NAME

PCRE - Perl-compatible regular expressions

SAVING AND RE-USING PRECOMPILED PCRE PATTERNS

If you are running an application that uses a large number of regular expression patterns, it may be useful to store them in a precompiled form instead of having to compile them every time the application is run. If you are not using any private character tables (see the **pcre_maketables()** documentation), this is relatively straightforward. If you are using private tables, it is a little bit more complicated. However, if you are using the just-in-time optimization feature, it is not possible to save and reload the JIT data.

If you save compiled patterns to a file, you can copy them to a different host and run them there. If the two hosts have different endianness (byte order), you should run the **pcre[16|32]_pattern_to_host_byte_order()** function on the new host before trying to match the pattern. The matching functions return PCRE_ERROR_BADENDIANNESS if they detect a pattern with the wrong endianness.

Compiling regular expressions with one version of PCRE for use with a different version is not guaranteed to work and may cause crashes, and saving and restoring a compiled pattern loses any JIT optimization data.

SAVING A COMPILED PATTERN

The value returned by **pcre[16|32]_compile()** points to a single block of memory that holds the compiled pattern and associated data. You can find the length of this block in bytes by calling **pcre[16|32]_fullinfo()** with an argument of PCRE_INFO_SIZE. You can then save the data in any appropriate manner. Here is sample code for the 8-bit library that compiles a pattern and writes it to a file. It assumes that the variable *fd* refers to a file that is open for output:

```
int erroroffset, rc, size;
char *error;
pcre *re;

re = pcre_compile("my pattern", 0, &error, &erroroffset, NULL);
if (re == NULL) { ... handle errors ... }
rc = pcre_fullinfo(re, NULL, PCRE_INFO_SIZE, &size);
if (rc < 0) { ... handle errors ... }
rc = fwrite(re, 1, size, fd);
if (rc != size) { ... handle errors ... }
```

In this example, the bytes that comprise the compiled pattern are copied exactly. Note that this is binary data that may contain any of the 256 possible byte values. On systems that make a distinction between binary and non-binary data, be sure that the file is opened for binary output.

If you want to write more than one pattern to a file, you will have to devise a way of separating them. For binary data, preceding each pattern with its length is probably the most straightforward approach. Another possibility is to write out the data in hexadecimal instead of binary, one pattern to a line.

Saving compiled patterns in a file is only one possible way of storing them for later use. They could equally well be saved in a database, or in the memory of some daemon process that passes them via sockets to the processes that want them.

If the pattern has been studied, it is also possible to save the normal study data in a similar way to the compiled pattern itself. However, if the PCRE_STUDY_JIT_COMPILE was used, the just-in-time data that is created cannot be saved because it is too dependent on the current environment. When studying generates additional information, **pcre[16|32]_study()** returns a pointer to a **pcre[16|32]_extra** data block. Its format is defined in the section on matching a pattern in the **pcreapi** documentation. The *study_data* field points to the binary study data, and this is what you must save (not the **pcre[16|32]_extra** block itself). The length of the study data can be obtained by calling **pcre[16|32]_fullinfo()** with an argument of PCRE_INFO_STUDYSIZE. Remember to check that **pcre[16|32]_study()** did return a non-NULL value before trying to save the study data.

RE-USING A PRECOMPILED PATTERN

Re-using a precompiled pattern is straightforward. Having reloaded it into main memory, called **pcre[16|32]_pattern_to_host_byte_order()** if necessary, you pass its pointer to **pcre[16|32]_exec()** or **pcre[16|32]_dfa_exec()** in the usual way.

However, if you passed a pointer to custom character tables when the pattern was compiled (the *tableptr* argument of **pcre[16|32]_compile()**), you must now pass a similar pointer to **pcre[16|32]_exec()** or **pcre[16|32]_dfa_exec()**, because the value saved with the compiled pattern will obviously be nonsense. A field in a **pcre[16|32]_extra()** block is used to pass this data, as described in the section on matching a pattern in the **pcreapi** documentation.

Warning: The tables that **pcre_exec()** and **pcre_dfa_exec()** use must be the same as those that were used when the pattern was compiled. If this is not the case, the behaviour is undefined.

If you did not provide custom character tables when the pattern was compiled, the pointer in the compiled pattern is NULL, which causes the matching functions to use PCRE’s internal tables. Thus, you do not need to take any special action at run time in this case.

If you saved study data with the compiled pattern, you need to create your own **pcre[16|32]_extra** data block and set the *study_data* field to point to the reloaded study data. You must also set the PCRE_EXTRA_STUDY_DATA bit in the *flags* field to indicate that study data is present. Then pass the **pcre[16|32]_extra** block to the matching function in the usual way. If the pattern was studied for just-in-time optimization, that data cannot be saved, and so is lost by a save/restore cycle.

COMPATIBILITY WITH DIFFERENT PCRE RELEASES

In general, it is safest to recompile all saved patterns when you update to a new PCRE release, though not all updates actually require this.

AUTHOR

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

REVISION

Last updated: 12 November 2013