NAME

PCRE - Perl-compatible regular expressions.

SYNOPSIS

#include <pcreposix.h>

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

int regexec(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*);**

DESCRIPTION

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

The functions described here are just wrapper functions that ultimately call the PCRE native API. Their prototypes are defined in the **pcreposix.h** header file, and on Unix systems the library itself is called **pcreposix.a**, so can be accessed by adding **-lpcreposix** to the command for linking an application that uses them. Because the POSIX functions call the native ones, it is also necessary to add **-lpcre**.

I have implemented only those POSIX option bits that can be reasonably mapped to PCRE native options. 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 PCRE as a replacement library. Other POSIX options are not even defined.

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

When PCRE 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 PCRE 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-byte encoding domains it is probably even less compatible.

The header for these functions is supplied as **pcreposix.h** to avoid any potential clash with other POSIX libraries. It can, of course, be renamed or aliased as **regex.h**, which is the "correct" name. 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.

COMPILING A PATTERN

The function **regcomp()** is called to compile a pattern into an internal form. The pattern is a C string terminated by a binary zero, and is passed in the argument *pattern*. 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.

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

REG_DOTALL

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

REG_NEWLINE

The PCRE_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_NOSUB

The PCRE_NO_AUTO_CAPTURE option is set when the regular expression is passed for compilation to the native function. In addition, when a pattern that is compiled with this flag is passed to **regexec()** for matching, the *nmatch* and *pmatch* arguments are ignored, and no captured strings are returned.

REG_UCP

The PCRE_UCP option is set when the regular expression is passed for compilation to the native function. This causes PCRE to use Unicode properties when matchine \d, \w, etc., instead of just recognizing ASCII values. Note that REG_UTF8 is not part of the POSIX standard.

REG_UNGREEDY

The PCRE_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_UTF8

The PCRE_UTF8 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_UTF8 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 PCRE 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 PCRE_MULTILINE has only *some* of the effects specified for REG_NEWLINE. It does not affect the way newlines are matched by . (they are not) or by a negative class such as [^a] (they are).

The yield of **regcomp()** is zero on success, and non-zero otherwise. The *preg* structure is filled in on success, and one member of the structure 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 **regcomp()** is non-zero, you must not attempt to use the contents of the *preg* structure. If, for example, you pass it to **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 PCRE to obey POSIX semantics, but then PCRE was never intended to be a POSIX engine. The following table lists the different possibilities for matching newline characters in PCRE:

	Default	Change with
. matches newline	no	PCRE_DOTALL
newline matches [^a]	yes	not changeable
$ matches \n at end	yes	PCRE_DOLLARENDONLY
$ matches \n in middle	no	PCRE_MULTILINE
^ matches \n in middle	no	PCRE_MULTILINE

This is the equivalent table for POSIX:

	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

PCRE's behaviour is the same as Perl's, except that there is no equivalent for PCRE_DOLLAR_ENDONLY in Perl. In both PCRE and Perl, there is no way to stop newline from matching [^a].

The default POSIX newline handling can be obtained by setting PCRE_DOTALL and PCRE_DOLLAR_ENDONLY, but there is no way to make PCRE behave exactly as for the REG_NEWLINE action.

MATCHING A PATTERN

The function **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 PCRE_NOTBOL option is set when calling the underlying PCRE matching function.

REG_NOTEMPTY

The PCRE_NOTEMPTY option is set when calling the underlying PCRE 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 PCRE_NOTEOL option is set when calling the underlying PCRE matching function.

REG_STARTEND

The string is considered to start at *string + pmatch[0].rm_so* and to have a terminating NUL located at *string + pmatch[0].rm_eo* (there need not actually be a NUL at that location), regardless of the value of *nmatch*. 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 of the string, not how it is matched.

If the pattern was compiled with the REG_NOSUB flag, no data about any matched strings is returned. The *nmatch* and *pmatch* arguments of **regexec()** are ignored.

If the value of *nmatch* is zero, or if the value *pmatch* is NULL, 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 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 **regerror()** function maps a non-zero errorcode from either **regcomp()** or **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*. The length of the message, including the zero, is limited to *errbuf_size*. The yield of the function is the size of buffer needed to hold the whole message.

MEMORY USAGE

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

AUTHOR

Philip Hazel
University Computing Service
Cambridge CB2 3QH, England.

REVISION

Last updated: 09 January 2012