NAME

PCRE - Perl-compatible regular expressions

UTF-8, UTF-16, UTF-32, AND UNICODE PROPERTY SUPPORT

As well as UTF-8 support, PCRE also supports UTF-16 (from release 8.30) and UTF-32 (from release 8.32), by means of two additional libraries. They can be built as well as, or instead of, the 8-bit library.

UTF-8 SUPPORT

In order process UTF-8 strings, you must build PCRE's 8-bit library with UTF support, and, in addition, you must call **pcre_compile()** with the PCRE_UTF8 option flag, or the pattern must start with the sequence (*UTF8) or (*UTF). When either of these is the case, both the pattern and any subject strings that are matched against it are treated as UTF-8 strings instead of strings of individual 1-byte characters.

UTF-16 AND UTF-32 SUPPORT

In order process UTF-16 or UTF-32 strings, you must build PCRE's 16-bit or 32-bit library with UTF support, and, in addition, you must call **pcre16_compile()** or **pcre32_compile()** with the PCRE_UTF16 or PCRE_UTF32 option flag, as appropriate. Alternatively, the pattern must start with the sequence (*UTF16), (*UTF32), as appropriate, or (*UTF), which can be used with either library. When UTF mode is set, both the pattern and any subject strings that are matched against it are treated as UTF-16 or UTF-32 strings instead of strings of individual 16-bit or 32-bit characters.

UTF SUPPORT OVERHEAD

If you compile PCRE with UTF support, but do not use it at run time, the library will be a bit bigger, but the additional run time overhead is limited to testing the PCRE_UTF[8|16|32] flag occasionally, so should not be very big.

UNICODE PROPERTY SUPPORT

If PCRE is built with Unicode character property support (which implies UTF support), the escape sequences \p{..}, \P{..}, and \X can be used. The available properties that can be tested are limited to the general category properties such as Lu for an upper case letter or Nd for a decimal number, the Unicode script names such as Arabic or Han, and the derived properties Any and L&. Full lists is given in the **pcrepattern** and **pcresyntax** documentation. Only the short names for properties are supported. For example, \p{L} matches a letter. Its Perl synonym, \p{Letter}, is not supported. Furthermore, in Perl, many properties may optionally be prefixed by "Is", for compatibility with Perl 5.6. PCRE does not support this.

Validity of UTF-8 strings

When you set the PCRE_UTF8 flag, the byte strings passed as patterns and subjects are (by default) checked for validity on entry to the relevant functions. The entire string is checked before any other processing takes place. From release 7.3 of PCRE, the check is according the rules of RFC 3629, which are themselves derived from the Unicode specification. Earlier releases of PCRE followed the rules of RFC 2279, which allows the full range of 31-bit values (0 to 0x7FFFFFFF). The current check allows only values in the range U+0 to U+10FFFF, excluding the surrogate area. (From release 8.33 the so-called "non-character" code points are no longer excluded because Unicode corrigendum #9 makes it clear that they should not be.)

Characters in the "Surrogate Area" of Unicode are reserved for use by UTF-16, where they are used in pairs to encode codepoints with values greater than 0xFFFF. The code points that are encoded by UTF-16 pairs are available independently in the UTF-8 and UTF-32 encodings. (In other words, the whole surrogate thing is a fudge for UTF-16 which unfortunately messes up UTF-8 and UTF-32.)

If an invalid UTF-8 string is passed to PCRE, an error return is given. At compile time, the only additional information is the offset to the first byte of the failing character. The run-time functions **pcre_exec()** and **pcre_dfa_exec()** also pass back this information, as well as a more detailed reason code if the caller has

provided memory in which to do this.

In some situations, you may already know that your strings are valid, and therefore want to skip these checks in order to improve performance, for example in the case of a long subject string that is being scanned repeatedly. If you set the PCRE_NO_UTF8_CHECK flag at compile time or at run time, PCRE assumes that the pattern or subject it is given (respectively) contains only valid UTF-8 codes. In this case, it does not diagnose an invalid UTF-8 string.

Note that passing PCRE_NO_UTF8_CHECK to **pcre_compile()** just disables the check for the pattern; it does not also apply to subject strings. If you want to disable the check for a subject string you must pass this option to **pcre_exec()** or **pcre_dfa_exec()**.

If you pass an invalid UTF-8 string when PCRE_NO_UTF8_CHECK is set, the result is undefined and your program may crash.

Validity of UTF-16 strings

When you set the PCRE_UTF16 flag, the strings of 16-bit data units that are passed as patterns and subjects are (by default) checked for validity on entry to the relevant functions. Values other than those in the surrogate range U+D800 to U+DFFF are independent code points. Values in the surrogate range must be used in pairs in the correct manner.

If an invalid UTF-16 string is passed to PCRE, an error return is given. At compile time, the only additional information is the offset to the first data unit of the failing character. The run-time functions **pcre16_exec()** and **pcre16_dfa_exec()** also pass back this information, as well as a more detailed reason code if the caller has provided memory in which to do this.

In some situations, you may already know that your strings are valid, and therefore want to skip these checks in order to improve performance. If you set the PCRE_NO_UTF16_CHECK flag at compile time or at run time, PCRE assumes that the pattern or subject it is given (respectively) contains only valid UTF-16 sequences. In this case, it does not diagnose an invalid UTF-16 string. However, if an invalid string is passed, the result is undefined.

Validity of UTF-32 strings

When you set the PCRE_UTF32 flag, the strings of 32-bit data units that are passed as patterns and subjects are (by default) checked for validity on entry to the relevant functions. This check allows only values in the range U+0 to U+10FFFF, excluding the surrogate area U+D800 to U+DFFF.

If an invalid UTF-32 string is passed to PCRE, an error return is given. At compile time, the only additional information is the offset to the first data unit of the failing character. The run-time functions **pcre32_exec()** and **pcre32_dfa_exec()** also pass back this information, as well as a more detailed reason code if the caller has provided memory in which to do this.

In some situations, you may already know that your strings are valid, and therefore want to skip these checks in order to improve performance. If you set the PCRE_NO_UTF32_CHECK flag at compile time or at run time, PCRE assumes that the pattern or subject it is given (respectively) contains only valid UTF-32 sequences. In this case, it does not diagnose an invalid UTF-32 string. However, if an invalid string is passed, the result is undefined.

General comments about UTF modes

1. Codepoints less than 256 can be specified in patterns by either braced or unbraced hexadecimal escape sequences (for example, \x{b3} or \xb3). Larger values have to use braced sequences.

2. Octal numbers up to \777 are recognized, and in UTF-8 mode they match two-byte characters for values greater than \177.

3. Repeat quantifiers apply to complete UTF characters, not to individual data units, for example: \x{100}{3}.

4. The dot metacharacter matches one UTF character instead of a single data unit.

5. The escape sequence \C can be used to match a single byte in UTF-8 mode, or a single 16-bit data unit in UTF-16 mode, or a single 32-bit data unit in UTF-32 mode, but its use can lead to some strange effects because it breaks up multi-unit characters (see the description of \C in the **pcrepattern** documentation). The use of \C is not supported in the alternative matching function **pcre[16|32]_dfa_exec()**, nor is it supported in UTF mode by the JIT optimization of **pcre[16|32]_exec()**. If JIT optimization is requested for a UTF pattern that contains \C, it will not succeed, and so the matching will be carried out by the normal interpretive function.

6. The character escapes \b, \B, \d, \D, \s, \S, \w, and \W correctly test characters of any code value, but, by default, the characters that PCRE recognizes as digits, spaces, or word characters remain the same set as in non-UTF mode, all with values less than 256. This remains true even when PCRE is built to include Unicode property support, because to do otherwise would slow down PCRE in many common cases. Note in particular that this applies to \b and \B, because they are defined in terms of \w and \W. If you really want to test for a wider sense of, say, "digit", you can use explicit Unicode property tests such as \p{Nd}. Alternatively, if you set the PCRE_UCP option, the way that the character escapes work is changed so that Unicode properties are used to determine which characters match. There are more details in the section on generic character types in the **pcrepattern** documentation.

7. Similarly, characters that match the POSIX named character classes are all low-valued characters, unless the PCRE_UCP option is set.

8. However, the horizontal and vertical white space matching escapes (\h, \H, \v, and \V) do match all the appropriate Unicode characters, whether or not PCRE_UCP is set.

9. Case-insensitive matching applies only to characters whose values are less than 128, unless PCRE is built with Unicode property support. A few Unicode characters such as Greek sigma have more than two codepoints that are case-equivalent. Up to and including PCRE release 8.31, only one-to-one case mappings were supported, but later releases (with Unicode property support) do treat as case-equivalent all versions of characters such as Greek sigma.

AUTHOR

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

REVISION

Last updated: 27 February 2013