

MBS Real Studio RegEx Plugin Documentation

Christian Schmitz

May 15, 2012

0.1 Introduction

This is the PDF version of the documentation for the Real Studio Plug-in from Monkeybread Software Germany. Plugin part: MBS Real Studio RegEx Plugin

0.2 Content

- 1 List of all topics 3
- 2 All items in this plugin 7
- 3 List of all classes 41

Chapter 1

List of Topics

• 2 Regular Expressions	7
– 2.1 class RegExMBS	7
* 2.1.1 Compile(pattern as string) as boolean	8
* 2.1.1 CompileMemory(pattern as memoryblock, ByteOffset as integer) as boolean	9
* 2.1.1 ConfigBSR as boolean	9
* 2.1.1 ConfigLinkSize as integer	9
* 2.1.1 ConfigMallocThreshold as integer	9
* 2.1.1 ConfigMatchLimit as integer	9
* 2.1.1 ConfigMatchLimitRecursion as integer	10
* 2.1.1 ConfigNewLine as integer	10
* 2.1.1 ConfigStackRecurse as boolean	10
* 2.1.1 ConfigUnicodeProperties as boolean	10
* 2.1.1 ConfigUTF8 as boolean	10
* 2.1.1 Escape(text as string) as string	11
* 2.1.1 Execute(text as string, start as integer) as integer	11
* 2.1.1 ExecuteMemory(text as memoryblock, ByteOffset as integer, ByteLength as integer) as integer	12
* 2.1.1 Offset(index as integer) as integer	12
* 2.1.1 OffsetCharacters(index as integer) as integer	13
* 2.1.1 Replace(NewText as string) as string	14
* 2.1.1 ReplaceAll(Target as string, NewText as string = "") as string	15
* 2.1.1 ReplaceSelection(NewText as string) as string	15
* 2.1.1 StringNumber(name as string) as integer	15
* 2.1.1 Study as boolean	16
* 2.1.1 Substring(index as integer) as string	16

* 2.1.1 Substring(name as string) as string	16
* 2.1.1 Version as string	16
* 2.1.2 CompileOptionAnchored as Boolean	17
* 2.1.2 CompileOptionAutoCallOut as Boolean	17
* 2.1.2 CompileOptionBSRAnyCRLF as Boolean	17
* 2.1.2 CompileOptionBSRUnicode as Boolean	17
* 2.1.2 CompileOptionCaseLess as Boolean	18
* 2.1.2 CompileOptionDollarEndOnly as Boolean	18
* 2.1.2 CompileOptionDotAll as Boolean	19
* 2.1.2 CompileOptionDuplicateNames as Boolean	19
* 2.1.2 CompileOptionExtended as Boolean	20
* 2.1.2 CompileOptionFirstLine as Boolean	20
* 2.1.2 CompileOptionJavaScriptCompat as Boolean	20
* 2.1.2 CompileOptionMultiline as Boolean	20
* 2.1.2 CompileOptionNewLineAny as Boolean	21
* 2.1.2 CompileOptionNewLineAnyCRLF as Boolean	22
* 2.1.2 CompileOptionNewLineCR as Boolean	22
* 2.1.2 CompileOptionNewLineCRLF as Boolean	22
* 2.1.2 CompileOptionNewLineLF as Boolean	23
* 2.1.2 CompileOptionNoAutoCapture as Boolean	23
* 2.1.2 CompileOptionNoStartOptimize as Boolean	23
* 2.1.2 CompileOptionNoUTF8Check as Boolean	23
* 2.1.2 CompileOptions as Integer	24
* 2.1.2 CompileOptionUngreedy as Boolean	24
* 2.1.2 CompileOptionUnicodeCodePoints as Boolean	24
* 2.1.2 CompileOptionUTF8 as Boolean	24
* 2.1.2 Count as Integer	25
* 2.1.2 ErrorMessage as String	26
* 2.1.2 ErrorOffset as integer	26
* 2.1.2 ExecuteOptionAnchored as Boolean	26
* 2.1.2 ExecuteOptionBSRAnyCRLF as Boolean	26
* 2.1.2 ExecuteOptionBSRUnicode as Boolean	27
* 2.1.2 ExecuteOptionNewLineAny as Boolean	27
* 2.1.2 ExecuteOptionNewLineAnyCRLF as Boolean	28
* 2.1.2 ExecuteOptionNewLineCR as Boolean	28
* 2.1.2 ExecuteOptionNewLineCRLF as Boolean	28
* 2.1.2 ExecuteOptionNewLineLF as Boolean	28
* 2.1.2 ExecuteOptionNoStartOptimize as Boolean	29
* 2.1.2 ExecuteOptionNotBOL as Boolean	30
* 2.1.2 ExecuteOptionNotEmpty as Boolean	30
* 2.1.2 ExecuteOptionNotEmptyAtStart as Boolean	30
* 2.1.2 ExecuteOptionNotEOL as Boolean	30

* 2.1.2 ExecuteOptionNoUTF8Check as Boolean	31
* 2.1.2 ExecuteOptionPartial as Boolean	31
* 2.1.2 ExecuteOptionPartialHard as Boolean	31
* 2.1.2 ExecuteOptions as Integer	32
* 2.1.2 ExtraHandle as Integer	32
* 2.1.2 Handle as Integer	32
* 2.1.2 Lasterror as Integer	32
* 2.1.2 Text as String	32
* 2.1.2 TextMemory as String	33
* 2.1.3 ErrorBadCount = -15	33
* 2.1.3 ErrorBadMagic = -4	33
* 2.1.3 ErrorBadNewLine = -23	33
* 2.1.3 ErrorBadOffset = -24	34
* 2.1.3 ErrorBadOption = -3	34
* 2.1.3 ErrorBadPartial = -13	34
* 2.1.3 ErrorBadUTF8 = -10	34
* 2.1.3 ErrorBadUTF8Offset = -11	34
* 2.1.3 ErrorCallOut = -9	35
* 2.1.3 ErrorDFAREcurese = -20	35
* 2.1.3 ErrorDFAUCond = -17	35
* 2.1.3 ErrorDFAUItem = -16	35
* 2.1.3 ErrorDFAUMLimit = -18	35
* 2.1.3 ErrorDFAWSSize = -19	36
* 2.1.3 ErrorInternal = -14	36
* 2.1.3 ErrorMatchLimit = -8	36
* 2.1.3 ErrorNoMatch = -1	36
* 2.1.3 ErrorNoSubstring = -7	36
* 2.1.3 ErrorNull = -2	37
* 2.1.3 ErrorNullWSLimit = -22	37
* 2.1.3 ErrorPartial = -12	37
* 2.1.3 ErrorRecursionLimit = -21	37
* 2.1.3 ErrorShortUTF8 = -25	37
* 2.1.3 ErrorUnknownNode = -6	38
* 2.1.3 ErrorUnknownOpcode = -5	38

Chapter 2

Regular Expressions

2.1 class RegExMBS

```
class RegExMBS
```

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** A class for fast Regular Expression Search in a perl compatible way.

Example:

```
dim r as new RegExMBS
dim searchString as string = ".o"

if r.Compile(searchString) then

dim s as string="Hello World"

dim start as integer = 0
while r.Execute(s,start)>0

dim p as integer = r.OffsetCharacters(0)
dim l as integer = r.OffsetCharacters(1)-r.OffsetCharacters(0)

MsgBox "Found "+searchString+" on position "+str(p)+" with length "+str(l)+" in """+s+"""

start = r.OffsetCharacters(1)
wend

else
```

```
MsgBox "failed to compile"
end if
```

Notes: uses the PCRE library. You may check the PCRE documentation.

2.1.1 Methods

Compile(pattern as string) as boolean

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Compiles a pattern.
Example:

```
dim r as new RegExMBS
dim searchString as string = ".o"

if r.Compile(searchString) then
msgbox "OK"
else
MsgBox "failed to compile"
end if
```

Notes:

Some predefined patterns like `\b` do not support Unicode well, so you may work around that by using your own pattern.

Returns true on success and false on failure.
 ErrorMessage, Lasterror, ErrorOffset and Handle are set.

The following table lists the error codes than may be returned by `Compile()`, along with the error messages that may be returned by both compiling functions.

CompileMemory(pattern as memoryblock, ByteOffset as integer) as boolean

Plugin Version: 6.3 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Compiles a pattern.
Notes:

Same as Compile, but the text is stored in a memoryblock and must be a 0 terminated C string.
Be careful to use valid UTF8 input and provide offset in byte units and not in characters.

ConfigBSR as boolean

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Returns an integer whose value indicates what character sequences the \R escape sequence matches by default.
Notes: A value of 0 means that \R matches any Unicode line ending sequence; a value of 1 means that \R matches only CR, LF, or CRLF. The default can be overridden when a pattern is compiled or matched.

ConfigLinkSize as integer

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Returns an integer that contains the number of bytes used for internal linkage in compiled regular expressions.
Notes: The value is 2, 3, or 4. Larger values allow larger regular expressions to be compiled, at the expense of slower matching. The default value of 2 is sufficient for all but the most massive patterns, since it allows the compiled pattern to be up to 64K in size.

ConfigMallocThreshold as integer

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** The output is an integer that contains the threshold above which the POSIX interface uses malloc() for output vectors.

ConfigMatchLimit as integer

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Returns an integer that gives the default limit for the number of internal matching function calls in a Execute execution.

ConfigMatchLimitRecursion as integer

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Returns an integer that gives the default limit for the depth of recursion when calling the internal matching function in a `Execute()` execution.

ConfigNewLine as integer

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** What newline character is used as default.

Notes: The output is an integer whose value specifies the default character sequence that is recognized as meaning "newline". The four values that are supported are: 10 for LF, 13 for CR, 3338 for CRLF, -2 for ANYCRLF, and -1 for ANY. Though they are derived from ASCII, the same values are returned in EBCDIC environments. The default should normally correspond to the standard sequence for your operating system.

ConfigStackRecurse as boolean

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Returns an integer that is set to one if internal recursion when running `Execute()` is implemented by recursive function calls that use the stack to remember their state.

Notes: This is the usual way that PCRE is compiled. The output is zero if PCRE was compiled to use blocks of data on the heap instead of recursive function calls. In this case, `malloc` and `free` are called to manage memory blocks on the heap, thus avoiding the use of the stack.

ConfigUnicodeProperties as boolean

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Returns true if unicode properties are available.

Notes: Should be true for the plugin.

ConfigUTF8 as boolean

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Whether UTF8 is supported.

Notes: If this ever is false, please complain. This plugin is designed to work only on UTF8 strings for best

performance.

Escape(text as string) as string

Plugin Version: 7.8 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Escapes the string.
Example:

```
dim r as RegExMBS
msgbox r.Escape("Hello [ ] ") // shows Hello \[ \]
```

Notes:

The string is converted to UTF8 and all the RegEx special characters are escaped.
Returns "" on low memory.

Execute(text as string, start as integer) as integer

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Performs a search with the compiled pattern.
Example:

```
dim r as RegExMbs
dim s as string
dim c as integer

s="123 ABC 456"

r=new RegExMBS
if r.Compile(" \D+ ") then
c=r.Execute(s,0)
MsgBox str(c)+" "+str(r.Offset(0))+" "+str(r.Offset(1))
// shows: 1 3 8
// 1 for ubound of the offset array
// 3 for 3 bytes before the matched pattern
// 8 for the 8 bytes before the end of the matched pattern
end if
```

Notes:

Returns the number of found offsets.

text must be in UTF-8 text encoding.

Start must be 0 for the first character and the byte offset for other characters. Do not pass values from OffsetCharacters here!

Return values from Execute:

If Execute() fails, it returns a negative number. The following are defined in the header file:

ExecuteMemory(text as memoryblock, ByteOffset as integer, ByteLength as integer) as integer

Plugin Version: 6.3 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Performs a search with the compiled pattern.

Notes:

Same as Execute, but the text is stored in a memoryblock.

Be careful to use valid UTF8 input and provide offset and length in byte units and not in characters.

Offset(index as integer) as integer

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Get the offset in the offset list with given index in bytes.

Example:

```

dim r as RegExMbs
dim s as string
dim c as integer

s="123 ABC 456"

r=new RegExMBS
if r.Compile("..") then
c=r.Execute(s,0)
MsgBox str(c)+" "+str(r.Offset(0))+" "+str(r.Offset(1))
// shows: 1 4 10
// 1 for ubound of the offset array
// 4 for 4 bytes before the matched pattern
// 10 for the 10 bytes before the end of the matched pattern
end if

```

```

r=new RegExMBS
if r.Compile(".\xF6.") then // finds using Unicode codepoint
c=r.Execute(s,0)
MsgBox str(c)+" "+str(r.Offset(0))+" "+str(r.Offset(1))
// shows: 1 4 10
// 1 for ubound of the offset array
// 4 for 4 bytes before the matched pattern
// 10 for the 10 bytes before the end of the matched pattern
end if

```

Notes:

If you found a pattern in a string you get here:

Invalid indexes return 0.
Count is the number of entries here.

OffsetCharacters(index as integer) as integer

Plugin Version: 6.3 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Get the offset in the offset list with given index in characters.

Example:

```

dim r as new RegExMBS
dim searchString as string = ".o"

if r.Compile(searchString) then

dim s as string=" Hello World"

if r.Execute(s,0)>0 then
dim lines(-1) as string

lines.Append str(R.Count)+" offset found."
lines.Append "In Bytes:"
lines.Append " Start of matched patern: "+str(R.Offset(0))
lines.Append " End of matched patern: "+str(R.Offset(1))
lines.Append " Length of matched patern: "+str(R.Offset(1)-r.Offset(0))

lines.Append "In Characters:"

```

```

lines.Append " Start of matched patern: "+str(R.OffsetCharacters(0))
lines.Append " End of matched patern: "+str(R.OffsetCharacters(1))
lines.Append " Length of matched patern: "+str(R.OffsetCharacters(1)-r.OffsetCharacters(0))

MsgBox Join(lines,EndOfLine)
end if

else
MsgBox "failed to compile"
end if

```

Notes:

This function is identical to `Offset()`, but returns characters instead of bytes.
 Works only with valid UTF-8 strings as input.
 Value is calculated on each function call based on `Offset(index)` and current text.

If you found a pattern in a string you get here:

Invalid indexes return 0.
 Count is the number of entries here.

Replace(NewText as string) as string

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Replaces the text on the current found position and returns the complete new text.

Notes:

You need to call `Execute` before.
`Lasterror` is set.
`NewText` must have UTF-8 text encoding.

`\0` references the whole found pattern, `\1` to `\15` the subexpressions.
`\t` is replaced with `chr(9)`, `\r` and `\n` with `chr(13)` and `\\` with `\`.

ReplaceAll(Target as string, NewText as string = "") as string

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Searches the target string for current pattern and replaces all occurrences with the new text.

Notes:

You need to call Compile before to initialize the pattern and you should call Study before to optimize the pattern.

Lasterror is set.

Target and NewText must have UTF-8 text encoding.

\0 references the whole found pattern, \1 to \15 the subexpressions.

\t is replaced with chr(9), \r and \n with chr(13) and \\ with \.

ReplaceSelection(NewText as string) as string

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Replaces the text on the current found position and returns new text for that selection.

Notes:

This method is for text editors where you will store result in editfield.seltext to replace the current selection.

Lasterror is set.

You need to call Execute before.

NewText must have UTF-8 text encoding.

\0 references the whole found pattern, \1 to \15 the subexpressions.

\t is replaced with chr(9), \r and \n with chr(13) and \\ with \.

StringNumber(name as string) as integer

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** This convenience function finds the number of a named substring capturing parenthesis in a compiled pattern.

Notes:

name: Name whose number is required

The yield of the function is the number of the parenthesis if the name is found, or PCRE_ERROR_NO-SUBSTRING otherwise.

Study as boolean

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** After you compiled a pattern study can optimize it.

Notes:

Only useful if you use Execute several times.

In that case you call one time Compile, one time Study and several times Execute.

Errormessage is set.

Substring(index as integer) as string

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Returns the subexpression found with the given index.

Notes:

Returns "" on any error.

Lasterror is set.

See also:

- 2.1.1 Substring(name as string) as string

16

Substring(name as string) as string

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Returns the subexpression found with the given name.

Notes:

Returns "" on any error.

Lasterror is set.

See also:

- 2.1.1 Substring(index as integer) as string

16

Version as string

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** The version of the PCRE library as an ASCII string.

2.1.2 Properties

CompileOptionAnchored as Boolean

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Option for Compile: Force pattern anchoring

Notes: (Read and Write property)

CompileOptionAutoCallOut as Boolean

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Option for Compile: Compile automatic callouts

Notes: (Read and Write property)

CompileOptionBSRAnyCRLF as Boolean

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** This option and CompileOptionBSRUnicode (which are mutually exclusive) control what the \R escape sequence matches.

Notes:

The choice is either to match only CR, LF, or CRLF, or to match any Unicode newline sequence. The default is specified when PCRE is built. It can be overridden from within the pattern, or by setting an option when a compiled pattern is matched.

(Read and Write property)

CompileOptionBSRUnicode as Boolean

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** This option and CompileOptionBSRAnyCRLF (which are mutually exclusive) control what the \R escape sequence matches.

Notes:

The choice is either to match only CR, LF, or CRLF, or to match any Unicode newline sequence. The default is specified when PCRE is built. It can be overridden from within the pattern, or by setting an option when

a compiled pattern is matched.
(Read and Write property)

CompileOptionCaseLess as Boolean

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Option for Compile:
Do caseless matching

Example:

```
dim r as new RegExMBS
dim searchString as string = "hello"

r.CompileOptionCaseLess=True

if r.Compile(searchString) then

dim s as string=" Hello World"

if r.Execute(s,0)>0 then
MsgBox "Found: " + mid(s, r.OffsetCharacters(0)+1, r.OffsetCharacters(1)-r.OffsetCharacters(0))
else
MsgBox "nothing found"
end if

else
MsgBox "failed to compile"
end if
```

Notes:

The current library version is compiled to match only ASCII characters caseless.
(Read and Write property)

CompileOptionDollarEndOnly as Boolean

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Option for Compile:
\$ not to match newline at end

Notes: (Read and Write property)

CompileOptionDotAll as Boolean

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Option for Compile:
 . matches anything including endofline

Example:

```

dim r as new RegExMBS
dim searchString as string = "<.+>"

r.CompileOptionDotAll = false // finds only if true

if r.Compile(searchString) then

dim s as string="<Step enable=""True"" +EndOfLine.unix+"id=""93"" name=""Beep"" />"

if r.Execute(s,0)>0 then
MsgBox "Found: " +mid(s, r.OffsetCharacters(0)+1, r.OffsetCharacters(1)-r.OffsetCharacters(0))
else
MsgBox "nothing found"
end if

else
MsgBox "failed to compile"
end if

```

Notes: (Read and Write property)

CompileOptionDuplicateNames as Boolean

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** If this bit is set, names used to identify capturing subpatterns need not be unique.

Notes:

This can be helpful for certain types of pattern when it is known that only one instance of the named subpattern can ever be matched. There are more details of named subpatterns below; see also the pcrepattern documentation.

(Read and Write property)

CompileOptionExtended as Boolean

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Option for Compile: Ignore whitespace and # comments
Notes: (Read and Write property)

CompileOptionFirstLine as Boolean

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Option for Compile: Force matching to be before newline
Notes: (Read and Write property)

CompileOptionJavaScriptCompat as Boolean

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** If this option is set, PCRE's behaviour is changed in some ways so that it is compatible with JavaScript rather than Perl.
Notes:

The changes are as follows:

(1) A lone closing square bracket in a pattern causes a compile-time error, because this is illegal in JavaScript (by default it is treated as a data character). Thus, the pattern `AB] CD` becomes illegal when this option is set.

(2) At run time, a back reference to an unset subpattern group matches an empty string (by default this causes the current matching alternative to fail). A pattern such as `(\1)(a)` succeeds when this option is set (assuming it can find an "a" in the subject), whereas it fails by default, for Perl compatibility.
(Read and Write property)

CompileOptionMultiline as Boolean

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Option for Compile: `^` and `$` match newlines within data
Example:

```

dim r as RegExMBS
dim n,i,c as integer
dim s as string

s=ReplaceLineEndings(EditField1.text,EndOfLine.UNIX)

r=new RegExMBS

'r.CompileOptionFirstLine=True
r.CompileOptionMultiline=True

if r.Compile("^...$ ") then
n=0
do
c=r.Execute(s,n)

if c>0 then
MsgBox r.Substring(0)
n=r.Offset(1)
end if

loop until c=0
end if

```

Notes: (Read and Write property)

CompileOptionNewLineAny as Boolean

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** This option override the default newline definition that was chosen when PCRE was built.

Notes:

Setting CompileOptionNewLineCR or CompileOptionNewLineLF specifies that a newline is indicated by a single character (CR or LF, respectively). Setting CompileOptionNewLineCRLF specifies that a newline is indicated by the two-character CRLF sequence. Setting CompileOptionNewLineAnyCRLF specifies that any of the three preceding sequences should be recognized. Setting CompileOptionNewLineAny specifies that any Unicode newline sequence should be recognized. The Unicode newline sequences are the three just mentioned, plus the single characters VT (vertical tab, U+000B), FF (formfeed, U+000C), NEL (next line, U+0085), LS (line separator, U+2028), and PS (paragraph separator, U+2029). The last two are recognized only in UTF-8 mode.

The newline setting in the options word uses three bits that are treated as a number, giving eight possibili-

ties. Currently only six are used (default plus the five values above). This means that if you set more than one newline option, the combination may or may not be sensible. For example, `CompileOptionNewLineCR` with `CompileOptionNewLineLF` is equivalent to `CompileOptionNewLineCRLF`, but other combinations may yield unused numbers and cause an error.

The only time that a line break in a pattern is specially recognized when compiling is when `PCRE_EXTENDED` is set. `CR` and `LF` are whitespace characters, and so are ignored in this mode. Also, an unescaped `#` outside a character class indicates a comment that lasts until after the next line break sequence. In other circumstances, line break sequences in patterns are treated as literal data.

(Read and Write property)

CompileOptionNewLineAnyCRLF as Boolean

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** This option override the default newline definition that was chosen when PCRE was built.

Notes:

See `CompileOptionNewLineAny` for more details.

(Read and Write property)

CompileOptionNewLineCR as Boolean

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** This option override the default newline definition that was chosen when PCRE was built.

Notes:

See `CompileOptionNewLineAny` for more details.

(Read and Write property)

CompileOptionNewLineCRLF as Boolean

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** This option override the default newline definition that was chosen when PCRE was built.

Notes:

See `CompileOptionNewLineAny` for more details.

(Read and Write property)

CompileOptionNewLineLF as Boolean

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** This option override the default newline definition that was chosen when PCRE was built.

Notes:

See CompileOptionNewLineAny for more details.
(Read and Write property)

CompileOptionNoAutoCapture as Boolean

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Option for Compile: Disable numbered capturing parentheses (named ones available)

Notes: (Read and Write property)

CompileOptionNoStartOptimize as Boolean

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** This is an option that acts at matching time; that is, it is really an option for Execute.

Notes:

If it is set at compile time, it is remembered with the compiled pattern and assumed at matching time.
(Read and Write property)

CompileOptionNoUTF8Check as Boolean

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Option for Compile: Do not check the pattern for UTF-8 validity.

Notes: (Read and Write property)

CompileOptions as Integer

Plugin Version: 6.3 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** The internal value of all the compile options.

Notes:

You can get and set the bits using the CompileOption* Boolean properties.
(Read only property)

CompileOptionUngreedy as Boolean

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Option for Compile: Invert greediness of quantifiers.

Notes:

Basically this is about whether to find the next matching item or the last matching item in the whole string. Matching the next item is always much faster.
(Read and Write property)

CompileOptionUnicodeCodePoints as Boolean

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Whether to support unicode code points for character classes.

Notes:

This option changes the way PCRE processes `\B`, `\b`, `\D`, `\d`, `\S`, `\s`, `\W`, `\w`, and some of the POSIX character classes. By default, only ASCII characters are recognized, but if `PCRE_UCP` is set, Unicode properties are used instead to classify characters. More details are given in the section on generic character types in the `prepattern` page. If you set `PCRE_UCP`, matching one of the items it affects takes much longer. The option is available only if PCRE has been compiled with Unicode property support.
(Read and Write property)

CompileOptionUTF8 as Boolean

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Option for Compile: Run in UTF-8 mode.

Notes: (Read and Write property)

Count as Integer

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Number of offsets found.

Example:

```
dim r as new RegExMBS
dim searchString as string = ".o"

if r.Compile(searchString) then

dim s as string=" Hello World"

if r.Execute(s,0)>0 then
dim lines(-1) as string

lines.Append str(R.Count)+" offset found."
lines.Append "In Bytes:"
lines.Append " Start of matched patern: "+str(R.Offset(0))
lines.Append " End of matched patern: "+str(R.Offset(1))
lines.Append " Length of matched patern: "+str(R.Offset(1)-r.Offset(0))

lines.Append "In Characters:"
lines.Append " Start of matched patern: "+str(R.OffsetCharacters(0))
lines.Append " End of matched patern: "+str(R.OffsetCharacters(1))
lines.Append " Length of matched patern: "+str(R.OffsetCharacters(1)-r.OffsetCharacters(0))

MsgBox Join(lines,EndOfLine)
end if

else
MsgBox "failed to compile"
end if
```

Notes: (Read only property)

ErrorMessage as String

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** The last message reported.

Notes:

Set by Study and Compile.
(Read only property)

ErrorOffset as integer

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** The last error offset from the compile function.

Notes: (Read only property)

ExecuteOptionAnchored as Boolean

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Option for Execute: Match only at the first position

Notes: (Read and Write property)

ExecuteOptionBSRAnyCRLF as Boolean

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** This option and ExecuteOptionBSRUnicode (which are mutually exclusive) control what the \R escape sequence matches.

Notes:

The choice is either to match only CR, LF, or CRLF, or to match any Unicode newline sequence. The default is specified when PCRE is built. It can be overridden from within the pattern, or by setting an option when a compiled pattern is matched.

(Read and Write property)

ExecuteOptionBSRUnicode as Boolean

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** This option and ExecuteOptionBSRAnyCRLF (which are mutually exclusive) control what the `\R` escape sequence matches.
Notes:

The choice is either to match only CR, LF, or CRLF, or to match any Unicode newline sequence. The default is specified when PCRE is built. It can be overridden from within the pattern, or by setting an option when a compiled pattern is matched.
(Read and Write property)

ExecuteOptionNewLineAny as Boolean

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** This option override the default newline definition that was chosen when PCRE was built.
Notes:

Setting ExecuteOptionNewLineCR or ExecuteOptionNewLineLF specifies that a newline is indicated by a single character (CR or LF, respectively). Setting ExecuteOptionNewLineCRLF specifies that a newline is indicated by the two-character CRLF sequence. Setting ExecuteOptionNewLineAnyCRLF specifies that any of the three preceding sequences should be recognized. Setting ExecuteOptionNewLineAny specifies that any Unicode newline sequence should be recognized. The Unicode newline sequences are the three just mentioned, plus the single characters VT (vertical tab, U+000B), FF (formfeed, U+000C), NEL (next line, U+0085), LS (line separator, U+2028), and PS (paragraph separator, U+2029). The last two are recognized only in UTF-8 mode.

The newline setting in the options word uses three bits that are treated as a number, giving eight possibilities. Currently only six are used (default plus the five values above). This means that if you set more than one newline option, the combination may or may not be sensible. For example, ExecuteOptionNewLineCR with ExecuteOptionNewLineLF is equivalent to ExecuteOptionNewLineCRLF, but other combinations may yield unused numbers and cause an error.

The only time that a line break in a pattern is specially recognized when compiling is when PCRE_EXTENDED is set. CR and LF are whitespace characters, and so are ignored in this mode. Also, an unescaped `#` outside a character class indicates a comment that lasts until after the next line break sequence. In other circumstances, line break sequences in patterns are treated as literal data.
(Read and Write property)

ExecuteOptionNewLineAnyCRLF as Boolean

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** This option override the default newline definition that was chosen when PCRE was built.

Notes:

See ExecuteOptionNewLineAny for more details.
(Read and Write property)

ExecuteOptionNewLineCR as Boolean

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** This option override the default newline definition that was chosen when PCRE was built.

Notes:

See ExecuteOptionNewLineAny for more details.
(Read and Write property)

ExecuteOptionNewLineCRLF as Boolean

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** This option override the default newline definition that was chosen when PCRE was built.

Notes:

See ExecuteOptionNewLineAny for more details.
(Read and Write property)

ExecuteOptionNewLineLF as Boolean

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** This option override the default newline definition that was chosen when PCRE was built.

Notes:

See ExecuteOptionNewLineAny for more details.
(Read and Write property)

ExecuteOptionNoStartOptimize as Boolean

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** There are a number of optimizations that Execute uses at the start of a match, in order to speed up the process.

Notes:

For example, if it is known that an unanchored match must start with a specific character, it searches the subject for that character, and fails immediately if it cannot find it, without actually running the main matching function. This means that a special item such as (*COMMIT) at the start of a pattern is not considered until after a suitable starting point for the match has been found. When callouts or (*MARK) items are in use, these "start-up" optimizations can cause them to be skipped if the pattern is never actually used. The start-up optimizations are in effect a pre-scan of the subject that takes place before the pattern is run.

The ExecuteOptionNoStartOptimize option disables the start-up optimizations, possibly causing performance to suffer, but ensuring that in cases where the result is "no match", the callouts do occur, and that items such as (*COMMIT) and (*MARK) are considered at every possible starting position in the subject string. If ExecuteOptionNoStartOptimize is set at compile time, it cannot be unset at matching time.

Setting ExecuteOptionNoStartOptimize can change the outcome of a matching operation. Consider the pattern

```
(*COMMIT)ABC
```

When this is compiled, PCRE records the fact that a match must start with the character "A". Suppose the subject string is "DEFABC". The start-up optimization scans along the subject, finds "A" and runs the first match attempt from there. The (*COMMIT) item means that the pattern must match the current starting position, which in this case, it does. However, if the same match is run with ExecuteOptionNoStartOptimize set, the initial scan along the subject string does not happen. The first match attempt is run starting from "D" and when this fails, (*COMMIT) prevents any further matches being tried, so the overall result is "no match". If the pattern is studied, more start-up optimizations may be used. For example, a minimum length for the subject may be recorded. Consider the pattern

```
(*MARK:A)(X—Y)
```

The minimum length for a match is one character. If the subject is "ABC", there will be attempts to match "ABC", "BC", "C", and then finally an empty string. If the pattern is studied, the final attempt does not take place, because PCRE knows that the subject is too short, and so the (*MARK) is never encountered. In this case, studying the pattern does not affect the overall match result, which is still "no match", but it does affect the auxiliary information that is returned.

(Read and Write property)

ExecuteOptionNotBOL as Boolean

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Option for Execute:
Subject is not the beginning of a line
Notes: (Read and Write property)

ExecuteOptionNotEmpty as Boolean

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Option for Execute:
An empty string is not a valid match
Notes: (Read and Write property)

ExecuteOptionNotEmptyAtStart as Boolean

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the options.
Notes:

This is like `ExecuteOptionNotEmpty`, except that an empty string match that is not at the start of the subject is permitted. If the pattern is anchored, such a match can occur only if the pattern contains `\K`. Perl has no direct equivalent of `ExecuteOptionNotEmpty` or `ExecuteOptionNotEmptyAtStart`, but it does make a special case of a pattern match of the empty string within its `split()` function, and when using the `/g` modifier. It is possible to emulate Perl's behaviour after matching a null string by first trying the match again at the same offset with `ExecuteOptionNotEmptyAtStart` and `ExecuteOptionAnchored`, and then if that fails, by advancing the starting offset (see below) and trying an ordinary match again. There is some code that demonstrates how to do this in the `pcdemo` sample program. In the most general case, you have to check to see if the newline convention recognizes CRLF as a newline, and if so, and the current character is CR followed by LF, advance the starting offset by two characters instead of one.

(Read and Write property)

ExecuteOptionNotEOL as Boolean

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Option for Execute:
Subject is not the end of a line
Notes: (Read and Write property)

ExecuteOptionNoUTF8Check as Boolean

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Option for Execute: Do not check the subject for UTF-8 validity

Notes: (Read and Write property)

ExecuteOptionPartial as Boolean

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** Option for Execute: Return PCRE_ERROR_PARTIAL for a partial match

Notes: (Read and Write property)

ExecuteOptionPartialHard as Boolean

Plugin Version: 11.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** This option (and ExecuteOptionPartial) turn on the partial matching feature.

Notes:

A partial match occurs if the end of the subject string is reached successfully, but there are not enough subject characters to complete the match. If this happens when ExecuteOptionPartial (but not ExecuteOptionPartialHard) is set, matching continues by testing any remaining alternatives. Only if no complete match can be found is ErrorPartial returned instead of PCRE_ERROR_NOMATCH. In other words, ExecuteOptionPartial says that the caller is prepared to handle a partial match, but only if no complete match can be found.

If ExecuteOptionPartialHard is set, it overrides ExecuteOptionPartial. In this case, if a partial match is found, Execute() immediately returns ErrorPartial, without considering any other alternatives. In other words, when ExecuteOptionPartialHard is set, a partial match is considered to be more important than an alternative complete match.

In both cases, the portion of the string that was inspected when the partial match was found is set as the first matching string. There is a more detailed discussion of partial and multi-segment matching, with examples, in the pcrepartial documentation.

(Read and Write property)

ExecuteOptions as Integer

Plugin Version: 6.3 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** The internal value of all the execute options.

Notes:

You can get and set the bits using the CompileOption* Boolean properties.
(Read only property)

ExtraHandle as Integer

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** The handle to the extra data structure.

Notes: (Read only property)

Handle as Integer

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** The handle to the pattern data structure.

Notes: (Read only property)

Lasterror as Integer

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** The last error code reported.

Notes:

0 is no error and -1 is some parameter error.

See Error* constants for other values.

(Read only property)

Text as String

Plugin Version: 6.2 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** The text used for the last successful compile call.

Notes: (Read only property)

TextMemory as String

Plugin Version: 6.3 Console & Web: Yes Mac: Yes, Win: Yes, Linux: Yes, . **Function:** The text used for the last successful compiletext call.

Notes: (Read only property)

2.1.3 Constants

ErrorBadCount = -15

Plugin Version: 11.2 Console & Web: No Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the RegEx error constants.

Notes: This error is given if the value of the ovecsize argument is negative.

ErrorBadMagic = -4

Plugin Version: 11.2 Console & Web: No Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the RegEx error constants.

Notes: PCRE stores a 4-byte "magic number" at the start of the compiled code, to catch the case when it is passed a junk pointer and to detect when a pattern that was compiled in an environment of one endianness is run in an environment with the other endianness. This is the error that PCRE gives when the magic number is not present.

ErrorBadNewLine = -23

Plugin Version: 11.2 Console & Web: No Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the RegEx error constants.

Notes: An invalid combination of Newline options was given.

ErrorBadOffset = -24

Plugin Version: 11.2 Console & Web: No Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the RegEx error constants.

Notes: The value of startoffset was negative or greater than the length of the subject, that is, the value in length.

ErrorBadOption = -3

Plugin Version: 11.2 Console & Web: No Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the RegEx error constants.

Notes: An unrecognized bit was set in the options argument.

ErrorBadPartial = -13

Plugin Version: 11.2 Console & Web: No Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the RegEx error constants.

Notes: This code is no longer in use. It was formerly returned when the PCRE_ PARTIAL option was used with a compiled pattern containing items that were not supported for partial matching. From release 8.00 onwards, there are no restrictions on partial matching.

ErrorBadUTF8 = -10

Plugin Version: 11.2 Console & Web: No Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the RegEx error constants.

Notes: A string that contains an invalid UTF-8 byte sequence was passed as a subject. However, if PCRE_ PARTIAL_ HARD is set and the problem is a truncated UTF-8 character at the end of the subject, ErrorShortUTF8 is used instead.

ErrorBadUTF8Offset = -11

Plugin Version: 11.2 Console & Web: No Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the RegEx error constants.

Notes: The UTF-8 byte sequence that was passed as a subject was valid, but the value of startoffset did

not point to the beginning of a UTF-8 character or the end of the subject.

ErrorCallOut = -9

Plugin Version: 11.2 Console & Web: No Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the RegEx error constants.

ErrorDFARecurse = -20

Plugin Version: 11.2 Console & Web: No Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the RegEx error constants.

ErrorDFAUCond = -17

Plugin Version: 11.2 Console & Web: No Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the RegEx error constants.

ErrorDFAUItem = -16

Plugin Version: 11.2 Console & Web: No Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the RegEx error constants.

ErrorDFAUMLimit = -18

Plugin Version: 11.2 Console & Web: No Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the RegEx error constants.

ErrorDFAWSSize = -19

Plugin Version: 11.2 Console & Web: No Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the RegEx error constants.

ErrorInternal = -14

Plugin Version: 11.2 Console & Web: No Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the RegEx error constants.

Notes: An unexpected internal error has occurred. This error could be caused by a bug in PCRE or by overwriting of the compiled pattern.

ErrorMatchLimit = -8

Plugin Version: 11.2 Console & Web: No Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the RegEx error constants.

Notes: The backtracking limit, as specified by the `match_limit` field in a `pcr_extra` structure (or defaulted) was reached.

ErrorNoMatch = -1

Plugin Version: 11.2 Console & Web: No Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the RegEx error constants.

Notes: The subject string did not match the pattern.

ErrorNoSubstring = -7

Plugin Version: 11.2 Console & Web: No Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the RegEx error constants.

Notes: This error is used by the substring functions.

ErrorNull = -2

Plugin Version: 11.2 Console & Web: No Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the RegEx error constants.

Notes: Either code or subject was passed as "", or ovector was "" and ovecsize was not zero.

ErrorNullWSLimit = -22

Plugin Version: 11.2 Console & Web: No Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the RegEx error constants.

ErrorPartial = -12

Plugin Version: 11.2 Console & Web: No Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the RegEx error constants.

Notes: The subject string did not match, but it did match partially.

ErrorRecursionLimit = -21

Plugin Version: 11.2 Console & Web: No Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the RegEx error constants.

Notes: The internal recursion limit, as specified by the match_limit_recursion field in a pcre_extra structure (or defaulted) was reached. See the description above.

ErrorShortUTF8 = -25

Plugin Version: 11.2 Console & Web: No Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the RegEx error constants.

Notes: The subject string ended with an incomplete (truncated) UTF-8 character, and the PCRE_PARTIAL_HARD option was set. Without this option, ErrorBadUTF8 is returned in this situation.

ErrorUnknownNode = -6

Plugin Version: 11.2 Console & Web: No Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the RegEx error constants.

ErrorUnknownOpcode = -5

Plugin Version: 11.2 Console & Web: No Mac: Yes, Win: Yes, Linux: Yes, . **Function:** One of the RegEx error constants.

Notes: While running the pattern match, an unknown item was encountered in the compiled pattern. This error could be caused by a bug in PCRE or by overwriting of the compiled pattern.

0 no error
1 \at end of pattern
2 \c at end of pattern
3 unrecognized character follows \
4 numbers out of order in { } quantifier
5 number too big in { } quantifier
6 missing terminating] for character class
7 invalid escape sequence in character class
8 range out of order in character class
9 nothing to repeat
10 operand of unlimited repeat could match the empty string
11 internal error: unexpected repeat
12 unrecognized character after (?
13 POSIX named classes are supported only within a class
14 missing)
15 reference to non-existent subpattern
16 erroffset passed as NULL
17 unknown option bit(s) set
18 missing) after comment
19 parentheses nested too deeply
20 regular expression too large
21 failed to get memory
22 unmatched parentheses
23 internal error: code overflow
24 unrecognized character after (?<
25 lookbehind assertion is not fixed length
26 malformed number after ?(
27 conditional group contains more than two branches
28 assertion expected after ?(
29 (?R or (?digits must be followed by)
30 unknown POSIX class name
31 POSIX collating elements are not supported
32 this version of PCRE is not compiled with PCRE_ UTF8 support
33 spare error
34 character value in \x { ... } sequence is too large
35 invalid condition ?(0)
36 \C not allowed in lookbehind assertion
37 PCRE does not support \L, \l, \N, \U, or \u
38 number after ?C is >255
39 closing) for ?C expected
40 recursive call could loop indefinitely
41 unrecognized character after ?P
42 syntax error after ?P
43 two named groups have the same name
44 invalid UTF-8 string
45 support for \P, \p, and \X has not been compiled

PCRE_ERROR_NOMATCH	-1	The subject string did not match the pattern.
PCRE_ERROR_NULL	-2	Either code or subject was passed as "".
PCRE_ERROR_BADOPTION	-3	An unrecognized bit was set in the options argument.
PCRE_ERROR_BADMAGIC	-4	PCRE stores a 4-byte "magic number" at the start of the compiled code, to catch the case when it is passed a junk pointer and to detect when a pattern that was compiled in an environment of one endianness is run in an environment with the other endianness. This is the error that PCRE gives when the magic number is not present.
PCRE_ERROR_UNKNOWN_NODE	-5	While running the pattern match, an unknown item was encountered in the compiled pattern. This error could be caused by a bug in PCRE or by overwriting of the compiled pattern.
PCRE_ERROR_NOMEMORY	-6	If a pattern contains back references, but the ovector that is passed to <code>Execute()</code> is not big enough to remember the referenced substrings, PCRE gets a block of memory at the start of matching to use for this purpose. If the call via <code>pcre_malloc()</code> fails, this error is given. The memory is automatically freed at the end of matching.
PCRE_ERROR_MATCHLIMIT	-8	The backtracking limit, as specified by the <code>match_limit</code> field in a <code>pcre_extra</code> structure (or defaulted) was reached.
PCRE_ERROR_RECURSIONLIMIT	-21	The internal recursion limit, as specified by the <code>match_limit_recursion</code> field in a <code>pcre_extra</code> structure (or defaulted) was reached.
PCRE_ERROR_CALLOUT	-9	This error is never generated by <code>Execute()</code> itself. It is provided for use by callout functions that want to yield a distinctive error code. See the <code>precallout</code> documentation for details.
PCRE_ERROR_BADUTF8	-10	A string that contains an invalid UTF-8 byte sequence was passed as a subject.
PCRE_ERROR_BADUTF8_OFFSET	-11	The UTF-8 byte sequence that was passed as a subject was valid, but the value of <code>startoffset</code> did not point to the beginning of a UTF-8 character.
PCRE_ERROR_PARTIAL	-12	The subject string did not match, but it did match partially. See the <code>prepartial</code> documentation for details of partial matching.
PCRE_ERROR_BADPARTIAL	-13	The <code>PCRE_PARTIAL</code> option was used with a compiled pattern containing items that are not supported for partial matching. See the <code>prepartial</code> documentation for details of partial matching.
PCRE_ERROR_INTERNAL	-14	An unexpected internal error has occurred. This error could be caused by a bug in PCRE or by overwriting of the compiled pattern.
PCRE_ERROR_BADCOUNT	-15	This error is given if the value of the <code>ovecsize</code> argument is negative.

index	offset
0	start of matched pattern
1	end of matched pattern
2	start of subexpression 1
3	end of subexpression 1
2*n	start of subexpression n
2*n+1	end of subexpression n

index	offset
0	start of matched pattern
1	end of matched pattern
2	start of subexpression 1
3	end of subexpression 1
2*n	start of subexpression n
2*n+1	end of subexpression n

Chapter 3

List of all classes

- RegExMBS

7