NSString (MiscRegex)

Declared In: <misckit/NSString+MiscRegex.h>

Category Description

NSString+MiscRegex provides regular expression support -- with some caveats -- for the NSString class, using the functions in the regexpr package included in the MiscKit. It provides similar functionality as the regular expression methods on the old MiscString, though most method names have changed. There are also more features, such as being able to get the strings matched by subportions of an expression, better handling of zero-length matches, ability to use the 'fastmap' option of the regexpr package for faster searching, and the ability to use sed-like tokens in replacement strings to represent portions of the matched text.

Problems with Unicode strings

Unfortunately, there are some problems with regular expression searching when it comes to Unicode strings

(which NSString normally supports). Unicode characters are 16 bits, which causes problems with traditional regular expression algorithms. I believe this is due to the use of character tables, which are 256 bytes with 8-bit characters (256 possibilities at one byte each), but 128K with 16-bit characters (65,536 possibilities at two bytes each). Composed character sequences are probably a large problem as well. In any event, I am not aware of any regular expression package that can search Unicode strings.

As a result, all the methods in this category operate on the -cString of the receiving NSString instance. This is not a problem with strings that have a 1-1 character mapping between itself and its cString (which should hopefully be most of them), but might be for strings that don't (such as those that have composed character sequences). For example, since NSRanges returned by these methods are based on the cString, they may not match the intended range of characters in the NSString itself. In that case, a call to -substringToIndex: using a returned NSRange would not return the exact substring intended -- and could even raise an exception, if that range was out of bounds for the Unicode string. NSString+MiscRegex methods that return NSStrings get those strings from the cString itself, so they don't have this problem. However, if any information was lost in the translation to the cString in the first place, then that information will also be lost in the returned NSStrings.

Regular expression syntax

The regular expression syntax used in these methods imitates that of GNU regular expressions, which is an extension of standard Unix regular expressions (as defined by ed(1), sed(1), grep(1), etc). See the documentation for the regexpr package for a full description.

Of particular note is the **re_set_syntax()** function, which sets a global syntax setting used by the regexpr routines. Different values allow for modifications to the default syntax, and the setting will affect the NSString+MiscRegex category (as well as any other code that uses the regexpr functions).

If the regular expression passed to these methods is malformed, NSExceptions are raised. The exceptionReason should describe what the problem with the expression was. Additionally, exceptions will be raised if there are internal errors during the actual search, but these should virtually never happen if the

expression itself is valid. To determine the validity of an expression, use the **-isValidRegex** method.

Using the Regex methods

Most of the methods are fairly straightforward. **-numOfRegex:...** returns the number of occurrences of the given expression in the string, **-stringsMatchedByRegex:....** returns all of the matches in an NSArray of NSStrings, and **-rangesMatchedByRegex:...** returns the ranges of the matches in an NSArray of NSValues (each NSValue encoding an NSRange). **-componentsSeparatedByRegex:...** returns an NSArray of the strings between the matches, much like the standard **-componentsSeparatedByString:** method.

-rangeOfRegex:.... returns the range of a particular match. The **-grep:...** methods can return different pieces of information about a match: the matched string, the string up to the match, the string after the match, the ranges of all three (returned in NSValues), substrings of the matched string corresponding to marked subexpressions in the regex, and the ranges of those substrings. See the section below for more information regarding these submatches.

The **-stringByReplacingRegex:withString:...** methods return a new string with the matched portion replaced with another string, and the **-stringByReplacingEveryOccurrenceOfRegex:...** methods return a new string with every match replaced. There are corresponding **-replaceRegex:** and **-replaceEveryOccurrenceOfRegex:** methods for NSMutableString, which do the replacements in place.

Because a match of zero length is perfectly possible with regular expressions, NSString's usual indicator for not finding the search term (an NSRange's 'length' member being equal to 0) isn't sufficient for NSString+MiscRegex. Instead, for any NSRanges returned by these methods, check if the 'location' member is equal to NSNotFound.

Marking subexpressions within a regular expression

Using parentheses, a part of the regular expression can be marked for later reference. For example, in the

expression "ab(c+)" [or "ab\(c+\)", depending on how the global syntax variable is set], the "c+" portion of the expression has been marked. Using the -grep:... methods, the substrings matched by these subexpressions can also be retrieved. By specifying the 'MiscSubstringsPiece', an NSArray of NSStrings is returned, which contains all of the substrings matched by all of the subexpressions. The substring for the full match is the first item (index zero), followed by the substrings matching all of the subexpressions. For example, if the expression above was used on the string "abccccdef", the array would contain two strings: "abcccc" (the entire match), and "cccc" (the part matched by "c+").

Marked subexpressions can also be referred to in the replace strings used in the replacement regex methods. See the option parameter 'MiscUseMatchSubstitutions' below.

Search options

The NSString+Regex methods take the same search options as NSStrings do (except for NSLiteralSearch), and add a couple of additional ones. As with the standard NSString methods, not every option is available with every method. NSCaseInsensitiveSearch and MiscFasterSearch are the only options that can be used anywhere.

Option	Effect
NSCaseInsensitiveSearch	The regular expression search ignores case distinctions between characters.
NSBackwardsSearch	The occurrences of expressions found go from the end of the range towards the front. Use of this option entails a speed penalty, since -numOfRegex: is used to determine how times the expressions occurs, then determines the correct part to match. The advantage of this is that backwards searches are symmetrical with normal ones -- the first occurrence going backwards will be the same as the last occurrence going forwards.
NSAnchoredSearch	Matches will only be found at the beginning or end of the range (at the end if

	used in combination with NSBackwardsSearch, at the beginning if not). No match at the beginning or end means nothing is found, even if a matching sequence of characters occurs elsewhere in the string.
MiscFasterSearch	Makes searches use the fastmap feature of the regexpr package, which does some additional precompilation on the expression making the ensuing search faster. However, the overhead may not be worth it if the searches are done on small strings.
MiscUseMatchSubstitutions	Turns on special processing of the *replaceString* parameter in the replace methods (-stringByReplacingRegex:, -stringByReplacingEveryOccurrenceOfRegex:, and the NSMutableString parallels). When this is set, tokens in the replaceString will be replaced with portions of the string actually matched by the expression: "$0" and "&" will be replaced by the entire match, while "$1" through "$9" will be replaced with the first through ninth marked subexpressions. For example, if the regex "ab(c+)" is used on the string "abccccddd", and the replaceString is "__$0__$1__", the end result would be "__abcccc__cccc__ddd". The $0 is replaced by the full match ("abcccc"), the $1 is replaced by the match of the first subexpression ("cccc"), and the "ddd" is the part of the string that wasn't matched (and therefore not replaced). If you want the literal character "$" or "&" to appear in the replaced string, use the tokens "$$" and "$&" in *replaceString*.

Using precompiled regex structures

Part of the overhead of a regular expression search is the compilation done on the expression before the search. If several operations are done with the same expression, it is compiled each time, which is inefficient. To get around this, every method that takes an NSString argument for the expression itself has a parallel method that

takes a pointer to a C structure containing a precompiled expression. This way, an expression can be compiled once and thereafter used in multiple operations. Using this feature is probably not necessary under most circumstances, but in performance-critical code it could make a difference, especially if there are many operations done with the same expression.

To precompile an expression, use the MiscNewRegexStruct() function, which takes an NSString expression along with an options mask and returns the pointer to a newly allocated, compiled regex structure. The options mask can contain NSCaseInsensitiveSearch and MiscFasterSearch, since those are the two options processed at this point. Accordingly, there is no point to passing either of those options to any of the struct-taking methods.

Use the MiscFreeRegexStruct() function to deallocate the memory used by the structure.

```
NSString  *expression;  // assume this exists
regexp_t  pattern;

pattern = MiscNewRegexStruct(expression, MiscFasterSearch);

[someString replaceRegexStruct:pattern withString:... ];
<...other operations using pattern...>

MiscFreeRegexStruct(pattern);
```

For space reasons, the degenerate versions of the struct-taking methods are not detailed in the main documentation section; only the full versions are. All degenerate versions of the methods exist, though (and are listed in the "Method Types" section).

Functions

MiscNewRegexStruct()

regexp_t **MiscNewRegexStruct(**NSString ***regexString*, unsigned *options***)**

Allocates space for a new regex structure, and compiles the regular expression *regexString* into it. Returns the pointer to the structure. The *options* mask can contain NSCaseInsensitiveSearch and MiscFasterSearch, as those are the two options processed by this function. If *regexString* is an invalid regular expression, an exception is raised.

MiscFreeRegexStruct()

void **MiscFreeRegexStruct(**regexp_t *pattern***)**

Frees the memory used by the structure pointed to by *pattern*, which should have been obtained by a previous call to MiscNewRegexStruct().

Method Types

Counting matches:

- numOfRegex:
- numOfRegex:options:
- numOfRegex:range:
- numOfRegex:options:range:
- numOfRegexStruct:
- numOfRegexStruct:options:
- numOfRegexStruct:range:

	- numOfRegexStruct:options:range:
Finding matches:	- rangeOfRegex: - rangeOfRegex:options: - rangeOfRegex:occurrenceNum: - rangeOfRegex:options:occurrenceNum: - rangeOfRegex:range: - rangeOfRegex:options:range: - rangeOfRegex:occurrenceNum:range: - rangeOfRegex:options:occurrenceNum:range: - rangeOfRegexStruct: - rangeOfRegexStruct:options: - rangeOfRegexStruct:occurrenceNum: - rangeOfRegexStruct:options:occurrenceNum: - rangeOfRegexStruct:range: - rangeOfRegexStructoptions:range: - rangeOfRegexStruct:occurrenceNum:range: - rangeOfRegexStruct:options:occurrenceNum:range: - rangesMatchedByRegex: - rangesMatchedByRegex:options: - rangesMatchedByRegexStruct: - rangesMatchedByRegexStruct:options:
Returning matched substrings:	- componentsSeparatedByRegex: - componentsSeparatedByRegex:options: - componentsSeparatedByRegexStruct: - componentsSeparatedByRegexStruct:options: - grep:forPiece:

- grep:forPiece:options:
- grep:forPiece:occurrenceNum:
- grep:forPiece:options:occurrenceNum:
- grepRegexStruct:forPiece:
- grepRegexStruct:forPiece:options:
- grepRegexStruct:forPiece:occurrenceNum:
- grepRegexStruct:forPiece:options:occurrenceNum:
- grep:forPieces:
- grep:forPieces:options:
- grep:forPieces:occurrenceNum:
- grep:forPieces:options:occurrenceNum:
- grepRegexStruct:forPieces:
- grepRegexStruct:forPieces:options:
- grepRegexStruct:forPieces:occurrenceNum:
- grepRegexStruct:forPieces:options:occurrenceNum:
- stringsMatchedByRegex:
- stringsMatchedByRegex:options:
- stringsMatchedByRegexStruct:
- stringsMatchedByRegexStruct:options:

Replacing matches:

- stringByReplacingEveryOccurrenceOfRegex:withString:
- stringByReplacingEveryOccurrenceOfRegex:withString:options:
- stringByReplacingEveryOccurrenceOfRegex:withString:range:
- stringByReplacingEveryOccurrenceOfRegex:withString:options:range:
- stringByReplacingEveryOccurrenceOfRegexStruct:withString:
- stringByReplacingEveryOccurrenceOfRegexStruct:withString:options:
- stringByReplacingEveryOccurrenceOfRegexStruct:withString:range:

	- stringByReplacingEveryOccurrenceOfRegexStruct:withString:options:range: - stringByReplacingRegex:withString: - stringByReplacingRegex:withString:options: - stringByReplacingRegex:withString:occurrenceNum: - stringByReplacingRegex:withString:options:occurrenceNum: - stringByReplacingRegex:withString:range: - stringByReplacingRegex:withString:options:range: - stringByReplacingRegex:withString:occurrenceNum:range: - stringByReplacingRegex:withString:options:occurrenceNum:range: - stringByReplacingRegexStruct:withString: - stringByReplacingRegexStruct:withString:options: - stringByReplacingRegexStruct:withString:occurrenceNum: - stringByReplacingRegexStruct:withString:options:occurrenceNum: - stringByReplacingRegexStruct:withString:range: - stringByReplacingRegexStruct:withString:options:range: - stringByReplacingRegexStruct:withString:occurrenceNum:range: - stringByReplacingRegexStruct:withString:options:occurrenceNum:range:
Validating expressions:	- isValidRegex

Instance Methods

componentsSeparatedByRegex:
componentsSeparatedByRegex:options:

- **componentsSeparatedByRegex:**(NSString *)*regex* **options:**(unsigned)*mask*

componentsSeparatedByRegexStruct:options:

- **componentsSeparatedByRegexStruct:**(regexp_t)*pattern* **options:**(unsigned)*mask*

Constructs and returns an NSArray containing substrings from the receiver that have been divided by matches of the expression *regex* (or *pattern*). The strings in the array appear in the order they did in the receiver.

The rules for this method follow that of NSString's **-componentsSeparatedByString:**. If the expression matches at the beginning of the string, the first string in the array will be ªº, since that is the string before that match. The same concept holds if the expression matches at the end of the string (the last string in the array will be ªº), and if matches occur back to back (ªº will be inserted into the array at that point).

Be careful of using an expression that can match the empty string, since there is then a potential match between every character! When this happens, the strings in the returned array will be at most one character long.

In the degenerate methods, *mask* defaults to 0. Possible options: NSCaseInsensitiveSearch, MiscFasterSearch.

See also: -componentsSeparatedByString: (NSString class cluster), **-stringsMatchedByRegex:**

grep:forPiece:
grep:forPiece:options:
grep:forPiece:occurrenceNum:
grep:forPiece:options:occurrenceNum:

- **grep:**(NSString *)*regex* **forPiece:**(NSString *)*key* **options:**(unsigned)*mask* **occurrenceNum:**(int)*n*

grepRegexStruct:forPiece:options:occurrenceNum:

- **grepRegexStruct:**(regexp_t)*pattern* **forPiece:**(NSString *)*key* **options:**(unsigned)*mask* **occurrenceNum:**(int)*n*

This is a single-piece version of **-grep:forPieces:...**. Returns the piece specified by *key*.

See also: -grep:ForPieces:...

grep:forPieces:
grep:forPieces:options:
grep:forPieces:occurrenceNum:
grep:forPieces:options:occurrenceNum:

- (NSDictionary *)**grep:**(NSString *)*regex* **forPieces:**(NSArray *)*keys* **options:**(unsigned)*mask* **occurrenceNum:**(int)*n*

grepRegexStruct:forPieces:options:occurrenceNum:

- (NSDictionary *)**grepRegexStruct:**(regexp_t)*pattern* **forPieces:**(NSArray *)*keys* **options:**(unsigned)*mask* **occurrenceNum:**(int)*n*

Returns, in a dictionary, various pieces of information about the n^{th} occurrence of the expression *regex* (or *pattern*) in the receiver going from left to right (or right to left if NSBackwardsSearch is specified). Use *n*=0 for the first occurrence.

The array of piece names *keys* (an NSArray of NSStrings) specifies which pieces to return in the dictionary -- information about keys not specified is not generated. The information associated with a piece can be pulled from the returned dictionary by using the piece name as the key.

The valid piece names (defined in NSString+MiscRegex.h) are:

MiscBeforePiece	The substring of the receiver up to the beginning of the match. If the desired occurrence is not found, then this will be the entire text of the receiver (the idea being that MiscBeforePiece + MiscMiddlePiece + MiscAfterPiece will always add

	up to the original string).
MiscMiddlePiece	The matched substring. If the desired occurrence is not found, this will be a blank string.
MiscAfterPiece	The substring of the receiver from the end of the match. If the desired occurrence is not found, this will be a blank string.
MiscBeforeRangePiece	The range of the receiver up to the beginning of the match. If the desired occurrence is not found, this will be the entire range of the receiver.
MiscMiddleRangePiece	The range of the match. If the desired occurrence is not found, the range will have NSNotFound in its location field.
MiscAfterRangePiece	The range of the receiver from the end of the match. If the desired occurrence is not found, the range will have NSNotFound in its location field.
MiscSubstringsPiece	An array of the submatched strings. The string at index 0 will be the matched text, and any successive strings will be the substrings matched by portions of the expression marked by parentheses. If the desired occurrence is not found, this will be an empty array. The substrings will be in the same order of the opening (left) parentheses in the expression. For example, with the expression "ab(cd(e+))", the array would have three elements: 1) the string matched by "abcde+" (the whole expression), 2) the string matched by "cde+" (the first marked subportion), and the string matched by "e+" (the second matched subportion).
MiscSubrangesPiece	An array of the submatched ranges. The range at index 0 will be the range matched by the entire expression, and any successive ranges will be the subranges matched by portions of the expression marked by parentheses. If the desired occurrence is not found, this will be an empty array. The subranges

will be in the same order as the left parentheses in the expression.

Any ranges returned for the range pieces will be contained in NSValue objects. To get the actual NSRange, use -getValue: on the NSValue instance

For further information on subportions of expressions, see ªMarking subexpressions within a regular expressionº in the category description.

In the degenerate methods, *mask* defaults to 0, *n* defaults to 0, and *range* defaults to the entire content of the receiver. Possible options: NSCaseInsensitiveSearch, NSBackwardsSearch, NSAnchoredSearch, MiscFasterSearch.

See also: -grep:ForPiece:...

isValidRegex

- (BOOL)**isValidRegex**

Returns YES if the receiving string is a valid regular expression, NO otherwise. This is useful for validating an expression before use with other methods in this category, since they will all raise exceptions on an invalid expression.

numOfRegex:
numOfRegex:options:
numOfRegex:range:
numOfRegex:options:range:

- (unsigned)**numOfRegex:**(NSString *)*regex* **options:**(unsigned)*mask* **range:**(NSRange)*range*

numOfRegexStruct:options:range:

- (unsigned)**numOfRegexStruct:**(regexp_t)*pattern* **options:**(unsigned)*mask* **range:**(NSRange)*range*

Returns the number of times the expression *regex* (or *pattern*) is found in the specified *range* of the receiving string. In the degenerate methods, *mask* defaults to 0 and *range* defaults to the entire content of the receiver. Possible options: NSCaseInsensitiveSearch, MiscFasterSearch.

See also: -rangeOfRegex:...

rangeOfRegex:
rangeOfRegex:options:
rangeOfRegex:occurrenceNum:
rangeOfRegex:options:occurrenceNum:
rangeOfRegex:range:
rangeOfRegex:options:range:
rangeOfRegex:occurrenceNum:range:
rangeOfRegex:options:occurrenceNum:range:

- (NSRange)**rangeOfRegex:**(NSString *)*regex* **options:**(unsigned)*mask* **occurrenceNum:**(int)*n* **range:** (NSRange)*range*

rangeOfRegexStruct:options:occurrenceNum:range:

- (NSRange)**rangeOfRegexStruct:**(regexp_t)*pattern* **options:**(unsigned)*mask* **occurrenceNum:**(int)*n* **range:**(NSRange)*range*

Returns the range of the n^{th} occurrence of the expression *regex* (or *pattern*) in the receiver going from left to right (or right to left if NSBackwardsSearch is specified). Use *n*=0 for the first occurrence. If the n^{th} occurrence is not found, the returned NSRange's location will equal NSNotFound.

In the degenerate methods, *mask* defaults to 0, *n* defaults to 0, and *range* defaults to the entire content of the receiver. Possible options: NSCaseInsensitiveSearch, NSBackwardsSearch, NSAnchoredSearch, MiscFasterSearch.

See also: -numOfRegex:...

rangesMatchedByRegex:
rangesMatchedByRegex:options:

- (NSArray *)**rangesMatchedByRegex:**(NSString *)*regex* **options:**(unsigned)*mask*

rangesMatchedByRegexStruct:options:

- (NSArray *)**rangesMatchedByRegexStruct:**(regexp_t)*pattern* **options:**(unsigned)*mask*

Constructs and returns an NSArray containing all the ranges matched by the expression *regex* (or *pattern*). The ranges in the array appear in the order they did in the receiver.

The ranges in the array are encoded with NSValue objects, so you have to use -getValue: to get the actual NSRange:

```
NSString     *expression;   // assume this exists
NSEnumerator *rangeEnum;
NSValue      *rangeValue;
NSRange      stringRange;

rangeEnum = [[someString rangesMatchedByRegex:expression] objectEnumerator];

while (rangeValue = [rangeEnum nextObject])
{
     [rangeValue getValue:&stringRange];
     ...
```

```
}
```

Be careful of using an expression that can match the empty string, since there is then a potential match between every character! Ranges with a length of 0 will be inserted into the returned array when this happens.

In the degenerate methods, *mask* defaults to 0. Possible options: NSCaseInsensitiveSearch, MiscFasterSearch.

See also: -stringsMatchedByRegex:, **-componentsSeparatedByRegex:**

stringByReplacingEveryOccurrenceOfRegex:withString:
stringByReplacingEveryOccurrenceOfRegex:withString:options:
stringByReplacingEveryOccurrenceOfRegex:withString:range:
stringByReplacingEveryOccurrenceOfRegex:withString:options:range:

- (NSString *)**stringByReplacingEveryOccurrenceOfRegex:**(NSString *)*regex* **withString:**(NSString *)*replaceString* **options:**(unsigned)*mask* **range:**(NSRange)*range*

stringByReplacingEveryOccurrenceOfRegexStruct:withString:options:range:

- (NSString *)**stringByReplacingEveryOccurrenceOfRegexStruct:**(regexp_t)*pattern* **withString:**(NSString *)*replaceString* **options:**(unsigned)*mask* **range:**(NSRange)*range*

Returns a new string with every occurrence of the expression *regex* (or *pattern*) in the receiver replaced with *replaceString*. If MiscUseMatchSubstitutions is specified, then special tokens can be used within *replaceString* to include portions of matched text (See ªSearch optionsº in the category description for more details).

In the degenerate methods, *mask* defaults to 0 and *range* defaults to the entire content of the receiver. Possible options: NSCaseInsensitiveSearch, MiscFasterSearch, MiscUseMatchSubstitutions.

See also: -stringByReplacingRegex:withString:...

stringByReplacingRegex:withString:
stringByReplacingRegex:withString:options:
stringByReplacingRegex:withString:occurrenceNum:
stringByReplacingRegex:withString:options:occurrenceNum:
stringByReplacingRegex:withString:range:
stringByReplacingRegex:withString:options:range:
stringByReplacingRegex:withString:occurrenceNum:range:
stringByReplacingRegex:withString:options:occurrenceNum:range:

- (NSString *)**stringByReplacingRegex:**(NSString *)*regex* **withString:**(NSString *)*replaceString* **options:**(unsigned)*mask* **occurrenceNum:**(int)*n* **range:**(NSRange)*range*

stringByReplacingRegexStruct:withString:options:occurrenceNum:range:

- (NSString *)**stringByReplacingRegexStruct:**(regexp_t)*pattern* **withString:**(NSString *)*replaceString* **options:**(unsigned)*mask* **occurrenceNum:**(int)*n* **range:**(NSRange)*range*

Returns a new, autoreleased NSString with the portion of the receiver matched by the n^{th} occurrence of the expression *regex* (or *pattern*) replaced by *replaceString*. The search for the n^{th} occurrence goes from left to right, unless NSBackwardsSearch is specified, in which case it goes from right to left. Use *n*=0 for the first occurrence. If MiscUseMatchSubstitutions is specified, then special tokens can be used within *replaceString* to include portions of matched text (See ªSearch optionsº in the category description for more details).

In the degenerate methods, *mask* defaults to 0, *n* defaults to 0, and *range* defaults to the entire content of the receiver. Possible options: NSCaseInsensitiveSearch, NSBackwardsSearch, NSAnchoredSearch, MiscFasterSearch, MiscUseMatchSubstitutions.

See also: -stringByReplacingEveryOccurrenceOfRegex:withString:...

stringsMatchedByRegex:
stringsMatchedByRegex:options:

- (NSArray *)**stringsMatchedByRegex:**(NSString *)*regex* **options:**(unsigned)*mask*

stringsMatchedByRegexStruct:options:

- (NSArray *)**stringsMatchedByRegexStruct:**(regexp_t)*pattern* **options:**(unsigned)*mask*

Constructs and returns an NSArray containing all the substrings from the receiver matched by the expression *regex* (or *pattern*). The strings in the array appear in the order they did in the receiver.

Be careful of using an expression that can match the empty string, since there is then a potential match between every character! Empty strings will be inserted into the returned array when this happens.

In the degenerate methods, *mask* defaults to 0. Possible options: NSCaseInsensitiveSearch, MiscFasterSearch.

See also: -**rangesMatchedByRegex:**, -**componentsSeparatedByRegex:**

NSMutableString (MiscRegex)

Declared In: <misckit/NSString+MiscRegex.h>

Category Description

NSMutableString+MiscRegex provides -replaceRegex... parallel methods for the -stringByReplacingRegex... methods of NSString+MiscRegex.

Method Types

Replacing matches:

- replaceEveryOccurrenceOfRegex:withString:
- replaceEveryOccurrenceOfRegex:withString:options:
- replaceEveryOccurrenceOfRegex:withString:range:
- replaceEveryOccurrenceOfRegex:withString:options:range:
- replaceEveryOccurrenceOfRegexStruct:withString:
- replaceEveryOccurrenceOfRegexStruct:withString:options:
- replaceEveryOccurrenceOfRegexStruct:withString:range:
- replaceEveryOccurrenceOfRegexStruct:withString:options:range:
- replaceRegex:withString:
- replaceRegex:withString:options:
- replaceRegex:withString:occurrenceNum:
- replaceRegex:withString:options:occurrenceNum:
- replaceRegex:withString:range:
- replaceRegex:withString:options:range:
- replaceRegex:withString:occurrenceNum:range:

- replaceRegex:withString:options:occurrenceNum:range:
- replaceRegexStruct:withString:
- replaceRegexStruct:withString:options:
- replaceRegexStruct:withString:occurrenceNum:
- replaceRegexStruct:withString:options:occurrenceNum:
- replaceRegexStruct:withString:range:
- replaceRegexStruct:withString:options:range:
- replaceRegexStruct:withString:occurrenceNum:range:
- replaceRegexStruct:withString:options:occurrenceNum:range:

Instance Methods

replaceEveryOccurrenceOfRegex:
replaceEveryOccurrenceOfRegex:withString:options:
replaceEveryOccurrenceOfRegex:withString:range:
replaceEveryOccurrenceOfRegex:withString:options:range:

- (void)**replaceEveryOccurrenceOfRegex:**(NSString *)*regex* **withString:**(NSString *)*replaceString* **options:**(unsigned)*mask* **range:**(NSRange)*range*

replaceEveryOccurrenceOfRegexStruct:withString:options:range:

- (void)**replaceEveryOccurrenceOfRegexStruct:**(regexp_t)*pattern* **withString:**(NSString *)*replaceString* **options:**(unsigned)*mask* **range:**(NSRange)*range*

Like -**stringByReplacingEveryOccurrenceOfRegex:withString**:..., except that it modifies the receiver in place instead of returning a modified string.

replaceRegex:withString:
replaceRegex:withString:options:
replaceRegex:withString:occurrenceNum:
replaceRegex:withString:options:occurrenceNum:
replaceRegex:withString:range:
replaceRegex:withString:options:range:
replaceRegex:withString:occurrenceNum:range:
replaceRegex:withString:options:occurrenceNum:range:

- (void)**replaceRegex:**(NSString *)*regex* **withString:**(NSString *)*replaceString* **options:**(unsigned)*mask* **occurrenceNum:**(int)*n* **range:**(NSRange)*range*

replaceRegexStruct:withString:options:occurrenceNum:range:

- (void)**replaceRegexStruct:**(regexp_t)*pattern* **withString:**(NSString *)*replaceString* **options:** (unsigned)*mask* **occurrenceNum:**(int)*n* **range:**(NSRange)*range*

Like -**stringByReplacingRegex:withString**:..., except that it modifies the receiver in place instead of returning a modified string.