than the matching function described here. Both regular expression parsing and substitution are possible to implement without support other than that required by the C Standard {7}, but matching is much more complex than substituting. The only ''difficult'' part of substitution, given the information supplied by *regexec* (), is finding the next character in a string when there can be multibyte characters. That is a much wider issue, and one that needs a more general solution.

The *errno* variable has not been used for error returns to avoid cluttering up the *errno* namespace for this feature.

In Draft 9, the interface was modified so that the matched substrings *rm_sp* and *rm_ep* are in a separate *regmatch_t* structure instead of in *regex_t* . This allows a single compiled regular expression to be used simultaneously in several contexts; in *main* () and a signal handler, perhaps, or in multiple threads of lightweight processes. (The *preg* argument to *regexec* () is declared with type `const`, so the implementation is not permitted to use the structure to store intermediate results.) It also allows an application to request an arbitrary number of substrings from a regular expression. (Previous versions reported only ten substrings.) The number of subexpressions in the regular expression is reported in *re_nsub* in *preg* . With this change to *regexec* (), consideration was given to dropping the REG _NOSUB flag, since the user can now specify this with a zero *nmatch* argument to *regexec* (). However, keeping REG _NOSUB allows an implementation to use a different (perhaps more efficient) algorithm if it knows in *regcomp* () that no subexpressions need be reported. The implementation is only required to fill in *pmatch* if *nmatch* is not zero and if REG _NOSUB is not specified. Note that the *size_t* type, as defined in the C Standard {7}, is unsigned, so the description of *regexec* () does not need to address negative values of *nmatch* .

The rules for reporting substrings of extended regular expressions are consistent with those used by Henry Spencer's ''almost public domain'' version of *regexec* ().

The REG _NOTBOL and REG _NOTEOL flags were added to *regexec* () in Draft 9. REG _NOTBOL was added to allow an application to do repeated searches for the same pattern in a line. If the pattern contains a circumflex character that should match the beginning of a line, then the pattern should only match when matched against the beginning of the line. Without the REG _NOTBOL flag, the application could rewrite the expression for subsequent matches, but in the general case this would require parsing the expression. The need for REG _NOTEOL is not as clear;

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

it was added for symmetry.

The addition of the *regerror* () function addresses the historical need for portable application programs to have access to error information more than ''Function failed to compile/match your regular expression for unknown reasons.'' 1

This interface provides for two different methods of dealing with error conditions. The specific error codes (REG_EBRACE, for example), defined in <regex.h>, allow an application to recover from an error if it is so able. Many applications, especially those that use patterns supplied by a user, will not try to deal with specific error cases, but will just use *regerror* () to obtain a human-readable error message to present to the user.

The *regerror* () function uses a scheme similar to *confstr* () to deal with the problem of allocating memory to hold the generated string. The scheme used by *strerror* () in the C Standard {7} was considered unacceptable since it creates difficulties for multithreaded applications. (POSIX.4a, a standard for threads, started balloting 1 in January 1991.) A different scheme used by *regerror* () in one draft of this stan- 1 dard was eliminated to improve internal consistency, and because the current interface produced greater consensus than the other.

The *preg* argument is provided to *regerror* () to allow an implementation to generate a more descriptive message than would be possible with *errcode* alone. An implementation might, for example, save the character offset of the offending character of the pattern in a field of *preg*, and then include that in the generated message string. The implementation may also ignore *preg*.

A REG_FILENAME flag was considered, but omitted. This flag caused *regexec* () to match patterns as described in 3.13 instead of regular expressions. This service is now provided by the *fnmatch* () function [see B.6].

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

## B.6 C Binding for Match Filename or Pathname

Function: *fnmatch* ()

### B.6.1 Synopsis

```
#include <fnmatch.h>
int fnmatch(const char *pattern , const char *string , int flags );
```

### B.6.2 Description

The *fnmatch* () function shall match patterns as described in 3.13.1 and 3.13.2. It checks the string specified by the *string* argument to see if it matches the pattern specified by the *pattern* argument.

The *flags* argument modifies the interpretation of *pattern* and *string* . It is the bitwise inclusive OR of zero or more of the flags shown in Table B-11, which are defined in the header <fnmatch.h>. If the FNM _PATHNAME flag is set in *flags* , then a slash character in *string* shall be explicitly matched by a slash in *pattern* ; it shall not be matched by either the asterisk or question-mark special characters, nor by a bracket expression. If the FNM _PATHNAME flag is not set, the slash character shall be treated as an ordinary character.

**Table B-11 – *fnmatch* () *flags* Argument**

| *flags* | Description |
|---|---|
| FNM _NOESCAPE | Disable backslash escaping |
| FNM _PATHNAME | Slash in *string* only matches slash in *pattern* |
| FNM _PERIOD | Leading period in *string* must be exactly matched by period in *pattern* |

If FNM _NOESCAPE is not set in *flags* , a backslash character (\) in *pattern* followed by any other character shall match that second character in *string* . In particular, '\\' shall match a backslash in *string* . If FNM _NOESCAPE is set, a backslash character shall be treated as an ordinary character.

If FNM _PERIOD is set in *flags* , then a leading period in *string* shall match a period in *pattern* as described by rule (2) in 3.13.2, where the location of ''leading'' is indicated by the value of FNM _PATHNAME :

— If FNM _PATHNAME is set, a period is ''leading'' if it is the first character in *string* or if it immediately follows a slash.

— If FNM _PATHNAME is not set, a period is ''leading'' only if it is the first character of *string* .

If FNM _PERIOD is not set, then no special restrictions shall be placed on matching a period.

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