NAME

PCRE2 - Perl-compatible regular expressions (revised API)

BUILDING PCRE2

PCRE2 is distributed with a **configure** script that can be used to build the library in Unix-like environments using the applications known as Autotools. Also in the distribution are files to support building using **CMake** instead of **configure**. The text file **README** contains general information about building with Autotools (some of which is repeated below), and also has some comments about building on various operating systems. There is a lot more information about building PCRE2 without using Autotools (including information about using **CMake** and building "by hand") in the text file called **NON-AUTOTOOLS-BUILD**. You should consult this file as well as the **README** file if you are building in a non-Unix-like environment.

PCRE2 BUILD-TIME OPTIONS

The rest of this document describes the optional features of PCRE2 that can be selected when the library is compiled. It assumes use of the **configure** script, where the optional features are selected or deselected by providing options to **configure** before running the **make** command. However, the same options can be selected in both Unix-like and non-Unix-like environments if you are using **CMake** instead of **configure** to build PCRE2.

If you are not using Autotools or **CMake**, option selection can be done by editing the **config.h** file, or by passing parameter settings to the compiler, as described in **NON-AUTOTOOLS-BUILD**.

The complete list of options for **configure** (which includes the standard ones such as the selection of the installation directory) can be obtained by running

```
./configure --help
```

The following sections include descriptions of "on/off" options whose names begin with --enable or --disable. Because of the way that **configure** works, --enable and --disable always come in pairs, so the complementary option always exists as well, but as it specifies the default, it is not described. Options that specify values have names that start with --with. At the end of a **configure** run, a summary of the configuration is output.

BUILDING 8-BIT, 16-BIT AND 32-BIT LIBRARIES

By default, a library called **libpcre2-8** is built, containing functions that take string arguments contained in arrays of bytes, interpreted either as single-byte characters, or UTF-8 strings. You can also build two other libraries, called **libpcre2-16** and **libpcre2-32**, which process strings that are contained in arrays of 16-bit and 32-bit code units, respectively. These can be interpreted either as single-unit characters or UTF-16/UTF-32 strings. To build these additional libraries, add one or both of the following to the

configure command:

--enable-pcre2-16
--enable-pcre2-32

If you do not want the 8-bit library, add

--disable-pcre2-8

as well. At least one of the three libraries must be built. Note that the POSIX wrapper is for the 8-bit library only, and that **pcre2grep** is an 8-bit program. Neither of these are built if you select only the 16-bit or 32-bit libraries.

BUILDING SHARED AND STATIC LIBRARIES

The Autotools PCRE2 building process uses **libtool** to build both shared and static libraries by default. You can suppress an unwanted library by adding one of

--disable-shared
--disable-static

to the **configure** command.

UNICODE AND UTF SUPPORT

By default, PCRE2 is built with support for Unicode and UTF character strings. To build it without Unicode support, add

--disable-unicode

to the **configure** command. This setting applies to all three libraries. It is not possible to build one library with Unicode support and another without in the same configuration.

Of itself, Unicode support does not make PCRE2 treat strings as UTF-8, UTF-16 or UTF-32. To do that, applications that use the library can set the PCRE2_UTF option when they call **pcre2_compile()** to compile a pattern. Alternatively, patterns may be started with (*UTF) unless the application has locked this out by setting PCRE2_NEVER_UTF.

UTF support allows the libraries to process character code points up to 0x10ffff in the strings that they handle. Unicode support also gives access to the Unicode properties of characters, using pattern escapes such as \P, \p, and \X. Only the general category properties such as *Lu* and *Nd*, script names, and some bi-directional properties are supported. Details are given in the **pcre2pattern** documentation.

Pattern escapes such as \d and \w do not by default make use of Unicode properties. The application can request that they do by setting the PCRE2_UCP option. Unless the application has set PCRE2_NEVER_UCP, a pattern may also request this by starting with (*UCP).

DISABLING THE USE OF \C

The \C escape sequence, which matches a single code unit, even in a UTF mode, can cause unpredictable behaviour because it may leave the current matching point in the middle of a multi-code-unit character. The application can lock it out by setting the PCRE2_NEVER_BACKSLASH_C option when calling **pcre2_compile()**. There is also a build-time option

--enable-never-backslash-C

(note the upper case C) which locks out the use of \C entirely.

JUST-IN-TIME COMPILER SUPPORT

Just-in-time (JIT) compiler support is included in the build by specifying

--enable-jit

This support is available only for certain hardware architectures. If this option is set for an unsupported architecture, a building error occurs. If in doubt, use

--enable-jit=auto

which enables JIT only if the current hardware is supported. You can check if JIT is enabled in the configuration summary that is output at the end of a **configure** run. If you are enabling JIT under SELinux you may also want to add

--enable-jit-sealloc

which enables the use of an execmem allocator in JIT that is compatible with SELinux. This has no effect if JIT is not enabled. See the **pcre2jit** documentation for a discussion of JIT usage. When JIT support is enabled, **pcre2grep** automatically makes use of it, unless you add

--disable-pcre2grep-jit

to the **configure** command.

NEWLINE RECOGNITION

By default, PCRE2 interprets the linefeed (LF) character as indicating the end of a line. This is the

normal newline character on Unix-like systems. You can compile PCRE2 to use carriage return (CR) instead, by adding

--enable-newline-is-cr

to the **configure** command. There is also an --enable-newline-is-lf option, which explicitly specifies linefeed as the newline character.

Alternatively, you can specify that line endings are to be indicated by the two-character sequence CRLF (CR immediately followed by LF). If you want this, add

--enable-newline-is-crlf

to the **configure** command. There is a fourth option, specified by

--enable-newline-is-anycrlf

which causes PCRE2 to recognize any of the three sequences CR, LF, or CRLF as indicating a line ending. A fifth option, specified by

--enable-newline-is-any

causes PCRE2 to recognize any Unicode newline sequence. The Unicode newline sequences are the three just mentioned, plus the single characters VT (vertical tab, U+000B), FF (form feed, U+000C), NEL (next line, U+0085), LS (line separator, U+2028), and PS (paragraph separator, U+2029). The final option is

--enable-newline-is-nul

which causes NUL (binary zero) to be set as the default line-ending character.

Whatever default line ending convention is selected when PCRE2 is built can be overridden by applications that use the library. At build time it is recommended to use the standard for your operating system.

WHAT \R MATCHES

By default, the sequence \R in a pattern matches any Unicode newline sequence, independently of what has been selected as the line ending sequence. If you specify

--enable-bsr-anycrlf

the default is changed so that \R matches only CR, LF, or CRLF. Whatever is selected when PCRE2 is built can be overridden by applications that use the library.

HANDLING VERY LARGE PATTERNS

Within a compiled pattern, offset values are used to point from one part to another (for example, from an opening parenthesis to an alternation metacharacter). By default, in the 8-bit and 16-bit libraries, two-byte values are used for these offsets, leading to a maximum size for a compiled pattern of around 64 thousand code units. This is sufficient to handle all but the most gigantic patterns. Nevertheless, some people do want to process truly enormous patterns, so it is possible to compile PCRE2 to use three-byte or four-byte offsets by adding a setting such as

```
--with-link-size=3
```

to the **configure** command. The value given must be 2, 3, or 4. For the 16-bit library, a value of 3 is rounded up to 4. In these libraries, using longer offsets slows down the operation of PCRE2 because it has to load additional data when handling them. For the 32-bit library the value is always 4 and cannot be overridden; the value of --with-link-size is ignored.

LIMITING PCRE2 RESOURCE USAGE

The **pcre2_match()** function increments a counter each time it goes round its main loop. Putting a limit on this counter controls the amount of computing resource used by a single call to **pcre2_match()**. The limit can be changed at run time, as described in the **pcre2api** documentation. The default is 10 million, but this can be changed by adding a setting such as

```
--with-match-limit=500000
```

to the **configure** command. This setting also applies to the **pcre2_dfa_match()** matching function, and to JIT matching (though the counting is done differently).

The **pcre2_match()** function uses heap memory to record backtracking points. The more nested backtracking points there are (that is, the deeper the search tree), the more memory is needed. There is an upper limit, specified in kibibytes (units of 1024 bytes). This limit can be changed at run time, as described in the **pcre2api** documentation. The default limit (in effect unlimited) is 20 million. You can change this by a setting such as

```
--with-heap-limit=500
```

which limits the amount of heap to 500 KiB. This limit applies only to interpretive matching in **pcre2_match()** and **pcre2_dfa_match()**, which may also use the heap for internal workspace when processing complicated patterns. This limit does not apply when JIT (which has its own memory

arrangements) is used.

You can also explicitly limit the depth of nested backtracking in the **pcre2_match()** interpreter. This limit defaults to the value that is set for --with-match-limit. You can set a lower default limit by adding, for example,

```
--with-match-limit-depth=10000
```

to the **configure** command. This value can be overridden at run time. This depth limit indirectly limits the amount of heap memory that is used, but because the size of each backtracking "frame" depends on the number of capturing parentheses in a pattern, the amount of heap that is used before the limit is reached varies from pattern to pattern. This limit was more useful in versions before 10.30, where function recursion was used for backtracking.

As well as applying to **pcre2_match()**, the depth limit also controls the depth of recursive function calls in **pcre2_dfa_match()**. These are used for lookaround assertions, atomic groups, and recursion within patterns. The limit does not apply to JIT matching.

CREATING CHARACTER TABLES AT BUILD TIME

PCRE2 uses fixed tables for processing characters whose code points are less than 256. By default, PCRE2 is built with a set of tables that are distributed in the file *src/pcre2_chartables.c.dist*. These tables are for ASCII codes only. If you add

```
--enable-rebuild-chartables
```

to the **configure** command, the distributed tables are no longer used. Instead, a program called **pcre2_dftables** is compiled and run. This outputs the source for new set of tables, created in the default locale of your C run-time system. This method of replacing the tables does not work if you are cross compiling, because **pcre2_dftables** needs to be run on the local host and therefore not compiled with the cross compiler.

If you need to create alternative tables when cross compiling, you will have to do so "by hand". There may also be other reasons for creating tables manually. To cause **pcre2_dftables** to be built on the local host, run a normal compiling command, and then run the program with the output file as its argument, for example:

```
cc src/pcre2_dftables.c -o pcre2_dftables
./pcre2_dftables src/pcre2_chartables.c
```

This builds the tables in the default locale of the local host. If you want to specify a locale, you must

use the -L option:

```
LC_ALL=fr_FR ./pcre2_dftables -L src/pcre2_chartables.c
```

You can also specify -b (with or without -L). This causes the tables to be written in binary instead of as source code. A set of binary tables can be loaded into memory by an application and passed to **pcre2_compile()** in the same way as tables created by calling **pcre2_maketables()**. The tables are just a string of bytes, independent of hardware characteristics such as endianness. This means they can be bundled with an application that runs in different environments, to ensure consistent behaviour.

USING EBCDIC CODE

PCRE2 assumes by default that it will run in an environment where the character code is ASCII or Unicode, which is a superset of ASCII. This is the case for most computer operating systems. PCRE2 can, however, be compiled to run in an 8-bit EBCDIC environment by adding

```
--enable-ebcdic --disable-unicode
```

to the **configure** command. This setting implies --enable-rebuild-chartables. You should only use it if you know that you are in an EBCDIC environment (for example, an IBM mainframe operating system).

It is not possible to support both EBCDIC and UTF-8 codes in the same version of the library. Consequently, --enable-unicode and --enable-ebcdic are mutually exclusive.

The EBCDIC character that corresponds to an ASCII LF is assumed to have the value 0x15 by default. However, in some EBCDIC environments, 0x25 is used. In such an environment you should use

```
--enable-ebcdic-nl25
```

as well as, or instead of, --enable-ebcdic. The EBCDIC character for CR has the same value as in ASCII, namely, 0x0d. Whichever of 0x15 and 0x25 is *not* chosen as LF is made to correspond to the Unicode NEL character (which, in Unicode, is 0x85).

The options that select newline behaviour, such as --enable-newline-is-cr, and equivalent run-time options, refer to these character values in an EBCDIC environment.

PCRE2GREP SUPPORT FOR EXTERNAL SCRIPTS

By default **pcre2grep** supports the use of callouts with string arguments within the patterns it is matching. There are two kinds: one that generates output using local code, and another that calls an external program or script. If --disable-pcre2grep-callout-fork is added to the **configure** command,

only the first kind of callout is supported; if --disable-pcre2grep-callout is used, all callouts are completely ignored. For more details of **pcre2grep** callouts, see the **pcre2grep** documentation.

PCRE2GREP OPTIONS FOR COMPRESSED FILE SUPPORT

By default, **pcre2grep** reads all files as plain text. You can build it so that it recognizes files whose names end in **.gz** or **.bz2**, and reads them with **libz** or **libbz2**, respectively, by adding one or both of

```
--enable-pcre2grep-libz
--enable-pcre2grep-libbz2
```

to the **configure** command. These options naturally require that the relevant libraries are installed on your system. Configuration will fail if they are not.

PCRE2GREP BUFFER SIZE

pcre2grep uses an internal buffer to hold a "window" on the file it is scanning, in order to be able to output "before" and "after" lines when it finds a match. The default starting size of the buffer is 20KiB. The buffer itself is three times this size, but because of the way it is used for holding "before" lines, the longest line that is guaranteed to be processable is the notional buffer size. If a longer line is encountered, **pcre2grep** automatically expands the buffer, up to a specified maximum size, whose default is 1MiB or the starting size, whichever is the larger. You can change the default parameter values by adding, for example,

```
--with-pcre2grep-bufsize=51200
--with-pcre2grep-max-bufsize=2097152
```

to the **configure** command. The caller of **pcre2grep** can override these values by using --buffer-size and --max-buffer-size on the command line.

PCRE2TEST OPTION FOR LIBREADLINE SUPPORT

If you add one of

```
--enable-pcre2test-libreadline
--enable-pcre2test-libedit
```

to the **configure** command, **pcre2test** is linked with the **libreadline** or**libedit** library, respectively, and when its input is from a terminal, it reads it using the **readline()** function. This provides line-editing and history facilities. Note that **libreadline** is GPL-licensed, so if you distribute a binary of **pcre2test** linked in this way, there may be licensing issues. These can be avoided by linking instead with **libedit**, which has a BSD licence.

Setting --enable-pcre2test-libreadline causes the **-lreadline** option to be added to the **pcre2test** build. In many operating environments with a sytem-installed readline library this is sufficient. However, in some environments (e.g. if an unmodified distribution version of readline is in use), some extra configuration may be necessary. The INSTALL file for **libreadline** says this:

```
"Readline uses the termcap functions, but does not link with
the termcap or curses library itself, allowing applications
which link with readline the to choose an appropriate library."
```

If your environment has not been set up so that an appropriate library is automatically included, you may need to add something like

```
LIBS="-ncurses"
```

immediately before the **configure** command.

INCLUDING DEBUGGING CODE

If you add

```
--enable-debug
```

to the **configure** command, additional debugging code is included in the build. This feature is intended for use by the PCRE2 maintainers.

DEBUGGING WITH VALGRIND SUPPORT

If you add

```
--enable-valgrind
```

to the **configure** command, PCRE2 will use valgrind annotations to mark certain memory regions as unaddressable. This allows it to detect invalid memory accesses, and is mostly useful for debugging PCRE2 itself.

CODE COVERAGE REPORTING

If your C compiler is gcc, you can build a version of PCRE2 that can generate a code coverage report for its test suite. To enable this, you must install **lcov** version 1.6 or above. Then specify

```
--enable-coverage
```

to the **configure** command and build PCRE2 in the usual way.

Note that using **ccache** (a caching C compiler) is incompatible with code coverage reporting. If you have configured **ccache** to run automatically on your system, you must set the environment variable

```
CCACHE_DISABLE=1
```

before running **make** to build PCRE2, so that **ccache** is not used.

When --enable-coverage is used, the following addition targets are added to the *Makefile*:

```
make coverage
```

This creates a fresh coverage report for the PCRE2 test suite. It is equivalent to running "make coverage-reset", "make coverage-baseline", "make check", and then "make coverage-report".

```
make coverage-reset
```

This zeroes the coverage counters, but does nothing else.

```
make coverage-baseline
```

This captures baseline coverage information.

```
make coverage-report
```

This creates the coverage report.

```
make coverage-clean-report
```

This removes the generated coverage report without cleaning the coverage data itself.

```
make coverage-clean-data
```

This removes the captured coverage data without removing the coverage files created at compile time (*.gcno).

```
make coverage-clean
```

This cleans all coverage data including the generated coverage report. For more information about code coverage, see the **gcov** and **lcov** documentation.

DISABLING THE Z AND T FORMATTING MODIFIERS

The C99 standard defines formatting modifiers z and t for size_t and ptrdiff_t values, respectively. By default, PCRE2 uses these modifiers in environments other than old versions of Microsoft Visual Studio when __STDC_VERSION__ is defined and has a value greater than or equal to 199901L (indicating support for C99). However, there is at least one environment that claims to be C99 but does not support these modifiers. If

--disable-percent-zt

is specified, no use is made of the z or t modifiers. Instead of %td or %zu, a suitable format is used depending in the size of long for the platform.

SUPPORT FOR FUZZERS

There is a special option for use by people who want to run fuzzing tests on PCRE2:

--enable-fuzz-support

At present this applies only to the 8-bit library. If set, it causes an extra library called libpcre2-fuzzsupport.a to be built, but not installed. This contains a single function called LLVMFuzzerTestOneInput() whose arguments are a pointer to a string and the length of the string. When called, this function tries to compile the string as a pattern, and if that succeeds, to match it. This is done both with no options and with some random options bits that are generated from the string.

Setting --enable-fuzz-support also causes a binary called **pcre2fuzzcheck** to be created. This is normally run under valgrind or used when PCRE2 is compiled with address sanitizing enabled. It calls the fuzzing function and outputs information about what it is doing. The input strings are specified by arguments: if an argument starts with "=" the rest of it is a literal input string. Otherwise, it is assumed to be a file name, and the contents of the file are the test string.

OBSOLETE OPTION

In versions of PCRE2 prior to 10.30, there were two ways of handling backtracking in the **pcre2_match()** function. The default was to use the system stack, but if

--disable-stack-for-recursion

was set, memory on the heap was used. From release 10.30 onwards this has changed (the stack is no longer used) and this option now does nothing except give a warning.

SEE ALSO

pcre2api(3), **pcre2-config**(3).

AUTHOR

Philip Hazel
Retired from University Computing Service
Cambridge, England.

REVISION

Last updated: 27 July 2022