REGULAR EXPRESSION TUTORIAL

USING THE UNIX LIBRARY TO DEFINE A NEW CATEGORY

My colleague Kompilesoft on www.cocoaforum.com wrote a little utility for one of our readers. The original question was about how to read and write files from Objective/C.

Of course there are many possibilities about how to achieve this. Remember that Obj/C is C, so you have the normal C library functions such as fopen, fread, fwrite and fclose. And if you want to use C++, you have access to the std library and may use cout and cin and all those things. In addition, Obj/C has it's own way of interacting with the file system and reading and writing files.

To demonstrate this, Kompilesoft wrote a little utility snr = Search aNd Replace. The syntax is:

```
$ snr from-string to-string file ...
```

Mind you, the original question did ask for a 'pattern' (regular expression) as the from-string.

This is of course what the unix utility sed provides and was suggested by a couple of readers.

Here's the thread on the forum and you can read what was said at the time:

http://www.cocoaforum.com/2010/10/07/how-to-parse-a-text-file/

A couple of weeks went by, and K was still talking about this and I got thinking: I'm sure there's an NSRegularExpression class in Cocoa and that'll be able to do the business. Right and Wrong! Right: the iOS SDK has this class, the desktop version of Cocoa does not! Yes, I was also surprised.

I went off a-googling to find something and was pleased to find the following code:

http://homepage.mac.com/jrc/contrib/

This has a 'Category' to add regular expressions to the NSString class. It's build using the regexp library in UNIX (see man regcomp). The code was written by: John R Chang. I was really impressed by what John had done, however it didn't do exactly what I wanted. There were two short-comings:

1 John's code didn't have a replace function (although it's easy to add a method to the extension for this purpose).
2 I think John's code (or regexp to be precise) only returns the first match for a regular expression.

Because of restriction 2, I decided to (more or less) rewrite John's code to handle multiple matches and of course I added the replace function (which was the reason to be interested in this in the first place).

WHAT IS A REGULAR EXPRESSION?

A regular expression is a text pattern. Regular expressions are used by many unix utilities such as grep, python, perl, awk, sed and many more. Regular expressions are a language in their own right to define patterns of text. There are many tutorials on-line about Regular Expressions and I'll leave you to google into this matter.

I'll give you an example: r.*mills would match robinmills rmills ratmills but not clanmills !

. = any character

* = the previous character may be repeated 0 or more times

Almost (all) regular expression code is processed in two steps. There is a compiler (step 1) which reads the regular expression and produces code for the (step 2) pattern matching engine. For my purposes here, the compilation step is carried out and immediately used on the contents of the file. So I've put the two steps together for convenience.

The NSRegularExpression call provided by the iOS platform defines the pattern (which is presumably instantly compiled, or compiled on demand). NSRegularExpression has several instance methods including the ability to modify an NSMutableString.

I could of course have written an NSRegularExpression class, however I chose not to do that for two reasons:

1 I'm sure the MacOS SDK will support this soon (probably with MacOS 10.7)
2 I thought it was more interesting to provide this as a 'Category'.

You can download the code from: **http://clanmills.com/acticles/cocoatutorials/RegularExpressions.zip**

ACKNOWLEDGMENTS

I acknowledge and respect the contribution to this project made by Kompilesoft and John Chang. Thanks, guys.

WHAT IS A CATEGORY?

A category is a very nice feature of Obj/C that enables you to add methods to every member of a class at run-time. It's very similar to JavaScripts prototype functions.

In JavaScript, every class has a prototype chain. If you change the prototype chain, every member of the class will acquire that method. This is a very simple way to extend existing classes.

In Objective C, this is called 'a category', and you define the interface of your category in a .h file. You define the implementation of the category in a .m (or .mm) file. Every member of the class now has access to the methods of your category. Calling it an extension would probably have been a better name for this.

Anyway, it's very useful. I've added two methods (thanks to John Chang) to NSString and written one additional method.

JavaScript

```
519 /Users/rmills/Documents/Tutorials $ js
js> x="now is the time"
js> String.prototype.robin = function()
{ return 'me robin, you : ' + this ; }
js> x.robin()
me robin, you: now is the time
js> y='abc'
js> y.robin()
me robin, you: abc
js> quit()
520 /Users/rmills/Documents/Tutorials $
```

Objective/C

```
NSString_RE.h
@interface NSString (RE)
- declare your methods
@end

NSString_RE.m
@implementation NSString (RE)
- define your methods
@end

somewhere.m
NSString* s ;
[s.myMethod];
```

```
- (NSArray *) substringsMatchingRegularExpression : (NSString *)pattern
                                         maxcount  : (int)maxcount
                                         options   : (int)options
                                         ranges    : (NSArray **)ranges
                                         error     : (NSError **)error;

- (BOOL) grep:(NSString *) pattern options   : (int)options ;

- (NSString *) stringWithRegularExpression   : (NSString*) pattern
                                         replace   : (NSString*) replace
                                         maxcount  : (int) maxcount
                                         options   : (int) options
                                         error     : (NSError**) error;
```

CODE DISCUSSION

This is a very simple command-line project. The code is very straightforward and easy to understand.

You can:

1) Use the New Project Wizard to create a command-line program (linked to the foundation classes).

2) Compile and run the project.

3) Copy the code from my snr.m to your project

4) Copy my NSString_RE.h and NSString_RE.m to your project.

5) Use Add/Existing Files in XCode to link in NSString_RE.m

6) Build and you should be working.

```
525 /Users/rmills/Projects/cocoatutorials/RegularExpressions $ ls -alt
drwxr-xr-x  7 rmills  staff   238B Oct 17 16:11 snr.xcodeproj
-rw-r--r--@ 1 rmills  staff   1.9K Oct 17 15:10 snr.m
-rw-r--r--  1 rmills  staff   3.6K Oct 17 15:06 NSString_RE.m
-rw-r--r--  1 rmills  staff   1.4K Oct 17 15:03 NSString_RE.h
-rw-r--r--@ 1 rmills  staff   3.0K Oct  7 15:00 snr.1
-rw-r--r--@ 1 rmills  staff   148B Oct  7 15:00 snr_Prefix.pch
526 /Users/rmills/Projects/cocoatutorials/RegularExpressions $
```

LICENSE

Robin Mills
Software Engineer

Windows/Mac/Linux
C++, Web, JavaScript, UI

400 N First St #311
San Jose, CA 95112

T (408) 288 7673
C (408) 394 1534
robin@clanmills.com

http://clanmills.com

Revision History

2010-10-17	Initial version.