NAME

PCRE - Perl-compatible regular expressions

SYNOPSIS

#include <pcre.h>

int (*pcre_callout)(pcre_callout_block *);

int (*pcre16_callout)(pcre16_callout_block *);

int (*pcre32_callout)(pcre32_callout_block *);

DESCRIPTION

PCRE provides a feature called "callout", which is a means of temporarily passing control to the caller of PCRE in the middle of pattern matching. The caller of PCRE provides an external function by putting its entry point in the global variable *pcre_callout* (*pcre16_callout* for the 16-bit library, *pcre32_callout* for the 32-bit library). By default, this variable contains NULL, which disables all calling out.

Within a regular expression, (?C) indicates the points at which the external function is to be called. Different callout points can be identified by putting a number less than 256 after the letter C. The default value is zero. For example, this pattern has two callout points:

```
  (?C1)abc(?C2)def
```

If the PCRE_AUTO_CALLOUT option bit is set when a pattern is compiled, PCRE automatically inserts callouts, all with number 255, before each item in the pattern. For example, if PCRE_AUTO_CALLOUT is used with the pattern

```
  A(\d{2}|--)
```

it is processed as if it were

```
(?C255)A(?C255)((?C255)\d{2}(?C255)|(?C255)-(?C255)-(?C255))(?C255)
```

Notice that there is a callout before and after each parenthesis and alternation bar. If the pattern contains a conditional group whose condition is an assertion, an automatic callout is inserted immediately before the condition. Such a callout may also be inserted explicitly, for example:

```
  (?(?C9)(?=a)ab|de)
```

This applies only to assertion conditions (because they are themselves independent groups).

Automatic callouts can be used for tracking the progress of pattern matching. The **pcretest** program has a pattern qualifier (/C) that sets automatic callouts; when it is used, the output indicates how the pattern is being matched. This is useful information when you are trying to optimize the performance of a particular pattern.

MISSING CALLOUTS

You should be aware that, because of optimizations in the way PCRE compiles and matches patterns, callouts sometimes do not happen exactly as you might expect.

At compile time, PCRE "auto-possessifies" repeated items when it knows that what follows cannot be part of the repeat. For example, a+[bc] is compiled as if it were a++[bc]. The **pcretest** output when this pattern is anchored and then applied with automatic callouts to the string "aaaa" is:

```
  --->aaaa
   +0 ^        ^
   +1 ^        a+
   +3 ^   ^    [bc]
  No match
```

This indicates that when matching [bc] fails, there is no backtracking into a+ and therefore the callouts that would be taken for the backtracks do not occur. You can disable the auto-possessify feature by passing PCRE_NO_AUTO_POSSESS to **pcre_compile()**, or starting the pattern with (*NO_AUTO_POSSESS). If this is done in **pcretest** (using the /O qualifier), the output changes to this:

```
  --->aaaa
   +0 ^        ^
   +1 ^        a+
   +3 ^   ^    [bc]
   +3 ^  ^     [bc]
   +3 ^ ^      [bc]
   +3 ^^       [bc]
  No match
```

This time, when matching [bc] fails, the matcher backtracks into a+ and tries again, repeatedly, until a+ itself fails.

Other optimizations that provide fast "no match" results also affect callouts. For example, if the pattern is

```
  ab(?C4)cd
```

PCRE knows that any matching string must contain the letter "d". If the subject string is "abyz", the lack of "d" means that matching doesn't ever start, and the callout is never reached. However, with "abyd", though the result is still no match, the callout is obeyed.

If the pattern is studied, PCRE knows the minimum length of a matching string, and will immediately give a "no match" return without actually running a match if the subject is not long enough, or, for unanchored patterns, if it has been scanned far enough.

You can disable these optimizations by passing the PCRE_NO_START_OPTIMIZE option to the matching function, or by starting the pattern with (*NO_START_OPT). This slows down the matching process, but does ensure that callouts such as the example above are obeyed.

THE CALLOUT INTERFACE

During matching, when PCRE reaches a callout point, the external function defined by *pcre_callout* or *pcre[16|32]_callout* is called (if it is set). This applies to both normal and DFA matching. The only argument to the callout function is a pointer to a **pcre_callout** or **pcre[16|32]_callout** block. These structures contains the following fields:

```
  int          version;
  int          callout_number;
  int         *offset_vector;
  const char   *subject;          (8-bit version)
  PCRE_SPTR16   subject;          (16-bit version)
  PCRE_SPTR32   subject;          (32-bit version)
  int          subject_length;
  int          start_match;
  int          current_position;
  int          capture_top;
  int          capture_last;
  void         *callout_data;
  int          pattern_position;
  int          next_item_length;
  const unsigned char *mark;       (8-bit version)
  const PCRE_UCHAR16  *mark;       (16-bit version)
  const PCRE_UCHAR32  *mark;       (32-bit version)
```

The *version* field is an integer containing the version number of the block format. The initial version was 0; the current version is 2. The version number will change again in future if additional fields are added, but the intention is never to remove any of the existing fields.

The *callout_number* field contains the number of the callout, as compiled into the pattern (that is, the number after ?C for manual callouts, and 255 for automatically generated callouts).

The *offset_vector* field is a pointer to the vector of offsets that was passed by the caller to the matching function. When **pcre_exec()** or **pcre[16|32]_exec()** is used, the contents can be inspected, in order to extract substrings that have been matched so far, in the same way as for extracting substrings after a match has completed. For the DFA matching functions, this field is not useful.

The *subject* and *subject_length* fields contain copies of the values that were passed to the matching function.

The *start_match* field normally contains the offset within the subject at which the current match attempt started. However, if the escape sequence \K has been encountered, this value is changed to reflect the modified starting point. If the pattern is not anchored, the callout function may be called several times from the same point in the pattern for different starting points in the subject.

The *current_position* field contains the offset within the subject of the current match pointer.

When the **pcre_exec()** or **pcre[16|32]_exec()** is used, the *capture_top* field contains one more than the number of the highest numbered captured substring so far. If no substrings have been captured, the value of *capture_top* is one. This is always the case when the DFA functions are used, because they do not support captured substrings.

The *capture_last* field contains the number of the most recently captured substring. However, when a recursion exits, the value reverts to what it was outside the recursion, as do the values of all captured substrings. If no substrings have been captured, the value of *capture_last* is -1. This is always the case for the DFA matching functions.

The *callout_data* field contains a value that is passed to a matching function specifically so that it can be passed back in callouts. It is passed in the *callout_data* field of a **pcre_extra** or **pcre[16|32]_extra** data structure. If no such data was passed, the value of *callout_data* in a callout block is NULL. There is a description of the **pcre_extra** structure in the **pcreapi** documentation.

The *pattern_position* field is present from version 1 of the callout structure. It contains the offset to the next item to be matched in the pattern string.

The *next_item_length* field is present from version 1 of the callout structure. It contains the length of the next item to be matched in the pattern string. When the callout immediately precedes an alternation bar, a closing parenthesis, or the end of the pattern, the length is zero. When the callout precedes an opening parenthesis, the length is that of the entire subpattern.

The *pattern_position* and *next_item_length* fields are intended to help in distinguishing between different automatic callouts, which all have the same callout number. However, they are set for all callouts.

The *mark* field is present from version 2 of the callout structure. In callouts from **pcre_exec()** or **pcre[16|32]_exec()** it contains a pointer to the zero-terminated name of the most recently passed (*MARK), (*PRUNE), or (*THEN) item in the match, or NULL if no such items have been passed. Instances of (*PRUNE) or (*THEN) without a name do not obliterate a previous (*MARK). In callouts from the DFA matching functions this field always contains NULL.

RETURN VALUES

The external callout function returns an integer to PCRE. If the value is zero, matching proceeds as normal. If the value is greater than zero, matching fails at the current point, but the testing of other matching possibilities goes ahead, just as if a lookahead assertion had failed. If the value is less than zero, the match is abandoned, the matching function returns the negative value.

Negative values should normally be chosen from the set of PCRE_ERROR_xxx values. In particular, PCRE_ERROR_NOMATCH forces a standard "no match" failure. The error number PCRE_ERROR_CALLOUT is reserved for use by callout functions; it will never be used by PCRE itself.

AUTHOR

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

REVISION

Last updated: 12 November 2013