CS436
Spring 2012

Lab 1B

Due Date

11:59 PM, Thursday February 9, 2012.
Email zipfile "LastName-Lab1B.zip" to cse436ta@gmail.com

Assignment 1B - WhatATool (Part I)

Assignment

In this assignment you will be getting your feet wet with Objective-C by writing a small command line tool. You will create and use various common framework classes in order to become familiar with the syntax of the language.

The Objective-C runtime provides a great deal of functionality, and the gcc compiler understands and compiles Objective-C syntax. The Objective-C language itself is a small set of powerful syntax additions to the standard C environment.

To that end, for this assignment, you'll be working in the most basic C environment available – the main() function.

This assignment is divided up into several small sections. Each section is a mini-exploration of a number of Objective-C classes and language features. Please put each section's code in a separate C function. The `main()` function should call each of the section functions in order. Next week will add more sections to this assignment.

IMPORTANT: The assignment walkthrough begins on page 3 of this document. The assignment walkthrough contains all of the section-specific details of what is expected.

The basic layout of your program should look something like this:

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

// sample function for one section, use a similar function per section
```

```
void PrintPathInfo() {
   // Code from path info section here
}

int main (int argc, const char * argv[]) {


       @autoreleasepool {

       PrintPathInfo(); // Section 1
       PrintProcessInfo(); // Section 2
       PrintBookmarkInfo(); // Section 3
       PrintIntrospectionInfo(); // Section 4

       }
       return 0;
}
```

Testing

In most assignments testing of the resulting application is the primary objective. In this case, testing/grading will be done both on the output of the tool, but also on the code of each section.

We will be looking at the following:

1. Your project should build without errors or warnings.

2. Your project should run without crashing.

3. Each section of the assignment describes a number of log messages that should be printed.

4. Each section of the assignment describes certain classes and methodology to be used to generate those log messages – the code generating those messages should follow the described methodology, and not be simply hard-coded log messages

.

A note about the `NSLog` function. It will generate a string that is prefixed with a timestamp, the name of the process, and the pid. It will also automatically append a newline to the log statement.

```
2009-09-09 13:49:42.275 WhatATool[360] Your message here.
```

This is expected. You are only being graded on the text not generated automatically by `NSLog`. You do not need to attempt to suppress what `NSLog` prints by default. To help make the output of your program more readable, it would be helpful to put some kind of header or separator between each of the sections.

Hints

Hints are distributed section by section but generally, the lecture #2 notes provide some very good clues with regards to Objective-C syntax and commonly used methods. Another source of good information is the Apple ADC site. In particular, these documents may be particularly helpful:

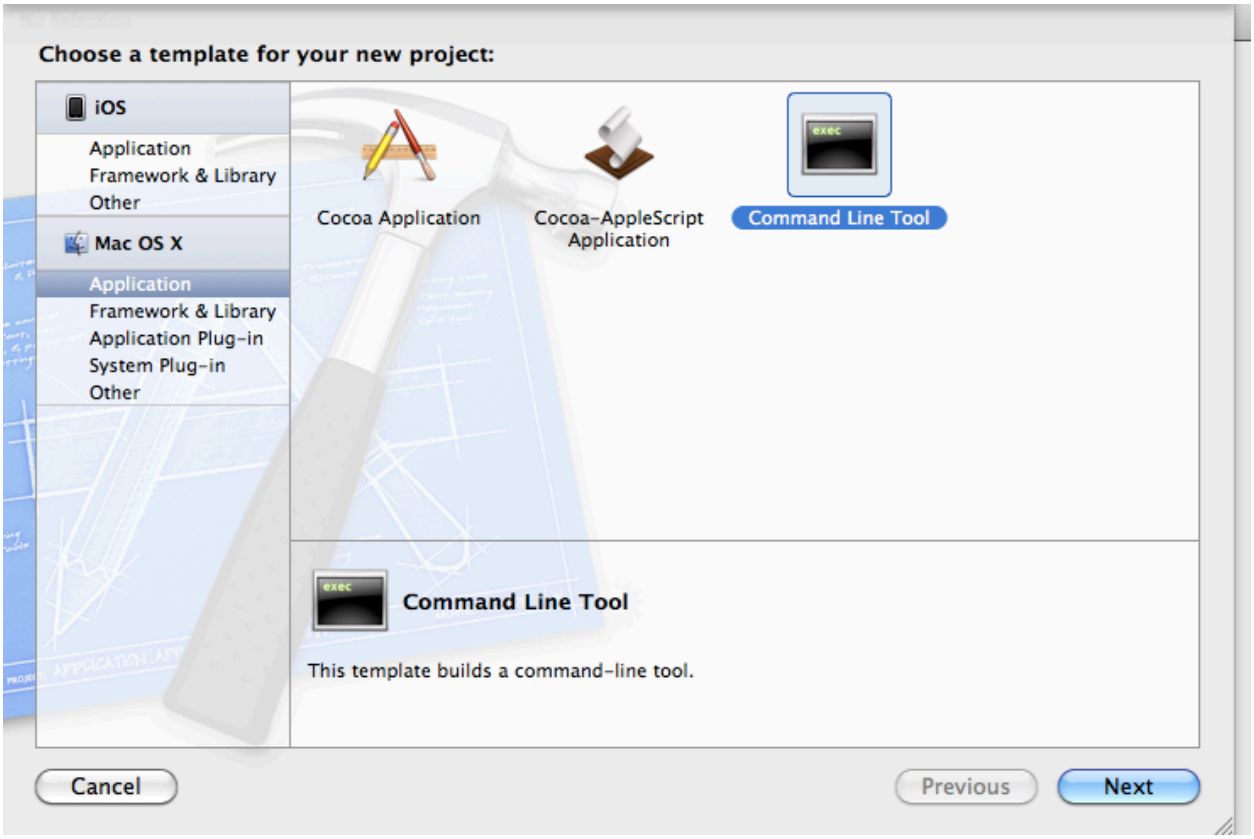
http://developer.apple.com/documentation/Cocoa/Conceptual/OOP_ObjC
http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC

Troubleshooting

Remember that an Objective-C NSString constant is prefixed with an @ sign. For example, `@"Hello World"`. The compiler will warn, and your program will crash if you use a C string where an NSString is required.

Assignment Walkthrough

In Xcode, create a new Foundation Tool project. Name it WhatATool.

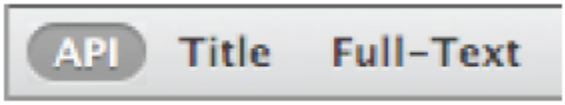


You will be doing the bulk of your work in the WhatATool.m file where a skeletal implementation of main() has been provided.

You should build and test at the end of each code-writing section.

Finding Answers / Getting to documentation

Some of the information required for this exercise exists in the class documentation, not in the course materials. You can use the Xcode documentation window to search for class documentation as well as conceptual articles. Open the Xcode documentation window by choosing Help > Documentation. The leftmost section of the search bar in the documentation window lets you search APIs, Titles or Full-Text of the documentation. For now, use the API Search. Select an appropriate Doc Set to search, for example the "Apple iPhone OS 2.0" doc set should contain all the materials you'll need for this assignment.



Section 1: Strings as file system paths

NSString has a set of methods that allow you to manipulate file system paths. You can find them in the NSString documentation grouped under the heading "Working with paths". In this section, begin with an NSString constant that is a tilde – the symbolic representation for your home directory in Unix.

```
NSString *path = @"~";
```

Starting with that string, find a path method that will expand the tilde in the path to the full path to your home directory. Use that method to make a new string, and log the full path in a format like:

```
My home folder is at '/Users/toddsmac'
```

NSString has a method that will return an array of path components. Each element in the returned array is a single component in the original path. Use this method to get an array of path components for the path you just logged. Use fast enumeration to enumerate through each item in the returned array, and log each path component. The result should look something like:

```
/
Users
toddsmac
```

Section Hints:
• With string convenience methods, a common pattern is to reuse the same variable:

```
NSString *aString = @"Bob";
aString = [aString stringWithSomeModification];
```

Section 2: Finding out a bit about our own process

Look up the class NSProcessInfo in the documentation.

You will find a class method that will return an NSProcessInfo object. (In fact, you should find a very handy code sample there as well). From this object, you can access the name and process identifier (pid) of the process.

Log these pieces of information to the console using `NSLog` in the format:

```
Process Name: 'WhatATool' Process ID: '6521'
```

Section Hints:

• We didn't discuss NSProcessInfo in lecture at all, so you aren't missing any slides or notes. All of the information you need is in the NSProcessInfo class documentation.

• Comparing the process info you logged with the prefix that NSLog generates can help verify you're logging the correct information.

Section 3: A little bookmark dictionary

In this section, you will build a small URL bookmark repository using a mutable dictionary. Each key is an NSString that serves as the description of the URL, the value is an NSURL.
Create a mutable dictionary that contains the following key/value pairs:
Key (NSString) Value (NSURL)

Washington University in St. Louis http://www.wustl.edu
Apple http://www.apple.com
Google http://www.google.com
Washington University Record http://record.wustl.edu
Washington University Library http://library.wustl.edu

Enumerate through the keys of the dictionary. While enumerating through the keys, check each key to see if it starts with @"Washington". If it does, retrieve the NSURL object for that key from the dictionary, and log both the key string and the NSURL object in the following format below. If the key does not start with @"Washington", no output should be printed.

```
Key: 'Washington University in St. Louis' URL: 'http://www.wustl.edu'
```

Section Hints:

• Use `+URLWithString`: to create the URL instances.

• NSString has methods to determine if a string has a particular prefix or suffix. *Remember to log only those keys that start with 'Washington'.*

• The methods for creating and adding items to a mutable dictionary are located in the Lecture #2 slides. Remember that an NSMutableDictionary is a subclass of NSDictionary and so inherits all of its methods.

• When using a dictionary and fast enumeration, you are enumerating the keys of the dictionary. You can use the key to then retrieve the value. NSDictionary also defines a method called valueEnumerator that returns an NSEnumerator object which will directly enumerate the values. You can use either approach.

Section 4: Selectors, Classes and Introspection

Objective-C has a number of facilities that add to its dynamic object-oriented capabilities. Many of these facilities deal with determining and using an object's capabilities at runtime.

Create a mutable array and add objects of various types to it. Create instance of the classes we've used elsewhere in this assignment to populate the array: NSString, NSURL, NSProcessInfo, NSDictionary, etc. Create some NSMutableString instances and put them in the array as well. Feel free to create other kinds of objects also.

Iterate through the objects in the array and do the following:

1. Print the class name of the object.
2. Log if the object is member of class NSString.
3. Log if the object is kind of class NSString.
4. Log if the object responds to the selector "lowercaseString".
5. If the object does respond to the lowercaseString selector, log the result of asking the object to perform that selector (using performSelector:)

For example, if an array contained an NSString, an NSURL and an NSDictionary the output might look something like this:

```
2008-01-10 20:56:03 WhatATool[360] Class name: NSCFString
2008-01-10 20:56:03 WhatATool[360] Is Member of NSString: NO
2008-01-10 20:56:03 WhatATool[360] Is Kind of NSString: YES
2008-01-10 20:56:03 WhatATool[360] Responds to lowercaseString: YES
2008-01-10 20:56:03 WhatATool[360] lowercaseString is: hello world!
2008-01-10 20:56:03 WhatATool[360] ====================================
2008-01-10 20:56:03 WhatATool[360] Class name: NSURL
2008-01-10 20:56:03 WhatATool[360] Is Member of NSString: NO
2008-01-10 20:56:03 WhatATool[360] Is Kind of NSString: NO
2008-01-10 20:56:03 WhatATool[360] Responds to lowercaseString: NO
2008-01-10 20:56:03 WhatATool[360] ====================================
2008-01-10 20:56:03 WhatATool[360] Class name: NSCFDictionary
2008-01-10 20:56:03 WhatATool[360] Is Member of NSString: NO
2008-01-10 20:56:03 WhatATool[360] Is Kind of NSString: NO
2008-01-10 20:56:03 WhatATool[360] Responds to lowercaseString: NO
2008-01-10 20:56:03 WhatATool[360] ====================================
```

Section Hints:

• When you ask various classes, such as NSString, for its class name, you may not get the result you expect. Your logs should report what the runtime reports back to you - don't worry if some of the class names may seem strange, these are implementation details of the NSString class. Encapsulation at work!