RegexKit*Lite*

Lightweight Objective-C Regular Expressions for Mac OS X using the ICU Library

2010-04-18 – 4.0

Contents

Introduction to RegexKit*Lite*

This document introduces RegexKit*Lite* for Mac OS X. RegexKit*Lite* enables easy access to regular expressions by providing a number of additions to the standard *Foundation* `NSString` class. RegexKit*Lite* acts as a bridge between the `NSString` class and the regular expression engine in the *International Components for Unicode*, or ICU, dynamic shared library that is shipped with Mac OS X.

RegexKit*Lite* Highlights

- Uses the regular expression engine from the ICU library which is shipped with Mac OS X.
- Automatically caches compiled regular expressions.
- Uses direct access to a strings `UTF-16` buffer if it is available.
- Caches the `UTF-16` conversion that is required by the ICU library when direct access to a strings `UTF-16` buffer is unavailable.
- Small size makes it ideal for use in iPhone applications.
- Multithreading safe.
- 64-bit support.
- Custom DTrace probe points.
- Support for Mac OS X 10.5 Garbage Collection.
- Support for the Blocks language extension.
- Uses *Core Foundation* for greater speed.
- Very easy to use, all functionality is provided by a category extension to the `NSString` class.
- Consists of two files, a header and the Objective-C source.
- Xcode 3 integrated documentation available (p. 50).
- Distributed under the terms of the *BSD License* (p. 118).

Who Should Read This Document

This document is intended for readers who would like to be able to use regular expressions in their Objective-C applications, whether those applications are for the Mac OS X desktop, or for the iPhone.

This document, and RegexKit*Lite*, is also intended for anyone who has the need to search and manipulate `NSString` objects. If you've ever used the `NSScanner`, `NSCharacterSet`, and `NSPredicate` classes, or any of the the `NSString rangeOf...` methods, RegexKit*Lite* is for you.

Regular expressions are a powerful way to search, match and extract, and manipulate strings. RegexKit*Lite*

can perform many of the same operations that `NSScanner`, `NSCharacterSet`, and `NSPredicate` perform, and usually do it with far fewer lines of code. As an example, *RegexKitLite Cookbook - Parsing CSV Data* (p. 39) contains an example that is just over a dozen lines, but is a full featured CSV, or Comma Separated Value, parser that takes a CSV input and turns it in to a `NSArray` of `NSArray`s.

Organization of This Document

This document follows the conventions and styles used by Apples documentation and is divided in to two main parts:

- The *Class Reference* part, RegexKit*Lite* `NSString` Additions Reference (p. 47).
- The *Programming Guide* part, which consists of the following chapters:
  - RegexKit*Lite* Overview (p. 5)
  - Using RegexKit*Lite* (p. 12)
  - ICU Syntax (p. 25)
  - RegexKit*Lite* Cookbook (p. 34)
  - Adding RegexKit*Lite* to your Project (p. 43)

Additional information can be found in the following sections:

- Release Information (p. 106), which contains the Release Notes for RegexKit*Lite* 4.0 (p. 112)
- License Information (p. 118), which contains the RegexKit*Lite BSD License*.

Supporting RegexKit*Lite* through Financial Donations

A significant amount of time and effort has gone in to the development of RegexKit*Lite*. Even though it is distributed under the terms of the *BSD License*, you are encouraged to contribute financially if you are using RegexKit*Lite* in a profitable commercial application. Should you decide to contribute to RegexKit*Lite*, please keep the following in mind:

- What it would have cost you in terms of hours, or consultant fees, to develop similar functionality.
- The ohloh.net Project Cost metric, $91,529 as of 4.0, does not factor in the cost of writing documentation, which is where most of the effort is spent.
- The target audience for RegexKit*Lite* is very small, so there are relatively few units "sold".

You can contribute by visiting SourceForge.net's donation page for RegexKit*Lite*.

Important: You are always required to acknowledge the use of RegexKit*Lite* in your product as specified in the terms of the *BSD License*.

Download

You can download RegexKit*Lite* distribution that corresponds to this version of the documentation here–`RegexKitLite-4.0.tar.bz2`, 135.9K. To be automatically notified when a new version of RegexKit*Lite* is available, add the RegexKit*Lite* documentation feed (p. 50) to Xcode.

Reporting Bugs

You can file bug reports, or review submitted bugs, at the following URL:

`http://sourceforge.net/tracker/?group_id=204582&atid=990188`

> **Note:** Anonymous bug reports are no longer accepted due to spam. A SourceForge.net account is required to file a bug report.

Contacting The Author

The author can be contacted at `regexkitlite@gmail.com`.

Conventions

The following typographic conventions are used in this document:

Italic
: Book titles, cross-references to other documents, new terms, and terms to be emphasized.

Colored Text
: Indicates a hyperlink, either to another part of this document, or an external URL.

`Fixed Width`
: Program code, applications, commands, files and directories, and the output of shell programs.

`Fixed␣Width␣A␣␣B␣C`
: Used when it is necessary to unambiguously identify spaces.

`Fixed Width Bold`
: C preprocessor defines and user typed input in a shell.

`Fixed Width Italic`
: Placeholder for user-supplied values and function or method parameters.

File ▸ Save As…
: Menus and menu selections are in a sans serif font.

`@"\\s*(\\w+)?";`
: A colored ring is used to draw attention to a specific part of the text, such as highlighting the difference between two examples.

↵
: Used to indicate when the return key is pressed in a shell.

This document includes text where the exact sequence of characters has specific meaning, such as program code and regular expressions. When printed, text that was meant to span a single line only will have to be

split across multiple lines in order to fit within the fixed width of a page. The '↩' glyph is used to visually indicate when a line has been split across multiple lines, and does not represent part of the intended sequence of characters. For example:

Description	Regex
Floating Point with Exponent	`[+\-]?(?:[0-9]*\.[0-9]+|↩` `[0-9]+\.)(?:[eE][+\-]?[0-9]+)?`

The same line shown without splitting it across multiple lines:

Description	Regex
Floating Point with Exponent	`[+\-]?(?:[0-9]*\.[0-9]+|[0-9]+\.)(?:[eE][+\-]?[0-9]+)?`

For aesthetic purposes, it is sometimes useful to indent the continuation of the split line. When this is done, the '↪' glyph is used to indicate where the continuation begins:

```
printf("%5d: a long, unbroken ABC↩
      ↪def string.\n", x);
```

The same line shown without splitting it across multiple lines:

```
printf("%5d: a long, unbroken ABCdef string.\n", x);
```

When the command that you need to type in to a shell is split across lines, a \ (backslash) is added to the end of the line where the split was made. These end of line backslashes are optional, you can either enter them as \↵, in which case the shell will wait for additional input, or they can be removed entirely, with the command spanning a single line. An example:

```
shell% gcc someFile.m …more gcc arguments… -framework Foundation \
       -licucore -o someFile↵
shell% 
```

Colophon

The text font is Adobe Warnock Pro, the fixed width font is Letter Gothic, and the sans serif font is Adobe Myriad Pro. MnSymbol10 and Apple Symbol are used for a small number of glyphs.

This document was typeset with XeLaTeX and TeXShop.app on Mac OS X and makes use of a number of freely available LaTeX packages– `fancyhdr`, `fancyvrb`, `hyperref`, `listings`, `pstricks`, `pst-blur`, `tabularx`, `titlesec`, `tocloft`, and others, either directly or indirectly through inclusion by another package. The author wishes to thank the authors of these packages, including the authors of TeX, LaTeX, XeLaTeX, and TeXShop, for making them freely available.

RegexKit*Lite* Overview

While RegexKit*Lite* is not a descendant of the `RegexKit.framework` source code, it does provide a small subset of *RegexKits* `NSString` methods for performing various regular expression tasks. These include determining the range that a regular expression matches within a string, easily creating a new string from the results of a match, splitting a string in to a `NSArray` with a regular expression, and performing search and replace operations with regular expressions using common `$n` substitution syntax.

RegexKit*Lite* uses the regular expression provided by the ICU library that ships with Mac OS X. The two files, `RegexKitLite.h` and `RegexKitLite.m`, and linking against the `/usr/lib/libicucore.dylib` ICU shared library is all that is required. Adding RegexKit*Lite* to your project only adds a few kilobytes of overhead to your applications size and typically only requires a few kilobytes of memory at run-time. Since a regular expression must first be compiled by the ICU library before it can be used, RegexKit*Lite* keeps a small 4-way set associative cache with a least recently used replacement policy of the compiled regular expressions.

See Also

RegexKit Framework
International Components for Unicode
Unicode Home Page

Official Support from Apple for ICU Regular Expressions

Mac OS X

As of Mac OS X 10.6, the author is not aware of any official support from Apple for linking to the `libicucore.dylib` library. On the other hand, the author is unaware of any official prohibition against it, either. Linking to the ICU library and making use of the ICU regular expression API is slightly different than making use of private, undocumented API's. There are a number of very good reasons why you shouldn't use private, undocumented API's, such as:

- The undocumented, private API is not yet mature enough for Apple to commit to supporting it. Once an API is made "public", developers expect future versions to at least be compatible with previously published versions.
- The undocumented, private API may expose implementation specific details that can change between versions. Public API's are the proper "abstraction layer boundary" that allows the provider of the API to hide implementation specific details.

The ICU library, on the other hand, contains a "published, public API" in which the ICU developers have committed to supporting in later releases, and RegexKit*Lite* uses only these public APIs. One could argue that Apple is not obligated to continue to include the ICU library in later versions of Mac OS X, but this seems unlikely for a number of reasons which will not be discussed here. With the introduction of iPhone OS 3.2, Apple now officially supports iPhone applications linking to the ICU library for the purpose of using its regular expression functionality. This is encouraging news for Mac OS X developers if one assumes that Apple will try to keep some kind of parity between the iPhone OS and Mac OS X API's.

iPhone OS < 3.2

Prior to iPhone OS 3.2, there was never any official support from Apple for linking to the `libicucore.dylib` library. It was unclear if linking to the library would violate the iPhone OS SDK Agreement prohibition against using undocumented API's, but a large number of iPhone applications choose to use RegexKit*Lite*, and the author is not aware of a single rejection because of it.

iPhone OS ≥ 3.2

Starting with iPhone OS 3.2, Apple now officially allows iPhone OS applications to link with the ICU library. The ICU library contains a lot of functionality for dealing with internationalization and localization, but Apple only officially permits the use of the ICU Regular Expression functionality.

Apple also provides a way to use ICU based regular expressions from *Foundation* by adding a new option to `NSStringCompareOptions`– `NSRegularExpressionSearch`. This new option can be used with the `NSString rangeOfString:options:` method, and the following example of its usage is given:

```
// finds phone number in format nnn-nnn-nnnn
NSRange r;
NSString *regEx = @"[0-9]{3}-[0-9]{3}-[0-9]{4}";
r = [textView.text rangeOfString:regEx options:NSRegularExpressionSearch];
if (r.location != NSNotFound) {
    NSLog(@"Phone number is %@", [textView.text substringWithRange:r]);
} else {
    NSLog(@"Not found.");
}
```

At this time, `rangeOfString:options:` is the only regular expression functionality Apple has added to *Foundation* and capture groups in a regular expression are not supported. Apple also gives the following note:

> **Note:** As noted in "ICU Regular-Expression Support," the ICU libraries related to regular expressions are included in iPhone OS 3.2. However, you should only use the ICU facilities if the `NSString` alternative is not sufficient for your needs.

RegexKit*Lite* provides a much richer API, such as the automatic extraction of a match as a `NSString`. Using RegexKit*Lite*, the example can be rewritten as:

```
// finds phone number in format nnn-nnn-nnnn
NSString *regEx = @"[0-9]{3}-[0-9]{3}-[0-9]{4}";
NSString *match = [textView.text stringByMatching:regEx];
if ([match isEqual:@""] == NO) {
    NSLog(@"Phone number is %@", match);
} else {
    NSLog(@"Not found.");
}
```

What's more, RegexKit*Lite* provides easy access to all the matches of a regular expression in a `NSString`:

```
// finds phone number in format nnn-nnn-nnnn
NSString *regEx = @"[0-9]{3}-[0-9]{3}-[0-9]{4}";
for(NSString *match in [textView.text componentsMatchedByRegex:regEx]) {
    NSLog(@"Phone number is %@", match);
}
```

To do the same thing using just `NSRegularExpressionSearch` would require significantly more code and effort on your part. RegexKit*Lite* also provides powerful search and replace functionality:

```
// finds phone number in format nnn-nnn-nnnn
NSString *regEx = @"([0-9]{3})-([0-9]{3}-[0-9]{4})";
// and transforms the phone number in to the format of (nnn) nnn-nnnn
NSString *replaced = [textView.text stringByReplacingOccurrencesOfRegex:regEx
                                                             withString:@"($1) $2"];
```

RegexKit*Lite* also has a number of performance enhancing features built in such as caching compiled regular expressions. Although the author does not have any benchmarks comparing `NSRegularExpressionSearch` to RegexKit*Lite*, it is likely that RegexKit*Lite* outperforms `NSRegularExpressionSearch`.

See Also

What's New in iPhone OS - ICU Regular-Expression Support
What's New in iPhone OS - Foundation Framework Changes
iPad Programming Guide - ICU Regular-Expression Support
iPad Programming Guide - Foundation-Level Regular Expressions
`NSRegularExpressionSearch`
`- rangeOfString:options:`

The iPhone 4.0 SDK Agreement

While iPhone OS 3.2 included official, Apple sanctioned use of linking of the ICU library for the purposes of using the ICU regular expression engine, the iPhone OS 4.0 SDK included the following change to the iPhone OS SDK Agreement:

3.3.1 Applications may only use Documented APIs in the manner prescribed by Apple and must not use or call any private APIs. Applications must be originally written in Objective-C, C, C++, or JavaScript as executed by the iPhone OS WebKit engine, and only code written in C, C++, and Objective-C may compile and directly link against the Documented APIs (e.g., Applications that link to Documented APIs through an intermediary translation or compatibility layer or tool are prohibited).

This raises a number of obvious questions:

- Does 3.3.1 apply to RegexKit*Lite*?
- Will the use of RegexKit*Lite* in an iPhone OS application be grounds for rejection under 3.3.1?

There is considerable speculation as to what is covered by this change, but at the time of this writing, there is no empirical evidence or official guidelines from Apple to make any kind of an informed decision as to whether or not the use of RegexKit*Lite* would violate 3.3.1. It is the authors opinion that RegexKit*Lite* could be considered as a *compatibility layer* between `NSString` and the now *Documented APIs* for regular expressions in the ICU library.

It is widely speculated that the motivation for the change to 3.3.1 was to prevent the development of Flash applications for the iPhone. The author believes that most reasonable people would consider the application of *compatibility layer* in this context to mean something entirely different than what it means when applied to RegexKit*Lite*.

At this time, the author is not aware of a single iPhone application that has been rejected due to the use of RegexKit*Lite*. If your application is rejected due to the use of RegexKit*Lite*, please let the author know by emailing `regexkitlite@gmail.com`. As always, `CAVEAT EMPTOR`.

The Difference Between RegexKit.framework and RegexKit*Lite*

RegexKit.framework and RegexKit*Lite* are two different projects. In retrospect, RegexKit*Lite* should have been given a more distinctive name. Table-1.1 summarizes some of the key differences between the two.

Table 1.1 – Differences between RegexKit.framework and RegexKit*Lite*

	RegexKit.framework	**RegexKit*Lite***
Regex Library	PCRE	ICU
Library Included	Yes, built into framework object file.	No, provided by Mac OS X.
Library Linked As	Statically linked into framework.	Dynamically linked to `/usr/lib/libicucore.dylib`.
Compiled Size	Approximately 371KB† per architecture.	Very small, approximately 16KB-20KB‡ per architecture.
Style	External, linked to framework.	Compiled directly in to final executable.
Feature Set	Large, with additions to many classes.	Minimal, `NSString` only.

† Version 0.6.0. About half of the 371KB is the PCRE library.
The default distribution framework shared library file is 1.4MB in size and includes the `ppc`, `ppc64`, `i386`, and `x86_64` architectures.
If 64-bit support is removed, the framework shared library file size drops to 664KB.

‡ Since the ICU library is part of Mac OS X, it does not add to the final size.

Compiled Regular Expression Cache

The `NSString` that contains the regular expression must be compiled in to an ICU `URegularExpression`. This can be an expensive, time consuming step, and the compiled regular expression can be reused again in another search, even if the strings to be searched are different. Therefore RegexKit*Lite* keeps a small cache of recently compiled regular expressions.

The cache is organized as a 4-way set associative cache, and the size of the cache can be tuned with the preprocessor define `RKL_CACHE_SIZE`. The default cache size, which should always be a prime number, is set to `13`. Since the cache is 4-way set associative, the total number of compiled regular expressions that can be cached is `RKL_CACHE_SIZE` times four, for a total of `13 * 4`, or `52`. The `NSString` *`regexString`* is mapped to a cache set using modular arithmetic: Cache set ≡ `[regexString hash]` mod `RKL_CACHE_SIZE`, i.e. `cacheSet = [regexString hash] % 13;`. Since RegexKit*Lite* uses *Core Foundation*, this is actually coded as `cacheSet = CFHash(regexString) % RKL_CACHE_SIZE;`.

Each of the four "ways" of a cache set are checked to see if it contains a `NSString` that was used to create the compiled regular expression that is identical to the `NSString` for the regular expression that is being checked. If there is an exact match, then the matching "way" is updated as the most recently used, and the compiled regular expression is used as-is. Otherwise, the least recently used, or LRU, "way" in the cache set is cleared and replaced with the compiled regular expression for the regular expression that wasn't in the cache.

In addition to the compiled regular expression cache, RegexKit*Lite* keeps a small lookaside cache that maps a regular expressions `NSString` pointer and `RKLRegexOptions` (p. 95) directly to a cached compiled regular expression. When a regular expressions `NSString` pointer and `RKLRegexOptions` (p. 95) is in the lookaside

cache, RegexKit*Lite* can bypass calling `CFHash(regexString)` and checking each of the four "ways" in a cache set since the lookaside cache has provided the exact cached compiled regular expression. The lookaside cache is quite small at just 64 bytes and it was added because `Shark.app` profiling during performance tuning showed that `CFHash()`, while quite fast, was the primary bottleneck when retrieving already compiled and cached regular expressions, typically accounting for ≅40% of the look up time.

Regular Expressions in Mutable Strings

When a regular expression is compiled, an immutable copy of the string is kept. For immutable `NSString` objects, the copy is usually the same object with its reference count increased by one. Only `NSMutableString` objects will cause a new, immutable `NSString` to be created.

If the regular expression being used is stored in a `NSMutableString`, the cached regular expression will continue to be used as long as the `NSMutableString` remains unchanged. Once mutated, the changed `NSMutableString` will no longer be a match for the cached compiled regular expression that was being used by it previously. Even if the newly mutated strings `hash` is congruent to the previous unmutated strings `hash` modulo `RKL_CACHE_SIZE`, that is to say they share the same cache set (i.e., `([mutatedString hash] % RKL_CACHE_SIZE) == ([unmutatedString hash] % RKL_CACHE_SIZE)`), the immutable copy of the regular expression string used to create the compiled regular expression is used to ensure true equality. The newly mutated string will have to go through the whole regular expression compile and cache creation process.

This means that `NSMutableString` objects can be safely used as regular expressions, and any mutations to those objects will immediately be detected and reflected in the regular expression used for matching.

Searching Mutable Strings

Unfortunately, the ICU regular expression API requires that the compiled regular expression be "set" to the string to be searched. To search a different string, the compiled regular expression must be "set" to the new string. Therefore, RegexKit*Lite* tracks the last `NSString` that each compiled regular expression was set to, recording the pointer to the `NSString` object, its hash, and its length. If any of these parameters are different from the last parameters used for a compiled regular expression, the compiled regular expression is "set" to the new string. Since mutating a string will likely change its `hash` value, it's generally safe to search `NSMutableString` objects, and in most cases the mutation will reset the compiled regular expression to the updated contents of the `NSMutableString`.

> **Caution:** Care must be taken when mutable strings are searched and there exists the possibility that the string has mutated between searches. See *NSString RegexKitLite Additions Reference - Cached Information and Mutable Strings* (p. 50) for more information.

Last Match Information

When performing a match, the arguments used to perform the match are kept. If those same arguments are used again, the actual matching operation is skipped because the compiled regular expression already contains the results for the given arguments. This is mostly useful when a regular expression contains multiple capture groups, and the results for different capture groups for the same match are needed. This means that there is only a small penalty for iterating over all the capture groups in a regular expression for a match, and essentially becomes the direct ICU regular expression API equivalent of `uregex_start()` and `uregex_end()`.

See Also
ICU4C C API - Regular Expressions
ICU Regular Expression Syntax (p. 25)

UTF-16 Conversion Cache

RegexKit*Lite* is ideal when the string being matched is a non-ASCII, Unicode string. This is because the regular expression engine used, ICU, can only operate on UTF-16 encoded strings. Since Cocoa keeps essentially all non-ASCII strings encoded in UTF-16 form internally, this means that RegexKit*Lite* can operate directly on the strings buffer without having to make a temporary copy and transcode the string in to ICU's required format.

Like all object oriented programming, the internal representation of an objects information is private. However, the ICU regular expression engine requires that the text to be search be encoded as a UTF-16 string. For pragmatic purposes, *Core Foundation* has several public functions that can provide direct access to the buffer used to hold the contents of the string, but such direct access is only available if the private buffer is already encoded in the requested direct access format. As a rough rule of thumb, 8-bit simple strings, such as ASCII, are kept in their 8-bit format. Non 8-bit simple strings are stored as UTF-16 strings. Of course, this is an implementation private detail, so this behavior should never be relied upon. It is mentioned because of the tremendous impact on matching performance and efficiency it can have if a string must be converted to UTF-16.

For strings in which direct access to the UTF-16 string is available, RegexKit*Lite* uses that buffer. This is the ideal case as no extra work needs to be performed, such as converting the string in to a UTF-16 string, and allocating memory to hold the temporary conversion. Of course, direct access is not always available, and occasionally the string to be searched will need to be converted in to a UTF-16 string.

RegexKit*Lite* has two conversion cache types. Each conversion cache type contains four buffers each, and buffers are re-used on a least recently used basis. If the selected cache type does not contain the contents of the NSString that is currently being searched in any of its buffers, the least recently used buffer is cleared and the current NSString takes it place. The first conversion cache type is fixed in size and set by the C preprocessor define RKL_FIXED_LENGTH, which defaults to 2048. Any string whose length is less than RKL_FIXED_LENGTH will use the fixed size conversion cache type. The second conversion cache type, for strings whose length is longer than RKL_FIXED_LENGTH, will use a dynamically sized conversion buffer. The memory allocation for the dynamically sized conversion buffer is resized for each conversion with realloc() to the size needed to hold the entire contents of the UTF-16 converted string.

This strategy was chosen for its relative simplicity. Keeping track of dynamically created resources is required to prevent memory leaks. As designed, there are only four pointers to dynamically allocated memory: the four pointers to hold the conversion contents of strings whose length is larger than RKL_FIXED_LENGTH. However, since realloc() is used to manage those memory allocations, it becomes very difficult to accidentally leak the buffers. Having the fixed sized buffers means that the memory allocation system isn't bothered with many small requests, most of which are transient in nature to begin with. The current strategy tries to strike the best balance between performance and simplicity.

Mutable Strings

When converted in to a UTF-16 string, the hash of the NSString is recorded, along with the pointer to the NSString object and the strings length. In order for the RegexKit*Lite* to use the cached conversion, all of these parameters must be equal to their values of the NSString to be searched. If there is any difference, the cached conversion is discarded and the current NSString, or NSMutableString as the case may be, is reconverted in to a UTF-16 string.

Caution: Care must be taken when mutable strings are searched and there exists the possibility that the string has mutated between searches. See *NSString RegexKitLite Additions Reference - Cached Information and Mutable Strings* (p. 50) for more information.

Multithreading Safety

RegexKit*Lite* is also multithreading safe. Access to the compiled regular expression cache and the conversion cache is protected by a single `OSSpinLock` to ensure that only one thread has access at a time. The lock remains held while the regular expression match is performed since the compiled regular expression returned by the ICU library is not safe to use from multiple threads. Once the match has completed, the lock is released, and another thread is free to lock the cache and perform a match.

Important: While it is safe to use the same regular expression from any thread at any time, the usual multithreading caveats apply. For example, it is not safe to mutate a `NSMutableString` in one thread while performing a match on the mutating string in another.

If Blocks functionality is enabled, and a RegexKit*Lite* method that takes a Block as one of its parameters is used, RegexKit*Lite* takes a slightly different approach in order to support the asynchronous, and possibly re-entrant, nature of Blocks.

First, an autoreleased Block helper proxy object is created and is used to keep track of any Block local resources needed to perform a Block-based enumeration.

Then the regular expression cache is checked exactly as before. Once a compiled regular expression is obtained, the ICU function `uregex_clone` is used to create a Block local copy of the regular expression. After the Block local copy has been made, the global compiled regular expression cache lock is unlocked.

If the string to be searched requires conversion to `UTF-16`, then a one time use Block local `UTF-16` conversion of the string is created.

These changes mean that RegexKit*Lite* Block-based enumeration methods are just as multithreading safe and easy to use as non-Block-based enumeration methods, such as the ability to continue to use RegexKit*Lite* methods without any restrictions from within the Block used for enumeration.

64-bit Support

RegexKit*Lite* is 64-bit clean. Internally, RegexKit*Lite* uses Cocoas standard `NSInteger` and `NSUInteger` types for representing integer values. The size of these types change between 32-bit and 64-bit automatically, depending on the target architecture. ICU, on the other hand, uses a signed 32-bit `int` type for many of its arguments, such as string offset values. Because of this, the maximum length of a string that RegexKit*Lite* will accept is the maximum value that can be represented by a signed 32-bit integer, which is approximately 2 gigabytes. Strings that are longer this limit will raise `NSRangeException`. This limitation may be significant to those who are switching to 64-bit because the size of the data they need to process exceeds what can be represented with 32-bits.

Note: Several numeric constants throughout this document will have either `L` or `UL` appended to them– for example `0UL`, or `2L`. This is to ensure that they are treated as 64-bit `long` or `unsigned long` values, respectively, when targeting a 64-bit architecture.

Using RegexKit*Lite*

The goal of RegexKit*Lite* is not to be a comprehensive Objective-C regular expression framework, but to provide a set of easy to use primitives from which additional functionality can be created. To this end, RegexKit*Lite* provides the following two core primitives from which everything else is built:

- `- (NSInteger)captureCountWithOptions:(RKLRegexOptions)options error:(NSError **)error;`
- `- (NSRange)rangeOfRegex:(NSString *)regex options:(RKLRegexOptions)options inRange:(NSRange)range capture:(NSInteger)capture error:(NSError **)error;`

There is often a need to create a new string of the characters that were matched by a regular expression. RegexKit*Lite* provides the following method which conveniently combines sending the receiver `substringWithRange:` with the range returned by `rangeOfRegex:`.

- `- (NSString *)stringByMatching:(NSString *)regex options:(RKLRegexOptions)options inRange:(NSRange)range capture:(NSInteger)capture error:(NSError **)error;`

RegexKit*Lite* 2.0 adds the ability to split strings by dividing them with a regular expression, and the ability to perform search and replace operations using common `$n` substitution syntax. `replaceOccurrencesOfRegex:withString:` (p. 87) is used to modify the contents of `NSMutableString` objects directly and `stringByReplacingOccurrencesOfRegex:withString:` (p. 93) will create a new, immutable `NSString` from the receiver.

- `- (NSArray *)componentsSeparatedByRegex:(NSString *)regex options:(RKLRegexOptions)options range:(NSRange)range error:(NSError **)error;`
- `- (NSUInteger)replaceOccurrencesOfRegex:(NSString *)regex options:(RKLRegexOptions)options withString:(NSString *)replacement range:(NSRange)range error:(NSError **)error;`
- `- (NSString *)stringByReplacingOccurrencesOfRegex:(NSString *)regex options:(RKLRegexOptions)options withString:(NSString *)replacement range:(NSRange)range error:(NSError **)error;`

RegexKit*Lite* 3.0 adds several new methods that return a `NSArray` containing the aggregated results of a number of individual regex operations.

- `- (NSArray *)arrayOfCaptureComponentsMatchedByRegex:(NSString *)regex options:(RKLRegexOptions)options range:(NSRange)range error:(NSError **)error;`
- `- (NSArray *)captureComponentsMatchedByRegex:(NSString *)regex options:(RKLRegexOptions)options range:(NSRange)range error:(NSError **)error;`
- `- (NSArray *)componentsMatchedByRegex:(NSString *)regex range:(NSRange)range;`

RegexKit*Lite* 4.0 adds several new methods that take advantage of the new blocks language extension.

- `- (BOOL)enumerateStringsMatchedByRegex:(NSString *)regex usingBlock:(void (^)(NSInteger captureCount, NSString * const capturedStrings[captureCount], const NSRange capturedRanges[captureCount], volatile BOOL * const stop))block;`
- `- (BOOL)enumerateStringsMatchedByRegex:(NSString *)regex options:(RKLRegexOptions)options inRange:(NSRange)range error:(NSError **)error enumerationOptions:(RKLRegexEnumerationOptions)enumerationOptions usingBlock:(void (^)(NSInteger captureCount, NSString * const capturedStrings[captureCount], const NSRange capturedRanges[captureCount], volatile BOOL * const stop))block;`
- `- (BOOL)enumerateStringsSeparatedByRegex:(NSString *)regex usingBlock:(void (^)(NSInteger captureCount, NSString * const capturedStrings[captureCount], const NSRange capturedRanges[captureCount], volatile BOOL * const stop))block;`
- `- (BOOL)enumerateStringsSeparatedByRegex:(NSString *)regex options:(RKLRegexOptions)options inRange:(NSRange)range error:(NSError **)error enumerationOptions:(RKLRegexEnumerationOptions)enumerationOptions usingBlock:(void (^)(NSInteger captureCount, NSString * const capturedStrings[captureCount], const NSRange capturedRanges[captureCount], volatile BOOL * const stop))block;`
- `- (NSString *)stringByReplacingOccurrencesOfRegex:(NSString *)regex usingBlock:(NSString *(^)(NSInteger captureCount, NSString * const capturedStrings[captureCount], const NSRange capturedRanges[captureCount], volatile BOOL * const stop))block;`
- `- (NSString *)stringByReplacingOccurrencesOfRegex:(NSString *)regex options:(RKLRegexOptions)options inRange:(NSRange)range error:(NSError **)error enumerationOptions:(RKLRegexEnumerationOptions)enumerationOptions usingBlock:(NSString *(^)(NSInteger captureCount, NSString * const capturedStrings[captureCount], const NSRange capturedRanges[captureCount], volatile BOOL * const stop))block;`
- `- (NSUInteger)replaceOccurrencesOfRegex:(NSString *)regex usingBlock:(NSString *(^)(NSInteger captureCount, NSString * const capturedStrings[captureCount], const NSRange capturedRanges[captureCount], volatile BOOL * const stop))block;`
- `- (NSUInteger)replaceOccurrencesOfRegex:(NSString *)regex options:(RKLRegexOptions)options inRange:(NSRange)range error:(NSError **)error enumerationOptions:(RKLRegexEnumerationOptions)enumerationOptions usingBlock:(NSString *(^)(NSInteger captureCount, NSString * const capturedStrings[captureCount], const NSRange capturedRanges[captureCount], volatile BOOL * const stop))block;`

There are no additional classes that supply the regular expression matching functionality, everything is accomplished with the two methods above. These methods are added to the existing `NSString` class via an Objective-C category extension. See *RegexKitLite `NSString` Additions Reference* (p. 47) for a complete list of methods.

The real workhorse is the `rangeOfRegex:options:inRange:capture:error:` (p. 84) method. The receiver of the message is an ordinary `NSString` class member that you wish to perform a regular expression match on. The parameters of the method are a `NSString` containing the regular expression *`regex`*, any `RKLRegexOptions` (p. 95) match *`options`*, the `NSRange` *`range`* of the receiver that is to be searched, the *`capture`* number

from the regular expression *regex* that you would like the result for, and an optional *error* parameter that will contain a `NSError` object if a problem occurs with the details of the error.

Important:	The C language assigns special meaning to the `\` character when inside a quoted `" "` string in your source code. The `\` character is the escape character, and the character that follows has a different meaning than normal. The most common example of this is `\n`, which translates in to the *new-line* character. Because of this, you are required to 'escape' any uses of `\` by prepending it with another `\`. In practical terms this means doubling any `\` in a regular expression, which unfortunately is quite common, that are inside of quoted `" "` strings in your source code. Failure to do so will result in numerous warnings from the compiler about unknown escape sequences. To match a single literal `\` with a regular expression requires no less than four backslashes: `"\\\\"`.

See Also

ICU Regular Expression Syntax (p. 25)
RegexKitLite Cookbook (p. 34)
RegexKitLite `NSString` Additions Reference (p. 47)
Regular Expression Options (p. 95)
RegexKitLite NSError and NSException User Info Dictionary Keys (p. 103)
Blocks Programming Topics

Finding the Range of a Match

A simple example:

```
NSString *searchString = @"This is neat.";
NSString *regexString  = @"(\\w+)\\s+(\\w+)\\s+(\\w+)";
NSRange   matchedRange = NSMakeRange(NSNotFound, 0UL);
NSError  *error        = NULL;

matchedRange = [searchString rangeOfRegex:regexString
                                  options:RKLNoOptions
                                  inRange:searchRange
                                  capture:2L
                                    error:&error];

NSLog(@"matchedRange: %@", NSStringFromRange(matchedRange));
// 2008-03-18 03:51:16.530 test[51583:813] matchedRange: {5, 2}
```

In the previous example, the `NSRange` that capture number `2` matched is `{5, 2}`, which corresponds to the word `is` in `searchString`. Once the `NSRange` is known, you can create a new string containing just the matching text:

```
NSString *matchedString = [searchString substringWithRange:matchedRange];

NSLog(@"matchedString: '%@'", matchedString);
// 2008-03-18 03:51:16.532 test[51583:813] matchedString: 'is'
```

RegexKit*Lite* can conveniently combine the two steps above with `stringByMatching:`. This example also demonstrates the use of one of the simpler convenience methods, where some of the arguments are automatically filled in with default values:

```
NSString *searchString  = @"This is neat.";
NSString *regexString   = @"(\\w+)\\s+(\\w+)\\s+(\\w+)";

NSString *matchedString = [searchString stringByMatching:regexString capture:2L];

NSLog(@"matchedString: '%@'", matchedString);
// 2008-03-18 03:53:42.949 test[51583:813] matchedString: 'is'
```

See Also

- `- rangeOfRegex:` (p. 83)
- `- stringByMatching:` (p. 89)

ICU Regular Expression Syntax (p. 25)

Search and Replace

You can perform search and replace operations on `NSString` objects and use common `$n` capture group substitution in the replacement string:

```
NSString *searchString      = @"This is neat.";
NSString *regexString       = @"\\b(\\w+)\\b";
NSString *replaceWithString = @"{$1}";
NSString *replacedString    = NULL;

replacedString = [searchString stringByReplacingOccurrencesOfRegex:regexString
                                                         withString:replaceWithString];

NSLog(@"replaced string: '%@'", replacedString);
// 2008-07-01 19:03:03.195 test[68775:813] replaced string: '{This} {is} {neat}.'
```

> **Important:** Search and replace methods will raise a `RKLICURegexException` (p. 103) if the *`replacementString`* contains `$n` capture references where *n* is greater than the number of capture groups in the regular expression.

In this example, the regular expression `\b(\w+)\b` has a single capture group, which is created with the use of `()` parenthesis. The text that was matched inside the parenthesis is available for use in the replacement text by using `$n`, where *n* is the parenthesized capture group you would like to use. Additional capture groups are numbered sequentially in the order that they appear from left to right. Capture group `0` (zero) is also available and is equivalent to all the text that the regular expression matched.

Mutable strings can be manipulated directly:

```
NSString        *regexString       = @"\\b(\\w+)\\b";
NSString        *replaceWithString = @"{$1}";
NSUInteger       replacedCount     = 0UL;

NSMutableString *mutableString = [NSMutableString stringWithString:@"This is neat."];

replacedCount = [mutableString replaceOccurrencesOfRegex:regexString
                                              withString:replaceWithString];

NSLog(@"count: %lu string: '%@'", (u_long)replacedCount, mutableString);
// 2008-07-01 21:25:43.433 test[69689:813] count: 3 string: '{This} {is} {neat}.'
```

Search and Replace using Blocks

RegexKit*Lite* 4.0 adds support for performing the same search and replacement on strings, except now the contents of the replacement string are created by the Block that is passed as the argument. For each match that is found in the string, the Block argument is called and passed the details of the match which includes a C array of `NSString` objects, one for each capture, along with a C array of `NSRange` structures with the range information for the current match.

The text that was matched will be replaced with the `NSString` object that the Block is required to return. This allows you complete control over the contents of the replaced text, such as doing complex transformations of the matched text, which is much more flexible and powerful than the simple, fixed replacement functionality provided by `stringByReplacingOccurrencesOfRegex:withString:` (p. 93).

The example below is essentially the same as the previous search and replace examples, except this example uses the `capitalizedString` method to capitalize the matched result, which is then used in the string that is returned as the replacement text. Note that the first letter in each word in `replacedString` is now capitalized.

```
NSString *searchString      = @"This is neat.";
NSString *regexString       = @"\\b(\\w+)\\b";
NSString *replacedString    = NULL;

replacedString = [searchString stringByReplacingOccurrencesOfRegex:regexString
                                                          usingBlock:
                  ^NSString *(NSInteger captureCount,
                              NSString * const capturedStrings[captureCount],
                              const NSRange capturedRanges[captureCount],
                              volatile BOOL * const stop) {
                    return([NSString stringWithFormat:@"{%@}",
                            [capturedStrings[1] capitalizedString]]);
                  }];

// 2010-04-14 21:00:42.726 test[35053:a0f] replaced string: '{This} {Is} {Neat}.'
```

See Also

- `replaceOccurrencesOfRegex:usingBlock:` (p. 85)
- `replaceOccurrencesOfRegex:withString:` (p. 87)
- `stringByReplacingOccurrencesOfRegex:usingBlock:` (p. 91)
- `stringByReplacingOccurrencesOfRegex:withString:` (p. 93)

ICU Regular Expression Syntax (p. 25)
ICU Replacement Text Syntax (p. 33)

Splitting Strings

Strings can be split with a regular expression using the `componentsSeparatedByRegex:` (p. 71) methods. This functionality is nearly identical to the preexisting `NSString` method `componentsSeparatedByString:`, except instead of only being able to use a fixed string as a separator, you can use a regular expression:

```
NSString *searchString = @"This is neat.";
NSString *regexString  = @"\\s+";
NSArray  *splitArray   = NULL;

splitArray = [searchString componentsSeparatedByRegex:regexString];
```

```
// splitArray == { @"This", @"is", @"neat." }

NSLog(@"splitArray: %@", splitArray);
```

The output from `NSLog()` when run from a shell:

```
shell% ./splitArray↵
2008-07-01 20:58:39.025 splitArray[69618:813] splitArray: (
    This,
    is,
    "neat."
)
shell% 
```

Unfortunately our example string `@"This␣is␣neat."` doesn't allow us to show off the power of regular expressions. As you can probably imagine, splitting the string with the regular expression `\s+` allows for one or more *white space* (p. 26) characters to be matched. This can be much more flexible than just a fixed string of `@"␣"`, which will split on a single space only. If our example string contained extra spaces, say `@"This␣␣␣is␣␣␣␣␣neat."`, the result would have been the same.

See Also

- `componentsSeparatedByRegex:` (p. 71)
- `enumerateStringsSeparatedByRegex:usingBlock:` (p. 78)

ICU Regular Expression Syntax (p. 25)

Creating an Array of Every Match

RegexKit*Lite* 3.0 adds several methods that conveniently perform a number of individual RegexKit*Lite* operations and aggregate the results in to a `NSArray`. Since the result is a `NSArray`, the standard Cocoa collection enumeration patterns can be used, such as `NSEnumerator` and Objective-C 2.0's `for…in` feature. One of the most common tasks is to extract all of the matches of a regular expression from a string. `componentsMatchedByRegex:` returns the entire text matched by a regular expression even if the regular expression contains additional capture groups, effectively capture group `0`. For example:

```
NSString *searchString = @"$10.23, $1024.42, $3099";
NSString *regexString  = @"\\$((\\d+)(?:\\.(\\d+)|\\.?))";
NSArray  *matchArray   = NULL;

matchArray = [searchString componentsMatchedByRegex:regexString];

// matchArray == { @"$10.23", @"$1024.42", @"$3099" };

NSLog(@"matchArray: %@", matchArray);
```

The output from `NSLog()` when run from a shell:

```
shell% ./matchArray↵
2009-05-06 03:20:03.546 matchArray[69939:10b] matchArray: (
    "$10.23",
    "$1024.42",
    "$3099"
)
shell% 
```

As the example above demonstrates, `componentsMatchedByRegex:` returns the entire text that the regular expression matched even though the regular expression contains capture groups. `arrayOfCaptureComponentsMatchedByRegex:` can be used if you need to get the text that the individual capture groups matched as well:

```
NSString *searchString  = @"$10.23, $1024.42, $3099";
NSString *regexString   = @"\\$((\\d+)(?:\\.(\\d+)|\\.?))";
NSArray  *capturesArray = NULL;

capturesArray = [searchString arrayOfCaptureComponentsMatchedByRegex:regexString];

/* capturesArray ==
[NSArray arrayWithObjects:
  [NSArray arrayWithObjects:  @"$10.23",   @"10.23",   @"10", @"23", NULL],
  [NSArray arrayWithObjects:@"$1024.42", @"1024.42", @"1024", @"42", NULL],
  [NSArray arrayWithObjects:   @"$3099",    @"3099", @"3099",   @"", NULL],
  NULL];
*/

NSLog(@"capturesArray: %@", capturesArray);
```

The output from `NSLog()` when run from a shell:

```
shell% ./capturesArray↵
2009-05-06 03:25:46.852 capturesArray[69981:10b] capturesArray: (
        (
        "$10.23",
        "10.23",
        10,
        23
    ),
        (
        "$1024.42",
        "1024.42",
        1024,
        42
    ),
        (
        "$3099",
        3099,
        3099,
        ""
    )
)
shell% ▊
```

See Also

- `arrayOfCaptureComponentsMatchedByRegex:` (p. 61)
- `captureComponentsMatchedByRegex:` (p. 66)
- `componentsMatchedByRegex:` (p. 69)

Enumerating Matches (p. 19)
Collections Programming Topics for Cocoa - Enumerators: Traversing a Collection's Elements

Enumerating Matches

The RegexKit*Lite* `componentsMatchedByRegex:` method enables you to quickly create a `NSArray` containing all the matches of a regular expression in a string. To enumerate the contents of the `NSArray`, you can send the array an `objectEnumerator` message.

An example using `componentsSeparatedByRegex:` (p. 71) and a `NSEnumerator`:

```
#import <Foundation/Foundation.h>
#import "RegexKitLite.h"

int main(int argc, char *argv[]) {
  NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

  NSString     *searchString    = @"one\ntwo\n\nfour\n";
  NSArray      *matchArray      = NULL;
  NSEnumerator *matchEnumerator = NULL;
  NSString     *regexString     = @"(?m)^.*$";
  NSString     *matchedString   = NULL;
  NSUInteger    line            = 0UL;

  NSLog(@"searchString: '%@'", searchString);
  NSLog(@"regexString : '%@'", regexString);

  matchArray      = [searchString componentsMatchedByRegex:regexString];
  matchEnumerator = [matchArray objectEnumerator];

  while((matchedString = [matchEnumerator nextObject]) != NULL) {
    NSLog(@"%lu: %lu '%@'", ++line, [matchedString length], matchedString);
  }

  [pool release];
  return(0);
}
```

The following shell transcript demonstrates compiling the example and executing it. Line number three clearly demonstrates that matches of zero length are possible. Without the additional logic in `nextObject` to handle this special case, the enumerator would never advance past the match.

> **Note:** In the shell transcript below, the `NSLog()` line that prints `searchString` has been annotated with the '⏎' glyph to help visually identify the corresponding `\n` *new-line* characters in `searchString`.

```
shell% gcc -g -o match match RegexKitLite.m -framework Foundation -licucore↵
shell% ./match↵
2008-03-21 15:56:17.469 match[44050:807] searchString: 'one⏎
two⏎
⏎
four⏎
'
2008-03-21 15:56:17.520 match[44050:807] regexString : '(?m)^.*$'
2008-03-21 15:56:17.575 match[44050:807] 1: 3 'one'
```

```
2008-03-21 15:56:17.580 match[44050:807] 2: 3 'two'
2008-03-21 15:56:17.584 match[44050:807] 3: 0 ''
2008-03-21 15:56:17.590 match[44050:807] 4: 4 'four'
shell% 
```

Enumerating Matches with Objective-C 2.0

You can enumerate all the matches of a regular expression in a string using Objective-C 2.0's `for…in` feature. Compared to using a `NSEnumerator`, using `for…in` not only takes fewer lines of code to accomplish the same thing, it is usually faster as well.

An example using the Objective-C 2.0 `for…in` feature:

```
#import <Foundation/Foundation.h>
#import "RegexKitLite.h"

int main(int argc, char *argv[]) {
  NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

  NSString   *searchString = @"one\ntwo\n\nfour\n";
  NSString   *regexString  = @"(?m)^.*$";
  NSUInteger  line         = 0UL;

  NSLog(@"searchString: '%@'", searchString);
  NSLog(@"regexString : '%@'", regexString);

  for(NSString *matchedString in [searchString componentsMatchedByRegex:regexString]) {
    NSLog(@"%lu: %lu '%@'", ++line, [matchedString length], matchedString);
  }

  [pool release];
  return(0);
}
```

Note: The output of the preceding example is identical to the `NSEnumerator` shell output (p. 19).

Enumerating Matches using Blocks

A third way to enumerate all the matches of a regular expression in a string is to use one of the Blocks-based enumeration methods.

An example using `enumerateStringsMatchedByRegex:usingBlock:` (p. 76):

```
#import <Foundation/Foundation.h>
#import "RegexKitLite.h"

int main(int argc, char *argv[]) {
  NSAutoreleasePool  *pool = [[NSAutoreleasePool alloc] init];

          NSString   *searchString = @"one\ntwo\n\nfour\n";
          NSString   *regexString  = @"(?m)^.*$";
```

```
  __block NSUInteger  line         = 0UL;

  NSLog(@"searchString: '%@'", searchString);
  NSLog(@"regexString : '%@'", regexString);

  [searchString enumerateStringsMatchedByRegex:regexString
                                    usingBlock:
   ^(NSInteger captureCount,
     NSString * const capturedStrings[captureCount],
     const NSRange capturedRanges[captureCount],
     volatile BOOL * const stop) {
     NSLog(@"%lu: %lu '%@'", ++line, [capturedStrings[0] length], capturedStrings[0]);
   }];

  [pool release];
  return(0);
}
```

Note: The output of the preceding example is identical to the `NSEnumerator` shell output (p. 19).

See Also

- `- componentsMatchedByRegex:` (p. 69)
- `- enumerateStringsMatchedByRegex:usingBlock:` (p. 76)

Regular Expression Enumeration Options (p. 97)
RegexKitLite NSString Additions Reference - Block-based Enumeration Methods (p. 51)
NSArray Class Reference
NSEnumerator Class Reference
Blocks Programming Topics
Collections Programming Topics for Cocoa - Enumerators: Traversing a Collection's Elements
The Objective-C 2.0 Programming Language - Fast Enumeration

DTrace

Important: DTrace support is not enabled by default. To enable DTrace support, use the `RKL_DTRACE` preprocessor flag: `-DRKL_DTRACE`

RegexKit*Lite* has two DTrace probe points that provide information about its internal caches:

- `RegexKitLite:::compiledRegexCache(unsigned long eventID, const char *regexUTF8, int options, int captures, int hitMiss, int icuStatusCode, const char *icuErrorMessage, double *hitRate);`
- `RegexKitLite:::utf16ConversionCache(unsigned long eventID, unsigned int lookupResultFlags, double *hitRate, const void *string, unsigned long NSRange_location, unsigned long NSRange_length, long length);`

Each of the probe points supply information via a number of arguments that are accessible through the DTrace variables `arg0` ... `argn`.

The first argument, *`eventID`* via `arg0`, is a unique event ID that is incremented each time the RegexKit*Lite* mutex lock is acquired. All the probes that fire while the mutex is held will share the same event ID. This

can help if you are trying to correlate multiple events across different CPUs.

> **Important:** Most uses of the `dtrace` command require superuser privileges. The examples given here use `sudo` to execute `dtrace` as the `root` user.

The following is available in `examples/compiledRegexCache.d` and demonstrates the use of all the arguments available via the `RegexKitLite:::compiledRegexCache` probe point:

```
#!/usr/sbin/dtrace -s

RegexKitLite*:::compiledRegexCache {
  this->eventID         = (unsigned long)arg0;
  this->regexUTF8       = copyinstr(arg1);
  this->options         = (unsigned int)arg2;
  this->captures        = (int)arg3;
  this->hitMiss         = (int)arg4;
  this->icuStatusCode   = (int)arg5;
  this->icuErrorMessage = (arg6 == 0) ? "" : copyinstr(arg6);
  this->hitRate         = (double *)copyin(arg7, sizeof(double));

  printf("%5d: %-60.60s Opt: %#8.8x Cap: %2d Hit: %2d Rate: %6.2f%% code: %5d msg: %s\n",
         this->eventID,
         this->regexUTF8,
         this->options,
         this->captures,
         this->hitMiss,
         *this->hitRate,
         this->icuStatusCode,
         this->icuErrorMessage);
}
```

Below is an example of the output from `compiledRegexCache.d`:

> **Note:** The output below has been significantly trimmed and altered to fit on a printed page.

```
shell% sudo dtrace -Z -q -s compiledRegexCache.d↵
 131: (\w+   Opt: 0x0 Cap: -1 Hit: -1 Rate:  63.36% code: 66310 msg: U_REGEX_MISMATCHED…
 164: \b\s*  Opt: 0x0 Cap:  0 Hit:  0 Rate:  60.98% code:     0 msg:
shell% ▉
```

An example that prints the number of times that a compiled regular expression was not in the cache per second:

```
shell% sudo dtrace -Z -q \
-n 'RegexKitLite*:::compiledRegexCache /arg4==0/ { @miss[pid, execname] = count(); }' \
-n 'tick-1sec { printa("%-8d %-40s %@d/sec\n", @miss); trunc(@miss); }'↵
67003    RegexKitLite_tests                       16/sec
67008    RegexKitLite_tests                       50/sec
^C

shell% ▉
```

See Also

RegexKitLite:::compiledRegexCache (p. 58)
Solaris Dynamic Tracing Guide (as .PDF)

The following is available in `examples/utf16ConversionCache.d` and demonstrates the use of all the arguments available via the `RegexKitLite:::utf16ConversionCache` probe point.

```
#!/usr/sbin/dtrace -s

enum {
  RKLCacheHitLookupFlag           = 1 << 0,
  RKLConversionRequiredLookupFlag = 1 << 1,
  RKLSetTextLookupFlag            = 1 << 2,
  RKLDynamicBufferLookupFlag      = 1 << 3,
  RKLErrorLookupFlag              = 1 << 4
};

RegexKitLite*:::utf16ConversionCache {
  this->eventID           = (unsigned long)arg0;
  this->lookupResultFlags = (unsigned int)arg1;
  this->hitRate           = (double *)copyin(arg2, sizeof(double));
  this->stringPtr         = (void *)arg3;
  this->NSRange_location  = (unsigned long)arg4;
  this->NSRange_length    = (unsigned long)arg5;
  this->length            = (long)arg6;

  printf("%5lu: flags: %#8.8x {Hit: %d Conv: %d SetText: %d Dyn: %d Error: %d}↩
        ↪ rate: %6.2f%% string: %#8.8p NSRange {%6lu, %6lu} length: %ld\n",
         this->eventID,
         this->lookupResultFlags,
         (this->lookupResultFlags & RKLCacheHitLookupFlag)           != 0,
         (this->lookupResultFlags & RKLConversionRequiredLookupFlag) != 0,
         (this->lookupResultFlags & RKLSetTextLookupFlag)            != 0,
         (this->lookupResultFlags & RKLDynamicBufferLookupFlag)      != 0,
         (this->lookupResultFlags & RKLErrorLookupFlag)              != 0,
         *this->hitRate,
         this->stringPtr,
         this->NSRange_location,
         this->NSRange_length,
         this->length);
}
```

Below is an example of the output from `utf16ConversionCache.d`:

> **Note:** The output below has been significantly trimmed and altered to fit on a printed page.

```
shell% sudo dtrace -Z -q -s utf16ConversionCache.d↵
   85: flags: 0x0 {Hit: 0 Conv: 0 Set: 0 Dyn: 0 Err: 0} rate: 59.18% R { 0, 18} L: 18
   86: flags: 0x4 {Hit: 0 Conv: 0 Set: 1 Dyn: 0 Err: 0} rate: 59.18% R { 0, 18} L: 18
   87: flags: 0x6 {Hit: 0 Conv: 1 Set: 1 Dyn: 0 Err: 0} rate: 58.00% R { 1, 37} L: 39
   88: flags: 0x3 {Hit: 1 Conv: 1 Set: 0 Dyn: 0 Err: 0} rate: 58.82% R { 1, 37} L: 39
  109: flags: 0x6 {Hit: 0 Conv: 1 Set: 1 Dyn: 0 Err: 0} rate: 53.62% R { 0, 56} L: 56
```

```
  110: flags: 0x6 {Hit: 0 Conv: 1 Set: 1 Dyn: 0 Err: 0} rate: 52.86% R { 0, 83} L: 97
  111: flags: 0x7 {Hit: 1 Conv: 1 Set: 1 Dyn: 0 Err: 0} rate: 53.52% R {46, 83} L: 97
shell% 
```

An example that prints the number of times that a string required a conversion to `UTF-16` and was not in the cache per second:

```
shell% sudo dtrace -Z -q \
       -n 'RegexKitLite*:::utf16ConversionCache /(arg1 & 0x3) == 0x2/ \
           { @miss[pid, execname] = count(); }' \
       -n 'tick-1sec { printa("%-8d %-40s %@d/sec\n", @miss); trunc(@miss); }'↵
67020    RegexKitLite_tests                       73/sec
67037    RegexKitLite_tests                       64/sec
^C

shell% 
```

See Also

`RegexKitLite:::utf16ConversionCache` (p. 59)
RegexKitLite:::utf16ConversionCache arg1 Flags (p. 102)
Solaris Dynamic Tracing Guide (as .PDF)

ICU Syntax

In this section:

ICU Regular Expression Syntax

For your convenience, the regular expression syntax from the ICU documentation is included below. When in doubt, you should refer to the official *ICU User Guide - Regular Expressions* documentation page.

See Also
ICU User Guide - Regular Expressions
Unicode Technical Standard #18 - Unicode Regular Expressions

Character	Description
`\a`	Match a `BELL`, `\u0007`
`\A`	Match at the beginning of the input. Differs from `^` in that `\A` will not match after a *`new-line`* within the input.
`\b`, outside of a `[`*Set*`]`	Match if the current position is a word boundary. Boundaries occur at the transitions between *`word`* `\w` and *`non-word`* `\W` characters, with combining marks ignored. **See also:** `RKLUnicodeWordBoundaries` (p. 96)
`\b`, within a `[`*Set*`]`	Match a `BACKSPACE`, `\u0008`.
`\B`	Match if the current position is not a *`word`* boundary.
`\c`*x*	Match a *Control-x* character.
`\d`	Match any character with the *Unicode General Category* of `\p{Nd}` (*Number, Decimal Digit*).
`\D`	Match any character that is not a *`decimal digit`*.
`\e`	Match an `ESCAPE`, `\u001B`.
`\E`	Terminates a `\Q`...`\E` quoted sequence.
`\f`	Match a `FORM FEED`, `\u000C`.
`\G`	Match if the current position is at the end of the previous match.
`\n`	Match a `LINE FEED`, `\u000A`.

Character	Description
\N{*Unicode Character Name*}	Match the named *Unicode Character*.
\p{*Unicode Property Name*}	Match any character with the specified *Unicode Property*.
\P{*Unicode Property Name*}	Match any character not having the specified *Unicode Property*.
\Q	Quotes all following characters until \E.
\r	Match a CARRIAGE RETURN, \u000D.
\s	Match a *white space* character. White space is defined as [\t\n\f\r\p{Z}].
\S	Match a non-*white space* character.
\t	Match a HORIZONTAL TABULATION, \u0009.
\u*hhhh*	Match the character with the hex value *hhhh*.
\U*hhhhhhhh*	Match the character with the hex value *hhhhhhhh*. Exactly eight hex digits must be provided, even though the largest Unicode code point is \U0010ffff.
\w	Match a word character. Word characters are [\p{Ll}\p{Lu}\p{Lt}\p{Lo}\p{Nd}].
\W	Match a non-word character.
\x{*h...*}	Match the character with hex value *hhhh*. From one to six hex digits may be supplied.
\x*hh*	Match the character with two digit hex value *hh*.
\X	Match a *Grapheme Cluster*.
\Z	Match if the current position is at the end of input, but before the final line terminator, if one exists.
\z	Match if the current position is at the end of input.
n	Back Reference. Match whatever the *n*th capturing group matched. *n* must be a number ≥ 1 and ≤ total number of capture groups in the pattern.
[*pattern*]	Match any one character from the set. See *ICU Regular Expression Character Classes* (p. 28) for a full description of what may appear in the pattern.
.	Match any character.
^	Match at the beginning of a line.
$	Match at the end of a line.
\	Quotes the following character. Characters that must be quoted to be treated as literals are * ? + [( ) { } ^ $ | \ . /

Operator	Description
|	Alternation. A|B matches either A or B.
*	Match zero or more times. Match as many times as possible.
+	Match one or more times. Match as many times as possible.
?	Match zero or one times. Prefer one.
{*n*}	Match exactly *n* times.
{*n*,}	Match at least *n* times. Match as many times as possible.
{*n*,*m*}	Match between *n* and *m* times. Match as many times as possible, but not more than *m*.
*?	Match zero or more times. Match as few times as possible.
+?	Match one or more times. Match as few times as possible.
??	Match zero or one times. Prefer zero.
{*n*}?	Match exactly *n* times.
{*n*,}?	Match at least *n* times, but no more than required for an overall pattern match.
{*n*,*m*}?	Match between *n* and *m* times. Match as few times as possible, but not less than *n*.
*+	Match zero or more times. Match as many times as possible when first encountered, do not retry with fewer even if overall match fails. Possessive match.
++	Match one or more times. Possessive match.
?+	Match zero or one times. Possessive match.
{*n*}+	Match exactly *n* times. Possessive match.
{*n*,}+	Match at least *n* times. Possessive match.
{*n*,*m*}+	Match between *n* and *m* times. Possessive match.
(...)	Capturing parentheses. Range of input that matched the parenthesized subexpression is available after the match.
(?:...)	Non-capturing parentheses. Groups the included pattern, but does not provide capturing of matching text. Somewhat more efficient than capturing parentheses.
(?>...)	Atomic-match parentheses. First match of the parenthesized subexpression is the only one tried; if it does not lead to an overall pattern match, back up the search for a match to a position before the (?> .
(?#...)	Free-format comment (?#*comment*).
(?=...)	Look-ahead assertion. True if the parenthesized pattern matches at the current input position, but does not advance the input position.
(?!...)	Negative look-ahead assertion. True if the parenthesized pattern does not match at the current input position. Does not advance the input position.
(?<=...)	Look-behind assertion. True if the parenthesized pattern matches text preceding the current input position, with the last character of the match being the input character just before the current position. Does not alter the input position. The length of possible strings matched by the look-behind pattern must not be unbounded (no * or + operators).
(?<!...)	Negative Look-behind assertion. True if the parenthesized pattern does not match text preceding the current input position, with the last character of the match being the input character just before the current position. Does not alter the input position. The length of possible strings matched by the look-behind pattern must not be unbounded (no * or + operators).

Operator	Description
`(?ismwx-ismwx:...)`	Flag settings. Evaluate the parenthesized expression with the specified flags *enabled* or *-disabled*.
`(?ismwx-ismwx)`	Flag settings. Change the flag settings. Changes apply to the portion of the pattern following the setting. For example, `(?i)` changes to a case insensitive match. **See also:** *Regular Expression Options* (p. 95)

See Also
ICU User Guide - Regular Expressions
Regular Expression Options (p. 95)

ICU Regular Expression Character Classes

The following was originally from *ICU User Guide - UnicodeSet*, but has been adapted to fit the needs of this documentation. Specifically, the ICU `UnicodeSet` documentation describes an ICU C++ object–`UnicodeSet`. The term `UnicodeSet` was effectively replaced with **Character Class**, which is more appropriate in the context of regular expressions. As always, you should refer to the original, official documentation when in doubt.

See Also
ICU User Guide - UnicodeSet
UTS #18 Unicode Regular Expressions - Subtraction and Intersection
UTS #18 Unicode Regular Expressions - Properties

Overview

A character class is a regular expression pattern that represents a set of Unicode characters or character strings. The following table contains some example character class patterns:

Pattern	Description
`[a-z]`	The lower case letters `a` through `z`
`[abc123]`	The six characters `a`, `b`, `c`, `1`, `2`, and `3`
`[\p{Letter}]`	All characters with the Unicode `General Category` of `Letter`.

String Values

In addition to being a set of Unicode code point characters, a character class may also contain string values. Conceptually, a character class is always a set of strings, not a set of characters. Historically, regular expressions have treated `[...]` character classes as being composed of single characters only, which is equivalent to a string that contains only a single character.

Character Class Patterns

Patterns are a series of characters bounded by square brackets that contain lists of characters and Unicode property sets. Lists are a sequence of characters that may have ranges indicated by a `-` between two characters, as in `a-z`. The sequence specifies the range of all characters from the left to the right, in Unicode order. For example, `[a␣c␣d-f␣m]` is equivalent to `[a␣c␣d␣e␣f␣m]`. Whitespace can be freely used for clarity as `[a␣c␣d-f␣m]` means the same as `[acd-fm]`.

Unicode property sets are specified by a Unicode property, such as `[:Letter:]`. ICU version 2.0 supports `General Category`, `Script`, and `Numeric Value` properties (ICU will support additional properties in the future). For a list of the property names, see the end of this section. The syntax for specifying the property names is an extension of either POSIX or Perl syntax with the addition of `=value`. For example, you can match letters by using the POSIX syntax `[:Letter:]`, or by using the Perl syntax `\p{Letter}`. The type can be omitted for the `Category` and `Script` properties, but is required for other properties.

The following table lists the standard and negated forms for specifying Unicode properties in both POSIX or Perl syntax. The negated form specifies a character class that includes everything but the specified property. For example, `[:^Letter:]` matches all characters that are not `[:Letter:]`.

Syntax Style	Standard	Negated
POSIX	`[:`*type*`=`*value*`:]`	`[:^`*type*`=`*value*`:]`
Perl	`\p{`*type*`=`*value*`}`	`\P{`*type*`=`*value*`}`

See Also
UTS #18 Unicode Regular Expressions - Properties

Character classes can then be modified using standard set operations... Union, Inverse, Difference, and Intersection.

- To union two sets, simply concatenate them. For example, `[[:letter:] [:number:]]`
- To intersect two sets, use the `&` operator. For example, `[[:letter:] & [a-z]]`
- To take the set-difference of two sets, use the `-` operator. For example, `[[:letter:] - [a-z]]`
- To invert a set, place a `^` immediately after the opening `[`. For example, `[^a-z]`. In any other location, the `^` does not have a special meaning.

The binary operators `&` and `-` have equal precedence and bind left-to-right. Thus `[[:letter:]-[a-z]-[\u0100-\u01FF]]` is equivalent to `[[[:letter:]-[a-z]]-[\u0100-\u01FF]]`. Another example is the set `[[ace][bdf] - [abc][def]]` is **not** the empty set, but instead the set `[def]`. This only really matters for the difference operation, as the intersection operation is commutative.

Another caveat with the `&` and `-` operators is that they operate between **sets**. That is, they must be immediately preceded and immediately followed by a set. For example, the pattern `[[:Lu:]-A]` is illegal, since it is interpreted as the set `[:Lu:]` followed by the incomplete range `-A`. To specify the set of uppercase letters except for `A`, enclose the `A` in a set: `[[:Lu:]-[A]]`.

Pattern	Description
`[a]`	The set containing `a`.
`[a-z]`	The set containing `a` through `z` and all letters in between, in Unicode order.
`[^a-z]`	The set containing all characters but `a` through `z`, that is, `U+0000` through `a-1` and `z+1` through `U+FFFF`.
`[[`*pat1*`][`*pat2*`]]`	The union of sets specified by *pat1* and *pat2*.
`[[`*pat1*`]&[`*pat2*`]]`	The intersection of sets specified by *pat1* and *pat2*.
`[[`*pat1*`]-[`*pat2*`]]`	The asymmetric difference of sets specified by *pat1* and *pat2*.
`[:Lu:]`	The set of characters belonging to the given Unicode category. In this case, Unicode uppercase letters. The long form for this is `[:UppercaseLetter:]`.
`[:L:]`	The set of characters belonging to all Unicode categories starting with `L`, that is, `[[:Lu:][:Ll:][:Lt:][:Lm:][:Lo:]]`. The long form for this is `[:Letter:]`.

See Also
UTS #18 Unicode Regular Expressions - Subtraction and Intersection

String Values in Character Classes

String values are enclosed in {*curly brackets*}. For example:

Pattern	Description
`[abc{def}]`	A set containing four members, the single characters `a`, `b`, and `c` and the string `def`.
`[{abc}{def}]`	A set containing two members, the string `abc` and the string `def`.
`[{a}{b}{c}][abc]`	These two sets are equivalent. Each contains three items, the three individual characters `a`, `b`, and `c`. A {*string*} containing a single character is equivalent to that same character specified in any other way.

Character Quoting and Escaping in ICU Character Class Patterns

Single Quote

Two single quotes represent a single quote, either inside or outside single quotes. Text within single quotes is not interpreted in any way, except for two adjacent single quotes. It is taken as literal text– special characters become non-special. These quoting conventions for ICU character classes differ

Backslash Escapes

Outside of single quotes, certain backslashed characters have special meaning:

Pattern	Description
`\u`*hhhh*	Exactly 4 hex digits; *h* in `[0-9A-Fa-f]`
`\U`*hhhhhhhh*	Exactly 8 hex digits
`\x`*hh*	1-2 hex digits
`\`*ooo*	1-3 octal digits; *o* in `[0-7]`
`\a`	U+0007 BELL
`\b`	U+0008 BACKSPACE
`\t`	U+0009 HORIZONTAL TAB
`\n`	U+000A LINE FEED
`\v`	U+000B VERTICAL TAB
`\f`	U+000C FORM FEED
`\r`	U+000D CARRIAGE RETURN
`\\`	U+005C BACKSLASH

Anything else following a backslash is mapped to itself, except in an environment where it is defined to have some special meaning. For example, `\p{Lu}` is the set of uppercase letters. Any character formed as the result of a backslash escape loses any special meaning and is treated as a literal. In particular, note that `\u` and `\U` escapes create literal characters.

Whitespace

Whitespace, as defined by the ICU API, is ignored unless it is quoted or backslashed.

Property Values

The following property value styles are recognized:

Style	Description
Short	Omits the `=type` argument. Used to prevent ambiguity and only allowed with the `Category` and `Script` properties.
Medium	Uses an abbreviated `type` and `value`.
Long	Uses a full `type` and `value`.

If the `type` or `value` is omitted, then the `=` equals sign is also omitted. The short style is only used for `Category` and `Script` properties because these properties are very common and their omission is unambiguous.

In actual practice, you can mix `type` names and `values` that are omitted, abbreviated, or full. For example, if `Category=Unassigned` you could use what is in the table explicitly, `\p{gc=Unassigned}`, `\p{Category=Cn}`, or `\p{Unassigned}`.

When these are processed, case and whitespace are ignored so you may use them for clarity, if desired. For example, `\p{Category = Uppercase Letter}` or `\p{Category = uppercase letter}`.

For a list of properties supported by ICU, see *ICU User Guide - Unicode Properties*.

See Also
ICU User Guide - Unicode Properties
UTS #18 Unicode Regular Expressions - Properties

Unicode Properties

The following tables list some of the commonly used Unicode Properties, which can be matched in a regular expression with `\p{`*Property*`}`. The tables were created from the Unicode 5.2 Unicode Character Database, which is the version used by ICU that ships with Mac OS X 10.6.

Category	
N	Number
Nd	DecimalNumber
Nl	LetterNumber
No	OtherNumber

Category	
M	Mark
Mn	NonspacingMark
Mc	SpacingMark
Me	EnclosingMark

Category	
Z	Separator
Zs	SpaceSeparator
Zl	LineSeparator
Zp	ParagraphSeparator

L	Letter
LC	CasedLetter
Lu	UppercaseLetter
Ll	LowercaseLetter
Lt	TitlecaseLetter
Lm	ModifierLetter
Lo	OtherLetter

P	Punctuation
Pc	ConnectorPunctuation
Pd	DashPunctuation
Ps	OpenPunctuation
Pe	ClosePunctuation
Pi	InitialPunctuation
Pf	FinalPunctuation
Po	OtherPunctuation

C	Other
Cc	Control
Cf	Format
Cs	Surrogate
Co	PrivateUse
Cn	Unassigned

Category	
S	Symbol
Sm	MathSymbol
Sc	CurrencySymbol
Sk	ModifierSymbol
So	OtherSymbol

Extended Property Class	
ASCII_Hex_Digit	Alphabetic
Bidi_Control	Dash
Default_Ignorable_Code_Point	Deprecated
Diacritic	Extender
Grapheme_Base	Grapheme_Extend
Grapheme_Link	Hex_Digit
Hyphen	IDS_Binary_Operator
IDS_Trinary_Operator	ID_Continue
ID_Start	Ideographic
Join_Control	Logical_Order_Exception
Lowercase	Math
Noncharacter_Code_Point	Other_Alphabetic
Other_Default_Ignorable_Code_Point	Other_Grapheme_Extend
Other_ID_Continue	Other_ID_Start
Other_Lowercase	Other_Math
Other_Uppercase	Pattern_Syntax
Pattern_White_Space	Quotation_Mark
Radical	STerm
Soft_Dotted	Terminal_Punctuation
Unified_Ideograph	Uppercase
Variation_Selector	White_Space
XID_Continue	XID_Start

Script					
Arabic	Armenian	Balinese	Bengali	Bopomofo	Braille
Buginese	Buhid	Canadian_Aboriginal	Carian	Cham	Cherokee
Common	Coptic	Cuneiform	Cypriot	Cyrillic	Deseret
Devanagari	Ethiopic	Georgian	Glagolitic	Gothic	Greek
Gujarati	Gurmukhi	Han	Hangul	Hanunoo	Hebrew
Hiragana	Inherited	Kannada	Katakana	Kayah_Li	Kharoshthi
Khmer	Lao	Latin	Lepcha	Limbu	Linear_B
Lycian	Lydian	Malayalam	Mongolian	Myanmar	New_Tai_Lue
Nko	Ogham	Ol_Chiki	Old_Italic	Old_Persian	Oriya
Osmanya	Phags_Pa	Phoenician	Rejang	Runic	Saurashtra
Shavian	Sinhala	Sundanese	Syloti_Nagri	Syriac	Tagalog
Tagbanwa	Tai_Le	Tamil	Telugu	Thaana	Thai
Tibetan	Tifinagh	Ugaritic	Unknown	Vai	Yi

Unicode Character Database

Unicode properties are defined in the Unicode Character Database, or **UCD**. From time to time the UCD is revised and updated. The properties available, and the definition of the characters they match, depend on the UCD that ICU was built with.

> **Note:** In general, the ICU and UCD versions change with each major operating system release.

See Also

UTS #18 Unicode Regular Expressions - Properties
UTS #18 Unicode Regular Expressions - Compatibility Properties
Unicode Character Database
The Unicode Standard - Unicode 5.2
Versions of the Unicode Standard

ICU Replacement Text Syntax

Character	Description
`$`*n*	The text of capture group *n* will be substituted for `$`*n*. *n* must be ≥ `0` and not greater than the number of capture groups. A `$` not followed by a digit has no special meaning, and will appear in the substitution text as itself, a `$`. **Important:** Methods will raise a `RKLICURegexException` (p. 103) if *n* is greater than the number of capture groups in the regular expression.
`\`	Treat the character following the backslash as a literal, suppressing any special meaning. Backslash escaping in substitution text is only required for `$` and `\`, but may proceed any character. The backslash itself will not be copied to the substitution text.

See Also

ICU User Guide - Replacement Text
- `replaceOccurrencesOfRegex:withString:options:range:error:` (p. 88)
- `stringByReplacingOccurrencesOfRegex:withString:options:range:error:` (p. 94)

RegexKit*Lite* Cookbook

> *Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems.*
> — Jamie Zawinski

This section contains a collection of regular expressions and example code demonstrating how RegexKit*Lite* makes some common programming choirs easier. RegexKit*Lite* makes it easy to match part of a string and extract just that part, or even create an entirely new string using just a few pieces of the original string. A great example of this is a string that contains a URL and you need to extract just a part of it, perhaps the host or maybe just the port used. This example demonstrates how easy it is to extract the port used from a URL, which is then converted in to a `NSInteger` value:

```
searchString = @"http://www.example.com:8080/index.html";
regexString  = @"\\bhttps?://[a-zA-Z0-9\\-.]+(?::(\\d+))?"
               @"(?:(?:/[a-zA-Z0-9\\-._?,'+\\&%$=~*!():@\\\\]*)+)?";

NSInteger portInteger = [[searchString stringByMatching:regexString
                                                capture:1L] integerValue];

NSLog(@"portInteger: '%ld'", (long)portInteger);
// 2008-10-15 08:52:52.500 host_port[8021:807] portInteger: '8080'
```

Pattern Matching Recipes

Inside you'll find more examples like this that you can use as the starting point for your own regular expression pattern matching solution. Keep in mind that these are meant to be examples to help get you started and not necessarily the ideal solution for every need. Trade-offs are usually made when creating a regular expression, matching an email address is a perfect example of this. A regular expression that precisely matches the formal definition of email address is both complicated and usually unnecessary. Knowing which trade-offs are acceptable requires that you understand what it is you're trying to match, the data that you're searching through, and the requirements and uses of the matched results. It won't take long until you gain an appreciation for Jamie Zawinski's infamous quote.

See Also

O'Reilly - Mastering Regular Expressions, 3rd edition by Jeffrey Friedl
O'Reilly - Regular Expressions Cookbook by Jan Goyvaerts and Steven Levithan
RegExLib.com - Regular Expression Library
ICU Userguide - Regular Expressions
Regular-Expressions.info - Regex Tutorial, Examples, and Reference
Wikipedia - Regular Expression

Numbers

Description	Regex	Examples
Integer	`[+\-]?[0-9]+`	`123` `-42` `+23`
Hex Number	`0[xX][0-9a-fA-F]+`	`0x0` `0xdeadbeef` `0xF3`
Floating Point	`[+\-]?(?:[0-9]*\.[0-9]+|[0-9]+\.)`	`123.` `.123` `+.42`
Floating Point with Exponent	`[+\-]?(?:[0-9]*\.[0-9]+|[0-9]+\.)(?:[eE][+\-]?[0-9]+)?`	`123.` `.123` `10.0E13` `1.23e-7`
Comma Separated Number	`[0-9]{1,3}(?:,[0-9]{3})*`	`42` `1,234` `1,234,567`
Comma Separated Number	`[0-9]{1,3}(?:,[0-9]{3})*(?:\.[0-9]+)?`	`42` `1,234` `1,234,567.89`

Extracting and Converting Numbers

`NSString` includes several methods for converting the contents of the string in to a numeric value in the various C primitive types. The following demonstrates the matching of an `int` and `double` in a `NSString`, and then converting the matched string in to its base type.

```
NSString *searchString = @"The int 5542 to convert";
NSString *regexString  = @"([+\\-]?[0-9]+)";
int       matchedInt   = [[searchString stringByMatching:regexString
                                                 capture:1L] intValue];
```

The variable `matchedInt` now contains the value of `5542`.

```
NSString *searchString  = @"The double 4321.9876 to convert";
NSString *regexString   = @"([+\\-]?(?:[0-9]*\\.[0-9]+|[0-9]+\\.))";
double    matchedDouble = [[searchString stringByMatching:regexString
                                                  capture:1L] doubleValue];
```

The variable `matchedDouble` now contains the value of `4321.9876`. `doubleValue` can even convert numbers that are in scientific notation, which represent numbers as $n \times 10^{exp}$:

```
NSString *searchString  = @"The double 1.010489e5 to convert";
NSString *regexString   = @"([+\\-]?(?:[0-9]*\\.[0-9]+|[0-9]+\\.)"
                           @"(?:[eE][+\\-]?[0-9]+)?)";
double    matchedDouble = [[searchString stringByMatching:regexString
                                                  capture:1L] doubleValue];
```

The variable `matchedDouble` now contains the value of `101048.9`.

Extracting and Converting Hex Numbers

Converting a string that contains a hex number in to a more basic type, such as an `int`, takes a little more work. Unfortunately, Foundation does not provide an easy way to convert a hex value in a string in to a more basic type as it does with `intValue` or `doubleValue`. Thankfully the standard C library provides a set of functions for performing such a conversion. For this example we will use the `strtol()` (*string to long*) function to convert the hex value we've extracted from `searchString`. We can not pass the pointer to the

`NSString` object that contains the matched hex value since `strtol()` is part of the standard C library which can only work on pointers to C strings. We use the `UTF8String` method to get a pointer to a compatible C string of the matched hex value.

```
NSString *searchString = @"A hex value: 0x0badf00d";
NSString *regexString  = @"\\b(0[xX][0-9a-fA-F]+)\\b";

NSString *hexString    = [searchString stringByMatching:regexString capture:1L];

// Use strtol() to convert the string to a long.
long hexLong = strtol([hexString UTF8String], NULL, 16);

NSLog(@"hexLong: 0x%lx / %ld", (u_long)hexLong, hexLong);
// 2008-09-01 09:40:44.848 hex_example[30583:807] hexLong: 0xbadf00d / 195948557
```

The full set of *string to...* functions are: `strtol()`, `strtoll()`, `strtoul()`, and `strtoull()`. These convert a string value, from base 2 to base 36, in to a `long`, `long long`, `unsigned long`, and `unsigned long long` respectively.

Adding Hex Value Conversions to `NSString`

Since it seems to be a frequently asked question, and a common search engine query for RegexKit web site visitors, here is a `NSString` category addition that converts the receivers text in to a `NSInteger` value. This is the same functionality as `intValue` or `doubleValue`, except that it converts hexadecimal text values instead of decimal text values.

> **Note:** The following code can also be found in the RegexKit*Lite* distributions `examples/` directory.

The example conversion code is fairly quick since it uses *Core Foundation* directly along with the stack to hold any temporary string conversions. Any whitespace at the beginning of the string will be skipped and the hexadecimal text to be converted may be optionally prefixed with either `0x` or `0X`. Returns `0` if the receiver does not begin with a valid hexadecimal text representation. Refer to `strtol(3)` for additional conversion details.

> **Important:** If the receiver needs to be converted in to an encoding that is compatible with `strtol()`, only the first sixty characters of the receiver are converted.

```
#import <Foundation/NSString.h>

@interface NSString (HexConversion)
-(NSInteger)hexValue;
@end

#import "NSString-HexConversion.h"
#import <CoreFoundation/CFString.h>
#include <stdlib.h>

@implementation NSString (HexConversion)

-(NSInteger)hexValue
{
  CFStringRef cfSelf = (CFStringRef)self;
```

```
  UInt8 buffer[64];
  const char *cptr;

  if((cptr = CFStringGetCStringPtr(cfSelf, kCFStringEncodingMacRoman)) == NULL) {
    CFRange range     = CFRangeMake(0L, CFStringGetLength(cfSelf));
    CFIndex usedBytes = 0L;
    CFStringGetBytes(cfSelf, range, kCFStringEncodingUTF8, '?', false, buffer, 60L,
                     &usedBytes);
    buffer[usedBytes] = 0;
    cptr              = (const char *)buffer;
  }

  return((NSInteger)strtol(cptr, NULL, 16));
}

@end
```

See Also

- `strtol(3)`
- `- intValue`
- `- doubleValue`

Text Files

Description	Regex
Empty Line	`(?m:^$)`
Empty or Whitespace Only Line	`(?m-s:^\s$)`
Strip Leading Whitespace	`(?m-s:^\s*(.*?)$)`
Strip Trailing Whitespace	`(?m-s:^(.*?)\s*$)`
Strip Leading and Trailing Whitespace	`(?m-s:^\s*(.*?)\s*$)`
Quoted String, Can Span Multiple Lines, May Contain `\"`	`"(?:[^"\\]*+|\\.)*"`
Quoted String, Single Line Only, May Contain `\"`	`"(?:[^"\\\r\n]*+|\\[^\r\n])*"`
HTML Comment	`(?s:<--.*?-->)`
Perl / Shell Comment	`(?m-s:#.*$)`
C, C++, or ObjC Comment	`(?m-s://.*$)`
C, C++, or ObjC Comment and Leading Whitespace	`(?m-s:\s*//.*$)`
C, C++, or ObjC Comment	`(?s:/\*.*?\*/)`

The Newline Debacle

Unfortunately, when processing text files, there is no standard 'newline' character or character sequence. Today this most commonly surfaces when converting text between Microsoft Windows / MS-DOS and Unix / Mac OS X. The reason for the proliferation of newline standards is largely historical and goes back many decades. Below is a table of the dominant newline character sequence 'standards':

Description	Sequence	C String	Control	Common Uses
Line Feed	`\u000A`	`\n`	`^J`	Unix, Amiga, Mac OS X
Vertical Tab	`\u000B`	`\v`	`^K`	
Form Feed	`\u000C`	`\f`	`^L`	
Carriage Return	`\u000D`	`\r`	`^M`	Apple][, Mac OS ≤ 9
Next Line (NEL)	`\u0085`			IBM / EBCDIC
Line Separator	`\u2028`			Unicode
Paragraph Separator	`\u2029`			Unicode
Carriage Return + Line Feed	`\u000D\u000A`	`\r\n`	`^M^J`	MS-DOS, Windows

Ideally, one should be flexible enough to accept any of these character sequences if one has to process text files, especially if the origin of those text files is not known. Thankfully, regular expressions excel at just such a task. Below is a regular expression pattern that will match any of the above character sequences. This is also the character sequence that the metacharacter `$` matches.

Description	Regex	Notes
Any newline	`(?:\r\n|[\n\v\f\r\x85\p{Zl}\p{Zp}])`	UTS #18 recommended. Character sequence that `$` matches.

See Also

UTS #18: Unicode Regular Expressions - Line Boundaries
Wikipedia - Newline

Matching the Beginning and End of a Line

It is often necessary to work with the individual lines of a file. There are two regular expression metacharacters, `^` and `$`, that match the *beginning* and *end* of a line, respectively. However, exactly what is matched by `^` and `$` depends on whether or not the *`multi-line`* option is enabled for the regular expression, which by default is disabled. It can be enabled for the entire regular expression by passing `RKLMultiline` via the `options:` method argument, or within the regular expression using the options syntax– `(?m:…)`.

If *`multi-line`* is ***disabled***, then `^` and `$` match the beginning and end of the entire string. If there is a *`newline`* character sequence at the very end of the string, then `$` will match the character just before the *`newline`* character sequence. Any *`newline`* character sequences in the middle of the string will ***not*** be matched.

If *`multi-line`* is ***enabled***, then `^` and `$` match the beginning and end of a line, where the end of a line is the *`newline`* character sequence. The metacharacter `^` matches either the first character in the string, or the first character following a *`newline`* character sequence. The metacharacter `$` matches either the last character in the string, or the character just before a *`newline`* character sequence.

Creating a `NSArray` Containing Every Line in a String

A common text processing pattern is to process a file one line at a time. Using the recommended regular expression for matching any newline and the `componentsSeparatedByRegex:` (p. 71) method, you can easily create a `NSArray` containing every line in a file and process it one line at a time:

```
NSString *fileNameString = @"example";
NSString *regexString    = @"(?:\r\n|[\n\v\f\r\302\205\\p{Zl}\\p{Zp}])";
NSError  *error          = NULL;
```

```
NSString *fileString     = [NSString stringWithContentsOfFile:fileNameString
                                                  usedEncoding:NULL
                                                         error:&error];

if(fileString) {
  NSArray *linesArray = [fileString componentsSeparatedByRegex:regexString];

  for(NSString *lineString in linesArray) { // ObjC 2.0 for…in loop.
    // Per line processing.

  }
} else {
  NSLog(@"Error reading file '%@'", fileNameString);
  if(error) { NSLog(@"Error: %@", error); }
}
```

The `componentsSeparatedByRegex:` (p. 71) method effectively 'chops off' the matched regular expression, or in this case any newline character. In the example above, within the `for…in` loop, `lineString` will not have a newline character at the end of the string.

Parsing CSV Data

Description	Regex
Split CSV line	`,(?=(?:(?:[^"\\]*+|\\")*"(?:[^"\\]*+|\\")*")*↩ (?!(?:[^"\\]*+|\\")*"(?:[^"\\]*+|\\")*$))`

This regular expression essentially works by ensuring that there are an even number of unescaped `"` quotes following a `,` comma. This is done by using *look-head assertions*. The first *look-head assertion*, `(?=`, is a pattern that matches zero or more strings that contain two `"` characters. Then, a *negative look-head assertion* matches a single, unpaired `"` quote character remaining at the `$` end of the line. It also uses *possessive matches* in the form of `*+` for speed, which prevents the regular expression engine from backtracking excessively. It's certainly not a beginners regular expression.

The following is used as a substitute for a CSV data file in the example below.

```
NSString *csvFileString = @"RegexKitLite,1.0,\"Mar 23, 2008\",27004\n"
                          @"RegexKitLite,1.1,\"Mar 28, 2008\",28081\n"
                          @"RegexKitLite,1.2,\"Apr 01, 2008\",28765\n"
                          @"RegexKitLite,2.0,\"Jul 07, 2008\",40569\n"
                          @"RegexKitLite,2.1,\"Jul 12, 2008\",40660\n";
```

This example really highlights the power of regular expressions when it comes to processing text. It takes just 17 lines, which includes comments, to parse a CSV data file of any newline type and create a *row* by *column* of `NSArray` values of the results while correctly handling `"` quoted values, including escaped `\"` quotes.

```
NSString   *newlineRegex      = @"(?:\r\n|[\n\v\f\r\\x85\\p{Zl}\\p{Zp}])";
NSString   *splitCSVLineRegex = @",(?=(?:(?:[^\"\\\\]*+|\\\\\")*\""
                                @"(?:[^\"\\\\]*+|\\\\\")*\")*"
                                @"(?!(?:[^\"\\\\]*+|\\\\\")*\""
                                @"(?:[^\"\\\\]*+|\\\\\")*$))";

// Create a NSArray of every line in csvFileString.
NSArray *csvLinesArray = [csvFileString componentsSeparatedByRegex:newlineRegex];
```

```
// Create an id array to hold the comma split line results.
id          splitLines[[csvLinesArray count]]; // C99 variable length array.
NSUInteger  splitLinesIndex = 0UL;             // Index of next splitLines[] member.

for(NSString *csvLineString in csvLinesArray) {                // ObjC 2 for…in loop.
  if([csvLineString isMatchedByRegex:@"^\\s*$"]) { continue; } // Skip empty lines.
  splitLines[splitLinesIndex++] =
      [csvLineString componentsSeparatedByRegex:splitCSVLineRegex];
}

// Gather up all the individual comma split results in to a single NSArray.
NSArray *splitLinesArray = [NSArray arrayWithObjects:&splitLines[0]
                                               count:splitLinesIndex];
```

Network and URL

Description	Regex
HTTP	`\bhttps?://[a-zA-Z0-9\-.]+↩` `(?:(?:/[a-zA-Z0-9\-._?,'+&%$= *!():@\\]*)+)?`
HTTP	`\b(https?)://([a-zA-Z0-9\-.]+)↩` `((?:/[a-zA-Z0-9\-._?,'+&%$= *!():@\\]*)+)?`
HTTP	`\b(https?)://(?:(\S+?)(?::(\S+?))?@)?([a-zA-Z0-9\-.]+)↩` `(?::(\d+))?((?:/[a-zA-Z0-9\-._?,'+&%$= *!():@\\]*)+)?`
E-Mail	`\b([a-zA-Z0-9%_.+\-]+)@([a-zA-Z0-9.\-]+?\.[a-zA-Z]{2,6})\b`
Hostname	`\b(?:[a-zA-Z0-9][a-zA-Z0-9\-]{0,61}?[a-zA-Z0-9]\.)+[a-zA-Z]{2,6}\b`
IP	`\b(?:\d{1,3}\.){3}\d{1,3}\b`
IP with Optional Netmask	`\b((?:\d{1,3}\.){3}\d{1,3})(?:/(\d{1,2}))?\b`
IP or Hostname	`\b(?:(?:\d{1,3}\.){3}\d{1,3}|(?:[a-zA-Z0-9][a-zA-Z0-9\-]{0,61}?↩` `[a-zA-Z0-9]\.)+[a-zA-Z]{2,6})\b`

Creating a `NSDictionary` of URL Information

The following example demonstrates how to match several fields in a URL and create a `NSDictionary` with the extracted results. Only the capture groups that result in a successful match will create a corresponding key in the dictionary.

```
NSString *searchString =
             @"http://johndoe:secret@www.example.com:8080/private/mail/index.html";
NSString *regexString  =
             @"\\b(https?)://(?:(\\S+?)(?::(\\S+?))?@)?([a-zA-Z0-9\\-.]+)"
             @"(?::(\\d+))?((?:/[a-zA-Z0-9\\-._?,'+\\&%$=~*!():@\\\\]*)+)?";

if([searchString isMatchedByRegex:regexString]) {
  NSString *protocolString = [searchString stringByMatching:regexString capture:1L];
```

```
  NSString *userString     = [searchString stringByMatching:regexString capture:2L];
  NSString *passwordString = [searchString stringByMatching:regexString capture:3L];
  NSString *hostString     = [searchString stringByMatching:regexString capture:4L];
  NSString *portString     = [searchString stringByMatching:regexString capture:5L];
  NSString *pathString     = [searchString stringByMatching:regexString capture:6L];

  NSMutableDictionary *urlDictionary = [NSMutableDictionary dictionary];

  if(protocolString) { [urlDictionary setObject:protocolString forKey:@"protocol"]; }
  if(userString)     { [urlDictionary setObject:userString     forKey:@"user"];     }
  if(passwordString) { [urlDictionary setObject:passwordString forKey:@"password"]; }
  if(hostString)     { [urlDictionary setObject:hostString     forKey:@"host"];     }
  if(portString)     { [urlDictionary setObject:portString     forKey:@"port"];     }
  if(pathString)     { [urlDictionary setObject:pathString     forKey:@"path"];     }

  NSLog(@"urlDictionary: %@", urlDictionary);
}
```

RegexKit*Lite* 4.0 adds a new method, `dictionaryByMatchingRegex:…`, that makes the creation of `NSDictionary` objects like this much easier, as the following example demonstrates:

```
NSString     *searchString  =
               @"http://johndoe:secret@www.example.com:8080/private/mail/index.html";

NSString     *regexString   =
               @"\\b(https?)://(?:(\\S+?)(?::(\\S+?))?@)?([a-zA-Z0-9\\-.]+)"
               @"(?::(\\d+))?((?:/[a-zA-Z0-9\\-._?,'+\\&%$=~*!():@\\\\]*)+)?";

NSDictionary *urlDictionary = NULL;

urlDictionary = [searchString dictionaryByMatchingRegex:regexString
                                    withKeysAndCaptures:@"protocol", 1,
                                                        @"user",     2,
                                                        @"password", 3,
                                                        @"host",     4,
                                                        @"port",     5,
                                                        @"path",     6,
                                                        NULL];

if(urlDictionary != NULL) {
  NSLog(@"urlDictionary: %@", urlDictionary);
}
```

> **Note:** Other than the difference in mutability for the dictionary containing the result, the RegexKit*Lite* 4.0 `dictionaryByMatchingRegex:…` example produces the same result as the more verbose, pre-4.0 example.

These examples can form the basis of a function or method that takes a `NSString` as an argument and returns a `NSDictionary` as a result, maybe even as a category addition to `NSString`. The following is the output when the examples above are compiled and run:

```
shell% ./http_example↵
2008-09-01 10:57:55.245 test_nsstring[31306:807] urlDictionary: {
    host = "www.example.com";
    password = secret;
    path = "/private/mail/index.html";
    port = 8080;
    protocol = http;
    user = johndoe;
}
shell% 
```

Adding RegexKit*Lite* to your Project

Note: The following outlines a typical set of steps that one would perform. This is not the only way, nor the required way to add RegexKit*Lite* to your application. They may not be correct for your project as each project is unique. They are an overview for those unfamiliar with adding additional shared libraries to the list of libraries your application links against.

Outline of Required Steps

The following outlines the steps required to use RegexKit*Lite* in your project.

- Linking your application to the ICU dynamic shared library.
- Adding the `RegexKitLite.m` and `RegexKitLite.h` files to your project and application target.
- Import the `RegexKitLite.h` header.

See Also
Xcode Build System Guide - Linking

Adding RegexKit*Lite* using Xcode

Important: These instructions apply to Xcode versions 2.4.1 and 3.0. Other versions should be similar, but may vary for specific details.

Unfortunately, adding additional dynamic shared libraries that your application links to is not a straightforward process in Xcode, nor is there any recommended standard way. Two options are presented below– the first is the 'easy' way that alters your applications Xcode build settings to pass an additional command line argument directly to the linker. The second option attempts to add the ICU dynamic shared library to the list of resources for your project and configuring your executable to link against the added resource.

The 'easy' way is the recommended way to link against the ICU dynamic shared library.

The Easy Way To Add The ICU Library

1. First, determine the build settings layer of your project that should have altered linking configuration change applied to. The build settings in Xcode are divided in to layers and each layer inherits the build settings from the layer above it. The top, global layer is Project Settings, followed by Target Settings, and finally the most specific layer Executable Settings . If your project is large enough to have multiple targets and executables, you probably have an idea which layer is appropriate. If you are unsure or unfamiliar with the different layers, Target Settings is recommended.

2. Select the appropriate layer from the Project menu. If you are unsure, Project ▸ Edit Active Target is recommended.

3. Select Build from the tab near the top of the Target Info window. Find the Other Linker Flags build setting from the many build settings available and edit it. Add `-licucore` [*dash ell icucore* as a single word, without spaces]. If there are already other flags present, it is recommended that you add `-licucore` to the end of the existing flags.

> **Important:** If other linker flags are present, there must be at least one space separating `-licucore` from the other linker flags. For example, `-flag1 -licucore -flag2`

> **Note:** The Configuration drop down menu controls which build configuration the changes you make are applied to. All Configurations should be selected if this is the first time your are making these changes.

4. Follow the *Add The RegexKitLite Source Files To Your Project* steps below.

See Also

Xcode Build System Guide - Build Settings

The Hard Way To Add The ICU Library

1. First, add the ICU dynamic shared library to your Xcode project. You may choose to add the library to any group in your project, and which groups are created by default is dependent on the template type you chose when you created your project. For a typical Cocoa application project, a good choice is the Frameworks group. To add the ICU dynamic shared library, control/right-click on the Framework group and choose Add ▸ Existing Files…

2. Next, you will need to choose the ICU dynamic shared library file to add. Exactly which file to choose depends on your project, but a fairly safe choice is to select `/Developer/SDKs/MacOSX10.6.sdk/↩ usr/lib/libicucore.dylib`. You may have installed your developer tools in a different location than the default `/Developer` directory, and the Mac OS X SDK version should be the one your project is targeting, typically the latest one available.

3. Then, in the dialog that follows, make sure that Copy items into… is unselected. Select the targets you will be using RegexKit*Lite* in and then click Add to add the ICU dynamic shared library to your project.

4. Once the ICU dynamic shared library is added to your project, you will need to add it to the libraries that your executable is linked with. To do so, expand the Targets group, and then expand the executable targets you will be using RegexKit*Lite* in. You will then need to select the `libicucore.dylib` file that you added in the previous step and drag it in to the Link Binary With Libraries group for each executable target that you will be using RegexKit*Lite* in. The order of the files within the Link Binary With Libraries group is not important, and for a typical Cocoa application the group will contain the `Cocoa.framework` file.

Add The RegexKit*Lite* Source Files To Your Project

1. Next, add the RegexKit*Lite* source files to your Xcode project. In the Groups & Files outline view on the left, control/right-click on the group that would like to add the files to, then select Add ▸ Existing Files…

> **Note:** You can perform the following steps once for each file (`RegexKitLite.h` and `RegexKitLite.m`), or once by selecting both files from the file dialog.

2. Select the `RegexKitLite.h` and / or `RegexKitLite.m` file from the file chooser dialog.
3. The next dialog will present you with several options. If you have not already copied the RegexKit*Lite* files in to your projects directory, you may want to click on the Copy items into… option. Select the targets that you would like add the RegexKit*Lite* functionality to.
4. Finally, you will need to include the `RegexKitLite.h` header file. The best way to do this is very dependent on your project. If your project consists of only half a dozen source files, you can add:

   ```
   #import "RegexKitLite.h"
   ```

   manually to each source file that makes uses of RegexKit*Lite*s features. If your project has grown beyond this, you've probably already organized a common "master" header to include to capture headers that are required by nearly all source files already.

Adding RegexKit*Lite* using the Shell

Using RegexKit*Lite* from the shell is also easy. Again, you need to add the header `#import` to the appropriate source files. Then, to link to the ICU library, you typically only need to add `-licucore`, just as you would any other library. Consider the following example:

```
#import <Foundation/NSObjCRuntime.h>
#import <Foundation/NSAutoreleasePool.h>
#import "RegexKitLite.h"

int main(int argc, char *argv[]) {
  NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

  // Copyright COPYRIGHT_SIGN APPROXIMATELY_EQUAL_TO 2008
  // Copyright \u00a9 \u2245 2008

  char     *utf8CString =  "Copyright \xC2\xA9 \xE2\x89\x85 2008";
  NSString *regexString = @"Copyright (.*) (\\d+)";

  NSString *subjectString = [NSString stringWithUTF8String:utf8CString];
  NSString *matchedString = [subjectString stringByMatching:regexString capture:1L];

  NSLog(@"subject: \"%@\"", subjectString);
  NSLog(@"matched: \"%@\"", matchedString);

  [pool release];
  return(0);
}
```

Compiled and run from the shell:

```
shell% cd examples↵
shell% gcc -g -I.. -o link_example link_example.m ../RegexKitLite.m \
       -framework Foundation -licucore↵
```

```
shell% ./link_example↵
2008-03-14 03:52:51.187 test[15283:807] subject: "Copyright © ≅ 2008"
2008-03-14 03:52:51.269 test[15283:807] matched: "© ≅"
shell% 
```

RegexKit*Lite* NSString Additions Reference

Extends by category	NSString, NSMutableString
RegexKit*Lite*	4.0
Declared in	RegexKitLite.h
Companion guides	ICU User Guide - Regular Expressions

Overview

RegexKit*Lite* is not meant to be a full featured regular expression framework. Because of this, it provides only the basic primitives needed to create additional functionality. It is ideal for developers who:

- Developing applications for the iPhone.
- Have modest regular expression needs.
- Require a very small footprint.
- Unable or unwilling to add additional, external frameworks.
- Deal predominantly in `UTF-16` encoded Unicode strings.
- Require the enhanced word breaking functionality provided by the ICU library.

RegexKit*Lite* consists of only two files, the header file `RegexKitLite.h` and `RegexKitLite.m`. The only other requirement is to link with the ICU library that comes with Mac OS X. No new classes are created, all functionality is provided as a category extension to the `NSString` and `NSMutableString` classes.

See Also

RegexKitLite Guide (p. 1)
ICU Regular Expression Syntax (p. 25)
Adding RegexKitLite to your Project (p. 43)
License Information (p. 118)
RegexKit Framework
International Components for Unicode
Unicode Home Page

Compile Time Preprocessor Tunables

The settings listed below are implemented using the C Preprocessor. Some of the setting are simple boolean `enabled` or `disabled` settings, while others specify a value, such as the number of cached compiled regular expressions. There are several ways to alter these settings, but if you are not familiar with this style of compile time configuration settings and how to alter them using the C Preprocessor, it is recommended that you use the default values provided.

Setting	Default	Description
`NS_BLOCK_ASSERTIONS`	*n/a*	RegexKit*Lite* contains a number of extra run-time assertion checks that can be disabled with this flag. The standard `NSException.h` assertion macros are not used because of the multithreading lock. This flag is typically set for Release style builds where the additional error checking is no longer necessary.
`RKL_APPEND_TO_ICU_FUNCTIONS`	*None*	This flag is useful if you are supplying your own version of the ICU library. When set, this preprocessor define causes the ICU functions used by RegexKit*Lite* to have the value of `RKL_APPEND_TO_ICU_FUNCTIONS` appended to them. For example, if `RKL_APPEND_TO_ICU_FUNCTIONS` is set to `_4_0` (i.e., `-DRKL_APPEND_TO_ICU_FUNCTIONS=_4_0`), it would cause `uregex_find()` to become `uregex_find_4_0()`.
`RKL_BLOCKS`	*Automatic*	Enables blocks support. This feature is automatically enabled if `NS_BLOCKS_AVAILABLE` is set to `1`, which is typically set if support for blocks is appropriate. At the time of this writing, this typically means that the Xcode setting for the minimum version of Mac OS X supported must be 10.6. This feature may be explicitly disabled under all circumstances by setting its value to `0`, or alternatively it can be explicitly enabled under all circumstances by setting its value to `1`. The behavior is undefined if `RKL_BLOCKS` is set to `1` and the compiler does not support the blocks language extension or if the run-time does not support blocks.
`RKL_CACHE_SIZE`	`13`	RegexKit*Lite* uses a 4-way set associative cache and `RKL_CACHE_SIZE` controls the number of sets in the cache. The total number of compiled regular expressions that can be cached is `RKL_CACHE_SIZE * 4`, for a default value of `52`. `RKL_CACHE_SIZE` should always be a prime number to maximize the use of the cache.
`RKL_DTRACE`	*Disabled*	This preprocessor define controls whether or not RegexKit*Lite* provider DTrace probe points are enabled. This feature may be explicitly disabled under all circumstances by setting its value to `0`.
`RKL_FAST_MUTABLE_CHECK`	*Disabled*	Enables the use of the undocumented, private *Core Foundation* `__CFStringIsMutable()` function to determine if the string to be searched is immutable. This can significantly increase the number of matches per second that can be performed on immutable strings since a number of mutation checks can be safely skipped.
`RKL_FIXED_LENGTH`	`2048`	Sets the size of the fixed length `UTF-16` conversion cache buffer. Strings that need to be converted to `UTF-16` that have a length less than this size will use the fixed length conversion cache. Using a fixed sized buffer for all small strings means less `malloc()` overhead, heap fragmentation, and reduces the chances of a memory leak occurring.
`RKL_METHOD_PREPEND`	*None*	When set, this preprocessor define causes the RegexKit*Lite* methods defined in `RegexKitLite.h` to have the value of `RKL_METHOD_PREPEND` prepended to them. For example, if `RKL_METHOD_PREPEND` is set to `xyz_` (i.e., `-DRKL_METHOD_PREPEND=xyz_`), it would cause `clearStringCache` (p. 61) to become `xyz_clearStringCache`.

Setting	Default	Description
`RKL_REGISTER_FOR_IPHONE_LOWMEM_NOTIFICATIONS`	*Automatic*	This preprocessor define controls whether or not extra code is included that attempts to automatically register with the `NSNotificationCenter` for the `UIApplicationDidReceiveMemoryWarningNotification` notification. This feature is automatically enabled if it can be determined at compile time that the iPhone is being targeted. This feature may be explicitly disabled under all circumstances by setting its value to `0`.
`RKL_STACK_LIMIT`	`131072`	The maximum amount of stack space that will be used before switching to heap based allocations. This can be useful for multithreading programs where the stack size of secondary threads is much smaller than the main thread.

See Also

Assertions and Logging - Using the Assertion Macros

Fast Mutable Checks

Setting `RKL_FAST_MUTABLE_CHECK` allows RegexKit*Lite* to quickly check if a string to search is immutable or not. Every call to RegexKit*Lite* requires checking a strings `hash` and `length` values to guard against a string mutating and using invalid cached data. If the same string is searched repeatedly and it is immutable, these checks aren't necessary since the string can never change while in use. While these checks are fairly quick, it can add approximately 15 to 20 percent of extra overhead, and not performing the checks is always faster.

Since checking a strings mutability requires calling an undocumented, private *Core Foundation* function, RegexKit*Lite* takes extra precautions and does not use the function directly. Instead, an internal, local stub function is created and called to determine if a string is mutable. The first time this function is called, RegexKit*Lite* uses `dlsym()` to look up the address of the `__CFStringIsMutable()` function. If the function is found, RegexKit*Lite* will use it from that point on to determine if a string is immutable. However, if the function is not found, RegexKit*Lite* has no way to determine if a string is mutable or not, so it assumes the worst case that all strings are potentially mutable. This means that the private *Core Foundation* `__CFStringIsMutable()` function can go away at any time and RegexKit*Lite* will continue to work, although with slightly less performance.

This feature is disabled by default, but should be fairly safe to enable due to the extra precautions that are taken. If this feature is enabled and the `__CFStringIsMutable()` function is not found for some reason, RegexKit*Lite* falls back to its default behavior which is the same as if this feature was not enabled.

iPhone Low Memory Notifications

The `RKL_REGISTER_FOR_IPHONE_LOWMEM_NOTIFICATIONS` preprocessor define controls whether or not extra code is compiled in that automatically registers for the iPhone UIKit `UIApplicationDidReceiveMemoryWarningNotification` notification. When enabled, an initialization function tagged with `__attribute__((constructor))` is executed by the linker at load time which causes RegexKit*Lite* to check if the low memory notification symbol is available. If the symbol is present then RegexKit*Lite* registers to receive the notification. When the notification is received, RegexKit*Lite* will automatically call `clearStringCache` (p. 61) to flush the caches and return the memory used to hold any cached compiled regular expressions.

This feature is normally automatically enabled if it can be determined at compile time that the iPhone is being targeted. This feature is safe to enable even if the target is Mac OS X for the desktop. It can also be explicitly disabled, even when targeting the iPhone, by setting `RKL_REGISTER_FOR_IPHONE_LOWMEM_NOTIFICATIONS` to `0` (i.e., `-DRKL_REGISTER_FOR_IPHONE_LOWMEM_NOTIFICATIONS=0`).

See Also
Memory Usage Performance Guidelines - Responding to Low-Memory Warnings in iPhone OS

Using RegexKit*Lite* with a Custom ICU Build

The details of building and linking to a custom build of ICU will not be covered here. ICU is a very large and complex library that can be configured and packaged in countless ways. Building and linking your application to a custom build of ICU is ***non-trivial***. Apple provides the full source to the version of ICU that they supply with Mac OS X. At the time of this writing, the latest version available was for Mac OS X 10.6.2– `ICU-400.38.tar.gz`.

RegexKit*Lite* provides the `RKL_APPEND_TO_ICU_FUNCTIONS` C preprocessor define if you would like to use RegexKit*Lite* with a custom ICU build that you supply. A custom version of ICU will typically have the ICU version appended to all of its functions, and `RKL_APPEND_TO_ICU_FUNCTIONS` allows you to append that version to the ICU functions that RegexKit*Lite* calls. For example, passing `-DRKL_APPEND_TO_ICU_FUNCTIONS=_4_0` to `gcc` would cause the ICU function `uregex_find()` used by RegexKit*Lite* to be called as `uregex_find_4_0()`.

Xcode 3 Integrated Documentation

This documentation is available in the Xcode DocSet format at the following URL:

```
feed://regexkit.sourceforge.net/RegexKitLiteDocSets.atom
```

For Xcode < 3.2, select Help ▸ Documentation. Then, in the lower left hand corner of the documentation window, there should be a gear icon with a drop down menu indicator which you should select and choose New Subscription… and enter the DocSet URL.

For Xcode ≥ 3.2, select Xcode ▸ Preferences…. Then select the Documentation preference group, typically the right most group, and press Add Documentation Set Publisher… and enter the DocSet URL.

Once you have added the URL, a new group should appear, inside which will be the RegexKit*Lite* documentation with a Get button. Click on the Get button and follow the prompts.

> **Note:** Xcode will ask you to enter an administrators password to install the documentation, which is explained here.

Cached Information and Mutable Strings

While RegexKit*Lite* takes steps to ensure that the information it has cached is valid for the strings it searches, there exists the possibility that out of date cached information may be used when searching mutable strings. For each compiled regular expression, RegexKit*Lite* caches the following information about the last `NSString` that was searched:

- The strings `length`, `hash` value, and the pointer to the `NSString` object.
- The pointer to the `UTF-16` buffer that contains the contents of the string, which may be an internal buffer if the string required conversion.
- The `NSRange` used for the `inRange:` parameter for the last search, and the `NSRange` result for capture `0` of that search.

An ICU compiled regular expression must be "set" to the text to be searched. Before a compiled regular expression is used, the pointer to the string object to search, its `hash`, `length`, and the pointer to the `UTF-16`

buffer is compared with the values that the compiled regular expression was last "set" to. If any of these values are different, the compiled regular expression is reset and "set" to the new string.

If a `NSMutableString` is mutated between two uses of the same compiled regular expression and its `hash`, `length`, or `UTF-16` buffer changes between uses, RegexKit*Lite* will automatically reset the compiled regular expression with the new values of the mutated string. The results returned will correctly reflect the mutations that have taken place between searches.

It is possible that the mutations to a string can go undetected, however. If the mutation keeps the `length` the same, then the only way a change can be detected is if the strings `hash` value changes. For most mutations the `hash` value will change, but it is possible for two different strings to share the same `hash`. This is known as a *hash collision*. Should this happen, the results returned by RegexKit*Lite* may not be correct.

Therefore, if you are using RegexKit*Lite* to search `NSMutableString` objects, and those strings may have mutated in such a way that RegexKit*Lite* is unable to detect that the string has changed, you must manually clear the internal cache to ensure that the results accurately reflect the mutations. To clear the cached information for a specific string you send the instance a `flushCachedRegexData` (p. 80) message:

```
NSMutableString *aMutableString; // Assumed to be valid.
[aMutableString flushCachedRegexData];
```

To clear all of the cached information in RegexKit*Lite*, which includes all the cached compiled regular expressions along with any cached information and `UTF-16` conversions for strings that have been searched, you use the following class method:

```
[NSString clearStringCache];
```

Warning:	When searching `NSMutableString` objects that have mutated between searches, failure to clear the cache may result in undefined behavior. Use `flushCachedRegexData` (p. 80) to selectively clear the cached information about a `NSMutableString` object.

See Also

`+ clearStringCache` (p. 61)
`- flushCachedRegexData` (p. 80)

Block-based Enumeration Methods

The RegexKit*Lite* Block-based enumeration methods are modeled after their `NSString` counterparts. There are a few differences, however.

- RegexKit*Lite* does not support mutating a `NSMutableString` object while it is under going Block-based enumeration.
- There is no support for concurrent enumeration.

While RegexKit*Lite* may not support mutating a `NSMutableString` during Block-based enumeration, it does provide the means to create a new string from the `NSString` object returned by the block used to enumerate the matches in a string, and in the case of `NSMutableString`, to replace the contents of that `NSMutableString` with the modified string at the end of the enumeration. This functionality is available via the following methods:

- `- stringByReplacingOccurrencesOfRegex:usingBlock:` (p. 91) (`NSString`)
- `- replaceOccurrencesOfRegex:usingBlock:` (p. 85) (`NSMutableString`)

Exception to the Cocoa Memory Management Rules for Block-based Enumeration

The standard *Cocoa Memory Management Rules* specify that objects returned by a method, or in this case the objects passed to a Block, remain valid throughout the scope of the calling method. Due to the potentially large volume of temporary strings that are created during a Block-based enumeration, RegexKit*Lite* makes an exception to this rule– the strings passed to a Block via `capturedStrings[]` are valid only until the closing brace of the Block:

```
[searchString enumerateStringsMatchedByRegex:regex
                                  usingBlock:
 ^(NSInteger captureCount,
   NSString * const capturedStrings[captureCount],
   const NSRange capturedRanges[captureCount],
   volatile BOOL * const stop) {
   // Block code.
 } /* <- capturedStrings[] is valid only up to this point. */ ];
```

If you need to refer to a string past the closing brace of the Block, you need to send that string a `retain` message. Of course, it is not always necessary to explicitly send a `capturedStrings[]` string a `retain` message when you need it to exist past the closing brace of a Block– adding a `capturedStrings[]` string to a `NSMutableDictionary` will send the string a `retain` as a side effect of adding it to the dictionary.

Memory management during RegexKit*Lite* Block-based enumeration is conceptually similar to the following pseudo-code:

```
NSInteger captureCount = [regex captureCount];

while(moreMatches) {
  NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

  BOOL      stop = NO;
  NSRange   capturedRanges[captureCount];
  NSString *capturedStrings[captureCount];

  for(capture = 0L; capture < captureCount; capture++) {
    capturedRanges[capture]  = [searchString rangeOfRegex:regex capture:capture];
    capturedStrings[capture] = [searchString stringByMatching:regex capture:capture];
  }

  // The following line represents the execution of the user supplied Block.
  enumerationBlock(captureCount, capturedStrings, capturedRanges, &stop);
  // ... and this represents when the user supplied Block has finished / returned.

  [pool release]; // capturedStrings[] are sent a release at this point.
  if(stop != NO) { break; }
}
```

While conceptually and behaviorally similar, it is important to note that RegexKit*Lite* does not actually use or create autorelease pools when performing Block-based enumeration. Instead, a `CFMutableArray` object is used to accumulate the temporary string objects during an iteration, and at the start of an iteration, any previously accumulated temporary string objects are removed from the array.

See Also

Blocks Programming Topics
Regular Expression Enumeration Options (p. 97)

Usage Notes

Convenience Methods

For convenience methods where an argument is not present, the default value used is given below.

Argument	Default Value
`capture:`	`0`
`options:`	`RKLNoOptions` (p. 96)
`range:`	The entire range of the receiver.
`enumerationOptions:`	`RKLRegexEnumerationNoOptions` (p. 98)

Exceptions Raised

Methods will raise an exception if their arguments are invalid, such as passing `NULL` for a required parameter. An invalid regular expression or `RKLRegexOptions` (p. 95) parameter will not raise an exception. Instead, a `NSError` object with information about the error will be created and returned via the address given with the optional *`error`* argument. If information about the problem is not required, *`error`* may be `NULL`. For convenience methods that do not have an *`error`* argument, the primary method is invoked with `NULL` passed as the argument for *`error`*.

Important: Methods raise `NSInvalidArgumentException` if *`regex`* is `NULL`, or if *`capture`* < `0`, or is not valid for *`regex`*.

Important: Methods raise `NSRangeException` if *`range`* exceeds the bounds of the receiver.

Important: Methods raise `NSRangeException` if the receivers length exceeds the maximum value that can be represented by a signed 32-bit integer, even on 64-bit architectures.

Important: Search and replace methods raise `RKLICURegexException` (p. 103) if *`replacement`* contains `$`*`n`* capture references where *`n`* is greater than the number of capture groups in the regular expression *`regex`*.

See Also

RegexKitLite NSError Error Domains (p. 103)
RegexKitLite NSError and NSException User Info Dictionary Keys (p. 103)

Tasks

DTrace Probe Points

`RegexKitLite:::compiledRegexCache` (p. 58)
This probe point fires each time the compiled regular expression cache is accessed.

`RegexKitLite:::utf16ConversionCache` (p. 59)
This probe point fires each time the `UTF-16` conversion cache is accessed.

Clearing Cached Information

`+ clearStringCache` (p. 61)
Clears the cached information about strings.

`- flushCachedRegexData` (p. 80)
Clears any cached information about the receiver.

Determining the Number of Captures

`+ captureCountForRegex:` (p. 59) Deprecated in RegexKit*Lite* 3.0
Returns the number of captures that *`regex`* contains.

`+ captureCountForRegex:options:error:` (p. 60) Deprecated in RegexKit*Lite* 3.0
Returns the number of captures that *`regex`* contains.

`- captureCount` (p. 68)
Returns the number of captures that the receiver contains.

`- captureCountWithOptions:error:` (p. 68)
Returns the number of captures that the receiver contains.

Finding all Captures of all Matches

`- arrayOfCaptureComponentsMatchedByRegex:` (p. 61)
Returns an array containing all the matches from the receiver that were matched by the regular expression *`regex`*. Each match result consists of an array of the substrings matched by all the capture groups present in the regular expression.

`- arrayOfCaptureComponentsMatchedByRegex:range:` (p. 62)
Returns an array containing all the matches from the receiver that were matched by the regular expression *`regex`* within *`range`*. Each match result consists of an array of the substrings matched by all the capture groups present in the regular expression.

`- arrayOfCaptureComponentsMatchedByRegex:options:range:error:` (p. 62)
Returns an array containing all the matches from the receiver that were matched by the regular expression *`regex`* within *`range`* using *`options`*. Each match result consists of an array of the substrings matched by all the capture groups present in the regular expression.

Getting all the Captures of a Match

`- captureComponentsMatchedByRegex:` (p. 66)
Returns an array containing the substrings matched by each capture group present in *`regex`* for the first match of *`regex`* in the receiver.

`- captureComponentsMatchedByRegex:range:` (p. 67)
Returns an array containing the substrings matched by each capture group present in *`regex`* for the first match of *`regex`* within *`range`* of the receiver.

`- captureComponentsMatchedByRegex:options:range:error:` (p. 67)
Returns an array containing the substrings matched by each capture group present in *`regex`* for the first match of *`regex`* within *`range`* of the receiver using *`options`*.

Finding all Matches

- `componentsMatchedByRegex:` (p. 69)
  Returns a `NSArray` containing all the substrings from the receiver that were matched by the regular expression *`regex`*.
- `componentsMatchedByRegex:capture:` (p. 69)
  Returns a `NSArray` containing all the substrings from the receiver that were matched by capture number *`capture`* from the regular expression *`regex`*.
- `componentsMatchedByRegex:range:` (p. 70)
  Returns a `NSArray` containing all the substrings from the receiver that were matched by the regular expression *`regex`* within *`range`*.
- `componentsMatchedByRegex:options:range:capture:error:` (p. 70)
  Returns a `NSArray` containing all the substrings from the receiver that were matched by the regular expression *`regex`* within *`range`* using *`options`*.
- `enumerateStringsMatchedByRegex:usingBlock:` (p. 76)
  Enumerates the matches in the receiver by the regular expression *`regex`* and executes *`block`* for each match found.
- `enumerateStringsMatchedByRegex:options:inRange:error:enumerationOptions:usingBlock:` (p. 77)
  Enumerates the matches in the receiver by the regular expression *`regex`* within *`range`* using *`options`* and executes *`block`* using *`enumerationOptions`* for each match found.

Dividing Strings

- `componentsSeparatedByRegex:` (p. 71)
  Returns a `NSArray` containing substrings of the receiver that have been divided by the regular expression *`regex`*.
- `componentsSeparatedByRegex:range:` (p. 72)
  Returns a `NSArray` containing substrings within range of the receiver that have been divided by the regular expression *`regex`*.
- `componentsSeparatedByRegex:options:range:error:` (p. 72)
  Returns a `NSArray` containing substrings within range of the receiver that have been divided by the regular expression *`regex`* using *`options`*.
- `enumerateStringsSeparatedByRegex:usingBlock:` (p. 78)
  Enumerates the strings of the receiver that have been divided by the regular expression *`regex`* and executes *`block`* for each divided string.
- `enumerateStringsSeparatedByRegex:options:inRange:error:enumerationOptions:usingBlock:` (p. 79)
  Enumerates the strings of the receiver that have been divided by the regular expression *`regex`* within *`range`* using *`options`* and executes *`block`* using *`enumerationOptions`* for each divided string.

Identifying Matches

- `isMatchedByRegex:` (p. 82)
  Returns a Boolean value that indicates whether the receiver is matched by *`regex`*.
- `isMatchedByRegex:inRange:` (p. 82)
  Returns a Boolean value that indicates whether the receiver is matched by *`regex`* within *`range`*.
- `isMatchedByRegex:options:inRange:error:` (p. 82)
  Returns a Boolean value that indicates whether the receiver is matched by *`regex`* within *`range`*.

Determining if a Regular Expression is Valid

- `isRegexValid` (p. 82)
  Returns a Boolean value that indicates whether the regular expression contained in the receiver is valid.
- `isRegexValidWithOptions:error:` (p. 83)
  Returns a Boolean value that indicates whether the regular expression contained in the receiver is valid using *`options`*.

Determining the Range of a Match

- `rangeOfRegex:` (p. 83)
  Returns the range for the first match of *`regex`* in the receiver.
- `rangeOfRegex:capture:` (p. 84)
  Returns the range of capture number *`capture`* for the first match of *`regex`* in the receiver.
- `rangeOfRegex:inRange:` (p. 84)
  Returns the range for the first match of *`regex`* within *`range`* of the receiver.
- `rangeOfRegex:options:inRange:capture:error:` (p. 84)
  Returns the range of capture number *`capture`* for the first match of *`regex`* within *`range`* of the receiver.

Modifying Mutable Strings

- `replaceOccurrencesOfRegex:withString:` (p. 87)
  Replaces all occurrences of the regular expression *`regex`* with the contents of replacement string after performing capture group substitutions, returning the number of replacements made.
- `replaceOccurrencesOfRegex:withString:range:` (p. 88)
  Replaces all occurrences of the regular expression *`regex`* within *`range`* with the contents of replacement string after performing capture group substitutions, returning the number of replacements made.
- `replaceOccurrencesOfRegex:withString:options:range:error:` (p. 88)
  Replaces all occurrences of the regular expression *`regex`* using *`options`* within *`range`* with the contents of replacement string after performing capture group substitutions, returning the number of replacements made.
- `replaceOccurrencesOfRegex:usingBlock:` (p. 85)
  Enumerates the matches in the receiver by the regular expression *`regex`* and executes *`block`* for each match found. Replaces the characters that were matched with the contents of the string returned by *`block`*, returning the number of replacements made.
- `replaceOccurrencesOfRegex:options:inRange:error:enumerationOptions:usingBlock:` (p. 86)
  Enumerates the matches in the receiver by the regular expression *`regex`* within *`range`* using *`options`* and executes *`block`* using *`enumerationOptions`* for each match found. Replaces the characters that were matched with the contents of the string returned by *`block`*, returning the number of replacements made.

Creating Temporary Strings from a Match

- `stringByMatching:` (p. 89)
  Returns a string created from the characters of the receiver that are in the range of the first match of *`regex`*.
- `stringByMatching:capture:` (p. 89)
  Returns a string created from the characters of the receiver that are in the range of the first match of *`regex`* for *`capture`*.

- `stringByMatching:inRange:` (p. 90)
  Returns a string created from the characters of the receiver that are in the range of the first match of *regex* within *range* of the receiver.
- `stringByMatching:options:inRange:capture:error:` (p. 90)
  Returns a string created from the characters of the receiver that are in the range of the first match of *regex* using *options* within *range* of the receiver for *capture*.

Replacing Substrings

- `stringByReplacingOccurrencesOfRegex:withString:` (p. 93)
  Returns a string created from the characters of the receiver in which all matches of the regular expression *regex* are replaced with the contents of the replacement string after performing capture group substitutions.
- `stringByReplacingOccurrencesOfRegex:withString:range:` (p. 94)
  Returns a string created from the characters within *range* of the receiver in which all matches of the regular expression *regex* are replaced with the contents of the replacement string after performing capture group substitutions.
- `stringByReplacingOccurrencesOfRegex:withString:options:range:error:` (p. 94)
  Returns a string created from the characters within *range* of the receiver in which all matches of the regular expression *regex* using *options* are replaced with the contents of the replacement string after performing capture group substitutions.
- `stringByReplacingOccurrencesOfRegex:usingBlock:` (p. 91)
  Enumerates the matches in the receiver by the regular expression *regex* and executes *block* for each match found. Returns a string created by replacing the characters that were matched in the receiver with the contents of the string returned by *block*.
- `stringByReplacingOccurrencesOfRegex:options:inRange:error:enumerationOptions:usingBlock:` (p. 92)
  Enumerates the matches in the receiver by the regular expression *regex* within *range* using *options* and executes *block* using *enumerationOptions* for each match found. Returns a string created by replacing the characters that were matched in the receiver with the contents of the string returned by *block*.

Creating a Dictionary from a Match

- `dictionaryByMatchingRegex:withKeysAndCaptures:` (p. 73)
  Creates and returns a dictionary containing the matches constructed from the specified set of *keys* and *captures* for the first match of *regex* in the receiver.
- `dictionaryByMatchingRegex:range:withKeysAndCaptures:` (p. 74)
  Creates and returns a dictionary containing the matches constructed from the specified set of *keys* and *captures* for the first match of *regex* within *range* of the receiver.
- `dictionaryByMatchingRegex:options:range:error:withKeysAndCaptures:` (p. 75)
  Creates and returns a dictionary containing the matches constructed from the specified set of *keys* and *captures* for the first match of *regex* within *range* of the receiver using *options*.

Creating a Dictionary for every Match

- `arrayOfDictionariesByMatchingRegex:withKeysAndCaptures:` (p. 63)
  Returns an array containing all the matches in the receiver that were matched by the regular expression *regex*. Each match result consists of a dictionary containing that matches substrings constructed from the specified set of *keys* and *captures*.

- `arrayOfDictionariesByMatchingRegex:range:withKeysAndCaptures:` (p. 64)
  Returns an array containing all the matches in the receiver that were matched by the regular expression *`regex`* within *`range`*. Each match result consists of a dictionary containing that matches substrings constructed from the specified set of *`keys`* and *`captures`*.
- `arrayOfDictionariesByMatchingRegex:options:range:error:withKeysAndCaptures:` (p. 65)
  Returns an array containing all the matches in the receiver that were matched by the regular expression *`regex`* within *`range`* using *`options`*. Each match result consists of a dictionary containing that matches substrings constructed from the specified set of *`keys`* and *`captures`*.

DTrace Probe Points

RegexKitLite:::compiledRegexCache

This probe point fires each time the compiled regular expression cache is accessed.

```
RegexKitLite:::compiledRegexCache(unsigned long eventID, const char *regexUTF8,
    int options, int captures, int hitMiss, int icuStatusCode,
    const char *icuErrorMessage, double *hitRate);
```

Parameters

`arg0`, *`eventID`*
The unique ID for this mutex lock acquisition.

`arg1`, *`regexUTF8`*
Up to 64 characters of the regular expression encoded in `UTF-8`. Must be copied with `copyinstr(arg1)`.

`arg2`, *`options`*
The `RKLRegexOptions` (p. 95) options used.

`arg3`, *`captures`*
The number of captures present in the regular expression, or `-1` if there was an error.

`arg4`, *`hitMiss`*
A boolean value that indicates whether or not this event was a cache hit or not, or `-1` if there was an error.

`arg5`, *`icuStatusCode`*
If an error occurs, this contains the error number returned by ICU.

`arg6`, *`icuErrorMessage`*
If an error occurs, this contains a `UTF-8` encoded string of the ICU error. Must be copied with `copyinstr(arg6)`.

`arg7`, *`hitRate`*
A pointer to a floating point value, between `0.0` and `100.0`, that represents the effectiveness of cache. Higher is better. Must be copied with `copyin(arg7, sizeof(double))`.

Discussion

An example of how to copy the `double` value pointed to by *`hitRate`*:

```
RegexKitLite*:::compiledRegexCache {
  this->hitRate = (double *)copyin(arg7, sizeof(double));
  printf("compiledRegexCache hitRate: %6.2f%%\n", this->hitRate);
}
```

See Also

`RegexKitLite:::utf16ConversionCache` (p. 59)
Using RegexKitLite - DTrace (p. 21)

Solaris Dynamic Tracing Guide (as .PDF)

RegexKitLite:::utf16ConversionCache

This probe point fires each time the `UTF-16` conversion cache is accessed.

```
RegexKitLite:::utf16ConversionCache(unsigned long eventID,
    unsigned int lookupResultFlags, double *hitRate, const void *string,
    unsigned long NSRange_location, unsigned long NSRange_length, long length);
```

Parameters

arg0, eventID
The unique ID for this mutex lock acquisition.

arg1, lookupResultFlags
A set of status flags about the result of the conversion cache lookup.

arg2, hitRate
A pointer to a floating point value, between `0.0` and `100.0`, that represents the effectiveness of cache. Higher is better. Must be copied with `copyin(arg2, sizeof(double))`.

arg3, string
A pointer to the `NSString` that this `UTF-16` conversion cache check is being performed on.

arg4, NSRange_location
The location value of the *range* argument from the invoking RegexKit*Lite* method.

arg5, NSRange_length
The length value of the *range* argument from the invoking RegexKit*Lite* method.

arg6, length
The length of the string.

Discussion

Only strings that require a `UTF-16` conversion count towards the value calculated for *hitRate*.

An example of how to copy the `double` value pointed to by *hitRate*:

```
RegexKitLite*:::utf16ConversionCache {
  this->hitRate = (double *)copyin(arg2, sizeof(double));
  printf("utf16ConversionCache hitRate: %6.2f%%\n", this->hitRate);
}
```

See Also

`RegexKitLite:::compiledRegexCache` (p. 58)
RegexKitLite:::utf16ConversionCache arg1 Flags (p. 102)
Using RegexKitLite - DTrace (p. 21)
Solaris Dynamic Tracing Guide (as .PDF)

Class Methods

captureCountForRegex:

Returns the number of captures that *regex* contains. Deprecated in RegexKit*Lite* 3.0. Use `captureCount` (p. 68) instead.

```
+ (NSInteger)captureCountForRegex:(NSString *)regex;
```

Discussion
Since the capture count of a regular expression does not depend on the string to be searched, this is a `NSString` class method. For example:

```
NSInteger regexCaptureCount = [NSString captureCountForRegex:@"(\\d+)\.(\\d+)"];
// regexCaptureCount would be set to 2.
```

Availability
Deprecated in RegexKit*Lite* 3.0.

Return Value
Returns `-1` if an error occurs. Otherwise the number of captures in *`regex`* is returned, or `0` if *`regex`* does not contain any captures.

See Also
- `+ captureCountForRegex:options:error:` (p. 60) Deprecated in RegexKit*Lite* 3.0
- `- captureCount` (p. 68)
- `- captureCountWithOptions:error:` (p. 68)

captureCountForRegex:options:error:

Returns the number of captures that *`regex`* contains. Deprecated in RegexKit*Lite* 3.0. Use `captureCountWithOptions:error:` (p. 68) instead.

```
+ (NSInteger)captureCountForRegex:(NSString *)regex options:(RKLRegexOptions)options
    error:(NSError **)error;
```

Discussion
The optional *`error`* parameter, if set and an error occurs, will contain a `NSError` object that describes the problem. This may be set to `NULL` if information about any errors is not required.

Since the capture count of a regular expression does not depend on the string to be searched, this is a `NSString` class method. For example:

```
NSInteger regexCaptureCount = [NSString captureCountForRegex:@"(\\d+)\.(\\d+)"
                                                      options:RKLNoOptions
                                                        error:NULL];
// regexCaptureCount would be set to 2.
```

Availability
Deprecated in RegexKit*Lite* 3.0.

Return Value
Returns `-1` if an error occurs. Otherwise the number of captures in *`regex`* is returned, or `0` if *`regex`* does not contain any captures.

See Also
- `+ captureCountForRegex:` (p. 59) Deprecated in RegexKit*Lite* 3.0
- `- captureCount` (p. 68)
- `- captureCountWithOptions:error:` (p. 68)
- *RegexKitLite NSError Error Domains* (p. 103)
- *RegexKitLite NSError and NSException User Info Dictionary Keys* (p. 103)
- *Regular Expression Options* (p. 95)

clearStringCache

Clears the cached information about strings and cached compiled regular expressions.

```
+ (void)clearStringCache;
```

Discussion
This method clears all the the cached state maintained by RegexKit*Lite*. This includes all the cached compiled regular expressions and any cached `UTF-16` conversions.

An example of clearing the cache:

```
[NSString clearStringCache]; // Clears all RegexKitLite cache state.
```

> **Warning:** When searching `NSMutableString` objects that have mutated between searches, failure to clear the cache may result in undefined behavior. Use `flushCachedRegexData` (p. 80) to selectively clear the cached information about a `NSMutableString` object.

> **Note:** You do not need to call `clearStringCache` (p. 61) or `flushCachedRegexData` (p. 80) when using the `NSMutableString` `replaceOccurrencesOfRegex:withString:` (p. 87) methods. The cache entry for that regular expression and `NSMutableString` is automatically cleared as necessary.

Availability
Available in RegexKit*Lite* 1.1 and later.

See Also
- `flushCachedRegexData` (p. 80)

NSString RegexKitLite Additions Reference - Cached Information and Mutable Strings (p. 50)

Instance Methods

arrayOfCaptureComponentsMatchedByRegex:

Returns an array containing all the matches from the receiver that were matched by the regular expression *regex*. Each match result consists of an array of the substrings matched by all the capture groups present in the regular expression.

```
- (NSArray *)arrayOfCaptureComponentsMatchedByRegex:(NSString *)regex;
```

Return Value
A `NSArray` object containing all the matches from the receiver by *regex*. Each match result consists of a `NSArray` which contains all the capture groups present in *regex*. Array index `0` represents all of the text matched by *regex* and subsequent array indexes contain the text matched by their respective capture group.

Discussion
A match result array index will contain an empty string, or `@""`, if a capture group did not match any text.

Availability
Available in RegexKit*Lite* 3.0 and later.

See Also
- `arrayOfCaptureComponentsMatchedByRegex:range:` (p. 62)
- `arrayOfCaptureComponentsMatchedByRegex:options:range:error:` (p. 62)

arrayOfCaptureComponentsMatchedByRegex:range:

Returns an array containing all the matches from the receiver that were matched by the regular expression *regex* within *range*. Each match result consists of an array of the substrings matched by all the capture groups present in the regular expression.

```
- (NSArray *)arrayOfCaptureComponentsMatchedByRegex:(NSString *)regex
    range:(NSRange)range;
```

Return Value
A `NSArray` object containing all the matches from the receiver by *regex*. Each match result consists of a `NSArray` which contains all the capture groups present in *regex*. Array index `0` represents all of the text matched by *regex* and subsequent array indexes contain the text matched by their respective capture group.

Discussion
A match result array index will contain an empty string, or `@""`, if a capture group did not match any text.

Availability
Available in RegexKit*Lite* 3.0 and later.

See Also
- `arrayOfCaptureComponentsMatchedByRegex:` (p. 61)
- `arrayOfCaptureComponentsMatchedByRegex:options:range:error:` (p. 62)

arrayOfCaptureComponentsMatchedByRegex:options:range:error:

Returns an array containing all the matches from the receiver that were matched by the regular expression *regex* within *range* using *options*. Each match result consists of an array of the substrings matched by all the capture groups present in the regular expression.

```
- (NSArray *)arrayOfCaptureComponentsMatchedByRegex:(NSString *)regex
    options:(RKLRegexOptions)options range:(NSRange)range error:(NSError **)error;
```

Parameters
regex
: A `NSString` containing a regular expression.

options
: A mask of options specified by combining `RKLRegexOptions` (p. 95) flags with the C bitwise OR operator. Either `0` or `RKLNoOptions` (p. 96) may be used if no options are required.

range
: The range of the receiver to search.

error
: An optional parameter that if set and an error occurs, will contain a `NSError` object that describes the problem. This may be set to `NULL` if information about any errors is not required.

Return Value
A `NSArray` object containing all the matches from the receiver by *regex*. Each match result consists of a `NSArray` which contains all the capture groups present in *regex*. Array index `0` represents all of the text matched by *regex* and subsequent array indexes contain the text matched by their respective capture group.

Discussion
If the receiver is not matched by *regex* then the returned value is a `NSArray` that contains no items.

A match result array index will contain an empty string, or `@""`, if a capture group did not match any text.

The match results in the array appear in the order they did in the receiver. For example, this code fragment:

```
NSString *list      = @"$10.23, $1024.42, $3099";
NSString *regex     = @"\\$((\\d+)(?:\\.(\\d+)|\\.?))";
NSArray  *listItems = [list arrayOfCaptureComponentsMatchedByRegex:regex];
```

produces a `NSArray` equivalent to:

```
[NSArray arrayWithObjects:
    [NSArray arrayWithObjects:  @"$10.23",   @"10.23",   @"10", @"23", NULL], // Idx 0
    [NSArray arrayWithObjects:@"$1024.42", @"1024.42", @"1024", @"42", NULL], // Idx 1
    [NSArray arrayWithObjects:   @"$3099",    @"3099", @"3099",   @"", NULL], // Idx 2
    NULL];
```

Availability
Available in RegexKit*Lite* 3.0 and later.

See Also
- `- arrayOfCaptureComponentsMatchedByRegex:` (p. 61)
- `- arrayOfCaptureComponentsMatchedByRegex:range:` (p. 62)

RegexKitLite NSError Error Domains (p. 103)
RegexKitLite NSError and NSException User Info Dictionary Keys (p. 103)
Regular Expression Options (p. 95)

arrayOfDictionariesByMatchingRegex:withKeysAndCaptures:

Returns an array containing all the matches in the receiver that were matched by the regular expression *regex*. Each match result consists of a dictionary containing that matches substrings constructed from the specified set of *keys* and *captures*.

```
- (NSArray *)arrayOfDictionariesByMatchingRegex:(NSString *)regex
    withKeysAndCaptures:(id)firstKey, ...;
```

Parameters

regex
: A `NSString` containing a regular expression.

firstKey
: The first key to add to the new dictionary.

...
: First the *capture* for *firstKey*, then a `NULL`-terminated list of alternating *keys* and *captures*. *Captures* are specified using `int` values.

> **Important:** Use of non-`int` sized *capture* arguments will result in undefined behavior. **Do not** append *capture* arguments with a `L` suffix.

> **Important:** Failure to `NULL`-terminate the *keys* and *captures* list will result in undefined behavior.

Return Value
A `NSArray` object containing all the matches from the receiver by *regex*. Each match result consists of a `NSDictionary` containing that matches substrings constructed from the specified set of *keys* and *captures*.

Discussion
If the receiver is not matched by *regex* then the returned value is a `NSArray` that contains no items.

A dictionary will not contain a given key if its corresponding capture group did not match any text.

Availability
Available in RegexKit*Lite* 4.0 and later.

See Also
- `arrayOfDictionariesByMatchingRegex:range:withKeysAndCaptures:` (p. 64)
- `arrayOfDictionariesByMatchingRegex:options:range:error:withKeysAndCaptures:` (p. 65)

RegexKitLite NSError Error Domains (p. 103)
RegexKitLite NSError and NSException User Info Dictionary Keys (p. 103)
Regular Expression Options (p. 95)

arrayOfDictionariesByMatchingRegex:range:withKeysAndCaptures:

Returns an array containing all the matches in the receiver that were matched by the regular expression *regex* within *range*. Each match result consists of a dictionary containing that matches substrings constructed from the specified set of *keys* and *captures*.

```
- (NSArray *)arrayOfDictionariesByMatchingRegex:(NSString *)regex range:(NSRange)range
    withKeysAndCaptures:(id)firstKey, ...;
```

Parameters

regex
: A `NSString` containing a regular expression.

range
: The range of the receiver to search.

firstKey
: The first key to add to the new dictionary.

...
: First the *capture* for *firstKey*, then a `NULL`-terminated list of alternating *keys* and *captures*. *Captures* are specified using `int` values.

> **Important:** Use of non-`int` sized *capture* arguments will result in undefined behavior. **Do not** append *capture* arguments with a `L` suffix.

> **Important:** Failure to `NULL`-terminate the *keys* and *captures* list will result in undefined behavior.

Return Value
A `NSArray` object containing all the matches from the receiver by *regex*. Each match result consists of a `NSDictionary` containing that matches substrings constructed from the specified set of *keys* and *captures*.

Discussion
If the receiver is not matched by *regex* then the returned value is a `NSArray` that contains no items.

A dictionary will not contain a given key if its corresponding capture group did not match any text.

Availability
Available in RegexKit*Lite* 4.0 and later.

See Also
- `arrayOfDictionariesByMatchingRegex:withKeysAndCaptures:` (p. 63)
- `arrayOfDictionariesByMatchingRegex:options:range:error:withKeysAndCaptures:` (p. 65)

RegexKitLite NSError Error Domains (p. 103)
RegexKitLite NSError and NSException User Info Dictionary Keys (p. 103)
Regular Expression Options (p. 95)

arrayOfDictionariesByMatchingRegex:options:range:error: withKeysAndCaptures:

Returns an array containing all the matches in the receiver that were matched by the regular expression *regex* within *range* using *options*. Each match result consists of a dictionary containing that matches substrings constructed from the specified set of *keys* and *captures*.

```
- (NSArray *)arrayOfDictionariesByMatchingRegex:(NSString *)regex
    options:(RKLRegexOptions)options range:(NSRange)range error:(NSError **)error
    withKeysAndCaptures:(id)firstKey, ...;
```

Parameters

regex
: A `NSString` containing a regular expression.

options
: A mask of options specified by combining `RKLRegexOptions` (p. 95) flags with the C bitwise OR operator. Either `0` or `RKLNoOptions` (p. 96) may be used if no options are required.

range
: The range of the receiver to search.

error
: An optional parameter that if set and an error occurs, will contain a `NSError` object that describes the problem. This may be set to `NULL` if information about any errors is not required.

firstKey
: The first key to add to the new dictionary.

...
: First the *capture* for *firstKey*, then a `NULL`-terminated list of alternating *keys* and *captures*. *Captures* are specified using `int` values.

> **Important:** Use of non-`int` sized *capture* arguments will result in undefined behavior. **Do not** append *capture* arguments with a `L` suffix.

> **Important:** Failure to `NULL`-terminate the *keys* and *captures* list will result in undefined behavior.

Return Value

A `NSArray` object containing all the matches from the receiver by *regex*. Each match result consists of a `NSDictionary` containing that matches substrings constructed from the specified set of *keys* and *captures*.

Discussion

If the receiver is not matched by *regex* then the returned value is a `NSArray` that contains no items.

A dictionary will not contain a given key if its corresponding capture group did not match any text. It is important to note that a regular expression can successfully match zero characters:

```
NSString *name      = @"Name: Bob\n"
                      @"Name: John Smith";
NSString *regex     = @"(?m)^Name:\\s*(\\w*)\\s*(\\w*)$";
```

```
    NSArray  *nameArray =
    [name arrayOfDictionariesByMatchingRegex:regex
                                     options:RKLNoOptions
                                       range:NSMakeRange(0UL, [name length])
                                       error:NULL
                          withKeysAndCaptures:@"first", 1, @"last", 2, NULL];

// 2010-04-16 01:15:30.061 RegexKitLite[42984:a0f] nameArray: (
//         { first = Bob,  last = ""    },
//         { first = John, last = Smith }
// )
```

Compared to this example, where the second capture group does not match any characters:

```
    NSString *name      = @"Name: Bob\n"
                          @"Name: John Smith";
    NSString *regex     = @"(?m)^Name:\\s*(\\w*)(?:\\s*|\\s+(\\w+))$";
    NSArray  *nameArray =
    [name arrayOfDictionariesByMatchingRegex:regex
                                     options:RKLNoOptions
                                       range:NSMakeRange(0UL, [name length])
                                       error:NULL
                          withKeysAndCaptures:@"first", 1, @"last", 2, NULL];

// 2010-04-16 01:15:30.061 RegexKitLite[42984:a0f] nameArray: (
//         { first = Bob                },
//         { first = John, last = Smith }
// )
```

Availability
Available in RegexKit*Lite* 4.0 and later.

See Also
- `- arrayOfDictionariesByMatchingRegex:withKeysAndCaptures:` (p. 63)
- `- arrayOfDictionariesByMatchingRegex:range:withKeysAndCaptures:` (p. 64)

RegexKitLite NSError Error Domains (p. 103)
RegexKitLite NSError and NSException User Info Dictionary Keys (p. 103)
Regular Expression Options (p. 95)

captureComponentsMatchedByRegex:

Returns an array containing the substrings matched by each capture group present in *regex* for the first match of *regex* in the receiver.

```
- (NSArray *)captureComponentsMatchedByRegex:(NSString *)regex;
```

Return Value
A `NSArray` containing the substrings matched by each capture group present in *regex* for the first match of *regex* in the receiver. Array index `0` represents all of the text matched by *regex* and subsequent array indexes contain the text matched by their respective capture group.

Discussion
A match result array index will contain an empty string, or `@""`, if a capture group did not match any text.

Availability
Available in RegexKit*Lite* 3.0 and later.

See Also
- `captureComponentsMatchedByRegex:range:` (p. 67)
- `captureComponentsMatchedByRegex:options:range:error:` (p. 67)

captureComponentsMatchedByRegex:range:

Returns an array containing the substrings matched by each capture group present in *regex* for the first match of *regex* within *range* of the receiver.

```
- (NSArray *)captureComponentsMatchedByRegex:(NSString *)regex range:(NSRange)range;
```

Return Value
A `NSArray` containing the substrings matched by each capture group present in *regex* for the first match of *regex* within *range* of the receiver. Array index `0` represents all of the text matched by *regex* and subsequent array indexes contain the text matched by their respective capture group.

Discussion
A match result array index will contain an empty string, or `@""`, if a capture group did not match any text.

Availability
Available in RegexKit*Lite* 3.0 and later.

See Also
- `captureComponentsMatchedByRegex:` (p. 66)
- `captureComponentsMatchedByRegex:options:range:error:` (p. 67)

captureComponentsMatchedByRegex:options:range:error:

Returns an array containing the substrings matched by each capture group present in *regex* for the first match of *regex* within *range* of the receiver using *options*.

```
- (NSArray *)captureComponentsMatchedByRegex:(NSString *)regex
    options:(RKLRegexOptions)options range:(NSRange)range error:(NSError **)error;
```

Parameters
regex
: A `NSString` containing a regular expression.

options
: A mask of options specified by combining `RKLRegexOptions` (p. 95) flags with the C bitwise OR operator. Either `0` or `RKLNoOptions` (p. 96) may be used if no options are required.

range
: The range of the receiver to search.

error
: An optional parameter that if set and an error occurs, will contain a `NSError` object that describes the problem. This may be set to `NULL` if information about any errors is not required.

Return Value
A `NSArray` containing the substrings matched by each capture group present in *regex* for the first match of *regex* within *range* of the receiver using *options*. Array index `0` represents all of the text matched by *regex* and subsequent array indexes contain the text matched by their respective capture group.

Discussion
If the receiver is not matched by *regex* then the returned value is a `NSArray` that contains no items.

A match result array index will contain an empty string, or `@""`, if a capture group did not match any text.

The returned value is for the first match of *regex* in the receiver. For example, this code fragment:

```
NSString *list      = @"$10.23, $1024.42, $3099";
NSString *regex     = @"\\$((\\d+)(?:\\.(\\d+)|\\.?))";
NSArray  *listItems = [list captureComponentsMatchedByRegex:regex];
```

produces a `NSArray` equivalent to:

```
[NSArray arrayWithObjects: @"$10.23", @"10.23", @"10", @"23", NULL];
```

Availability
Available in RegexKit*Lite* 3.0 and later.

See Also
- `- captureComponentsMatchedByRegex:` (p. 66)
- `- captureComponentsMatchedByRegex:range:` (p. 67)

RegexKitLite NSError Error Domains (p. 103)
RegexKitLite NSError and NSException User Info Dictionary Keys (p. 103)
Regular Expression Options (p. 95)

captureCount

Returns the number of captures that *regex* contains.

```
- (NSInteger)captureCount;
```

Discussion
Returns the capture count of the receiver, which should be a valid regular expression. For example:

```
NSInteger regexCaptureCount = [@"(\\d+)\.(\\d+)" captureCount];
// regexCaptureCount would be set to 2.
```

Return Value
Returns `-1` if an error occurs. Otherwise the number of captures in *regex* is returned, or `0` if *regex* does not contain any captures.

Availability
Available in RegexKit*Lite* 3.0 and later.

See Also
- `- captureCountWithOptions:error:` (p. 68)

captureCountWithOptions:error:

Returns the number of captures that *regex* contains.

```
- (NSInteger)captureCountWithOptions:(RKLRegexOptions)options error:(NSError **)error;
```

Discussion
The optional *error* parameter, if set and an error occurs, will contain a `NSError` object that describes the problem. This may be set to `NULL` if information about any errors is not required.

Returns the capture count of the receiver, which should be a valid regular expression. For example:

```
NSInteger regexCaptureCount = [@"(\\d+)\.(\\d+)" captureCountWithOptions:RKLNoOptions
                                                                   error:NULL];
// regexCaptureCount would be set to 2.
```

Return Value
Returns `-1` if an error occurs. Otherwise the number of captures in *`regex`* is returned, or `0` if *`regex`* does not contain any captures.

Availability
Available in RegexKit*Lite* 3.0 and later.

See Also
- `captureCount` (p. 68)
RegexKitLite NSError Error Domains (p. 103)
RegexKitLite NSError and NSException User Info Dictionary Keys (p. 103)
Regular Expression Options (p. 95)

componentsMatchedByRegex:

Returns an array containing all the substrings from the receiver that were matched by the regular expression *`regex`*.

```
- (NSArray *)componentsMatchedByRegex:(NSString *)regex;
```

Return Value
A `NSArray` object containing all the substrings from the receiver that were matched by *`regex`*.

Availability
Available in RegexKit*Lite* 3.0 and later.

See Also
- `componentsMatchedByRegex:capture:` (p. 69)
- `componentsMatchedByRegex:range:` (p. 70)
- `componentsMatchedByRegex:options:range:capture:error:` (p. 70)

componentsMatchedByRegex:capture:

Returns an array containing all the substrings from the receiver that were matched by capture number *`capture`* from the regular expression *`regex`*.

```
- (NSArray *)componentsMatchedByRegex:(NSString *)regex capture:(NSInteger)capture;
```

Return Value
A `NSArray` object containing all the substrings for capture group *`capture`* from the receiver that were matched by *`regex`*.

Discussion
An array index will contain an empty string, or `@""`, if the capture group did not match any text.

Availability
Available in RegexKit*Lite* 3.0 and later.

See Also

- componentsMatchedByRegex: (p. 69)
- componentsMatchedByRegex:range: (p. 70)
- componentsMatchedByRegex:options:range:capture:error: (p. 70)

componentsMatchedByRegex:range:

Returns an array containing all the substrings from the receiver that were matched by the regular expression *regex* within *range*.

```
- (NSArray *)componentsMatchedByRegex:(NSString *)regex range:(NSRange)range;
```

Return Value
A NSArray object containing all the substrings from the receiver that were matched by *regex* within *range*.

Availability
Available in RegexKit*Lite* 3.0 and later.

See Also
- componentsMatchedByRegex: (p. 69)
- componentsMatchedByRegex:capture: (p. 69)
- componentsMatchedByRegex:options:range:capture:error: (p. 70)

componentsMatchedByRegex:options:range:capture:error:

Returns an array containing all the substrings from the receiver that were matched by capture number *capture* from the regular expression *regex* within *range* using *options*.

```
- (NSArray *)componentsMatchedByRegex:(NSString *)regex
    options:(RKLRegexOptions)options range:(NSRange)range capture:(NSInteger)capture
    error:(NSError **)error;
```

Parameters
regex
: A NSString containing a regular expression.

options
: A mask of options specified by combining RKLRegexOptions (p. 95) flags with the C bitwise OR operator. Either 0 or RKLNoOptions (p. 96) may be used if no options are required.

range
: The range of the receiver to search.

capture
: The string matched by *capture* from *regex* to return. Use 0 for the entire string that *regex* matched.

error
: An optional parameter that if set and an error occurs, will contain a NSError object that describes the problem. This may be set to NULL if information about any errors is not required.

Return Value
A NSArray object containing all the substrings from the receiver that were matched by capture number *capture* from *regex* within *range* using *options*.

Discussion
If the receiver is not matched by *regex* then the returned value is a NSArray that contains no items.

An array index will contain an empty string, or `@""`, if a capture group did not match any text.

The match results in the array appear in the order they did in the receiver.

Example:

```
NSString *list      = @"$10.23, $1024.42, $3099";
NSArray  *listItems = [list componentsMatchedByRegex:@"\\$((\\d+)(?:\\.(\\d+)|\\.?))"];

// listItems == [NSArray arrayWithObjects:@"$10.23", @"$1024.42", @"$3099", NULL];
```

Example of extracting a specific capture group:

```
NSString *list      = @"$10.23, $1024.42, $3099";
NSRange   listRange = NSMakeRange(0UL, [list length]);
NSArray  *listItems = [list componentsMatchedByRegex:@"\\$((\\d+)(?:\\.(\\d+)|\\.?))"
                                             options:RKLNoOptions
                                               range:listRange
                                             capture:3L
                                               error:NULL];

// listItems == [NSArray arrayWithObjects:@"23", @"42", @"", NULL];
```

Availability
Available in RegexKit*Lite* 3.0 and later.

See Also
- `componentsMatchedByRegex:` (p. 69)
- `componentsMatchedByRegex:capture:` (p. 69)
- `componentsMatchedByRegex:range:` (p. 70)

RegexKitLite NSError Error Domains (p. 103)
RegexKitLite NSError and NSException User Info Dictionary Keys (p. 103)
Regular Expression Options (p. 95)

componentsSeparatedByRegex:

Returns an array containing substrings from the receiver that have been divided by the regular expression *regex*.

```
- (NSArray *)componentsSeparatedByRegex:(NSString *)regex;
```

Return Value
A `NSArray` object containing the substrings from the receiver that have been divided by *regex*.

Availability
Available in RegexKit*Lite* 2.0 and later.

See Also
- `componentsSeparatedByRegex:range:` (p. 72)
- `componentsSeparatedByRegex:options:range:error:` (p. 72)

componentsSeparatedByRegex:range:

Returns an array containing substrings within *range* of the receiver that have been divided by the regular expression *regex*.

```
- (NSArray *)componentsSeparatedByRegex:(NSString *)regex range:(NSRange)range;
```

Return Value
A `NSArray` object containing the substrings from the receiver that have been divided by *regex*.

Availability
Available in RegexKit*Lite* 2.0 and later.

See Also
- `componentsSeparatedByRegex:` (p. 71)
- `componentsSeparatedByRegex:options:range:error:` (p. 72)

componentsSeparatedByRegex:options:range:error:

Returns an array containing substrings within *range* of the receiver that have been divided by the regular expression *regex* using *options*.

```
- (NSArray *)componentsSeparatedByRegex:(NSString *)regex
    options:(RKLRegexOptions)options range:(NSRange)range error:(NSError **)error;
```

Parameters
regex
: A `NSString` containing a regular expression.

options
: A mask of options specified by combining `RKLRegexOptions` (p. 95) flags with the C bitwise OR operator. Either `0` or `RKLNoOptions` (p. 96) may be used if no options are required.

range
: The range of the receiver to search.

error
: An optional parameter that if set and an error occurs, will contain a `NSError` object that describes the problem. This may be set to `NULL` if information about any errors is not required.

Return Value
A `NSArray` object containing the substrings from the receiver that have been divided by *regex*.

Discussion
The substrings in the array appear in the order they did in the receiver. For example, this code fragment:

```
NSString *list      = @"Norman, Stanley, Fletcher";
NSArray  *listItems = [list componentsSeparatedByRegex:@",\\s*"];
```

produces an array `{ @"Norman", @"Stanley", @"Fletcher" }`.

If the receiver begins or ends with *regex*, then the first or last substring is, respectively, empty. For example, the string `", Norman, Stanley, Fletcher"` creates an array that has these contents: `{ @"", @"Norman", @"Stanley", @"Fletcher" }`.

If the receiver has no separators that are matched by *regex*– for example, `"Norman"`– the array contains the string itself, in this case `{ @"Norman" }`.

If *regex* contains capture groups– for example, `@",(\\s*)"`– the array will contain the text matched by each capture group as a separate element appended to the normal result. An additional element will be

created for each capture group. If an individual capture group does not match any text the result in the array will be a zero length string– `@""`. As an example– the regular expression `@"(\\s*)"` would produce the array `{ @"Norman", @" ", @"Stanley", @" ", @"Fletcher" }`.

Availability
Available in RegexKit*Lite* 2.0 and later.

See Also
- `componentsSeparatedByRegex:` (p. 71)
- `componentsSeparatedByRegex:range:` (p. 72)

RegexKitLite NSError Error Domains (p. 103)
RegexKitLite NSError and NSException User Info Dictionary Keys (p. 103)
Regular Expression Options (p. 95)

dictionaryByMatchingRegex:withKeysAndCaptures:

Creates and returns a dictionary containing the matches constructed from the specified set of *keys* and *captures* for the first match of *regex* in the receiver.

```
- (NSDictionary *)dictionaryByMatchingRegex:(NSString *)regex
    withKeysAndCaptures:(id)firstKey, ...;
```

Parameters

regex
: A `NSString` containing a regular expression.

firstKey
: The first key to add to the new dictionary.

...
: First the *capture* for *firstKey*, then a `NULL`-terminated list of alternating *keys* and *captures*. *Captures* are specified using `int` values.

> **Important:** Use of non-`int` sized *capture* arguments will result in undefined behavior. **Do not** append *capture* arguments with a `L` suffix.

> **Important:** Failure to `NULL`-terminate the *keys* and *captures* list will result in undefined behavior.

Return Value
A `NSDictionary` containing the matched substrings constructed from the specified set of *keys* and *captures*.

Discussion
The returned value is for the first match of *regex* in the receiver.

If the receiver is not matched by *regex* then the returned value is a `NSDictionary` that contains no items.

A dictionary will not contain a given key if its corresponding capture group did not match any text.

Availability
Available in RegexKit*Lite* 4.0 and later.

See Also
- `dictionaryByMatchingRegex:range:withKeysAndCaptures:` (p. 74)
- `dictionaryByMatchingRegex:options:range:error:withKeysAndCaptures:` (p. 75)

RegexKitLite NSError Error Domains (p. 103)
RegexKitLite NSError and NSException User Info Dictionary Keys (p. 103)
Regular Expression Options (p. 95)

dictionaryByMatchingRegex:range:withKeysAndCaptures:

Creates and returns a dictionary containing the matches constructed from the specified set of *keys* and *captures* for the first match of *regex* within *range* of the receiver.

```
- (NSDictionary *)dictionaryByMatchingRegex:(NSString *)regex range:(NSRange)range
    withKeysAndCaptures:(id)firstKey, ...;
```

Parameters

regex
: A `NSString` containing a regular expression.

range
: The range of the receiver to search.

firstKey
: The first key to add to the new dictionary.

...
: First the *capture* for *firstKey*, then a `NULL`-terminated list of alternating *keys* and *captures*. *Captures* are specified using `int` values.

> **Important:** Use of non-`int` sized *capture* arguments will result in undefined behavior. **Do not** append *capture* arguments with a `L` suffix.

> **Important:** Failure to `NULL`-terminate the *keys* and *captures* list will result in undefined behavior.

Return Value
A `NSDictionary` containing the matched substrings constructed from the specified set of *keys* and *captures*.

Discussion
The returned value is for the first match of *regex* in the receiver.

If the receiver is not matched by *regex* then the returned value is a `NSDictionary` that contains no items.

A dictionary will not contain a given key if its corresponding capture group did not match any text.

Availability
Available in RegexKit*Lite* 4.0 and later.

See Also
- `dictionaryByMatchingRegex:withKeysAndCaptures:` (p. 73)
- `dictionaryByMatchingRegex:options:range:error:withKeysAndCaptures:` (p. 75)
RegexKitLite NSError Error Domains (p. 103)
RegexKitLite NSError and NSException User Info Dictionary Keys (p. 103)
Regular Expression Options (p. 95)

dictionaryByMatchingRegex:options:range:error:withKeysAndCaptures:

Creates and returns a dictionary containing the matches constructed from the specified set of *keys* and *captures* for the first match of *regex* within *range* of the receiver using *options*.

```
- (NSDictionary *)dictionaryByMatchingRegex:(NSString *)regex
    options:(RKLRegexOptions)options range:(NSRange)range error:(NSError **)error
    withKeysAndCaptures:(id)firstKey, ...;
```

Parameters

regex

A `NSString` containing a regular expression.

options

A mask of options specified by combining `RKLRegexOptions` (p. 95) flags with the C bitwise OR operator. Either `0` or `RKLNoOptions` (p. 96) may be used if no options are required.

range

The range of the receiver to search.

error

An optional parameter that if set and an error occurs, will contain a `NSError` object that describes the problem. This may be set to `NULL` if information about any errors is not required.

firstKey

The first key to add to the new dictionary.

...

First the *capture* for *firstKey*, then a `NULL`-terminated list of alternating *keys* and *captures*. *Captures* are specified using `int` values.

> **Important:** Use of non-`int` sized *capture* arguments will result in undefined behavior. **Do not** append *capture* arguments with a `L` suffix.

> **Important:** Failure to `NULL`-terminate the *keys* and *captures* list will result in undefined behavior.

Return Value

A `NSDictionary` containing the matched substrings constructed from the specified set of *keys* and *captures*.

Discussion

The returned value is for the first match of *regex* in the receiver.

If the receiver is not matched by *regex* then the returned value is a `NSDictionary` that contains no items.

A dictionary will not contain a given key if its corresponding capture group did not match any text. It is important to note that a regular expression can successfully match zero characters:

```
NSString     *name           = @"Name: Joe";
NSString     *regex          = @"Name:\\s*(\\w*)\\s*(\\w*)";
NSDictionary *nameDictionary =
                [name dictionaryByMatchingRegex:regex
                                        options:RKLNoOptions
                                          range:NSMakeRange(0UL, [name length])
                                          error:NULL
                          withKeysAndCaptures:@"first", 1, @"last", 2, NULL];

// 2010-01-29 12:19:54.559 RegexKitLite[64944:a0f] nameDictionary: {
```

```
//     first = Joe;
//     last = "";
// }
```

Compared to this example, where the second capture group does not match any characters:

```
NSString     *name           = @"Name: Joe";
NSString     *regex          = @"Name:\\s*(\\w*)\\s*(\\w+)?";
NSDictionary *nameDictionary =
                [name dictionaryByMatchingRegex:regex
                                        options:RKLNoOptions
                                          range:NSMakeRange(0UL, [name length])
                                          error:NULL
                           withKeysAndCaptures:@"first", 1, @"last", 2, NULL];

// 2010-01-29 12:12:52.177 RegexKitLite[64893:a0f] nameDictionary: {
//     first = Joe;
// }
```

Availability
Available in RegexKit*Lite* 4.0 and later.

See Also
- `- dictionaryByMatchingRegex:withKeysAndCaptures:` (p. 73)
- `- dictionaryByMatchingRegex:range:withKeysAndCaptures:` (p. 74)

RegexKitLite NSError Error Domains (p. 103)
RegexKitLite NSError and NSException User Info Dictionary Keys (p. 103)
Regular Expression Options (p. 95)

enumerateStringsMatchedByRegex:usingBlock:

Enumerates the matches in the receiver by the regular expression `regex` and executes `block` for each match found.

```
- (BOOL)enumerateStringsMatchedByRegex:(NSString *)regex
    usingBlock:(void (^)(NSInteger captureCount,
                         NSString * const capturedStrings[captureCount],
                         const NSRange capturedRanges[captureCount],
                         volatile BOOL * const stop))block;
```

Parameters

`regex`

A `NSString` containing a regular expression.

`block`

The block that is executed for each match of `regex` in the receiver. The block takes four arguments:

Parameters

`captureCount`

The number of strings that `regex` captured. `captureCount` is always at least `1`.

`capturedStrings`

An array containing the substrings matched by each capture group present in `regex`. The size of the array is `captureCount`. If a capture group did not match anything, it will contain

a pointer to a string that is equal to `@""`. This argument may be `NULL` if *`enumerationOptions`* had `RKLRegexEnumerationCapturedStringsNotRequired` (p. 98) set.

`capturedRanges`
An array containing the ranges matched by each capture group present in *`regex`*. The size of the array is *`captureCount`*. If a capture group did not match anything, it will contain a `NSRange` equal to `{NSNotFound, 0}`.

`stop`
A reference to a `BOOL` value that the block can use to stop the enumeration by setting `*stop = YES;`, otherwise it should not touch `*stop`.

Return Value
Returns `YES` if there was no error, otherwise returns `NO`.

Availability
Available in RegexKit*Lite* 4.0 and later.

See Also
- `enumerateStringsMatchedByRegex:options:inRange:error:enumerationOptions:usingBlock:` (p. 77)

RegexKitLite NSString Additions Reference - Block-based Enumeration Methods (p. 51)
Blocks Programming Topics

enumerateStringsMatchedByRegex:options:inRange:error: enumerationOptions:usingBlock:

Enumerates the matches in the receiver by the regular expression *`regex`* within *`range`* using *`options`* and executes *`block`* using *`enumerationOptions`* for each match found.

```
- (BOOL)enumerateStringsMatchedByRegex:(NSString *)regex
    options:(RKLRegexOptions)options inRange:(NSRange)range error:(NSError **)error
    enumerationOptions:(RKLRegexEnumerationOptions)enumerationOptions
    usingBlock:(void (^)(NSInteger captureCount,
                         NSString * const capturedStrings[captureCount],
                         const NSRange capturedRanges[captureCount],
                         volatile BOOL * const stop))block;
```

Parameters
`regex`
A `NSString` containing a regular expression.

`options`
A mask of options specified by combining `RKLRegexOptions` (p. 95) flags with the C bitwise OR operator. Either `0` or `RKLNoOptions` (p. 96) may be used if no options are required.

`range`
The range of the receiver to search.

`error`
An optional parameter that if set and an error occurs, will contain a `NSError` object that describes the problem. This may be set to `NULL` if information about any errors is not required.

`enumerationOptions`
A mask of options specified by combining `RKLRegexEnumerationOptions` (p. 97) flags with the C bitwise OR operator. Either `0` or `RKLRegexEnumerationNoOptions` (p. 98) may be used if no options are required.

block
: The block that is executed for each match of *regex* in the receiver. The block takes four arguments:

Parameters

captureCount
: The number of strings that *regex* captured. *captureCount* is always at least 1.

capturedStrings
: An array containing the substrings matched by each capture group present in *regex*. The size of the array is *captureCount*. If a capture group did not match anything, it will contain a pointer to a string that is equal to @"". This argument may be NULL if *enumerationOptions* had RKLRegexEnumerationCapturedStringsNotRequired (p. 98) set.

capturedRanges
: An array containing the ranges matched by each capture group present in *regex*. The size of the array is *captureCount*. If a capture group did not match anything, it will contain a NSRange equal to {NSNotFound, 0}.

stop
: A reference to a BOOL value that the block can use to stop the enumeration by setting **stop* = YES;, otherwise it should not touch **stop*.

Return Value
Returns YES if there was no error, otherwise returns NO and indirectly returns a NSError object if *error* is not NULL.

Availability
Available in RegexKit*Lite* 4.0 and later.

See Also
- enumerateStringsMatchedByRegex:usingBlock: (p. 76)
RegexKitLite NSError Error Domains (p. 103)
RegexKitLite NSError and NSException User Info Dictionary Keys (p. 103)
Regular Expression Options (p. 95)
Regular Expression Enumeration Options (p. 97)
RegexKitLite NSString Additions Reference - Block-based Enumeration Methods (p. 51)
Blocks Programming Topics

enumerateStringsSeparatedByRegex:usingBlock:

Enumerates the strings of the receiver that have been divided by the regular expression *regex* and executes *block* for each divided string.

```
- (BOOL)enumerateStringsSeparatedByRegex:(NSString *)regex
    usingBlock:(void (^)(NSInteger captureCount,
                         NSString * const capturedStrings[captureCount],
                         const NSRange capturedRanges[captureCount],
                         volatile BOOL * const stop))block;
```

Parameters

regex
: A NSString containing a regular expression.

block
: The block that is executed for each match of *regex* in the receiver. The block takes four arguments:

Parameters

captureCount
: The number of strings that *regex* captured. *captureCount* is always at least 1.

capturedStrings
: An array containing the substrings matched by each capture group present in *regex*. The size of the array is *captureCount*. If a capture group did not match anything, it will contain a pointer to a string that is equal to @"". This argument may be NULL if *enumerationOptions* had RKLRegexEnumerationCapturedStringsNotRequired (p. 98) set.

capturedRanges
: An array containing the ranges matched by each capture group present in *regex*. The size of the array is *captureCount*. If a capture group did not match anything, it will contain a NSRange equal to {NSNotFound, 0}.

stop
: A reference to a BOOL value that the block can use to stop the enumeration by setting *stop = YES;, otherwise it should not touch *stop.

Return Value
Returns YES if there was no error, otherwise returns NO.

Availability
Available in RegexKit*Lite* 4.0 and later.

See Also
- enumerateStringsSeparatedByRegex:options:inRange:error:enumerationOptions:usingBlock: (p. 79)
- componentsSeparatedByRegex: (p. 71)

RegexKitLite NSString Additions Reference - Block-based Enumeration Methods (p. 51)
Blocks Programming Topics

enumerateStringsSeparatedByRegex:options:inRange:error: enumerationOptions:usingBlock:

Enumerates the strings of the receiver that have been divided by the regular expression *regex* within *range* using *options* and executes *block* using *enumerationOptions* for each divided string.

```
- (BOOL)enumerateStringsSeparatedByRegex:(NSString *)regex
    options:(RKLRegexOptions)options inRange:(NSRange)range error:(NSError **)error
    enumerationOptions:(RKLRegexEnumerationOptions)enumerationOptions
    usingBlock:(void (^)(NSInteger captureCount,
                         NSString * const capturedStrings[captureCount],
                         const NSRange capturedRanges[captureCount],
                         volatile BOOL * const stop))block;
```

Parameters
regex
: A NSString containing a regular expression.

options
: A mask of options specified by combining RKLRegexOptions (p. 95) flags with the C bitwise OR operator. Either 0 or RKLNoOptions (p. 96) may be used if no options are required.

range
: The range of the receiver to search.

error
: An optional parameter that if set and an error occurs, will contain a NSError object that describes

the problem. This may be set to `NULL` if information about any errors is not required.

enumerationOptions
: A mask of options specified by combining `RKLRegexEnumerationOptions` (p. 97) flags with the C bitwise OR operator. Either `0` or `RKLRegexEnumerationNoOptions` (p. 98) may be used if no options are required.

block
: The block that is executed for each match of *regex* in the receiver. The block takes four arguments:

Parameters

captureCount
: The number of strings that *regex* captured. *captureCount* is always at least `1`.

capturedStrings
: An array containing the substrings matched by each capture group present in *regex*. The size of the array is *captureCount*. If a capture group did not match anything, it will contain a pointer to a string that is equal to `@""`. This argument may be `NULL` if *enumerationOptions* had `RKLRegexEnumerationCapturedStringsNotRequired` (p. 98) set.

capturedRanges
: An array containing the ranges matched by each capture group present in *regex*. The size of the array is *captureCount*. If a capture group did not match anything, it will contain a `NSRange` equal to `{NSNotFound, 0}`.

stop
: A reference to a `BOOL` value that the block can use to stop the enumeration by setting `*stop = YES;`, otherwise it should not touch `*stop`.

Return Value
Returns `YES` if there was no error, otherwise returns `NO` and indirectly returns a `NSError` object if *error* is not `NULL`.

Availability
Available in RegexKit*Lite* 4.0 and later.

See Also
- `enumerateStringsSeparatedByRegex:usingBlock:` (p. 78)
- `componentsSeparatedByRegex:options:range:error:` (p. 72)

RegexKitLite NSError Error Domains (p. 103)
RegexKitLite NSError and NSException User Info Dictionary Keys (p. 103)
Regular Expression Options (p. 95)
Regular Expression Enumeration Options (p. 97)
RegexKitLite NSString Additions Reference - Block-based Enumeration Methods (p. 51)
Blocks Programming Topics

flushCachedRegexData

Clears any cached information about the receiver.

```
- (void)flushCachedRegexData;
```

Discussion
This method should be used when performing searches on `NSMutableString` objects and there is the possibility that the string has mutated in between calls to RegexKit*Lite*.

This method clears the cached information for the receiver **only**. This is more selective than `clearStringCache` (p. 61), which clears all the cached information from RegexKit*Lite*, including all the cached compiled regular expressions.

RegexKit*Lite* automatically detects the vast majority of string mutations and clears any cached information for the mutated string. To detect mutations, RegexKit*Lite* records a strings `length` and `hash` value at the point in time when it caches data for a string. Cached data for a string is invalidated if either of these values change between calls to RegexKit*Lite*. The problem case is when a string is mutated but the strings length remains the same **and** the hash value for the mutated string is identical to the hash value of the string before it was mutated. This is known as a *hash collision*. Since RegexKit*Lite* is unable to detect that a string has mutated when this happens, the programmer needs to explicitly inform RegexKit*Lite* that any cached data about the receiver needs to be cleared by sending `flushCachedRegexData` to the mutated string.

While it is possible to have "perfect mutation detection", and therefore guarantee that only valid cached data is used, it has a significant performance penalty. The first problem is that when caching information about a string, an immutable copy of that string needs to be made. The second problem is that determining that two strings are not identical is usually very fast and cheap– if their lengths are not the same, no further checks are required. The most expensive case is when two strings are identical because it requires a character by character comparison of the entire string to guarantee that they are equal. The most expensive case also happens to be the most common case, by far. To make matters worst, Cocoa provides no public way to determine if an instance is a mutable `NSMutableString` or an immutable `NSString` object. Therefore RegexKit*Lite* must assume the worst case that all strings are mutable and have potentially mutated between calls to RegexKit*Lite*.

RegexKit*Lite* is optimized for the common case which is when regular expression operations are performed on strings that are not mutating. The majority of mutations to a string can be quickly and cheaply detected by RegexKit*Lite* automatically. Since the programmer has the context of the string that is to be matched, and whether or not the string is being mutated, RegexKit*Lite* relies on the programmer to inform it whether or not the possibility exists that the string could have mutated in a way that is undetectable.

An example of clearing a strings cached information:

```
NSMutableString *mutableSearchString; // Assumed to be valid.
NSString *foundString = [mutableSearchString stringByMatching:@"\\d+"]; // Searched..

[mutableSearchString replaceCharactersInRange:NSMakeRange(5UL, 10UL)
                                   withString:@"[replaced]"]; // Mutated..

[mutableSearchString flushCachedRegexData]; // Clear cached information
                                            // about mutableSearchString.
```

Warning: Failure to clear the cached information for a `NSMutableString` object that has mutated between searches may result in undefined behavior.

Note: You do not need to call `clearStringCache` (p. 61) or `flushCachedRegexData` (p. 80) when using the `NSMutableString` `replaceOccurrencesOfRegex:withString:` (p. 87) methods. The cached information for that `NSMutableString` is automatically cleared as necessary.

Availability
Available in RegexKit*Lite* 3.0 and later.

See Also
`+ clearStringCache` (p. 61)
NSString RegexKitLite Additions Reference - Cached Information and Mutable Strings (p. 50)

isMatchedByRegex:

Returns a Boolean value that indicates whether the receiver is matched by *regex*.

```
- (BOOL)isMatchedByRegex:(NSString *)regex;
```

Availability
Available in RegexKit*Lite* 1.0 and later.

See Also
- `isMatchedByRegex:inRange:` (p. 82)
- `isMatchedByRegex:options:inRange:error:` (p. 82)

isMatchedByRegex:inRange:

Returns a Boolean value that indicates whether the receiver is matched by *regex* within *range*.

```
- (BOOL)isMatchedByRegex:(NSString *)regex inRange:(NSRange)range;
```

Availability
Available in RegexKit*Lite* 1.0 and later.

See Also
- `isMatchedByRegex:` (p. 82)
- `isMatchedByRegex:options:inRange:error:` (p. 82)

isMatchedByRegex:options:inRange:error:

Returns a Boolean value that indicates whether the receiver is matched by *regex* within *range*.

```
- (BOOL)isMatchedByRegex:(NSString *)regex options:(RKLRegexOptions)options
    inRange:(NSRange)range error:(NSError **)error;
```

Discussion
The optional *error* parameter, if set and an error occurs, will contain a `NSError` object that describes the problem. This may be set to `NULL` if information about any errors is not required.

Availability
Available in RegexKit*Lite* 1.0 and later.

See Also
- `isMatchedByRegex:` (p. 82)
- `isMatchedByRegex:inRange:` (p. 82)

RegexKitLite NSError Error Domains (p. 103)
RegexKitLite NSError and NSException User Info Dictionary Keys (p. 103)
Regular Expression Options (p. 95)

isRegexValid

Returns a Boolean value that indicates whether the regular expression contained in the receiver is valid.

```
- (BOOL)isRegexValid;
```

Availability
Available in RegexKit*Lite* 3.0 and later.

See Also
- `isRegexValidWithOptions:error:` (p. 83)

isRegexValidWithOptions:error:

Returns a Boolean value that indicates whether the regular expression contained in the receiver is valid using *options*.

```
- (BOOL)isRegexValidWithOptions:(RKLRegexOptions)options error:(NSError **)error;
```

Parameters
options
A mask of options specified by combining `RKLRegexOptions` (p. 95) flags with the C bitwise OR operator. Either `0` or `RKLNoOptions` (p. 96) may be used if no options are required.

error
An optional parameter that if set and an error occurs, will contain a `NSError` object that describes the problem. This may be set to `NULL` if information about any errors is not required.

Discussion
This method can be used to determine if a regular expression is valid. For example:

```
NSError  *error       = NULL;
NSString *regexString = @"[a-z"; // Missing the closing ]
if([regexString isRegexValidWithOptions:RKLNoOptions error:&error] == NO) {
  NSLog(@"The regular expression is invalid. Error: %@", error);
}
```

Availability
Available in RegexKit*Lite* 3.0 and later.

See Also
- `isRegexValid` (p. 82)
RegexKitLite NSError Error Domains (p. 103)
RegexKitLite NSError and NSException User Info Dictionary Keys (p. 103)
Regular Expression Options (p. 95)

rangeOfRegex:

Returns the range for the first match of *regex* in the receiver.

```
- (NSRange)rangeOfRegex:(NSString *)regex;
```

Return Value
A `NSRange` structure giving the location and length of the first match of *regex* in the receiver. Returns `{NSNotFound, 0}` if the receiver is not matched by *regex* or an error occurs.

Availability
Available in RegexKit*Lite* 1.0 and later.

See Also
- `rangeOfRegex:capture:` (p. 84)

- rangeOfRegex:inRange: (p. 84)
- rangeOfRegex:options:inRange:capture:error: (p. 84)

rangeOfRegex:capture:

Returns the range of capture number *capture* for the first match of *regex* in the receiver.

```
- (NSRange)rangeOfRegex:(NSString *)regex capture:(NSInteger)capture;
```

Return Value
A NSRange structure giving the location and length of capture number *capture* for the first match of *regex* in the receiver. Returns {NSNotFound, 0} if the receiver is not matched by *regex* or an error occurs.

Availability
Available in RegexKit*Lite* 1.0 and later.

See Also
- rangeOfRegex: (p. 83)
- rangeOfRegex:inRange: (p. 84)
- rangeOfRegex:options:inRange:capture:error: (p. 84)

rangeOfRegex:inRange:

Returns the range for the first match of *regex* within *range* of the receiver.

```
- (NSRange)rangeOfRegex:(NSString *)regex inRange:(NSRange)range;
```

Return Value
A NSRange structure giving the location and length of the first match of *regex* within *range* of the receiver. Returns {NSNotFound, 0} if the receiver is not matched by *regex* within *range* or an error occurs.

Availability
Available in RegexKit*Lite* 1.0 and later.

See Also
- rangeOfRegex: (p. 83)
- rangeOfRegex:capture: (p. 84)
- rangeOfRegex:options:inRange:capture:error: (p. 84)

rangeOfRegex:options:inRange:capture:error:

Returns the range of capture number *capture* for the first match of *regex* within *range* of the receiver.

```
- (NSRange)rangeOfRegex:(NSString *)regex options:(RKLRegexOptions)options
    inRange:(NSRange)range capture:(NSInteger)capture error:(NSError **)error;
```

Parameters
regex
A NSString containing a regular expression.

options
A mask of options specified by combining RKLRegexOptions (p. 95) flags with the C bitwise OR operator. Either 0 or RKLNoOptions (p. 96) may be used if no options are required.

range
: The range of the receiver to search.

capture
: The matching range of the capture number from *regex* to return. Use `0` for the entire range that *regex* matched.

error
: An optional parameter that if set and an error occurs, will contain a `NSError` object that describes the problem. This may be set to `NULL` if information about any errors is not required.

Return Value
A `NSRange` structure giving the location and length of capture number *capture* for the first match of *regex* within *range* of the receiver. Returns `{NSNotFound, 0}` if the receiver is not matched by *regex* within *range* or an error occurs.

Availability
Available in RegexKit*Lite* 1.0 and later.

See Also
- `rangeOfRegex:` (p. 83)
- `rangeOfRegex:capture:` (p. 84)
- `rangeOfRegex:inRange:` (p. 84)

RegexKitLite NSError Error Domains (p. 103)
RegexKitLite NSError and NSException User Info Dictionary Keys (p. 103)
Regular Expression Options (p. 95)

replaceOccurrencesOfRegex:usingBlock:

Enumerates the matches in the receiver by the regular expression *regex* and executes *block* for each match found. Replaces the characters that were matched with the contents of the string returned by *block*, returning the number of replacements made.

```
- (NSInteger)replaceOccurrencesOfRegex:(NSString *)regex
    usingBlock:(NSString *(^)(NSInteger captureCount,
                              NSString * const capturedStrings[captureCount],
                              const NSRange capturedRanges[captureCount],
                              volatile BOOL * const stop))block;
```

Parameters
regex
: A `NSString` containing a regular expression.

block
: The block that is executed for each match of *regex* in the receiver. The block takes four arguments:

  Parameters
  captureCount
  : The number of strings that *regex* captured. *captureCount* is always at least `1`.

  capturedStrings
  : An array containing the substrings matched by each capture group present in *regex*. The size of the array is *captureCount*. If a capture group did not match anything, it will contain a pointer to a string that is equal to `@""`. This argument may be `NULL` if *enumerationOptions* had `RKLRegexEnumerationCapturedStringsNotRequired` (p. 98) set.

  capturedRanges
  : An array containing the ranges matched by each capture group present in *regex*. The size

of the array is `captureCount`. If a capture group did not match anything, it will contain a `NSRange` equal to `{NSNotFound, 0}`.

`stop`

A reference to a `BOOL` value that the block can use to stop the enumeration by setting `*stop = YES`; otherwise it should not touch `*stop`.

Discussion
This method modifies the receivers contents. An exception will be raised if it is sent to an immutable object.

Return Value
Returns `-1` if there was an error, otherwise returns the number of replacements performed.

Availability
Available in RegexKit*Lite* 4.0 and later.

See Also
`- replaceOccurrencesOfRegex:options:inRange:error:enumerationOptions:usingBlock:` (p. 86)
RegexKitLite NSString Additions Reference - Block-based Enumeration Methods (p. 51)
Blocks Programming Topics

replaceOccurrencesOfRegex:options:inRange:error: enumerationOptions:usingBlock:

Enumerates the matches in the receiver by the regular expression `regex` within `range` using `options` and executes `block` using `enumerationOptions` for each match found. Replaces the characters that were matched with the contents of the string returned by `block`, returning the number of replacements made.

```
- (NSInteger)replaceOccurrencesOfRegex:(NSString *)regex
    options:(RKLRegexOptions)options inRange:(NSRange)range error:(NSError **)error
    enumerationOptions:(RKLRegexEnumerationOptions)enumerationOptions
    usingBlock:(NSString *(^)(NSInteger captureCount,
                              NSString * const capturedStrings[captureCount],
                              const NSRange capturedRanges[captureCount],
                              volatile BOOL * const stop))block;
```

Parameters
`regex`

A `NSString` containing a regular expression.

`options`

A mask of options specified by combining `RKLRegexOptions` (p. 95) flags with the C bitwise OR operator. Either `0` or `RKLNoOptions` (p. 96) may be used if no options are required.

`range`

The range of the receiver to search.

`error`

An optional parameter that if set and an error occurs, will contain a `NSError` object that describes the problem. This may be set to `NULL` if information about any errors is not required.

`enumerationOptions`

A mask of options specified by combining `RKLRegexEnumerationOptions` (p. 97) flags with the C bitwise OR operator. Either `0` or `RKLRegexEnumerationNoOptions` (p. 98) may be used if no options are required.

`block`

The block that is executed for each match of `regex` in the receiver. The block takes four arguments:

Parameters

captureCount
: The number of strings that *regex* captured. *captureCount* is always at least 1.

capturedStrings
: An array containing the substrings matched by each capture group present in *regex*. The size of the array is *captureCount*. If a capture group did not match anything, it will contain a pointer to a string that is equal to @"". This argument may be NULL if *enumerationOptions* had RKLRegexEnumerationCapturedStringsNotRequired (p. 98) set.

capturedRanges
: An array containing the ranges matched by each capture group present in *regex*. The size of the array is *captureCount*. If a capture group did not match anything, it will contain a NSRange equal to {NSNotFound, 0}.

stop
: A reference to a BOOL value that the block can use to stop the enumeration by setting **stop = YES*; otherwise it should not touch **stop*.

Discussion
This method modifies the receivers contents. An exception will be raised if it is sent to an immutable object.

Return Value
Returns -1 if there was an error and indirectly returns a NSError object if *error* is not NULL, otherwise returns the number of replacements performed.

Availability
Available in RegexKit*Lite* 4.0 and later.

See Also
- replaceOccurrencesOfRegex:usingBlock: (p. 85)
RegexKitLite NSError Error Domains (p. 103)
RegexKitLite NSError and NSException User Info Dictionary Keys (p. 103)
Regular Expression Options (p. 95)
Regular Expression Enumeration Options (p. 97)
RegexKitLite NSString Additions Reference - Block-based Enumeration Methods (p. 51)
Blocks Programming Topics

replaceOccurrencesOfRegex:withString:

Replaces all occurrences of the regular expression *regex* with the contents of *replacement* string after performing capture group substitutions, returning the number of replacements made.

```
- (NSInteger)replaceOccurrencesOfRegex:(NSString *)regex
    withString:(NSString *)replacement;
```

Important: Raises RKLICURegexException (p. 103) if *replacement* contains $*n* capture references where *n* is greater than the number of capture groups in the regular expression *regex*.

Discussion
This method modifies the receivers contents. An exception will be raised if it is sent to an immutable object.

Return Value
Returns -1 if there was an error, otherwise returns the number of replacements performed.

Availability

Available in RegexKit*Lite* 2.0 and later.

See Also
- `replaceOccurrencesOfRegex:withString:range:` (p. 88)
- `replaceOccurrencesOfRegex:withString:options:range:error:` (p. 88)

ICU Replacement Text Syntax (p. 33)

replaceOccurrencesOfRegex:withString:range:

Replaces all occurrences of the regular expression *regex* within *range* with the contents of *replacement* string after performing capture group substitutions, returning the number of replacements made.

```
- (NSInteger)replaceOccurrencesOfRegex:(NSString *)regex
    withString:(NSString *)replacement range:(NSRange)range;
```

Important: Raises `RKLICURegexException` (p. 103) if *replacement* contains `$`*n* capture references where *n* is greater than the number of capture groups in the regular expression *regex*.

Discussion
This method modifies the receivers contents. An exception will be raised if it is sent to an immutable object.

Return Value
Returns `-1` if there was an error, otherwise returns the number of replacements performed.

Availability
Available in RegexKit*Lite* 2.0 and later.

See Also
- `replaceOccurrencesOfRegex:withString:` (p. 87)
- `replaceOccurrencesOfRegex:withString:options:range:error:` (p. 88)

ICU Replacement Text Syntax (p. 33)

replaceOccurrencesOfRegex:withString:options:range:error:

Replaces all occurrences of the regular expression *regex* using *options* within *range* with the contents of *replacement* string after performing capture group substitutions, returning the number of replacements made.

```
- (NSInteger)replaceOccurrencesOfRegex:(NSString *)regex
    options:(RKLRegexOptions)options withString:(NSString *)replacement
    range:(NSRange)range error:(NSError **)error;
```

Parameters

regex
: A `NSString` containing a regular expression.

options
: A mask of options specified by combining `RKLRegexOptions` (p. 95) flags with the C bitwise OR operator. Either `0` or `RKLNoOptions` (p. 96) may be used if no options are required.

range
: The range of the receiver to search.

replacement
: The string to use as the replacement text for matches by *regex*. See *ICU Replacement Text Syntax* (p. 33) for more information.

> **Important:** Raises `RKLICURegexException` (p. 103) if *replacement* contains `$`*n* capture references where *n* is greater than the number of capture groups in the regular expression *regex*.

error
: An optional parameter that if set and an error occurs, will contain a `NSError` object that describes the problem. This may be set to `NULL` if information about any errors is not required.

Discussion
This method modifies the receivers contents. An exception will be raised if it is sent to an immutable object.

Return Value
Returns `-1` if there was an error and indirectly returns a `NSError` object if *error* is not `NULL`, otherwise returns the number of replacements performed.

Availability
Available in RegexKit*Lite* 2.0 and later.

See Also
- `replaceOccurrencesOfRegex:withString:` (p. 87)
- `replaceOccurrencesOfRegex:withString:range:` (p. 88)

ICU Replacement Text Syntax (p. 33)
RegexKitLite NSError Error Domains (p. 103)
RegexKitLite NSError and NSException User Info Dictionary Keys (p. 103)
Regular Expression Options (p. 95)

stringByMatching:

Returns a string created from the characters of the receiver that are in the range of the first match of *regex*.

```
- (NSString *)stringByMatching:(NSString *)regex;
```

Return Value
A `NSString` containing the substring of the receiver matched by *regex*. Returns `NULL` if the receiver is not matched by *regex* or an error occurs.

Availability
Available in RegexKit*Lite* 1.0 and later.

See Also
- `stringByMatching:capture:` (p. 89)
- `stringByMatching:inRange:` (p. 90)
- `stringByMatching:options:inRange:capture:error:` (p. 90)

stringByMatching:capture:

Returns a string created from the characters of the receiver that are in the range of the first match of *regex* for *capture*.

```
- (NSString *)stringByMatching:(NSString *)regex capture:(NSInteger)capture;
```

Return Value
A `NSString` containing the substring of the receiver matched by capture number *`capture`* of *`regex`*. Returns `NULL` if the receiver is not matched by *`regex`* or an error occurs.

Availability
Available in RegexKit*Lite* 1.0 and later.

See Also
- `stringByMatching:` (p. 89)
- `stringByMatching:inRange:` (p. 90)
- `stringByMatching:options:inRange:capture:error:` (p. 90)

stringByMatching:inRange:

Returns a string created from the characters of the receiver that are in the range of the first match of *`regex`* within *`range`* of the receiver.

```
- (NSString *)stringByMatching:(NSString *)regex inRange:(NSRange)range;
```

Return Value
A `NSString` containing the substring of the receiver matched by *`regex`* within *`range`* of the receiver. Returns `NULL` if the receiver is not matched by *`regex`* within *`range`* or an error occurs.

Availability
Available in RegexKit*Lite* 1.0 and later.

See Also
- `stringByMatching:` (p. 89)
- `stringByMatching:capture:` (p. 89)
- `stringByMatching:options:inRange:capture:error:` (p. 90)

stringByMatching:options:inRange:capture:error:

Returns a string created from the characters of the receiver that are in the range of the first match of *`regex`* using *`options`* within *`range`* of the receiver for *`capture`*.

```
- (NSString *)stringByMatching:(NSString *)regex options:(RKLRegexOptions)options
    inRange:(NSRange)range capture:(NSInteger)capture error:(NSError **)error;
```

Parameters
`regex`
: A `NSString` containing a regular expression.

`options`
: A mask of options specified by combining `RKLRegexOptions` (p. 95) flags with the C bitwise OR operator. Either `0` or `RKLNoOptions` (p. 96) may be used if no options are required.

`range`
: The range of the receiver to search.

`capture`
: The string matched by *`capture`* from *`regex`* to return. Use `0` for the entire string that *`regex`* matched.

`error`
: An optional parameter that if set and an error occurs, will contain a `NSError` object that describes the problem. This may be set to `NULL` if information about any errors is not required.

Return Value
A `NSString` containing the substring of the receiver matched by capture number *`capture`* of *`regex`* within *`range`* of the receiver. Returns `NULL` if the receiver is not matched by *`regex`* within *`range`* or an error occurs.

Availability
Available in RegexKit*Lite* 1.0 and later.

See Also
- `- stringByMatching:` (p. 89)
- `- stringByMatching:capture:` (p. 89)
- `- stringByMatching:inRange:` (p. 90)

RegexKitLite NSError Error Domains (p. 103)
RegexKitLite NSError and NSException User Info Dictionary Keys (p. 103)
Regular Expression Options (p. 95)

stringByReplacingOccurrencesOfRegex:usingBlock:

Enumerates the matches in the receiver by the regular expression *`regex`* and executes *`block`* for each match found. Returns a string created by replacing the characters that were matched in the receiver with the contents of the string returned by *`block`*.

```
- (NSString *)stringByReplacingOccurrencesOfRegex:(NSString *)regex
    usingBlock:(NSString *(^)(NSInteger captureCount,
                              NSString * const capturedStrings[captureCount],
                              const NSRange capturedRanges[captureCount],
                              volatile BOOL * const stop))block;
```

Parameters
`regex`
A `NSString` containing a regular expression.

`block`
The block that is executed for each match of *`regex`* in the receiver. The block takes four arguments:

Parameters
`captureCount`
The number of strings that *`regex`* captured. *`captureCount`* is always at least `1`.

`capturedStrings`
An array containing the substrings matched by each capture group present in *`regex`*. The size of the array is *`captureCount`*. If a capture group did not match anything, it will contain a pointer to a string that is equal to `@""`. This argument may be `NULL` if *`enumerationOptions`* had `RKLRegexEnumerationCapturedStringsNotRequired` (p. 98) set.

`capturedRanges`
An array containing the ranges matched by each capture group present in *`regex`*. The size of the array is *`captureCount`*. If a capture group did not match anything, it will contain a `NSRange` equal to `{NSNotFound, 0}`.

`stop`
A reference to a `BOOL` value that the block can use to stop the enumeration by setting `*stop = YES`; otherwise it should not touch *`*stop`*.

Return Value
A `NSString` created from the characters of the receiver in which all matches of the regular expression *`regex`* are replaced with the contents of the `NSString` returned by *`block`*. If the receiver is not matched by *`regex`* then the string that is returned is a copy of the receiver as if `stringWithString:` had been sent to it.

Returns `NULL` if there was an error.

Availability
Available in RegexKit*Lite* 4.0 and later.

See Also
- `stringByReplacingOccurrencesOfRegex:options:inRange:error:enumerationOptions:usingBlock:` (p. 92)
RegexKitLite NSString Additions Reference - Block-based Enumeration Methods (p. 51)
Blocks Programming Topics

stringByReplacingOccurrencesOfRegex:options:inRange:error:enumerationOptions:usingBlock:

Enumerates the matches in the receiver by the regular expression *regex* within *range* using *options* and executes *block* using *enumerationOptions* for each match found. Returns a string created by replacing the characters that were matched in the receiver with the contents of the string returned by *block*.

```
- (NSString *)stringByReplacingOccurrencesOfRegex:(NSString *)regex
    options:(RKLRegexOptions)options inRange:(NSRange)range error:(NSError **)error
    enumerationOptions:(RKLRegexEnumerationOptions)enumerationOptions
    usingBlock:(NSString *(^)(NSInteger captureCount,
                              NSString * const capturedStrings[captureCount],
                              const NSRange capturedRanges[captureCount],
                              volatile BOOL * const stop))block;
```

Parameters
regex
A `NSString` containing a regular expression.

options
A mask of options specified by combining `RKLRegexOptions` (p. 95) flags with the C bitwise OR operator. Either `0` or `RKLNoOptions` (p. 96) may be used if no options are required.

range
The range of the receiver to search.

error
An optional parameter that if set and an error occurs, will contain a `NSError` object that describes the problem. This may be set to `NULL` if information about any errors is not required.

enumerationOptions
A mask of options specified by combining `RKLRegexEnumerationOptions` (p. 97) flags with the C bitwise OR operator. Either `0` or `RKLRegexEnumerationNoOptions` (p. 98) may be used if no options are required.

block
The block that is executed for each match of *regex* in the receiver. The block takes four arguments:

Parameters
captureCount
The number of strings that *regex* captured. *captureCount* is always at least `1`.

capturedStrings
An array containing the substrings matched by each capture group present in *regex*. The size of the array is *captureCount*. If a capture group did not match anything, it will contain a pointer to a string that is equal to `@""`. This argument may be `NULL` if *enumerationOptions* had `RKLRegexEnumerationCapturedStringsNotRequired` (p. 98) set.

capturedRanges
: An array containing the ranges matched by each capture group present in *regex*. The size of the array is *captureCount*. If a capture group did not match anything, it will contain a NSRange equal to {NSNotFound, 0}.

stop
: A reference to a BOOL value that the block can use to stop the enumeration by setting *stop = YES; otherwise it should not touch *stop.

Return Value
A NSString created from the characters within *range* of the receiver in which all matches of the regular expression *regex* using *options* are replaced with the contents of the NSString returned by *block*. Returns the characters within *range* as if substringWithRange: had been sent to the receiver if the substring is not matched by *regex*.

Returns NULL if there was an error and indirectly returns a NSError object if *error* is not NULL.

Availability
Available in RegexKit*Lite* 4.0 and later.

See Also
- - stringByReplacingOccurrencesOfRegex:usingBlock: (p. 91)
- *RegexKitLite NSError Error Domains* (p. 103)
- *RegexKitLite NSError and NSException User Info Dictionary Keys* (p. 103)
- *Regular Expression Options* (p. 95)
- *Regular Expression Enumeration Options* (p. 97)
- *RegexKitLite NSString Additions Reference - Block-based Enumeration Methods* (p. 51)
- *Blocks Programming Topics*

stringByReplacingOccurrencesOfRegex:withString:

Returns a string created from the characters of the receiver in which all matches of the regular expression *regex* are replaced with the contents of the *replacement* string after performing capture group substitutions.

```
- (NSString *)stringByReplacingOccurrencesOfRegex:(NSString *)regex
    withString:(NSString *)replacement;
```

Important: Raises RKLICURegexException (p. 103) if *replacement* contains $*n* capture references where *n* is greater than the number of capture groups in the regular expression *regex*.

Return Value
A NSString created from the characters of the receiver in which all matches of the regular expression *regex* are replaced with the contents of the *replacement* string after performing capture group substitutions. If the receiver is not matched by *regex* then the string that is returned is a copy of the receiver as if stringWithString: had been sent to it.

Returns NULL if there was an error.

Availability
Available in RegexKit*Lite* 2.0 and later.

See Also
- - stringByReplacingOccurrencesOfRegex:withString:range: (p. 94)
- - stringByReplacingOccurrencesOfRegex:withString:options:range:error: (p. 94)

ICU Replacement Text Syntax (p. 33)

stringByReplacingOccurrencesOfRegex:withString:range:

Returns a string created from the characters within *range* of the receiver in which all matches of the regular expression *regex* are replaced with the contents of the *replacement* string after performing capture group substitutions.

```
- (NSString *)stringByReplacingOccurrencesOfRegex:(NSString *)regex
    withString:(NSString *)replacement range:(NSRange)range;
```

Important: Raises `RKLICURegexException` (p. 103) if *replacement* contains `$n` capture references where *n* is greater than the number of capture groups in the regular expression *regex*.

Return Value
A `NSString` created from the characters within *range* of the receiver in which all matches of the regular expression *regex* are replaced with the contents of the *replacement* string after performing capture group substitutions. Returns the characters within *range* as if `substringWithRange:` had been sent to the receiver if the substring is not matched by *regex*.

Returns `NULL` if there was an error.

Availability
Available in RegexKit*Lite* 2.0 and later.

See Also
- `stringByReplacingOccurrencesOfRegex:withString:` (p. 93)
- `stringByReplacingOccurrencesOfRegex:withString:options:range:error:` (p. 94)

ICU Replacement Text Syntax (p. 33)

stringByReplacingOccurrencesOfRegex:withString:options:range:error:

Returns a string created from the characters within *range* of the receiver in which all matches of the regular expression *regex* using *options* are replaced with the contents of the *replacement* string after performing capture group substitutions.

```
- (NSString *)stringByReplacingOccurrencesOfRegex:(NSString *)regex
    options:(RKLRegexOptions)options withString:(NSString *)replacement
    range:(NSRange)range error:(NSError **)error;
```

Parameters

regex
: A `NSString` containing a regular expression.

options
: A mask of options specified by combining `RKLRegexOptions` (p. 95) flags with the C bitwise OR operator. Either `0` or `RKLNoOptions` (p. 96) may be used if no options are required.

replacement
: The string to use as the replacement text for matches by *regex*. See *ICU Replacement Text Syntax* (p. 33) for more information.

> **Important:** Raises `RKLICURegexException` (p. 103) if *`replacement`* contains `$`*`n`* capture references where *`n`* is greater than the number of capture groups in the regular expression *`regex`*.

`range`
The range of the receiver to search.

`error`
An optional parameter that if set and an error occurs, will contain a `NSError` object that describes the problem. This may be set to `NULL` if information about any errors is not required.

Return Value
A `NSString` created from the characters within *`range`* of the receiver in which all matches of the regular expression *`regex`* using *`options`* are replaced with the contents of the *`replacement`* string after performing capture group substitutions. Returns the characters within *`range`* as if `substringWithRange:` had been sent to the receiver if the substring is not matched by *`regex`*.

Returns `NULL` if there was an error and indirectly returns a `NSError` object if *`error`* is not `NULL`.

Availability
Available in RegexKit*Lite* 2.0 and later.

See Also
- `- stringByReplacingOccurrencesOfRegex:withString:` (p. 93)
- `- stringByReplacingOccurrencesOfRegex:withString:range:` (p. 94)

ICU Replacement Text Syntax (p. 33)
RegexKitLite NSError Error Domains (p. 103)
RegexKitLite NSError and NSException User Info Dictionary Keys (p. 103)
Regular Expression Options (p. 95)

Constants

RKLRegexOptions

Type for regular expression options.

```
typedef uint32_t RKLRegexOptions;
```

Discussion
See *Regular Expression Options* (p. 95) for possible values.

Availability
Available in RegexKit*Lite* 1.0 and later.

Declared In
`RegexKitLite.h`

Regular Expression Options

The following flags control various aspects of regular expression matching. The flag values may be specified at the time that a regular expression is used, or they may be specified within the pattern itself using the `(?ismwx-ismwx)` pattern options.

```
enum {
```

```
    RKLNoOptions             = 0,
    RKLCaseless              = 2,
    RKLComments              = 4,
    RKLDotAll                = 32,
    RKLMultiline             = 8,
    RKLUnicodeWordBoundaries = 256
};
```

Constants

`RKLNoOptions`
No regular expression options specified.
Available in RegexKit*Lite* 1.0 and later.

`RKLCaseless`
If set, matching will take place in a case-insensitive manner.
Available in RegexKit*Lite* 1.0 and later.

`RKLComments`
If set, allow use of *`white space`* and `#comments` within patterns.
Available in RegexKit*Lite* 1.0 and later.

`RKLDotAll`
If set, a `.` in a pattern will match a *`line terminator`* in the input text. By default, it will not. Note that a *`carriage-return`* / *`line-feed`* pair in text behave as a single line terminator, and will match a single `.` (period) in a regular expression pattern.
Available in RegexKit*Lite* 1.0 and later.

`RKLMultiline`
Control the behavior of `^` and `$` in a pattern. By default these will only match at the start and end, respectively, of the input text. If this flag is set, `^` and `$` will also match at the start and end of each line within the input text.
Available in RegexKit*Lite* 1.0 and later.

`RKLUnicodeWordBoundaries`
Controls the behavior of `\b` in a pattern. If set, word boundaries are found according to the definitions of word found in *Unicode UAX 29 - Text Boundaries*. By default, word boundaries are identified by means of a simple classification of characters as either *`word`* or *`non-word`*, which approximates traditional regular expression behavior. The results obtained with the two options can be quite different in runs of *`spaces`* and other *`non-word`* characters.
Available in RegexKit*Lite* 1.0 and later.

Discussion

Options for controlling the behavior of a regular expression pattern can be controlled in two ways. When the method supports it, options may specified by combining `RKLRegexOptions` (p. 95) flags with the C bitwise OR operator. For example:

```
matchedRange = [aString rangeOfRegex:@"^(blue|red)$"
                             options:(RKLCaseless | RKLMultiline)
                             inRange:range
                               error:NULL];
```

The other way is to specify the options within the regular expression itself, of which there are two ways. The first specifies the options for everything following it, and the other sets the options on a per capture group basis. Options are either *enabled*, or following a `-`, *disabled*. The syntax for both is nearly identical:

Option	Example	Description
`(?ixsmw-ixsmw)...`	`(?i)...`	Enables the `RKLCaseless` option for everything that follows it. Useful at the beginning of a regular expression to set the desired options.
`(?ixsmw-ixsmw:...)`	`(?iw-m:...)`	Enables the `RKLCaseless` and `RKLUnicodeWordBoundaries` options and disables `RKLMultiline` for the capture group enclosed by the parenthesis.

The following table lists the regular expression pattern option character and its corresponding `RKLRegexOptions` flag:

Character	Option
i	`RKLCaseless`
x	`RKLComments`
s	`RKLDotAll`
m	`RKLMultiline`
w	`RKLUnicodeWordBoundaries`

Availability
Available in RegexKit*Lite* 1.0 and later.

See Also
ICU Regular Expression Syntax (p. 25)
ICU User Guide - Regular Expressions

Declared In
`RegexKitLite.h`

RKLRegexEnumerationOptions

Type for regular expression enumeration options.

```
typedef NSUInteger RKLRegexEnumerationOptions;
```

Discussion
See *Regular Expression Enumeration Options* (p. 97) for possible values.

> **Important:** `RKLRegexEnumerationOptions` is only available if `RKL_BLOCKS` is set.

Availability
Available in RegexKit*Lite* 4.0 and later.

Declared In
`RegexKitLite.h`

Regular Expression Enumeration Options

The following flags control various aspects of regular expression enumeration.

```
enum {
   RKLRegexEnumerationNoOptions                                = 0UL,
   RKLRegexEnumerationCapturedStringsNotRequired               = 1UL << 9,
   RKLRegexEnumerationReleaseStringReturnedByReplacementBlock = 1UL << 10,
   RKLRegexEnumerationFastCapturedStringsXXX                   = 1UL << 11
};
```

Constants

`RKLRegexEnumerationNoOptions`

No regular expression enumeration options specified.
Available in RegexKit*Lite* 4.0 and later.

`RKLRegexEnumerationCapturedStringsNotRequired`

If set, this indicates that the block does not require the captured strings and `NULL` will be passed to the blocks *`capturedStrings`*`[]` argument.

> **Important:** Methods will raise `NSInvalidArgumentException` if `RKLRegexEnumerationCapturedStringsNotRequired` (p. 98) and `RKLRegexEnumerationFastCapturedStringsXXX` (p. 98) are both set.

Available in RegexKit*Lite* 4.0 and later.

`RKLRegexEnumerationReleaseStringReturnedByReplacementBlock`

If set, RegexKit*Lite* will send the string returned by the block a `release` message. If garbage collection is active, this flag does nothing. This flag is only valid for the `stringByReplacingOccurrencesOfRegex:...` and `replaceOccurrencesOfRegex:...` groups of methods. See *Regular Expression Enumeration Options - Using RKLRegexEnumerationReleaseStringReturnedByReplacementBlock* (p. 99) for more information.

> **Important:** Methods will raise `NSInvalidArgumentException` if `RKLRegexEnumerationReleaseStringReturnedByReplacementBlock` (p. 98) is set and the block enumeration method is not a 'Search and Replace' method.

Available in RegexKit*Lite* 4.0 and later.

`RKLRegexEnumerationFastCapturedStringsXXX`

If set, RegexKit*Lite* will create the captured strings that are passed to the blocks *`capturedStrings`*`[]` argument in a special way. The enumeration identifier ends in `...XXX` to act as a visual reminder that special care must be taken. See *Regular Expression Enumeration Options - Using RKLRegexEnumerationFastCapturedStringsXXX* (p. 100) for more information.

> **Important:** Methods will raise `NSInvalidArgumentException` if `RKLRegexEnumerationCapturedStringsNotRequired` (p. 98) and `RKLRegexEnumerationFastCapturedStringsXXX` (p. 98) are both set.

> **Caution:** `RKLRegexEnumerationFastCapturedStringsXXX` (p. 98) is an advanced feature option. ***Do not*** use this option unless you fully understand the consequences. Incorrect usage of this option will result in undefined behavior and will probably result in your program crashing.

Available in RegexKit*Lite* 4.0 and later.

Discussion

> **Important:** `RKLRegexEnumerationOptions` is only available if `RKL_BLOCKS` is set.

When the method supports it, enumeration options may specified by combining `RKLRegexEnumerationOptions` (p. 97) flags with the C bitwise OR operator. For example:

```
RKLRegexEnumerationOptions enumerationOptions =
                        (RKLRegexEnumerationCapturedStringsNotRequired |
                         RKLRegexEnumerationReleaseStringReturnedByReplacementBlock);
```

A typical use of `stringByReplacingOccurrencesOfRegex:` might be something like the following:

```
replacedString =
   [aString stringByReplacingOccurrencesOfRegex:@"\\w+"
                                        options:RKLNoOptions
                                        inRange:NSMakeRange(0UL, [aString length])
                                          error:NULL
                                enumerationOptions:RKLRegexEnumerationNoOptions
                                        usingBlock:
                                       ^(NSInteger captureCount,
                                         NSString * const capturedStrings[captureCount],
                                         const NSRange capturedRanges[captureCount],
                                         volatile BOOL * const stop) {
  return((NSString *)[NSString stringWithFormat:@"<%@>", capturedStrings[0]]);
}];
```

> **Note:** The above example casts the value returned by `stringWithFormat:`, which has a return type of `id`, to `NSString *` because of a bug in `gcc 4.2.1-5646`. Without the cast, the compiler gives an error that the block does not return a compatible type. `clang` does not require a cast in order to compile.

An alternate way to work around the `gcc` bug is to explicitly declare the return type of the block. The example below shows how this is done with the changes highlighted:

```
stringByReplacingOccurrencesOfRegex:… usingBlock:
                        ^NSString *(NSInteger captureCount,
                                    NSString * const capturedStrings[captureCount],
                                    const NSRange capturedRanges[captureCount],
                                    volatile BOOL * const stop) {
  return(/*(NSString *)*/[NSString stringWithFormat:@"<%@>", capturedStrings[0]]);
}];
```

Using `RKLRegexEnumerationReleaseStringReturnedByReplacementBlock`

The use of convenience methods like `stringWithFormat:` can create a lot of temporary strings that are autoreleased. The memory used to hold these temporary strings won't be reclaimed until the autorelease pool that they are in is released. When performing a replace on a large string, this can end up being a considerable amount of memory. If this is becomes a problem in your application, the `RKLRegexEnumerationReleaseStringReturnedByReplacementBlock` enumeration option can be used to reclaim the memory used by these temporary strings immediately. `RKLRegexEnumerationReleaseStringReturnedByReplacementBlock` essentially transfers ownership of the string that is returned by the block to RegexKit*Lite*, which then becomes responsible for sending the string a `release` message when it is no longer needed. Typically this is right after RegexKit*Lite* has appended the contents of the returned string to the temporary internal string of all the accumulated replacement strings returned by the block. For example:

```
replacedString = [aString stringByReplacingOccurrencesOfRegex:@"\\w+"
                                                       options:RKLNoOptions
```

```
                    inRange:NSMakeRange(0UL, [aString length])
                      error:NULL
         enumerationOptions:RKLRegexEnumerationReleaseStringReturnedByReplacementBlock
                 usingBlock:
                    ^NSString *(NSInteger captureCount,
                                NSString * const capturedStrings[captureCount],
                                const NSRange capturedRanges[captureCount],
                                volatile BOOL * const stop) {
  return([[NSString alloc] initWithFormat:@"<%@>", capturedStrings[0]]);
}];
```

When using `RKLRegexEnumerationReleaseStringReturnedByReplacementBlock`, it is also possible to directly return the strings that were passed via *`capturedStrings`*`[]`. RegexKit*Lite* automatically checks if the pointer of the string returned by the block is equal to the pointer of any of the strings it created and passed via *`capturedStrings`*`[]`, in which case RegexKit*Lite* does not send the string that was returned a `release` message since this would result in over releasing the string object. For example:

```
replacedString = [aString stringByReplacingOccurrencesOfRegex:@"\\w+"
                    options:RKLNoOptions
                    inRange:NSMakeRange(0UL, [aString length])
                      error:NULL
         enumerationOptions:RKLRegexEnumerationReleaseStringReturnedByReplacementBlock
                 usingBlock:
                    ^NSString *(NSInteger captureCount,
                                NSString * const capturedStrings[captureCount],
                                const NSRange capturedRanges[captureCount],
                                volatile BOOL * const stop) {
  if([capturedStrings[0] isEqual:@"with"]) { return(capturedStrings[0]); }
  else { return([[NSString alloc] initWithFormat:@"<%@>", capturedStrings[0]]); }
}];
```

Using `RKLRegexEnumerationFastCapturedStringsXXX`

Normally, RegexKit*Lite* instantiates a new string object for each string passed via *`capturedStrings`*`[]`. When `RKLRegexEnumerationFastCapturedStringsXXX` is set, RegexKit*Lite* creates up to *`captureCount`* strings using the *Core Foundation* function `CFStringCreateMutableWithExternalCharactersNoCopy` and passes these strings via *`capturedStrings`*`[]`.

> **Caution:** `RKLRegexEnumerationFastCapturedStringsXXX` is an advanced feature option. ***Do not*** use this option unless you fully understand the consequences. Incorrect usage of this option will result in undefined behavior and will probably result in your program crashing.

The reason for doing this is strings created with `CFStringCreateMutableWithExternalCharactersNoCopy` can have the strings buffer set directly with `CFStringSetExternalCharactersNoCopy`. This allows RegexKit*Lite* to quickly update the strings that were captured by a match and is ***much*** faster than instantiating new strings for each match iteration since the only thing that needs to be updated is the captured strings pointer and length. In fact, the speed of `RKLRegexEnumerationFastCapturedStringsXXX` is nearly has fast as `RKLRegexEnumerationCapturedStringsNotRequired`.

Special care must be taken when using the strings passed via *`capturedStrings`*`[]` when `RKLRegexEnumerationFastCapturedStringsXXX` has been set. The following rules must be followed:

- The lifetime of the string is the duration of the blocks execution. The string ***can not*** be used or referenced

past the closing brace of the block.

- ***Do not*** mutate any of `capturedStrings[]`.
- ***Do not*** `retain` or `release` any of the `capturedStrings[]`.
- If you need the string to exist past the end of the block, a `-copy` of the string must be made:

```
NSString *copiedString = [[capturedStrings[0] copy] autorelease];
```

Example usages:

- When using `capturedStrings[]` as an argument to `stringWithFormat:`, you do not need to make a copy of the captured string:

```
[searchString stringByReplacingOccurrencesOfRegex:regex
                  usingBlock:
                     ^NSString *(NSInteger captureCount,
                                 NSString * const capturedStrings[captureCount],
                                 const NSRange capturedRanges[captureCount],
                                 volatile BOOL * const stop) {
  NSString *replacedString = NULL;
  // An example of when it is safe to use capturedStrings[] directly.
  replacedString = [NSString stringWithFormat:@"{1: '%@'}", capturedStrings[1]];
  return(replacedString);
}];
```

- Adding to a collection:

```
// Do not add a captured string directly to a collection.
[aDictionary setObject:capturedStrings[0] forKey:@"aKey"];

// Do add a copy of the captured string to a collection.
[aDictionary setObject:[[capturedStrings[0] copy] autorelease] forKey:@"aKey"];
```

- Assigning a captured string to a variable whose scope extends past the blocks:

```
__block NSString *foundString = NULL;

[searchString enumerateStringsMatchedByRegex:regex
                                  usingBlock:
                                 ^(NSInteger captureCount,
                                   NSString * const capturedStrings[captureCount],
                                   const NSRange capturedRanges[captureCount],
                                   volatile BOOL * const stop) {

  // Do not assign a captured string to a variable
  //        whose scope extends past the Blocks.
  foundString = capturedStrings[0];

  // Do assign a copy of the captured string to a
  //    variable whose scope extends past the Blocks.
  foundString = [[capturedStrings[0] copy] autorelease];
}];
```

Availability
Available in RegexKit*Lite* 4.0 and later.

See Also
RegexKitLite NSString Additions Reference - Block-based Enumeration Methods (p. 51)
Blocks Programming Topics

Declared In
`RegexKitLite.h`

RegexKitLite:::utf16ConversionCache *arg1* Flags

The following flags are used to indicate the status of the result from *arg1 lookupResultFlags* of the `RegexKitLite:::utf16ConversionCache` DTrace probe point.

```
enum {
   RKLCacheHitLookupFlag           = 1 << 0,
   RKLConversionRequiredLookupFlag = 1 << 1,
   RKLSetTextLookupFlag            = 1 << 2,
   RKLDynamicBufferLookupFlag      = 1 << 3,
   RKLErrorLookupFlag              = 1 << 4,
   RKLEnumerationBufferLookupFlag  = 1 << 5
};
```

Constants

`RKLCacheHitLookupFlag`
If set, there was a successful lookup hit. This flag is only set if `RKLConversionRequiredLookupFlag` is also set.
Available in RegexKit*Lite* 3.0 and later.

`RKLConversionRequiredLookupFlag`
If set, direct access to a strings `UTF-16` buffer was not available and a conversion was required.
Available in RegexKit*Lite* 3.0 and later.

`RKLSetTextLookupFlag`
If set, the ICU `uregex_setText()` function was called in order to set the compiled regular expression to a buffer.
Available in RegexKit*Lite* 3.0 and later.

`RKLDynamicBufferLookupFlag`
If set, the strings size was large enough to require the use of the dynamically sized conversion buffer.
Available in RegexKit*Lite* 3.0 and later.

`RKLErrorLookupFlag`
If set, there was some kind of error during the `UTF-16` conversion process.
Available in RegexKit*Lite* 3.0 and later.

`RKLEnumerationBufferLookupFlag`
If set, the probe was fired due to a blocks enumeration. The conversion result is one time use only and is only available to the block enumeration that required it.
Available in RegexKit*Lite* 4.0 and later.

See Also
`RegexKitLite:::utf16ConversionCache` (p. 59)

RegexKit*Lite* NSError Error Domains

The following `NSError` error domains are defined.

```
extern NSString * const RKLICURegexErrorDomain;
```

Constants

`RKLICURegexErrorDomain`
ICU Regular Expression Errors.

Availability

Available in RegexKit*Lite* 1.0 and later.

See Also

NSError Class Reference
RegexKitLite NSError and NSException User Info Dictionary Keys (p. 103)

Declared In

`RegexKitLite.h`

RegexKit*Lite* NSException Exception Names

The following `NSException` exception names are defined.

```
extern NSString * const RKLICURegexException;
```

Constants

`RKLICURegexException`
ICU Regular Expression Exceptions

Discussion

Returns a user info dictionary populated with keys as defined in *RegexKitLite NSError and NSException User Info Dictionary Keys* (p. 103).

Availability

Available in RegexKit*Lite* 2.0 and later.

See Also

NSException Class Reference
RegexKitLite NSError and NSException User Info Dictionary Keys (p. 103)

Declared In

`RegexKitLite.h`

RegexKit*Lite* NSError and NSException User Info Dictionary Keys

```
extern NSString * const RKLICURegexEnumerationOptionsErrorKey;
extern NSString * const RKLICURegexErrorCodeErrorKey;
extern NSString * const RKLICURegexErrorNameErrorKey;
extern NSString * const RKLICURegexLineErrorKey;
extern NSString * const RKLICURegexOffsetErrorKey;
extern NSString * const RKLICURegexPreContextErrorKey;
extern NSString * const RKLICURegexPostContextErrorKey;
extern NSString * const RKLICURegexRegexErrorKey;
```

```
extern NSString * const RKLICURegexRegexOptionsErrorKey;
extern NSString * const RKLICURegexReplacedCountErrorKey;
extern NSString * const RKLICURegexReplacedStringErrorKey;
extern NSString * const RKLICURegexReplacementStringErrorKey;
extern NSString * const RKLICURegexSubjectRangeErrorKey;
extern NSString * const RKLICURegexSubjectStringErrorKey;
```

Constants

`RKLICURegexEnumerationOptionsErrorKey`
The `RKLRegexEnumerationOptions` (p. 97) enumeration options specified.
Available in RegexKit*Lite* 4.0 and later.

`RKLICURegexErrorCodeErrorKey`
The error code returned by the ICU library function calls *`status`* argument.
Available in RegexKit*Lite* 2.0 and later.

`RKLICURegexErrorNameErrorKey`
The string returned by invoking `u_errorName` with the error code `RKLICURegexErrorCodeErrorKey`.
Example: `U_REGEX_RULE_SYNTAX`.
Available in RegexKit*Lite* 1.0 and later.

`RKLICURegexLineErrorKey`
The line number where the error occurred in the regular expression.
Available in RegexKit*Lite* 1.0 and later.

`RKLICURegexOffsetErrorKey`
The offset from the beginning of the line where the error occurred in the regular expression.
Available in RegexKit*Lite* 1.0 and later.

`RKLICURegexPreContextErrorKey`
Up to `16` characters leading up to the cause of the error in the regular expression.
Available in RegexKit*Lite* 1.0 and later.

`RKLICURegexPostContextErrorKey`
Up to `16` characters after the cause error in the regular expression.
Available in RegexKit*Lite* 1.0 and later.

`RKLICURegexRegexErrorKey`
The regular expression that caused the error.
Available in RegexKit*Lite* 1.0 and later.

`RKLICURegexRegexOptionsErrorKey`
The `RKLRegexOptions` regular expression options specified.
Available in RegexKit*Lite* 1.0 and later.

`RKLICURegexReplacedCountErrorKey`
The number of times a replacement was performed.
Available in RegexKit*Lite* 4.0 and later.

`RKLICURegexReplacedStringErrorKey`
The replaced string result after performing the search and replace operations.
Available in RegexKit*Lite* 4.0 and later.

`RKLICURegexReplacementStringErrorKey`
The replacement template string used to perform search and replace operations.
Available in RegexKit*Lite* 4.0 and later.

`RKLICURegexSubjectRangeErrorKey`
The range of the subject string that was being matched by the regular expression.
Available in RegexKit*Lite* 4.0 and later.

`RKLICURegexSubjectStringErrorKey`
The string that was being matched by the regular expression.
Available in RegexKit*Lite* 4.0 and later.

Discussion
Different error keys will be present depending on the type of error encountered.

See Also
NSError Class Reference
NSException Class Reference
RegexKitLite NSError Error Domains (p. 103)
RegexKitLite NSException Exception Names (p. 103)

Declared In
`RegexKitLite.h`

Release Information

1.0 - 2008/03/23

Initial release.

1.1 - 2008/03/28

Changes:

- Added the method `clearStringCache` (p. 61).
- Updated the documentation with information about searching `NSMutableString` strings, and when to use `clearStringCache` (p. 61).

Bug fixes:

- Fixed a bug that for strings that required `UTF-16` conversion, the conversion from the previous string that required conversion may have been re-used for the current string, even though the two strings are different and the new string requires conversion.
- Updated the internal inconsistency exception macro to correctly handle non-`ASCII` file names.

1.2 - 2008/04/01

Changes:

- Updated and clarified the documentation regarding adding RegexKit*Lite* to an Xcode project.
- Created Xcode 3 DocSet documentation.

2.0 - 2008/07/07

New features:

- The ability to split a string in to a `NSArray` with a regular expression.
- Search and replace using common `$`*n* capture substitution in the replacement string.
- Support for Leopards Garbage Collection

Changes:

- A new compile time configuration option, `RKL_FAST_MUTABLE_CHECK`, enables the use of a private, non-public *Core Foundation* function to determine if the string that will be searched is immutable. This allows many of the mutation safety checks to be skipped for that string and can significantly improve the performance of repeated matches on the same immutable `NSString`. For safety, this option is disabled by default.

- Gives a warning at compile time if the minimum Mac OS X target is < 10.3, which is the first version that shipped with the ICU library.
- Compile time tunable setting of how much of the stack can be used before switching to heap based storage. Changed by setting `RKL_STACK_LIMIT` to the maximum number of bytes to use from the stack. Defaults to `131072`, or `128Kbytes`. By default in Cocoa the main thread has a stack size of `8Mbytes` and additional threads have a stack size of `512Kbytes`.
- New `NSException` name for RegexKit*Lite* specific exceptions, `RKLICURegexException` (p. 103).
- Garbage Collection is safe to use in "mixed-mode" Garbage Collection applications. When compiled with Garbage Collection support (`-fobjc-gc`), RegexKit*Lite* will dynamically select either Garbage Collection or manual memory management at run-time depending on whether or not Garbage Collection is enabled and active.

New `NSString` Methods:

- `- (NSArray *)componentsSeparatedByRegex:(NSString *)regex;`
- `- (NSString *)stringByReplacingOccurrencesOfRegex:(NSString *)regex withString:(NSString *)replacement;`

New `NSMutableString` Methods:

- `- (NSUInteger)replaceOccurrencesOfRegex:(NSString *)regex withString:(NSString *)replacement;`

2.1 - 2008/07/12

Bug fixes:

- Fixed a bug dealing with the `UTF-16` conversion cache. A bug could cause the text in the conversion buffer to be incorrectly used for the wrong string.

2.2 - 2008/10/19

This release contains several large documentation additions and a few bug fixes. No new major functionality was added.

Documentation Changes:

- Added a section covering *ICU Regular Expression Character Classes* (p. 28)
- Added a section covering *Unicode Properties* (p. 31), which are specified in a regular expression with `\p{`*Unicode Property Name*`}` and `\P{`*Unicode Property Name*`}`.
- Added the *RegexKitLite Cookbook* (p. 34) section. Special *Copy to Clipboard* functionality is enabled for Safari users that allows a regular expression in this document that is selected and *Copied to the Clipboard* to be automatically escaped so the regular expression can be pasted directly in to your source code, ready to be used, with no additional work required.
- Added the *Epilogue* (p. 115) section. This section is meant to be less formal, where I can place bits and pieces that don't fit elsewhere or editorialize on some topic.
- A number of small additions, deletions, and edits through out the documentation.

Changes:

- A new compile time configuration option, `RKL_METHOD_PREPEND`, can be used to add custom prefix to

all the RegexKit*Lite* methods. For example, setting `RKL_METHOD_PREPEND` to `xyz_` would cause `stringByReplacingOccurrencesOfRegex:withString:` to become `xyz_stringByReplacingOccurrencesOfRegex:withString:`. This is for users who prefer that any category additions to the core classes, such as `NSString`, have a user defined prefix to prevent the possibility of user defined method names interfering with later system defined method names.

- Support for iPhone low memory notifications. When it can be determined at compile time that the iPhone is being targeted, the preprocessor flag `RKL_REGISTER_FOR_IPHONE_LOWMEM_NOTIFICATIONS` is enabled to add code that automatically registers for low memory notifications on the iPhone. When a low memory notification is received, RegexKit*Lite* will flush all its caches, freeing the memory used to hold the cached data.

Bug fixes:

- Bug 2105213

  Fixed a bug in `stringByReplacingOccurrencesOfRegex:withString:` (p. 93) and `replaceOccurrencesOfRegex:withString:` (p. 87) where if the receiver was an empty string (i.e., `@""`), then RegexKit*Lite* would throw an exception because the ICU library reported an error. This turned out to be a bug in the ICU library itself in the `uregex_reset()` function (and the methods it calls to perform its work). A bug was opened with the ICU project: ICU ticket #6545. A work-around for the buggy behavior was put in place so that if the ICU library reports a `U_INDEX_OUTOFBOUNDS_ERROR` error for a string with a length of zero, that error is ignored since it is spurious. Thanks go to Andy Kim for reporting this.

- Bug 2050825

  Fixed a bug with `NSScannedOption` when targeting the iPhone. `NSScannedOption` is not available on the iPhone, so C preprocessor statements were added to ensure that `NSScannedOption` is not referenced when Objective-C Garbage Collection is not enabled. Thanks go to Shaun Inman for reporting this first.

- Fixed a bug in `stringByReplacingOccurrencesOfRegex:withString:` (p. 93) and `replaceOccurrencesOfRegex:withString:` (p. 87) where an internal assertion exception would be thrown if the string to search and the replacement string were both zero length strings (i.e., `@""`).

3.0 - 2009/05/06

This release of RegexKit*Lite* is a major release that includes new features, new APIs, and a number of bug fixes.

New Features:

- DTrace support. This release includes RegexKit*Lite* specific DTrace provider probe points that allow you to monitor the effectiveness of RegexKit*Lite*'s caches. DTrace support can be enabled with the compile time configuration option `RKL_DTRACE`.
- Easy support for custom built ICU libraries. A new compile time configuration option, `RKL_APPEND_TO_ICU_FUNCTIONS`, can be used to append a string to the ICU functions called by RegexKit*Lite*. Custom builds of the ICU library include the ICU version appended to all functions by default and this feature allows you to easily retarget RegexKit*Lite* to your custom build. Example usage: `-DRKL_APPEND_TO_ICU_FUNCTIONS=_4_0`

New `NSString` Methods:

- `- (NSArray *)arrayOfCaptureComponentsMatchedByRegex:(NSString *)regex;`
- `- (NSArray *)captureComponentsMatchedByRegex:(NSString *)regex;`

- `- (NSInteger) captureCount;`
- `- (NSArray *) componentsMatchedByRegex:(NSString *)regex;`
- `- (void)flushCachedRegexData;`
- `- (BOOL)isRegexValid;`

Deprecated Functionality:

- `+ (NSInteger)captureCountForRegex:(NSString *)regex;` Deprecated in RegexKit*Lite* 3.0

  Use of `captureCountForRegex:` is deprecated in favor of `captureCount` (p. 68).

- `- (NSEnumerator *)matchEnumeratorWithRegex:(NSString *)regex;` and `RKLMatchEnumerator` Deprecated in RegexKit*Lite* 3.0

  The example code for `RKLMatchEnumerator` available in `examples/RKLMatchEnumerator.h` and `examples/RKLMatchEnumerator.m` has been deprecated in favor of `componentsMatchedByRegex:` (p. 69).

Changes that effect the results that are returned:

- When the regular expression matched the tail end of the string to search, `componentsSeparatedByRegex:` (p. 71) would incorrectly add a zero length string to the results returned.

```
NSArray *splitArray = [@"Bob, Tom, Sam," componentsSeparatedByRegex:@",\\s*"];

// { @"Bob", @"Tom", @"Sam", @"" } <- RegexKitLite ≤ v2.2.
// { @"Bob", @"Tom", @"Sam"      } <- RegexKitLite ≥ v3.0.
```

- The results returned by `componentsSeparatedByRegex:` (p. 71) were found to differ from the expected results when zero-width assertions, such as `\b`, were used. Prior to v3.0, regular expressions that matched zero characters would be considered a 'match', and a zero length string would be added to the results array. The expected results are the results that are returned by the `perl split()` function, which does not create a zero length string in such cases. ICU ticket #6826.

```
splitArray = [@"I|at|ice I eat rice" componentsSeparatedByRegex:@"\\b\\s*"];

// [1]: { @"", @"I", @"|", @"at", @"|", @"ice", @"", @"I", @"", @"eat", @"", @"rice" }
// [2]: {     @"I", @"|", @"at", @"|", @"ice",     @"I",     @"eat",     @"rice" }

// [1]: ICU  & RegexKitLite ≤ v2.2
// [2]: perl & RegexKitLite ≥ v3.0
```

- As part of the 64-bit tidy, a check of the length of a string that is passed to RegexKit*Lite*, and therefore ICU, was added to ensure that the length is less than `INT_MAX`. If the length of the string is ≥ `INT_MAX`, then RegexKit*Lite* will raise `NSRangeException`. The value of `INT_MAX` is $2^{31} - 1$ (`0x7fffffff`). This was done because ICU uses the `int` type for representing offset values.

Other Changes:

- Ticket #2027975 - Request for `captureCount` like functionality.
- Ticket #2779301 - Request for `componentsMatchedByRegex:` functionality.
- Ticket #2779965 - Request that documentation be updated with how to match a literal `\` with a regex specified using a string literal: `@"\\\\";`.
- Ticket #2786878 - Request for `isRegexValid` functionality.
- The GCC variable `cleanup __attribute__()` is now used to provide an extra safety net around the use

of the RegexKit*Lite* spin lock. The `cleanup` function ensures that if the spin lock was locked by a function that it was also unlocked by the function. If a function obtains the lock but does not unlock it, the `cleanup` function forcibly unlocks the spin lock.

- Minor GC changes. Some minor changes in the way heap buffers were allocated was required to support the new methods that return a `NSArray`.
- 64-bit cleanup. Various literal numeric constant values had either `L` or `UL` appended to them if they were used with `NSInteger`, `NSUInteger`, or various other '`long`' data types.
- 64-bit cleanup. Added checks to verify that a strings length is < `INT_MAX` and throw an exception if it isn't. This check was added because ICU uses signed 32-bit `int` values to represent offsets.
- Changed some macros to static inline functions for compile-time prototype checking.
- Updated `examples/NSString-HexConversion.m`. 64-bit tidies.
- Updated `examples/RKLMatchEnumerator.h`. Added the deprecated attribute to `matchEnumeratorWithRegex:`.
- Updated `examples/RKLMatchEnumerator.m`. 64-bit tidies. Added a preprocessor warning that `RKLMatchEnumerator` has been deprecated in favor of `componentsSeparatedByRegex:`.
- Updated the documentations visual style to better match the style currently used by Apple.
- Many small DocSet tweaks and improvements.

Bug fixes:

- Bug #2319200 - The documentation for `stringByReplacingOccurrencesOfRegex:withString:` (and related methods) was updated. The text for the Returned Value section was obviously copy and pasted from somewhere else and never updated. This has been fixed.
- Bug #2408447 - `stringByReplacingOccurrencesOfRegex:` bug fixed. There was a bug in ICU that would cause a search and replace operation to fail with `U_BUFFER_OVERFLOW_ERROR`, which would then cause RegexKit*Lite* to throw `RKLICURegexException` (p. 103), if the length of the replaced string was significantly longer than the original string. ICU ticket #6656.
- `componentsSeparatedByRegex:` bug fixed. When the regular expression matched the tail end of the string, an extra zero length string was incorrectly added to the results.
- `componentsSeparatedByRegex:` bug fixed. The results returned by the ICU function `uregex_split()` was found to be different that the results returned by the `perl split()` function. ICU ticket #6826
- `rangeOfRegex`, `stringByMatching:` bug fixed. A bug was fixed where the `inRange:` parameter was not correctly honored.
- If RegexKit*Lite* was able to get direct access to a strings `UTF-16` buffer, there was a very remote chance that RegexKit*Lite* would continue to use a pointer to a strings older `UTF-16` buffer if the string mutated and allocated a new buffer. The pointer to a strings direct buffer is now verified before each use.
- `isMatchedByRegex:inRange:` was never documented. Fixed.

3.1 - 2009/05/15

This release of RegexKit*Lite* is a bug fix release.

Bug Fixes:

- Bug #2790480 - If a regular expression had an error, as detected by ICU, then under some circumstances

the `lastCacheSlot` variable would not be properly cleared. As a result, this would cause the pointer to the compiled regular expression to be `NULL`. The `NULL` value would be caught by the run-time assertion checks if they were not disabled via `NS_BLOCK_ASSERTIONS`, otherwise it would most likely lead to a crash.

3.2 - 2009/11/04

This release of RegexKit*Lite* is a bug fix release.

Bug Fixes:

- Bug #2828966 - Fixed a minor bug that caused compiling on Mac OS X 10.4 to fail. Changed the header include line from `#include <objc/runtime.h>`, which is available only on Mac OS X ≥ 10.5, to `#include <objc/objc-runtime.h>`, which is available on Mac OS X ≥ 10.4.
- Bug #2862398 - Modified the logic for determining if the cached `UTF-16` conversion is valid for the string to be searched. Previously, RegexKit*Lite* used a combination of the string to be searched `hash` and `length` to determine if the cached `UTF-16` conversion was a match. For speed, `NSString` only uses a fixed number of characters to calculate the hash. This means that as the length of a string grows, the hash function becomes less effective at detecting differences between strings. The original intent behind the `UTF-16` conversion cache logic was to be able to re-use the `UTF-16` conversion for strings that were "the same" (i.e., hash equality), but were different instantiations (i.e., pointer inequality). On reflection, it was decided that the more likely real-world case would be strings that would share the same hash value, but actually be slightly different. RegexKit*Lite* now requires that the `NSString` pointer, as well as the `hash` and `length`, be the same as the string used for the cached `UTF-16` conversion.
- Bug #2879356 - `arrayOfCaptureComponentsMatchedByRegex:`, `captureComponentsMatchedByRegex:`, `componentsMatchedByRegex:`, and `componentsSeparatedByRegex:` bug fixed. Originally reported by Jesse Grosjean. Duplicate of bug #2890342. If a regex matched a part of the string that included the very end of the string, subsequent matches would incorrectly return that the regex did not match the string. Clearing the cache, either manually or because another regex match replaced the affected cache slot, would clear the condition.
- Bug #2890810 - `stringByReplacingOccurrencesOfRegex:` bug fixed. If the size of the buffer needed to hold the replaced string was larger than the initial estimated buffer size, ICU should return an error of `U_BUFFER_OVERFLOW_ERROR`. However, due to a bug in ICU (see bug #2408447, ICU ticket #6656), this error was not reported correctly and a workaround was implemented. A rare corner case could cause ICU to return `U_STRING_NOT_TERMINATED_WARNING` during the time that RegexKit*Lite* was calculating the size of the buffer needed to hold all of the replaced string. When this happened it would cause RegexKit*Lite* to miss the `U_BUFFER_OVERFLOW_ERROR` error needed to resize the buffer.

3.3 - 2009/11/07

This release of RegexKit*Lite* is a bug fix release.

Bug Fixes:

- Bug #2893824 - Fixed a minor bug that caused compiling with the iPhone device SDK to fail. This is related to the RegexKit*Lite* 3.2 fix for bug #2828966. It turns out that the iPhone simulator SDK has the header file `objc/objc-runtime.h`, but the iPhone device SDK only has the header file `objc/runtime.h`. RegexKit*Lite* will now conditionally use `#include <objc/runtime.h>` when targeting the iPhone or Mac OS X ≥ 10.5, and `#include <objc/objc-runtime.h>` when targeting Mac OS X ≤ 10.4.

4.0 - 2010/04/18

This release of RegexKit*Lite* is a major release that includes new features, new APIs, and bug fixes.

API Compatibility Changes:

- The `NSMutableString replaceOccurrencesOfRegex:...` methods now return a result of type `NSInteger`, whereas RegexKit*Lite* < 4.0 returned a result of type `NSUInteger`. This change, while technically breaking API compatibility, is likely to be completely backwards compatible for most users. The change was made so that errors, such as an invalid regular expression, can be distinguished from a search and replace operation that replaced zero occurrences. An error condition that is not an exception now returns a value of `-1`. Prior to RegexKit*Lite* 4.0, the only way to distinguish between a search and replace operation that replaced zero occurrences and an error that prevented a search and replace from taking place was to use a `replaceOccurrencesOfRegex:...` method with an `error:` parameter.

Changes that effect the results that are returned:

- The results returned by `componentsSeparatedByRegex:` (p. 71) were found to differ from the expected results for some regex patterns. For example, prior to v4.0, when the regular expression `.` (dot, match any character) was used to split a string, RegexKit*Lite* returned an array of seven zero length strings. The expected results are the results that are returned by the `perl split()` function– the equivalent of a `NSArray` with zero items in it.

```
NSArray *splitArray = [@"abc.def" componentsSeparatedByRegex:@"."];

// { @"", @"", @"", @"", @"", @"", @"" } <- RegexKitLite ≤ v3.3.
// {                                     } <- RegexKitLite ≥ v4.0 & perl.
```

New features:

- Documentation now available in PDF format.

  The RegexKit*Lite* documentation is now available in PDF format. The PDF version is ideal for making printed copies of the documentation.

- Improved compiled regular expression cache.

  RegexKit*Lite* versions < 4.0 used a direct mapped, or 1-way set associative, cache. RegexKit*Lite* 4.0 uses a 4-way set associative cache. In general, a *n*-way set associative cache, where $n > 1$, has fewer misses than a direct mapped cache. Entries in a set are managed on a least recently used, or LRU, basis. This means that when a compiled regular expression is added to the cache, the entry in a set that is the least recently used is the one chosen to hold the new compiled regular expression.

  In benchmark tests, the old compiled regular expression cache required a ***dramatically*** larger cache size to achieve the same effectiveness (i.e., the likelihood that the regular expression was already in the cache) as the new compiled regular expression cache. Retrieving a cached compiled regular expression is ***541.3*** times faster[†] than actually compiling the regular expression:

Retrieve	Time	Rate (per second)
Cached	51 ns	19,680,762
Compile	27,560 ns	36,285

  [†] Timing done a MacBook Pro 2.66GHz running Mac OS X 10.6.2.

- Improved `UTF-16` conversion cache.

The UTF-16 conversion cache now uses the same 4-way set associative cache system as the compiled regular expression cache. For strings that required a UTF-16 conversion, RegexKit*Lite* versions < 4.0 used two buffers for caching UTF-16 conversions– a small, fixed sized buffer for 'small' strings (≤ 2048 characters by default), and a dynamically sized buffer for 'large' strings. RegexKit*Lite* 4.0 keeps the same fixed and dynamic size dichotomy, but now each type contains four buffers which are reused on a least recently used basis.

- Blocks support.

  Although support for blocks was introduced with Mac OS X 10.6, you can still use blocks on Mac OS X 10.5 and iPhone ≥ 2.2 by using Plausible Blocks. This solution provides a compiler and run-time that has been back-ported from the Mac OS X 10.6 Snow Leopard sources.

  Note: RegexKit*Lite* checks if the C preprocessor define NS_BLOCKS_AVAILABLE is set to 1 to automatically determine if blocks functionality should be enabled. Typically NS_BLOCKS_AVAILABLE is only set to 1 when using the standard developer tools and the minimum version of Mac OS X supported is set to 10.6. RegexKit*Lite* blocks support can be explicitly enabled by setting RKL_BLOCKS to 1 (i.e., -DRKL_BLOCKS=1). The behavior is undefined if RKL_BLOCKS is set to 1 and the compiler does not support the blocks language extension or if the run-time does not support blocks.

  See Also
  Plausible Blocks - PLBlocks

New NSString Methods:

- - dictionaryByMatchingRegex:withKeysAndCaptures: (p. 73)
- - arrayOfDictionariesByMatchingRegex:withKeysAndCaptures: (p. 63)

New Block-based NSString Methods:

- - enumerateStringsMatchedByRegex:usingBlock: (p. 76)
- - enumerateStringsSeparatedByRegex:usingBlock: (p. 78)
- - stringByReplacingOccurrencesOfRegex:usingBlock: (p. 91)

New Block-based NSMutableString Methods:

- - replaceOccurrencesOfRegex:usingBlock: (p. 85)

Other changes:

- Improved performance. Many small performance tweaks and optimizations were made to frequently used code paths.
- The meaning of RKL_CACHE_SIZE changed. This compile-time preprocessor tunable now controls the number of "sets" in the compiled regular expression cache, and each "set" contains four entries. This means that the maximum number of potential compiled regular expressions that can be cached is RKL_CACHE_SIZE times four. Like previous versions, RKL_CACHE_SIZE should always be a prime number to maximize the use of the cache.
- RKL_CACHE_SIZE changed to 13. The total number of compiled regular expressions that can be cached is 13 * 4, or 52. This is a 126% increase from previous versions of RegexKit*Lite* which had a default value of 23 for RKL_CACHE_SIZE.
- Added the flag RKLEnumerationBufferLookupFlag (p. 102) to the DTrace RegexKitLite:::utf16Con

`versionCache` (p. 102) flags.

- Added a number of new keys to *RegexKitLite NSError and NSException User Info Dictionary Keys* (p. 103).

Bug fixes:

- If the ICU library returned a status of `U_STRING_NOT_TERMINATED_WARNING` during a search and replace operation, RegexKit*Lite* erred on the side of caution and treated it as a failure due to the numerous search and replace bugs in the past. RegexKit*Lite* 4.0 continues to err on the side of caution, but now treats `U_STRING_NOT_TERMINATED_WARNING` as if the ICU library had returned a status code of `U_BUFFER_OVERFLOW_ERROR`.
- The `NSMutableString replaceOccurrencesOfRegex:...` methods could have returned a value > `0`, the replaced count, even if no actual replacements were performed. This would occur if an error was encountered during the search and replace, in which case the replaced count was not correctly reset to `0`. RegexKit*Lite* 4.0 changed the `replaceOccurrencesOfRegex:...` API so that a value of `-1` could be returned to indicate that an error was encountered.
- `componentsSeparatedByRegex:` bug fixed. The results returned were found to be different that the results returned by the `perl split()` function.

Epilogue

Coding Style

One noticeable break in style conventions is in line lengths. There was a time when 80 column limits made a lot of sense as it was the lowest common denominator. Today, a modern computer screen can display much more than just 80 columns. Even an iPhone, which has a screen size of 320x480, can display 96 columns by 24 rows of the usual Terminal.app Monaco 10pt font (5x13 pixels) in landscape mode. Because of this, my personal style is not to have an arbitrary limit on line lengths. This allows for much more code to fit on the screen at once, which I've heard referred to as "Man–Machine Interface Bandwidth". While you can always page up and down, the simple movement of your eye is almost always an order of magnitude faster. Paging through code also tends to break your concentration as you briefly try to mentally orientate yourself with the freshly displayed text and where the section of code is that you're looking for.

I try to group a line around relevancy so that based on the start of the line you can quickly determine if the rest of the line is applicable. Clearly the number of spaces used to indent a block plays a similar role, it allows you to quickly visually establish the logical boundaries of what lines of code are applicable. I also try to horizontally align related statements since your eye tends to be extremely sensitive to such visual patterns. For example, in variable declaration and initialization, I try to align the type declaration and the = (equal sign) across multiple lines. This tends to cause the declaration type, the variable name, and the value assigned to visually *pop*. Without the alignment, you typically have to scan back and forth along a line to separate and find a variable name and its initialization value. Sometimes line breaks and horizontal alignment are done purely on what's subjectively aesthetically pleasing and allows the eye to quickly *flow* over the code.

The source code of RegexKit*Lite* isn't exactly what you'd call clean, there's more than a few crufty C barnacles in there. There is usually a choice between two polar opposites and in this case it's between elegant, easy to maintain and comprehend code, and speed. If you use regular expressions for very long, you will undoubtedly encounter a situation where you need to scan through tens of megabytes of text and the speed of your regular expression matching loop needs to be faster. ***A lot faster***. RegexKit*Lite* was written to go fast, and the source code style reflects this choice.

The Need for Speed

A significant amount of time was spent using Shark to optimize the critical sections of RegexKit*Lite*. This included tweaking even the most insignificant details, such as the order of boolean expressions in `if()` statements to minimize the number of branches that would have to be evaluated to determine if the statement is true or false. Wherever possible, *Core Foundation* is used directly. This avoids the overhead of an Objective-C message dispatch, which would invariably end up calling the exact same *Core Foundation* function anyways.

Even the cache has what is essentially a cache– the last regular expression used. Each time a regular expression is used, the compiled ICU regular expression must be retrieved from the cache, or if it does not exist in the cache, instantiated. Checking the cache involves calculating the remainder of the regular expression strings hash modulo the cache size prime, which is a moderately expensive division and multiplication op-

eration. However, checking if the regular expression being retrieved this time is exactly the same as the last regular expression retrieved is just a fast and simple comparison check. As it turns out, this is very often the case. Even the functions are arranged in such a way that the compiler will often inline everything in to one large aggregate function, eliminating the overhead of a function call in many places.

Normally, this kind of micro-optimization is completely unjustified. A rough rule of thumb is that 80% of your programs execution time is spent in 20% of your programs code. The only code worth optimizing is the 20% that is heavily executed. If your program makes heavy use of regular expressions, such as a loop that scans megabytes of text using regular expressions, RegexKit*Lite* is almost guaranteed to be a part of the 20% of code where most of the execution time is spent. The release notes for CotEditor would seem to indicate that all this effort has paid off for at least one user:

```
More than 10.4 when running on the color process to review the details of the
definition of a regular expression search RegexKitLite adopted, 0.9.3, as compared
to the speed of color from 1.5 to 4 times as much improved.
```

Note: This is an automatic machine translation from Japanese to English.

Documentation

Clearly documentation has been a high priority for this project. Documentation is always hard to write and good documentation is exponentially harder still. The vast majority of 'development effort and time' is spent on the documentation. It's always hard to judge the quality and effectiveness of something you wrote, so hopefully the extra effort is worth it and appreciated.

As an aside and a small rant, I have no idea how anyone manages to build so-called 'web applications'. I waste a truly unbelievable amount of time trying to accomplish the simplest of things in HTML, which is then multiplied when I check for 'compatibility' with different browsers and their various popular versions. What a joke, and that's just for this 'simple' documentation. Though I do have to give kudos to the Safari / WebKit guys, it always seems to be a lot easier to get the result you're looking for with the WebKit engine. The little things add so much: shadows, round rects, gradients, CSS animations, the canvas element, etc.

Questions about licensing, including in your software, etc

What do I need to do to include RegexKit*Lite* in my software?

Not much. The *BSD License* is very permissive. In short, you just need to 'acknowledge' your use of it. Safari's Help ▸ Acknowledgements is a good example of this.

I am selling a 'closed-source' commercial application. I would like to use RegexKit*Lite* in it, how much will it cost?

RegexKit*Lite* is free for any use.

Do I have to distribute the source with my application?

No. Unlike the *GNU GPL License*, there is no requirement that you include or make the source code available, even if you release a "compiled, binaries only" end product.

What about modifications to RegexKit*Lite*? Do I have to make any changes that I make available? Or contribute them back to the author of RegexKit*Lite*?

You may make any modifications you want and are under no obligation to release those changes to anyone.

Why the *BSD License*?

It's a well known license. If you are part of a Large Corporate Organization, chances are the Corporate Lawyers have Decided whether or not the use of source code licensed under the *BSD License* is Acceptable or not. This can be a godsend for anyone who has to deal with such situations.

It also expresses a few things I think are perfectly reasonable:

1. Give credit where credit is due.
2. There is no warranty, so use it at your own risk.
3. You can't hold the author responsible.

The first point is prescribed by most professional ethics already. Plus, it's always nice to see where your stuff ends up and how it's being used. The other points make explicit what should already be obvious. After all, you get what you pay for.

License Information

RegexKit*Lite* is distributed under the terms of the *BSD License* as specified below.

See Also

Wikipedia - BSD License
The Linux Information Project - BSD License Definition

License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the Zang Industries nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

```
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```