A handy reference to learning and
using the Objective-C programming language

Objective-C Recipes

A Problem-Solution Approach

Matthew Campbell

Objective-C Recipes

A Problem-Solution Approach

Matthew Campbell

Apress®

Objective-C Recipes

ISBN 978-1-4302-4371-7

ISBN 978-1-4302-4372-4 (eBook)

President and Publisher: Paul Manning
Lead Editor: Steve Anglin
Developmental Editor: Matthew Moodie and Louise Corrigan
Technical Reviewer: Anselm Bradford
Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Louise Corrigan, Morgan Ertel, Jonathan Gennick, Jonathan Hassell, Robert Hutchinson, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Gwenan Spearing, Matt Wade, Tom Welsh
Coordinating Editor: Corbin Collins
Copy Editor: Mary Behr
Compositor: Bytheway Publishing Services
Indexer: SPi Global
Artist: SPi Global
Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `www.springeronline.com`.

For information on translations, please e-mail `rights@apress.com`, or visit `www.apress.com`.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at `www.apress.com/bulk-sales`.

Any source code or other supplementary materials referenced by the author in this text is available to readers at `www.apress.com`. For detailed information about how to locate your book's source code, go to `www.apress.com/source-code/`.

Contents at a Glance

Contents

About the Author

Matthew Campbell has trained over 800 new iOS developers at the Mobile App Mastery Institute and iOS Code Camp. He also developed Tasting Notes, a universal app for wine lovers. Matt is the lead blogger for `http://HowToMakeiPhoneApps.com`, a blog about creating iPhone apps.

About the Technical Reviewer

Anselm Bradford is a lecturer in digital media at the Auckland University of Technology (AUT) in New Zealand where he researches interactive media, web media, and visual communication. He has been a technical reviewer on several iOS-related books and is the lead author of *HTML5 Mastery* and a co-author of *CSS3 Solutions*. He may be found @anselmbradford on Twitter and occasionally blogs at `AnselmBradford.com`.

Acknowledgments

It's tempting to think that a book like this is the sole work of the person whose name is stamped on the front cover. Of course, that's not true, and this book never would have happened at all without the support and occasional ego massaging from the supportive editors at Apress.

In particular, I'd like to acknowledge Louise Corrigan, whose comments peppered throughout our shared documents encouraged me to finish each chapter. I'd also like to acknowledge our technical reviewer, Anselm Bradford, who helped me make sure that the code wasn't going horribly wrong and would work for you.

I'd like to acknowledge Corbin Collins, who helped keep us all on track. It is way too easy to miss a deadline or two without the occasional nudge to keep us all in line, and Corbin provided that.

Finally, I'd like to give a shout out to all the readers of the `http://HowToMakeiPhoneApps.com` blog and the Mobile App Mastery Institute students. Everything in this book is possible because of your generous support and attention throughout the years. I never would have written this book without your feedback and validation.

Preface

Today, learning programming is about learning how to shape our world. Objective-C programmers are in a unique position to create applications that people all over the world can use in their daily lives.

Objective-C is a delight to use. While other programming languages can feel clumsy at times, Objective-C will show you its power and reach with grace. Problems that seem intractable in other programming languages melt away in Objective-C.

At its core, this book is about exploring Objective-C in the language's natural environment. Objective-C has a story to tell in code that is about computer science and solving problems in an elegant way.

Chapter 1

Application Development

This chapter covers some of the essentials involved with getting an Objective-C application set up from the command line and Xcode. You will see how to code command line Mac desktop apps and iOS apps for the iPhone and iPad.

The recipes in this chapter will show you how to:

- Compile an Objective-C program from the command line
- Code a custom class with properties and methods
- Implement both instance and class methods
- Extend existing classes using a category
- Code and compile a Mac command line application
- Use Xcode to set up a Mac application
- Use Xcode to set up an iOS application
- Add user controls to applications using Delegation and Target-Action patterns

> **NOTE:** Most of this book assumes that you are using a Mac with Xcode 4.2, which you can obtain from the Mac App Store at `www.apple.com/mac/app-store/`.

1.1 Creating a Terminal Application

Problem

You want to use Terminal to build a simple Objective-C program that doesn't depend on the extra features that come with Xcode. Your program will use Objective-C to write out a message to the terminal console window on your Mac.

Solution

Use your favorite text editor to create a file in your home directory, which is at `/Users/[yourusername]/`. You can use the text editor vi from your terminal or the GUI-based TextEdit program that comes with your Mac. If you use TextEdit, make sure to save the file that you create as plain text.

In this file, you will add a main function (which, incidentally, would look the same if written in C), import the Foundation framework, and add Objective-C code to write out a `Hello World` message to the console.

To compile this program, you will use a tool called `clang` to create an executable file that you can run from your terminal screen.

How It Works

The code that Objective-C needs to start is always located in a function called `main`, which takes some arguments and returns an integer value. In the first line of code, you import `Foundation`, which is a framework necessary for working with Objective-C objects.

Inside of your main function you must set up an autorelease pool, which is used by Objective-C to manage memory. Once you do that, you can use the `NSString` class to build a Hello World string and `NSLog` to write this string to the console screen.

The terminal command that is used to compile code is called `clang` and it compiles Objective-C programs. Here are some options that you may set when using `clang` to compile your Objective-C programs:

- `-fobj` means that Objective-C is the programming language.
- `-arc` specifies Automatic Reference Counting.

- `-framework` is used to link to the Foundation framework.
- `-o` specifies the name of the executable file that will be created.

> **NOTE:** If your Mac is running OSX 10.7 or greater, then you can use Automatic Reference Counting (ARC). ARC is a new feature available in OSX 10.7 used for memory management and you can get it by adding `-arc` to the statement that you use to compile your program. If you aren't sure what version of OSX you are using just omit `-arc` for now. See Chapter 8 for more details on ARC and memory management in general.

The Code

This is what the code in your plain text file should look like:

```
#import <Foundation/Foundation.h>
int main (int argc, const char * argv[]){
        @autoreleasepool {
                NSString *helloString = @"Hello World";
                NSLog(@"%@", helloString);
        }
        return 0;
}
```

Usage

Open up your terminal and type in the following commands to compile your code. Make sure to navigate to the location where you placed your code file before compiling.

```
clang -fobjc -framework Foundation main.m -o maccommandlineapp
```

For this example, I'm assuming that the code was placed in a file named `main.m` and that the output file will be called `maccommandlineapp`.

Hit return to compile the code. Once the program is compiled, type in `open maccommandlineapp` and press return to run and test your work.

Another window should open up with output that looks like this:

```
Hello World
```

```
logout

[Process completed]
```

1.2 Writing to the Console

Problem

As you're testing code, you would like to be able to write out values to the console window. Objects and primitive type values can be reported but each requires specific string formatters to work with `NSLog`.

Solution

Substitute object and primitives values into `NSLog` to report the values of these variables to the console screen.

How It Works

Object and primitive type values may be reported to the console using `NSLog`. Each type has a different specifier that must be used as a placeholder for the value. You type out the string that you would like to appear in the console while putting in specifiers into the string where you would like to see values reported. You can put as many specifiers into the string as you like, but you must make sure to include each value in the call to `NSLog`.

For example, if you had an integer variable named `myInteger` and a character variable named `myCharacter` and you wanted to report each of these values to the console, you would do something like this:

```
NSLog(@"myCharacter = %c and myInteger = %i", myCharacter, myInteger);
```

> **WARNING:** Each specifier that you include in the NSLog string must have a corresponding value in the comma-separated list to the right or the compiler will throw an error more '%' conversions than data arguments at compile time.

There are a few more specifiers that you may use. See Table 1-1 for a list of commonly used format specifiers.

Table 1-1. *List of Specifiers Used with NSLog*

Specifier	Data Type
%@	Objective-C object (looks at description method)
%d, %D, %i	Int (signed 32-bit integer)
%u, %U	Unsigned int (unsigned 32-bit integer)
%f	Double (64-bit floating point number)
%e	Double (64-bit floating point number in scientific notation)
%c	Unsigned char (unsigned 8-bit character)
%C	Unichar (16-bit character)
%p	Pointer (printed in hexadecimal)
%%	Escape character so you can print the % sign

The Code

Here is how you report the values of various variables to the console using NSLog:

```
#import <Foundation/Foundation.h>
int main (int argc, const char * argv[]){
        @autoreleasepool {

                //To print out primitive types:
                int myInteger = 1;
                NSLog(@"myInteger = %i", myInteger);

                float myFloatingPointNumber = 2;
                NSLog(@"myFloatingPointNumber = %f", myFloatingPointNumber);
                NSLog(@"myFloatingPointNumber in scientific notation = %e",  ➥
                                                        myFloatingPointNumber);


                char myCharacter = 'A';
                NSLog(@"myCharacter = %c", myCharacter);

                //To print out the % symbol
                NSLog(@"Percent Sign looks like %%");
```

```
        //To print out Objective-C objects:
        NSString *myString = @"My String";
        NSLog(@"myString = %@", myString);
        NSLog(@"myString's pointer = %p", myString);

        //To print out a series of values
        NSLog(@"myCharacter = %c and myInteger = %i", myCharacter, myInteger);

    }
    return 0;
}
```

Usage

To test this code, compile the files with `clang` as you did in Recipe 1-1.

```
clang -fobjc -framework Foundation main.m -o maccommandlineapp
```

Run the app by typing `open maccommandlineapp` in your terminal window and you should see output that looks like this:

```
myInteger = 1
myFloatingPointNumber = 2.000000
myFloatingPointNumber in scientific notation = 2.000000e+00
myCharacter = A
Percent Sign looks like %
myString = My String
myString's pointer = 0x105880110
myCharacter = A and myInteger = 1
logout

[Process completed]
```

> **NOTE:** In your output, the pointer for myString will have a different value than mine.