NAME

PCRE2 - Perl-compatible regular expressions (revised API)

EXPERIMENTAL PATTERN CONVERSION FUNCTIONS

This document describes a set of functions that can be used to convert "foreign" patterns into PCRE2 regular expressions. This facility is currently experimental, and may be changed in future releases. Two kinds of pattern, globs and POSIX patterns, are supported.

THE CONVERT CONTEXT

pcre2_convert_context *pcre2_convert_context_create(
  pcre2_general_context ***_gcontext_);**

pcre2_convert_context *pcre2_convert_context_copy(
  pcre2_convert_context ***_cvcontext_);**

void pcre2_convert_context_free(pcre2_convert_context ***_cvcontext_);**

int pcre2_set_glob_escape(pcre2_convert_context ***_cvcontext_,**
  uint32_t _escape_char_**);**

int pcre2_set_glob_separator(pcre2_convert_context ***_cvcontext_,**
  uint32_t _separator_char_**);**

A convert context is used to hold parameters that affect the way that pattern conversion works. Like all PCRE2 contexts, you need to use a context only if you want to override the defaults. There are the usual create, copy, and free functions. If custom memory management functions are set in a general context that is passed to **pcre2_convert_context_create()**, they are used for all memory management within the conversion functions.

There are only two parameters in the convert context at present. Both apply only to glob conversions. The escape character defaults to grave accent under Windows, otherwise backslash. It can be set to zero, meaning no escape character, or to any punctuation character with a code point less than 256. The separator character defaults to backslash under Windows, otherwise forward slash. It can be set to forward slash, backslash, or dot.

The two setting functions return zero on success, or PCRE2_ERROR_BADDATA if their second argument is invalid.

THE CONVERSION FUNCTION

int pcre2_pattern_convert(PCRE2_SPTR _pattern_**, PCRE2_SIZE** _length_**,**

uint32_t *options***, PCRE2_UCHAR *****buffer***,**
PCRE2_SIZE ****blength*, pcre2_convert_context ****cvcontext***);**

void pcre2_converted_pattern_free(PCRE2_UCHAR ****converted_pattern*);**

The first two arguments of **pcre2_pattern_convert()** define the foreign pattern that is to be converted. The length may be given as PCRE2_ZERO_TERMINATED. The **options** argument defines how the pattern is to be processed. If the input is UTF, the PCRE2_CONVERT_UTF option should be set. PCRE2_CONVERT_NO_UTF_CHECK may also be set if you are sure the input is valid. One or more of the glob options, or one of the following POSIX options must be set to define the type of conversion that is required:

```
PCRE2_CONVERT_GLOB
PCRE2_CONVERT_GLOB_NO_WILD_SEPARATOR
PCRE2_CONVERT_GLOB_NO_STARSTAR
PCRE2_CONVERT_POSIX_BASIC
PCRE2_CONVERT_POSIX_EXTENDED
```

Details of the conversions are given below. The **buffer** and **blength** arguments define how the output is handled:

If **buffer** is NULL, the function just returns the length of the converted pattern via **blength**. This is one less than the length of buffer needed, because a terminating zero is always added to the output.

If **buffer** points to a NULL pointer, an output buffer is obtained using the allocator in the context or **malloc()** if no context is supplied. A pointer to this buffer is placed in the variable to which **buffer** points. When no longer needed the output buffer must be freed by calling **pcre2_converted_pattern_free()**. If this function is called with a NULL argument, it returns immediately without doing anything.

If **buffer** points to a non-NULL pointer, **blength** must be set to the actual length of the buffer provided (in code units).

In all cases, after successful conversion, the variable pointed to by **blength** is updated to the length actually used (in code units), excluding the terminating zero that is always added.

If an error occurs, the length (via **blength**) is set to the offset within the input pattern where the error was detected. Only gross syntax errors are caught; there are plenty of errors that will get passed on for **pcre2_compile()** to discover.

The return from **pcre2_pattern_convert()** is zero on success or a non-zero PCRE2 error code. Note that PCRE2 error codes may be positive or negative: **pcre2_compile()** uses mostly positive codes and **pcre2_match()** negative ones; **pcre2_convert()** uses existing codes of both kinds. A textual error message can be obtained by calling **pcre2_get_error_message()**.

CONVERTING GLOBS

Globs are used to match file names, and consequently have the concept of a "path separator", which defaults to backslash under Windows and forward slash otherwise. If PCRE2_CONVERT_GLOB is set, the wildcards * and ? are not permitted to match separator characters, but the double-star (**) feature (which does match separators) is supported.

PCRE2_CONVERT_GLOB_NO_WILD_SEPARATOR matches globs with wildcards allowed to match separator characters. PCRE2_CONVERT_GLOB_NO_STARSTAR matches globs with the double-star feature disabled. These options may be given together.

CONVERTING POSIX PATTERNS

POSIX defines two kinds of regular expression pattern: basic and extended. These can be processed by setting PCRE2_CONVERT_POSIX_BASIC or PCRE2_CONVERT_POSIX_EXTENDED, respectively.

In POSIX patterns, backslash is not special in a character class. Unmatched closing parentheses are treated as literals.

In basic patterns, ? + | {} and () must be escaped to be recognized as metacharacters outside a character class. If the first character in the pattern is * it is treated as a literal. ^ is a metacharacter only at the start of a branch.

In extended patterns, a backslash not in a character class always makes the next character literal, whatever it is. There are no backreferences.

Note: POSIX mandates that the longest possible match at the first matching position must be found. This is not what **pcre2_match()** does; it yields the first match that is found. An application can use **pcre2_dfa_match()** to find the longest match, but that does not support backreferences (but then neither do POSIX extended patterns).

AUTHOR

Philip Hazel
University Computing Service
Cambridge, England.

REVISION

Last updated: 28 June 2018