PCRE (Perl Compatible Regular Expressions)

PCRE (Perl Compatible Regular Expressions) is a library of functions for matching text strings using regular expressions. The syntax used is that of Perl regular expressions, although the API and underlying algorithm are different. PCRE is intended to be cross platform, and is normally used as a linkable C library. For convenience, however, the RISC OS version is implemented as a relocatable module providing SWI calls to access the main features of PCRE. These are documented on the following pages, while this section provides an overview:

Strings in PCRE

A string in PCRE can include any character in the default character set. This is normally set at compile time, and for RISC OS is ISO 8859-1. PCRE can also understand Unicode in the UTF-8 encoding, and this can be enabled at runtime if required. Because a string can include nulls and other control characters which normally act as a terminator, any function which expects a string parameter, also expects a corresponding value for the length. This is normally the number of bytes, not characters, which may be different if UTF-8 support is enabled. You will need to calculate this value yourself. Some functions are able to work with null terminated, C-style strings. If this is the case you can set the length to -1.

By default, a line feed is understood as marking the end of a line, although this can be changed at runtime. This option affects the behaviour of the \N and $ elements in the pattern string.

A string passed into PCRE can be either the pattern string containing the regular expression, or the subject string which will be searched for sequences matching the pattern. A sequence of characters from the subject string which matches all or part of the pattern is known as a substring. PCRE provides a number of functions to extract substrings after a successful match.

Compiling a pattern

Before a pattern can be used it must be compiled into PCRE's internal representation. This is done using PCRE_Compile (SWI &88000). There are a number of options which may be passed to this function which affect the interpretation of the pattern string, please see the documentation of the SWI for details of these. If there are no errors and the pattern compiles successfully, a pointer to the compiled representation is returned. The memory for this is automatically claimed from the module area, and you should remember to free it after use using PCRE_FreeCode (SWI &88005). In the event that the pattern cannot be compiled, a null pointer is returned, and an error is raised. The following code fragment shows how a simple pattern can be compiled:

```
patt$="pattern string"
SYS "PCRE_Compile",patt$,LEN(patt$),0 TO code%
IF code%=0 THEN PRINT "Error compiling pattern"
```

Matching

Once the pattern string has been compiled, you can try matching it against the subject string. This is normally performed using the call PCRE_Match (SWI &88001), although there is an alternative function, PCRE_DFAMatch (SWI &88002) which works in a slightly different way. In addition to the compiled pattern, you must provide a pointer to a match data block which will be used to store

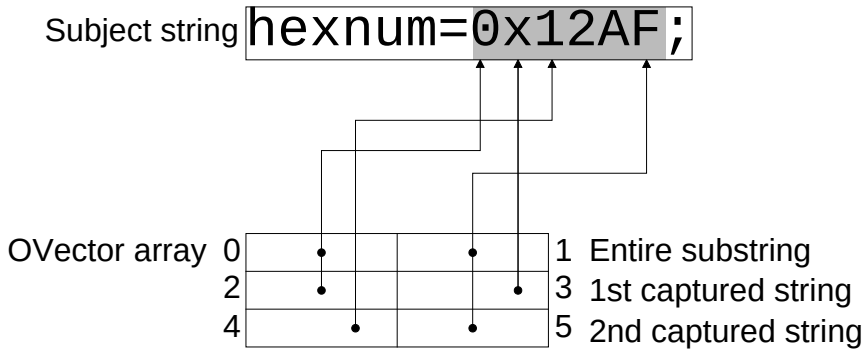
details about the substrings which match the pattern. Normally you will claim this data block yourself using PCRE_CreateData (SWI &88006). To automatically allocate a block of the correct size, either set R5=0 when calling PCRE_Match, or use PCRE_CreateData with R0=0 on entry.

Following a call to PCRE_Match, there are a number of possible return values. A positive value should be interpreted as meaning that the pattern matched successfully, and that the specified number of captured substrings are returned in the supplied match data block. A return value of zero means that the pattern matched successfully, but the match data block was not large enough to store all the captured substrings. If the pattern does not match -1 is returned, although this is not normally considered as an error. Formal errors with the matching process, such as a corrupted pattern, are indicated with a return value of -2 or less. The process of matching a compiled pattern against a subject string is illustrated by the following:

```
subj$=”subject string”
SYS “PCRE_DataCreate”,3 TO match_data%
SYS “PCRE_Match”,code%,subj$,LEN(subj$),0,0,match_data% TO rc%
IF rc%=-1 THEN
PRINT “No match!”
ELSE
IF rc%<-1 THEN PRINT “Error while matching”
ENDIF
```

Extracting substrings

Following a successful match, details of the substrings which match the pattern are stored in the match data block. The layout of this data structure is not normally exposed to application programs, and may change in the future. However, one component is an array of pairs of offsets. Each pair specifies the start and end of a substring. The first pair refers to the substring matched by the entire pattern, the second pair to the first captured substring and so on. This is shown diagrammatically below:



The normal way to get a matched substring is to use PCRE_CopySubstring (SWI &88003) which will copy the specified substring into another buffer for further processing. You can find out the size of the substring without copying it (useful for allocating memory) by calling PCRE_CopySubstring with R3=0. Alternatively PCRE_GetOVector (SWI &88004) will return the start and end offset of the substring. The following code fragment returns the substring matched by the entire pattern:

```
DIM str% 256
SYS “PCRE_SubstringCopy”,match_data%,subj$,0,str%,256 TO ,,,,len%
str%?len%=&0D:REM add CR to terminate string in BASIC
PRINT “Matched: “$str%
```

Freeing memory

Once you have finished using a pattern, then it is your responsibility to free any memory used. This includes both the memory occupied by the compiled pattern, and any memory explicitly requested for the match data block. The former can be de-allocated using the call PCRE_FreeCode (SWI &88005), and the latter using PCRE_FreeData (SWI &88007).

PCRE_Compile (SWI &88000)

Compiles a regular expression to PCRE's internal representation.

On entry

R0 = Pointer to regular expression string
R1 = Length of regular expression string in bytes, or -1 for a null terminated string
R2 = Options flags

On exit

R0 = Pointer to compiled regular expression

Use

This function compiles a regular expression into the internal representation suitable for use with PCRE_Match. It takes a pointer to the regular expression pattern, and returns a pointer to the internal representation. The pattern string can freely include nulls and other terminator characters, so it is also necessary to pass the length of this string.

The memory to store the compiled representation is automatically allocated from the module area, but the user should free this memory with PCRE_CodeFree when the pattern is no longer required.

The options typically control meaning of various elements in the regular expression are interpreted. The available options are as follows:

Bit	C Library name	Meaning if set
31	PCRE2_ANCHORED	Force pattern anchoring
1	PCRE2_ALT_BSUX	Alternative handling of \u, \U, and \x
2	PCRE2_AUTO_CALLOUT	Compile automatic callouts
3	PCRE2_CASELESS	Do caseless matching
4	PCRE2_DOLLAR_ENDONLY	$ not to match newline at end
5	PCRE2_DOTALL	. matches anything including NL
6	PCRE2_DUPNAMES	Allow duplicate names for subpatterns
7	PCRE2_EXTENDED	Ignore white space and # comments
8	PCRE2_FIRSTLINE	Force matching to be before newline
9	PCRE2_MATCH_UNSET_BACKREF	Match unset back references
10	PCRE2_MULTILINE	^ and $ match newlines within data
11	PCRE2_NEVER_UCP	Lock out PCRE2_UCP, e.g. via (*UCP)
12	PCRE2_NEVER_UTF	Lock out PCRE2_UTF, e.g. via (*UTF)
13	PCRE2_NO_AUTO_CAPTURE	Disable numbered capturing parentheses (named ones available)
14	PCRE2_NO_AUTO_POSSESS	Disable auto-possessification
15	PCRE2_NO_DOTSTAR_ANCHOR	Disable automatic anchoring for .*
16	PCRE2_NO_START_OPTIMIZE	Disable match-time start optimizations
30	PCRE2_NO_UTF_CHECK	Do not check the pattern for UTF validity (only relevant if PCRE2_UTF is set)
17	PCRE2_UCP	Use Unicode properties for \d, \w, etc.
18	PCRE2_UNGREEDY	Invert greediness of quantifiers

If compilation of the pattern fails, then this function sets R0 to null and returns an error.

PCRE_Match
(SWI &88001)

Matches a compiled expression against a supplied subject string.

On entry

R0 = Pointer to compiled regular expression
R1 = Pointer to subject string
R2 = Length of subject string in bytes, or -1 for a null terminated string
R3 = Offset in subject string at which to start matching
R4 = Options flags
R5 = Pointer to match data block or 0 to create one automatically

On exit

R0 = Number of captured substrings
R3 = Updated with the offset of the end of the matched string
R5 = Preserved, or set to point to the newly created match data

Use

This function matches a compiled regular expression (as returned by PCRE_Compile), against the specified subject string. As with the pattern string, the subject can include nulls, so the length of the string must be specified as well. Offsets and lengths of any captued substrings are stored in the match data block provided. The function returns the number of captured substrings in R0, with the substring matched by the entire pattern being counted as the first captured string. If the pattern did not match -1 is returned. To access the captured substrings, it is recommended to use PCRE_CopySubstring.

A return value of 0 means that the pattern matched, but there was insufficient space in the match data block to store all the captured substrings. In this case you will still be able to access the entire matched substring.

Normally matching starts at the begining of the subject string, but by specifying a non-zero offset in R3 it is possible to start in the middle, following a successful previous match, for example.

There are a number of options which affect the behaviour of matching, which are listed below:

Bit	**C Library name**	**Meaning if set**
31	PCRE2_ANCHORED	Match only at the first position
0	PCRE2_NOTBOL	Subject string is not the beginning of a line
1	PCRE2_NOTEOL	Subject string is not the end of a line
2	PCRE2_NOTEMPTY	An empty string is not a valid match
3	PCRE2_NOTEMPTY_ATSTART	An empty string at the start of the subject is not a valid match
30	PCRE2_NO_UTF_CHECK	Do not check the subject for UTF validity (only relevant if PCRE2_UTF was set at compile time)
4	PCRE2_PARTIAL_SOFT	Return PCRE2_ERROR_PARTIAL for a partial match if no full matches are found
5	PCRE2_PARTIAL_HARD	Return PCRE2_ERROR_PARTIAL for a partial match if that is found before a full match

PCRE_DFAMatch (SWI &88002)

Matches a compiled expression against a supplied subject string using the DFA algorithm.

On entry

R0 = Pointer to compiled regular expression
R1 = Pointer to subject string
R2 = Length of subject string in bytes
R3 = Offset in subject string at which to start matching
R4 = Options
R5 = Pointer to match data block

On exit

R0 = Number of captured substrings
R3 = Updated with the offset of the end of the (first) matched string

Use

This function is similar to PCRE_Match, but uses a Discrete Finite Automata (DFA) algorithm to perform the matching. For those patterns where there is only one possible match, the result is identical to PCRE_Match, but when there is more than one possible match from a given starting position, this function returns all of them.

There are some further differences which mean that this function is not strictly ‘Perl compatible’. In summary these are:

- Greedy and lazy quantifiers work in the same way, as all possible matches are returned anyway.
- There is no easy way to keep track of captured substrings, and therefore these are not available. For the same reason back references are not supported.
- The \eK escape sequence is not supported and causes an error if it is encountered.
- Even in UTF-8 mode, the \eC escape sequence is not supported as matching always proceeds one character at a time.
- Except for (*FAIL), backtracking control verbs are not supported.

For full details of the differences between the two algorithms please refer to the original PCRE documentation.

PCRE_CopySubstring (SWI &88003)

Extracts a substring following a successful match

On entry

R0 = Pointer to match data block
R1 = Pointer to original subject string supplied to PCRE_Match
R2 = Number of substring to extract
R3 = Pointer to buffer to receive substring, or 0 to return required length
R4 = Length of buffer

On exit

R0 = Preserved, or error code in case of failure
R4 = Updated with length of extracted substring

Use

This function extracts a captured substring following a successful call to PCRE_Match, and copies it into the supplied buffer. As well as the match data block, you must also provide the original subject string, and the number of the substring to extract. Substring number 0 refers to the entire substring matched by the expression, number 1 the first captured string and so on.

If R3 is zero on entry then this function calculates the length of the specified substring and returns it in R4. In this case it does not copy the substring.

If the call fails, an error is returned and R0 is updated with a negative number indicating an error code.

PCRE_GetOVector (SWI &88004)

Extract the location and length of substrings following a successful match

On entry

R0 = Pointer to match data block
R1 = Number of substring

On exit

R0 = Start offset of captured substring in bytes
R1 = Length of captured substring in bytes

Use

Given a match data block (returned from PCRE_Match), this call extracts the start and end offsets of a single captured substring from the table of entries. If you want to extract the actual captured string then use PCRE_CopySubstring. Substring number 0 refers to the entire substring matched by the expression, number 1 the first captured string and so on.

If the call fails, usually because an invalid substring number is supplied, then an error is returned and R1 is set to -1.

PCRE_FreeCode (SWI &88005)

Frees the memory used by a compiled regular expression.

On entry

R0 = Pointer to compiled regular expression

On exit

All registers preserved

Use

This call frees the memory used by a compiled regular expression once it is no longer needed. Please note that although the memory was allocated automatically, it remains the user’s responsibility to free it using this call.

PCRE_CreateData (SWI &88006)

Prepares a block of memory to hold data following a successful match

On entry

R0 = Required size as number of captured substrings, or 0 to calculate this automatically
R1 = Pointer to compiled regular expression if R0=0 on entry.

On exit

R0 = Pointer to newly created match data block

Use

This call allocates memory and prepares it for use as a match data block to be used by PCRE_Match and PCRE_DFAMatch. The argument in R0 specifies the number of substrings which must be stored. At very least there must be space to store details about the substring matched my the entire pattern, and if R0 is equal to 0 it is instead treated as 1. By default the memory is allocated from the module area and a pointer returned in R0. If the call fails then a null pointer is returned and an error raised.

If R0 is set to zero on entry, then the call additionally takes a pointer to a compiled regular expression and allocates a block of the neccessary size. This is calculated by scanning the regular expression and counting the number of capturing parentheses.

PCRE_FreeData (SWI &88007)

Frees the memory used by a match data block

On entry

R0 = Pointer to match data block

On exit

All registers preserved

Use

This call frees a block of memory which had previously been obtained using PCRE_CreateData. On entry R0 points to the start of the block of memory and the SWI returns with all registers preserved. It is currently the user’s responsibility to free all memory used in this way before their program finishes.

*PCRE_Config

Prints information about the features supported by this version of the PCRE library.

Syntax

*PCRE_Config

Parameters

None

Use

This command prints information about which options were enabled when this version of the PCRE library was compiled. Some of these options can be overridden at run time, but this command only shows the defaults.

Example

```
*PCRE_Config

JIT support: No
Unicode support: Yes
Internal link size: 2
Default newline sequence: 2
Internal resource limit: 1000000
Parentheses nesting limit: 250
Maximum recursion depth: 1000000
Use stack for recursion: Yes
```