B++ Regular Expressions

Classes

`class`	**RegEx**
	B++ Regular Expression class. More...

Detailed Description

B++ provides a simple implementation of Regular Expressions, based on T-REX Library from Alberto Demichelis (see http://tiny-rex.sourceforge.net).
It supports essentially a subset of ECMAScript syntax (see Refernce at www.cplusplus.com).

Description below contains syntax elements supported by B++:

Syntax

Special Characters

`\`	Quotes next (meta)character
`^`	Matches begin of string
`.`	Matches any character
`\t`	Matches tab character
`\v`	Matches vertical tab character
`\n`	Matches new line character
`\r`	Matches carriage return character
`\f`	Matches form feed character
`|`	Separates two alternative patterns or subpatterns

Note:
Escape sequences for character codes, e.g. \x*hh*, \u*hhhh*, are not supported, but 'normal' B++ escape sequences may be used in an regualar expression.

Example: `"^\x20"` matches a space character at begin of text, where `"^\\s"` matches any whitespace character on start of text.

Assertions

`^`	Matches begin of target text

$	Matches end of target text
\b	Matches a word boundary
\B	Matches position inside a word, i. e. both, character before and behind must be word characters

Brackets

()	Grouping (creates a stored capture)
(?:)	Builds group (subexpression) too, but creates no stored capture
[]	Defines character class

Note:
Due to a bug in T-REX library stored capture groups are not correctly filled, if alternative patterns (separated by '|') are used. In that case groups of submatches are filled if first alternative matches, otherwise only main match is filled.

Predefined Character Classes

\l	lowercase characters
\u	uppercase characters
\a	letters
\A	non letters
\w	word characters, includes decimal digits, letters and underline character (_)
\W	non word characters
\s	whitespaces
\S	non whitespaces
\d	digits
\D	non digits
\x	hexadecimal digits
\X	non hexadecimal digits
\c	control characters
\C	non control characters
\p	punctation
\P	non punctation

Note:
T-REX library uses character classification functions from C-library (isalpha, isalnum ...) for character classes. Therefore members of character classes may be locale dependent.

Quantifiers

*	Matches 0 or more times
+	Matches 1 or more times
?	Matches 1 or 0 times
{n}	Matches exactly n times
{n,}	Matches at least n times
{n,m}	Matches at least n but not more than m times

B++ Reference © 2005-2026 by R.-Erik Ebert

All parts of B++ are provided under terms and conditions of MIT license.

RegEx Class Reference

B++ Regular Expressions

B++ Regular Expression class. More...

```
#include <bppregex.h>
```

Public Member Functions

	RegEx (const **string** pattern=null, const **string** flags=null)
	Ctor.
	~RegEx ()
	Destructor releases native **RegEx** object.
void	**setFlags** (const **string** flags)
	Sets flags for evaluation behaviour of expression.
string	**getFlags** () const
	Gets currently active flags for evaluation behaviour of expression.
bool	**compile** (const **string** pattern, const **string** flags=null)
	Compiles regular expression (pattern).
string	**getError** () const
	Gets error message.
bool	**match** (const **string** txt)
	Matches entire text.
bool	**search** (const **string** txt, **uint** start=0, **uint** len=NPOS)
	Searches for first match.

`bool next ()`
Finds next match.
`uint getMatches (const string txt, vector dst, uint start=0, uint len=NPOS)`
Fills vector with texts of all matches.
`uint getMatchPositions (const string txt, vector dst, uint start=0, uint len=NPOS)`
Fills vector with positions and sizes of all matches.
`uint getSubExpressions (vector dst)`
Gets descriptions for current match and all its submatches.
`uint getSubExpressions (string txt, vector dst, uint start=0, uint len=NPOS)`
Gets descriptions for all matches and their submatches.
`bool operator() (const string txt, uint start=0, uint len=NPOS)`
Replaces matches in a string producing a new string.
`bool operator() ()`
Searches for next match.
`var operator. (const string key) const`
Allows simplified access to some properties of **RegEx** object.
`var operator.= (const string key, var value)`
Allows simplified access to some properties of **RegEx** object.

Private Attributes

`string _pattern`
stored pattern source
`uint _flags`
stored flag mask

Detailed Description

B++ Regular Expression class.

Class provides support of Regular Expressions for B++. It is a wrapper around native class **BppRegEx**.

> **Note**
>
> Other than most predefined functions, arguments of public members of this class behave like strong typed arguments in B++ sources, i. e. they use implicit type conversion if needed (and possible).

Definition at line **100** of file **bppregex.h**.

Constructor & Destructor Documentation

◆ RegEx()

```
RegEx::RegEx ( const string pattern = null,
               const string flags = null )
```

Ctor.

Parameters

pattern Pattern to prepare. If specified and not empty, constructor calls **compile** method.

flags Flags specification for expression, it may contain letters 'm' (multi line), 'i' (ignore case) and 'g' (global search); all other characters are ignored.

See also

compile, **setFlags**

◆ ~RegEx()

```
RegEx::~RegEx ( )
```

Destructor releases native **RegEx** object.

Member Function Documentation

◆ compile()

```
bool RegEx::compile ( const string pattern,
                      const string flags = null )
```

Compiles regular expression (pattern).

Parameters

pattern Pattern to prepare. If specified and not empty, constructor calls **compile** method.

flags Flags specification for expression, it may contain letters 'm' (multi line), 'i' (ignore case) and 'g' (global search); all other characters are ignored. If not specified, flags remain unchanged.

Returns

true on success. If return value is *false*, caller get an error description by **getError()** method.

Note

flags are stored only. They don't affect compilation of pattern and may be changed later by setFlags method if needed.

See also

getError, **setFlags**

◆ getError()

```
string RegEx::getError ( ) const
```

Gets error message.

Returns

Error description of last error occurred while call to compile method.

See also

compile

◆ getFlags()

```
string RegEx::getFlags ( ) const
```

Gets currently active flags for evaluation behaviour of expression.

Returns

Flags specification for expression, result may contain letters 'm' (multi line), 'i' (ignore case) and 'g' (global search).

See also

setFlags, **compile**

◆ getMatches()

```
uint RegEx::getMatches ( const string txt,
                         vector       dst,
                         uint         start = 0,
                         uint         len = NPOS )
```

Fills vector with texts of all matches.

Function gets matching substrings in *txt*.

Parameters

txt Text to search for matches.

dst **Vector** to store results in. Funktion simply appends new string for each match to *dest*.

start Start position of search in *txt*.

len Maximum length of search, if *len+start* exceeds lemgth of *txt* search is done to end of *txt*.

Returns

Number of matching substrings appended to *dst*.

Note

Behavior of method depends on 'g' flag. If it is set, all matches are appended to *dst*, otherwise search is aborted after first match.

See also

getMatchPositions, **search**, **next**

References **uint()**, and **vector()**.

◆ getMatchPositions()

```
uint RegEx::getMatchPositions ( const string txt,
                                vector       dst,
                                uint         start = 0,
                                uint         len = NPOS )
```

Fills vector with positions and sizes of all matches.

Function gets positions and length of matching substrings in *txt*.

Parameters

txt Text to search for matches.

dst **Vector** to store results in. For each match function appends a pair (vector of two elements) to *dest*, where first element represents position of match in *txt* and second its length.

start Start position of search in *txt*.

len Maximum length of search, if *len+start* exceeds lemgth of *txt* search is done to end of *txt*.

Returns

Number of matching substrings appended to *dst*.

> **Note**
>
> Behavior of method depends on 'g' flag. If it is set, all matches are appended to *dst*, otherwise search is aborted after first match.

See also

getMatches, **search**, **next**

References **uint()**, and **vector()**.

◆ getSubExpressions() [1/2]

```
uint RegEx::getSubExpressions ( string txt,
                                vector dst,
                                uint   start = 0,
                                uint   len = NPOS )
```

Gets descriptions for all matches and their submatches.

Function combines **search**, **next** and **getSubExpressions** functions. It iterates through specified text collecting all matches and their submatches.

Parameters

txt Text to analyze.

dst **Vector** to store results into. For each match a new vector is appended to *dst*, each containing a vector of submatches.

start Start position of search in *txt*.

len Maximum length of search, if *len+start* exceeds lemgth of *txt* search is done to end of *txt*.

Returns

Number of items (matches found) appended to *dst*.

See also

getSubExpressions(vector)

References **uint()**, and **vector()**.

◆ getSubExpressions() [2/2]

```
uint RegEx::getSubExpressions ( vector dst )
```

Gets descriptions for current match and all its submatches.

Function gets submatch descriptors of last match.

Parameters

dst **Vector** to store results in. For each submatch a dictionary of two elements is appended to *dst*, where item 'pos' (**BppInt**) contains position and item 'text' (**BppString**) the matching text. First element appended is the main match itself.

Returns

Number of elements appended to *dst*.

Note

Method is intended for use with **search**, **next** and **match** functions.

Example

```
// Find numbers behind a letter
RegEx rex;
if (rex->compile("\\a(\\d+)")) {
   for(bool ok = rex->search("abc123x4y5z");ok;ok=rex->next())
   {
      vector v = [];
      uint cnt = rex->getSubExpressions(v);
      for (uint i=0;i<cnt;++i) {
       dictionary d = v[i];
        print(i, " pos: ", d.pos, " text: ", d.text, "\n");
      }
   }
}
else print(getError(),"\n");
```

See also

search, **next**, **match**

References **uint()**, and **vector()**.

Referenced by **BppXML::parse()**.

◆ match()

```
bool RegEx::match ( const string txt )
```

Matches entire text.

Function checks if expression matches entire text.

Returns

true on match. On match a call to **getSubExpressions(vector)** can be done to get any submatches.

Example

```
// checks if adr is a valid mail address
bool checkMailAdress(string adr)
{
   RegEx rex("\w+[\\w\\.]*\w+@\\w+\\.\\a+");
   return rex->match(adr);
}
```

◆ next()

```
bool RegEx::next ( )
```

Finds next match.

Function is intended for use together with **search**. It finds next match.

Returns

true if another match was found.

See also

search

◆ operator()() [1/2]

```
bool RegEx::operator() ( )
```

Searches for next match.

Operator is simply an alias of **next** method, intended for simplified iteration through matches.

Returns

true if another match was found.

See also

operator()(string,uint,int)

operator()() [2/2]

```
bool RegEx::operator() ( const string txt,
                         uint        start = 0,
                         uint        len = NPOS )
```

Replaces matches in a string producing a new string.

Function builds new string from *txt* replacing first match or all matches in *txt* by *repl*.

Parameters

txt Text to use for replace matches.

repl Replacement expression. The expression is a string, optionally containing following placeholders/variables:

- **$&** or **$0** - Inserts entire matching text.
- **$`** (back tick) - Inserts all text before match.
- **$'** (apostrophe) - Inserts all text behind match.
- **$*n*** - Inserts text of *n*-th submatch, where *n* is a positive decimal number. If *n* exceeds number of submatches available, string representation of *n* is inserted.
- **$$** - Inserts Dollar character ($).

Returns

Result of replace operation

Note

Behavior depends on 'g' flag. If set, all matches are replaced, otherwise only first match.

See also

strReplace */ string replace(string txt, string repl);

/** Replaces matches in a string modifying the source string. Function replaces first match or all matches in *txt* by *repl*.

Parameters

txt Text to replace matches in.

repl Replacement expression. The expression is a string, optionally containing following placeholders/variables:

- **$&** or **$0** - Inserts entire matching text.
- **$`** (back tick) - Inserts all text before match.
- **$'** (apostrophe) - Inserts all text behind match.
- **$*n*** - Inserts text of *n*-th submatch, where *n* is a positive decimal number. If *n* exceeds number of submatches available, string representation of *n* is inserted.
- **$$** - Inserts Dollar character ($).

Returns

Number of matches replaced

Note

Function modifies *txt* argument. Use #replace to create new string instead.

Behavior depends on 'g' flag. If set, all matches are replaced, otherwise only first match.

See also

replace */ uint strReplace(string txt, string repl);

/** Searches for first match. Operator is simply an alias of **search** method, intended for simplified iteration through matches.

Parameters

txt Text to search in.

start Start position for search in *txt*.

len Length of substring in *text* used for search. If *len* is larger than length of *txt* searching is done until end of *txt*.

Returns

true if a match was found. If operator returns `true`, **getSubExpressions(vector)** can be called do get positions and values of match and its submatches (if any). Operator may be called without arguments to get next match.

Note

Operator searches for first match only, regardless of 'g' flag.

Example

```
// Count number of digits in string using operator()
RegEx rex;
if (rex->compile("\\d")) {
   int cnt = 0;
   for(bool ok = rex("abc123x4y5z"); ok; ok=rex()) ++cnt;
   print("Number of matches: ",cnt, "\n");
}
else print(getError(),"\n");
```

See also

next, **getSubExpressions**

References **uint()**.

◆ operator.()

```
var RegEx::operator. ( const string key ) const
```

Allows simplified access to some properties of **RegEx** object.

Operator provides read access to following properties:

Key	Result type	Description
flags	string	Returns flags currently set as string, may contain 'g', 'm' and 'i' character.
global	bool	Returns `true` if 'g' flag is set.
ignoreCase	bool	Returns `true` if 'i' flag is set.
multiline	bool	Returns `true` if 'm' flag is set.
source	string	Returns copy of pattern string used for expression.
text	string	Returns copy of text string used for last search operation.
pos	uint	Returns position for start of next search.
len	uint	Returns search length.

Parameters

key Key of property to get.

Returns

Result value dependent on *key*.

Exceptions

Operator throws exception of `string` type on invalid *key*.

References **var**.

◆ operator.=()

```
var RegEx::operator.= ( const string key,
                        var          value )
```

Allows simplified access to some properties of **RegEx** object.

Operator provides write access to following properties:

Key	Value type	Description
flags	string	Sets all flags of **RegEx** object (does same as **setFlags**).
global	bool	Sets/Resets 'g' flag.
ignoreCase	bool	Sets/Resets 'i' flag.
multiline	bool	Sets/Resets 'm' flag.
text	string	Sets text string used for next search operation.
pos	uint	Sets position for start of next search.
len	uint	Sets search length.

Parameters

key Key of property to set.

value New value of property specified by *key*.

Returns

value

Exceptions

Operator throws exception of `string` type on invalid *key* or invalid type of *value*.

References **var**.

◆ search()

```
bool RegEx::search ( const string txt,
                     uint         start = 0,
                     uint         len = NPOS )
```

Searches for first match.

Function searches for first match in specified text.

Parameters

txt Text to search in.

start Start position for search in *txt*.

len Length of substring in *text* used for search. If *len* is larger than length of *txt* searching is done until end of *txt*.

Returns

true if a match was found. If function returns `true`, **getSubExpressions(vector)** can be called do get positions and values of match and its submatches (if any). Function **next()** may be called to get next match.

Note

This function searches for first match only, regardless of 'g' flag.

Example

```
// Count number of digits in string
RegEx rex;
if (rex->compile("\\d")) {
   int cnt = 0;
   for(bool ok = rex->search("abc123x4y5z"); ok; ok=rex->next()) ++cnt;
   print("Number of matches: ",cnt, "\n");
}
else print(getError(),"\n");
```

See also

next, **getSubExpressions**

References **uint()**.

◆ setFlags()

```
void RegEx::setFlags ( const string flags )
```

Sets flags for evaluation behaviour of expression.

Parameters

flags Flags specification for expression, it may contain letters 'm' (multi line), 'i' (ignore case) and 'g' (global search); all other characters are ignored.

Note

Flags may be modified after compiling expression too.

Member Data Documentation

◆ _flags

```
uint RegEx::_flags
```

private

stored flag mask

Definition at line **364** of file **bppregex.h**.

◆ _pattern

```
string RegEx::_pattern
```

private

stored pattern source

Definition at line **362** of file **bppregex.h**.

The documentation for this class was generated from the following file:

- **bppregex.h**

B++ Reference © 2005-2026 by R.-Erik Ebert

All parts of B++ are provided under terms and conditions of MIT license.