NAME

PCRE2 - Perl-compatible regular expressions (revised API)

SYNOPSIS

#include <pcre2posix.h>

int pcre2_regcomp(regex_t ****preg*, const char ****pattern***,**
int *cflags***);**

int pcre2_regexec(const regex_t ****preg*, const char ****string***,**
size_t *nmatch***, regmatch_t** *pmatch***[], int** *eflags***);**

size_t pcre2_regerror(int *errcode***, const regex_t ****preg***,**
char ****errbuf*, size_t** *errbuf_size***);**

void pcre2_regfree(regex_t ****preg*);**

DESCRIPTION

This set of functions provides a POSIX-style API for the PCRE2 regular expression 8-bit library. There are no POSIX-style wrappers for PCRE2's 16-bit and 32-bit libraries. See the **pcre2api** documentation for a description of PCRE2's native API, which contains much additional functionality.

The functions described here are wrapper functions that ultimately call the PCRE2 native API. Their prototypes are defined in the **pcre2posix.h** header file, and they all have unique names starting with **pcre2_**. However, the **pcre2posix.h** header also contains macro definitions that convert the standard POSIX names such **regcomp()** into **pcre2_regcomp()** etc. This means that a program can use the usual POSIX names without running the risk of accidentally linking with POSIX functions from a different library.

On Unix-like systems the PCRE2 POSIX library is called **libpcre2-posix**, so can be accessed by adding **-lpcre2-posix** to the command for linking an application. Because the POSIX functions call the native ones, it is also necessary to add **-lpcre2-8**.

Although they were not defined as protypes in **pcre2posix.h**, releases 10.33 to 10.36 of the library contained functions with the POSIX names **regcomp()** etc. These simply passed their arguments to the PCRE2 functions. These functions were provided for backwards compatibility with earlier versions of PCRE2, which had only POSIX names. However, this has proved troublesome in situations where a program links with several libraries, some of which use PCRE2's POSIX interface while others use the real POSIX functions. For this reason, the POSIX names have been removed since release 10.37.

Calling the header file **pcre2posix.h** avoids any conflict with other POSIX libraries. It can, of course, be renamed or aliased as **regex.h**, which is the "correct" name, if there is no clash. It provides two structure types, *regex_t* for compiled internal forms, and *regmatch_t* for returning captured substrings. It also defines some constants whose names start with "REG_"; these are used for setting options and identifying error codes.

USING THE POSIX FUNCTIONS

Those POSIX option bits that can reasonably be mapped to PCRE2 native options have been implemented. In addition, the option REG_EXTENDED is defined with the value zero. This has no effect, but since programs that are written to the POSIX interface often use it, this makes it easier to slot in PCRE2 as a replacement library. Other POSIX options are not even defined.

There are also some options that are not defined by POSIX. These have been added at the request of users who want to make use of certain PCRE2-specific features via the POSIX calling interface or to add BSD or GNU functionality.

When PCRE2 is called via these functions, it is only the API that is POSIX-like in style. The syntax and semantics of the regular expressions themselves are still those of Perl, subject to the setting of various PCRE2 options, as described below. "POSIX-like in style" means that the API approximates to the POSIX definition; it is not fully POSIX-compatible, and in multi-unit encoding domains it is probably even less compatible.

The descriptions below use the actual names of the functions, but, as described above, the standard POSIX names (without the **pcre2_** prefix) may also be used.

COMPILING A PATTERN

The function **pcre2_regcomp()** is called to compile a pattern into an internal form. By default, the pattern is a C string terminated by a binary zero (but see REG_PEND below). The *preg* argument is a pointer to a **regex_t** structure that is used as a base for storing information about the compiled regular expression. (It is also used for input when REG_PEND is set.)

The argument *cflags* is either zero, or contains one or more of the bits defined by the following macros:

REG_DOTALL

The PCRE2_DOTALL option is set when the regular expression is passed for compilation to the native function. Note that REG_DOTALL is not part of the POSIX standard.

REG_ICASE

The PCRE2_CASELESS option is set when the regular expression is passed for compilation to the native function.

REG_NEWLINE

The PCRE2_MULTILINE option is set when the regular expression is passed for compilation to the native function. Note that this does *not* mimic the defined POSIX behaviour for REG_NEWLINE (see the following section).

REG_NOSPEC

The PCRE2_LITERAL option is set when the regular expression is passed for compilation to the native function. This disables all meta characters in the pattern, causing it to be treated as a literal string. The only other options that are allowed with REG_NOSPEC are REG_ICASE, REG_NOSUB, REG_PEND, and REG_UTF. Note that REG_NOSPEC is not part of the POSIX standard.

REG_NOSUB

When a pattern that is compiled with this flag is passed to **pcre2_regexec()** for matching, the *nmatch* and *pmatch* arguments are ignored, and no captured strings are returned. Versions of the PCRE library prior to 10.22 used to set the PCRE2_NO_AUTO_CAPTURE compile option, but this no longer happens because it disables the use of backreferences.

REG_PEND

If this option is set, the **reg_endp** field in the *preg* structure (which has the type const char *) must be set to point to the character beyond the end of the pattern before calling **pcre2_regcomp()**. The pattern itself may now contain binary zeros, which are treated as data characters. Without REG_PEND, a binary zero terminates the pattern and the **re_endp** field is ignored. This is a GNU extension to the POSIX standard and should be used with caution in software intended to be portable to other systems.

REG_UCP

The PCRE2_UCP option is set when the regular expression is passed for compilation to the native function. This causes PCRE2 to use Unicode properties when matchine \d, \w, etc., instead of just recognizing ASCII values. Note that REG_UCP is not part of the POSIX standard.

REG_UNGREEDY

The PCRE2_UNGREEDY option is set when the regular expression is passed for compilation to the

native function. Note that REG_UNGREEDY is not part of the POSIX standard.

REG_UTF

The PCRE2_UTF option is set when the regular expression is passed for compilation to the native function. This causes the pattern itself and all data strings used for matching it to be treated as UTF-8 strings. Note that REG_UTF is not part of the POSIX standard.

In the absence of these flags, no options are passed to the native function. This means the the regex is compiled with PCRE2 default semantics. In particular, the way it handles newline characters in the subject string is the Perl way, not the POSIX way. Note that setting PCRE2_MULTILINE has only *some* of the effects specified for REG_NEWLINE. It does not affect the way newlines are matched by the dot metacharacter (they are not) or by a negative class such as [^a] (they are).

The yield of **pcre2_regcomp()** is zero on success, and non-zero otherwise. The *preg* structure is filled in on success, and one other member of the structure (as well as *re_endp*) is public: *re_nsub* contains the number of capturing subpatterns in the regular expression. Various error codes are defined in the header file.

NOTE: If the yield of **pcre2_regcomp()** is non-zero, you must not attempt to use the contents of the *preg* structure. If, for example, you pass it to **pcre2_regexec()**, the result is undefined and your program is likely to crash.

MATCHING NEWLINE CHARACTERS

This area is not simple, because POSIX and Perl take different views of things. It is not possible to get PCRE2 to obey POSIX semantics, but then PCRE2 was never intended to be a POSIX engine. The following table lists the different possibilities for matching newline characters in Perl and PCRE2:

	Default	Change with
. matches newline	no	PCRE2_DOTALL
newline matches [^a]	yes	not changeable
$ matches \n at end	yes	PCRE2_DOLLAR_ENDONLY
$ matches \n in middle	no	PCRE2_MULTILINE
^ matches \n in middle	no	PCRE2_MULTILINE

This is the equivalent table for a POSIX-compatible pattern matcher:

	Default	Change with

. matches newline	yes	REG_NEWLINE
newline matches [^a]	yes	REG_NEWLINE
$ matches \n at end	no	REG_NEWLINE
$ matches \n in middle	no	REG_NEWLINE
^ matches \n in middle	no	REG_NEWLINE

This behaviour is not what happens when PCRE2 is called via its POSIX API. By default, PCRE2's behaviour is the same as Perl's, except that there is no equivalent for PCRE2_DOLLAR_ENDONLY in Perl. In both PCRE2 and Perl, there is no way to stop newline from matching [^a].

Default POSIX newline handling can be obtained by setting PCRE2_DOTALL and PCRE2_DOLLAR_ENDONLY when calling **pcre2_compile()** directly, but there is no way to make PCRE2 behave exactly as for the REG_NEWLINE action. When using the POSIX API, passing REG_NEWLINE to PCRE2's **pcre2_regcomp()** function causes PCRE2_MULTILINE to be passed to **pcre2_compile()**, and REG_DOTALL passes PCRE2_DOTALL. There is no way to pass PCRE2_DOLLAR_ENDONLY.

MATCHING A PATTERN

The function **pcre2_regexec()** is called to match a compiled pattern *preg* against a given *string*, which is by default terminated by a zero byte (but see REG_STARTEND below), subject to the options in *eflags*. These can be:

REG_NOTBOL

The PCRE2_NOTBOL option is set when calling the underlying PCRE2 matching function.

REG_NOTEMPTY

The PCRE2_NOTEMPTY option is set when calling the underlying PCRE2 matching function. Note that REG_NOTEMPTY is not part of the POSIX standard. However, setting this option can give more POSIX-like behaviour in some situations.

REG_NOTEOL

The PCRE2_NOTEOL option is set when calling the underlying PCRE2 matching function.

REG_STARTEND

When this option is set, the subject string starts at *string + pmatch[0].rm_so* and ends at *string + pmatch[0].rm_eo*, which should point to the first character beyond the string. There may be binary

zeros within the subject string, and indeed, using REG_STARTEND is the only way to pass a subject string that contains a binary zero.

Whatever the value of *pmatch[0].rm_so*, the offsets of the matched string and any captured substrings are still given relative to the start of *string* itself. (Before PCRE2 release 10.30 these were given relative to *string* + *pmatch[0].rm_so*, but this differs from other implementations.)

This is a BSD extension, compatible with but not specified by IEEE Standard 1003.2 (POSIX.2), and should be used with caution in software intended to be portable to other systems. Note that a non-zero *rm_so* does not imply REG_NOTBOL; REG_STARTEND affects only the location and length of the string, not how it is matched. Setting REG_STARTEND and passing *pmatch* as NULL are mutually exclusive; the error REG_INVARG is returned.

If the pattern was compiled with the REG_NOSUB flag, no data about any matched strings is returned. The *nmatch* and *pmatch* arguments of **pcre2_regexec()** are ignored (except possibly as input for REG_STARTEND).

The value of *nmatch* may be zero, and the value *pmatch* may be NULL (unless REG_STARTEND is set); in both these cases no data about any matched strings is returned.

Otherwise, the portion of the string that was matched, and also any captured substrings, are returned via the *pmatch* argument, which points to an array of *nmatch* structures of type *regmatch_t*, containing the members *rm_so* and *rm_eo*. These contain the byte offset to the first character of each substring and the offset to the first character after the end of each substring, respectively. The 0th element of the vector relates to the entire portion of *string* that was matched; subsequent elements relate to the capturing subpatterns of the regular expression. Unused entries in the array have both structure members set to -1.

A successful match yields a zero return; various error codes are defined in the header file, of which REG_NOMATCH is the "expected" failure code.

ERROR MESSAGES

The **pcre2_regerror()** function maps a non-zero errorcode from either **pcre2_regcomp()** or **pcre2_regexec()** to a printable message. If *preg* is not NULL, the error should have arisen from the use of that structure. A message terminated by a binary zero is placed in *errbuf*. If the buffer is too short, only the first *errbuf_size* - 1 characters of the error message are used. The yield of the function is the size of buffer needed to hold the whole message, including the terminating zero. This value is greater than *errbuf_size* if the message was truncated.

MEMORY USAGE

Compiling a regular expression causes memory to be allocated and associated with the *preg* structure. The function **pcre2_regfree()** frees all such memory, after which *preg* may no longer be used as a compiled expression.

AUTHOR

Philip Hazel
University Computing Service
Cambridge, England.

REVISION

Last updated: 26 April 2021