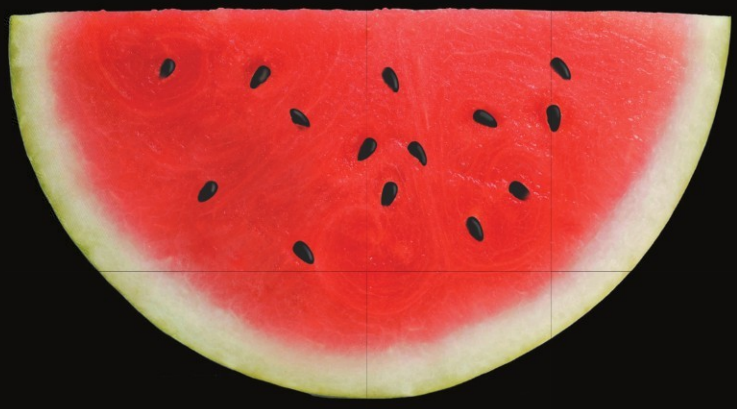
Core concepts and techniques
for iOS developers



iOS 5 Recipes

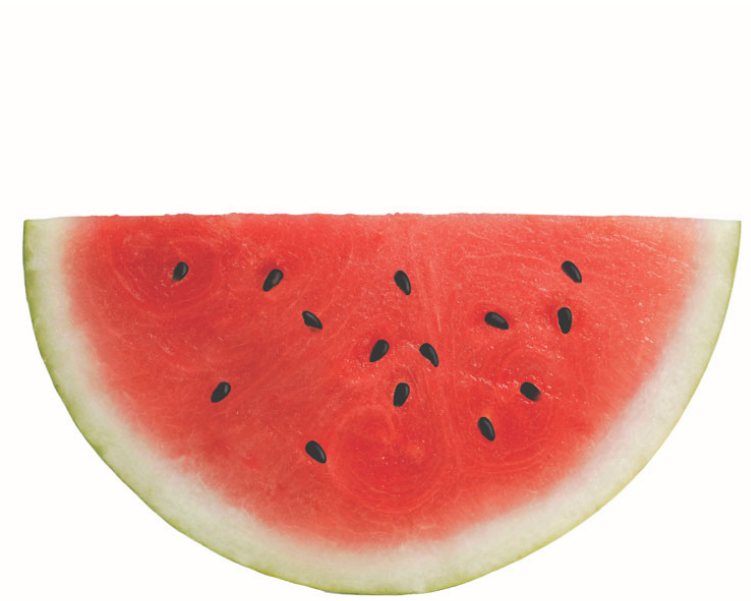
A Problem-Solution Approach

Shawn Grimes | **Colin Francis**

Apress®

iOS 5 Recipes

A Problem-Solution Approach



Shawn Grimes
Colin Francis

Apress®

iOS 5 Recipes: A Problem-Solution Approach

ISBN 978-1-4302-4005-1

ISBN 978-1-4302-4006-8 (eBook)

President and Publisher: Paul Manning
Lead Editor: Michelle Lowman
Development Editor: Ralph Moore
Technical Reviewer: Anselm Bradford
Editorial Board: Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, 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: Anita Castro
Copy Editor: Mary Ann Fugate
Compositor: MacPS, LLC
Indexer: SPi Global
Artist: SPi Global
Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media, LLC., 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.

The source code for this book is available to readers at www.apress.com. You will need to answer questions pertaining to this book in order to successfully download the code.

I dedicate this book to my wife, Stephanie, and my family, who have always supported me and encouraged me.

—Shawn Grimes

This work is dedicated to my grandfather, Larry Cohan.

—Colin Francis

Contents at a Glance

Contents

About the Authors

In 2010, **Shawn Grimes** taught himself Objective-C and iOS development and wrote his first iOS app for the iPad. From Baltimore, Maryland, Shawn attended Capitol College in Laurel, Maryland and graduated in 2003 with a bachelor's degree in software and Internet applications. He founded Shawn's Bits, LLC to create additional apps and present workshops for other aspiring iOS developers. To help local developers, he co-runs the Baltimore Mobile Developers group with Chris Stone. Shawn and his wife, Stephanie, run Campfire Apps, LLC, a mobile app development company focused on children's apps.

Colin Francis is an iOS developer from Gaithersburg, Maryland. After extensively studying computer science, he trained himself in iOS development and worked with Shawn Grimes in Baltimore. Now he lives in Miami, developing iOS apps independently with a focus on utilities and audio-focused software applications.

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. His experience with Internet-related development stretches back to 1996, when he hand-coded his first web site. He may be found at @anselmbradford on Twitter and occasionally blogs at AnselmBradford.com.

Acknowledgments

First, I would like to thank Colin, who took on this project with me and led the way to getting it completed. It has always been a pleasure working with him, and his appetite for knowledge is an inspiration to me.

I would also like to thank my wonderful family, who has inspired me and always supported me: Terri, Larry, Amber, Gloria, Wayne, Kelly, Debbi, Billy, Tom, Mark, Derek, Devin, Bethany, Lauren, Kelsie, Matt, Pam, Mike, Jackie, Gus, Chris, Sam, Brynn, and Courtney.

Finally, I would like to thank my friends, who put up with me taking my laptop everywhere with me so I could work on the book and kept Stephanie company while I was working on this book. Special thanks to Jessop, Lauren, and Henry.

Shawn Grimes

Working on this book has been an immense pleasure, but it was a task that I could not have faced without the full support of my family. I thank every one of them for supporting me and providing suggestions when I was stuck. A huge "thank you" as well goes to my mother, for all her help and support, no matter the occasion.

It has been a terrific experience working with Shawn. Ever since I met him and his wonderful wife, Stephanie, I have particularly enjoyed working on a huge variety of iOS projects with them both. Shawn's technical experience has helped guide me through many tasks with ease, and his generous nature makes him incredibly easy to work with. When he originally brought the project of writing this book to me, I was apprehensive, but with his assistance it was easily turned into the completed product that you see today.

I would like to thank everyone I have worked with in writing this book. Anselm, Ralph, and Mary have been fantastic reviewers, and it was through their intense and dedicated focus that this book has turned out so well. This book also would never have seen the light of day if not for the incredible organizational efforts of both Mark and Anita, as well as the multitude of other individuals at Apress.

Finally, I would like to especially thank all of my friends, of whom there are too many to name individually, who have helped me throughout the process of writing this book. Through my countless hours spent writing, they have constantly been a source of support, providing constant and often incredible suggestions, even if they could not decipher the subject. If not for their original insistence and encouragement to take on such a project, I would never have reached this point.

Colin Francis

Introduction

Once you have already acquired an understanding of the syntax structure of programming in Objective-C for iOS development, the most important part of creating applications is learning to work with the various tools and frameworks provided by Apple. In order to fully develop iPhone and iPad applications, you must have a detailed understanding not only of your development environment, but also of the various elements and functionalities that you are able to use. Regardless of whether your application is playing music, taking pictures, printing documents, or filtering images, this book will help guide you through the setup and building of your functionality.

What to Expect from This Book

The first few chapters of this book are devoted to acquiring a basic understanding of your development environment. You will learn a variety of ways to work within Xcode and Interface Builder, as well as the various standard user interface elements with which you can build your application. The remaining 13 chapters focus on specific examples, or recipes, of a variety of different applications, in order to demonstrate exactly how to implement each functionality from start to finish.

How This Book Is Organized

The example-based chapters of this book do not particularly build off of one another, in the hope that you can simply open up to any chapter of specific interest and start building a certain type of application. However, it is highly recommended that you read the first three chapters in order to acquire a solid understanding of working with Xcode and Interface Builder, if you have not already. Some of the methods used in these early chapters, such as those used to create properties, are referenced throughout the text and should be fully understood.

Throughout this book, it is assumed that you are developing in the latest versions of iOS (5.0) and Xcode (4.2) at the time of writing. This means that every recipe in this text assumes that you will be using ARC (Automatic Reference Counting), and as such does not include significant memory management. This also means that depending on when you are reading this, your results may look slightly different, though the basic functionality should remain similar.

Many of the recipes in this book cannot be fully tested on the iOS simulator, and as such will require both an Apple device and a provisioning profile, which can be acquired when you subscribe to Apple's iOS Developer Program. Each recipe that cannot be tested in the simulator will mention this fact.

Source Code and Errata

All the source code used in this book is available online for download at www.apress.com, and it is entirely free for use in any application, whether commercial or personal. A number of people have worked hard to keep this code as perfect and error-free as possible, but a few typos or bugs may become apparent with extensive use. Any corrections to the text or code are available in this book's "Errata" section, also at www.apress.com.

Contact Information

If you have any questions or comments regarding the book or its source code, we would be happy to assist. You can contact either author:

Colin Francis:
E-mail: cmfrancis24@gmail.com

Shawn Grimes:
E-Mail: shawn@shawnsbits.com
Web: www.shawnsbits.com

Chapter 1

Xcode 4 Tips and Tricks

Xcode 4 brought forth a number of changes to the look and feel of Xcode as well as changes in functionality. As with any major change to the way people do things, it was met with mixed reviews and some complaints. In this chapter, we'll steer clear of the shortcomings of Xcode 4 and insted focus on its strengths and improvements, which are many.

Xcode 4: An Introduction

The very first thing you'll notice about Xcode 4 is its unified interface window. Everything has been brought into one window, and the new interface has introduced the common interface element of tabs instead of multiple windows.

Figure 1–1 shows an example of the Xcode user interface, including its various display panes. These panels help you to navigate, build, and debug your application. Their visibility can be adjusted easily using the View buttons in the upper right-hand corner in order to provide more viewing space for the Editor.

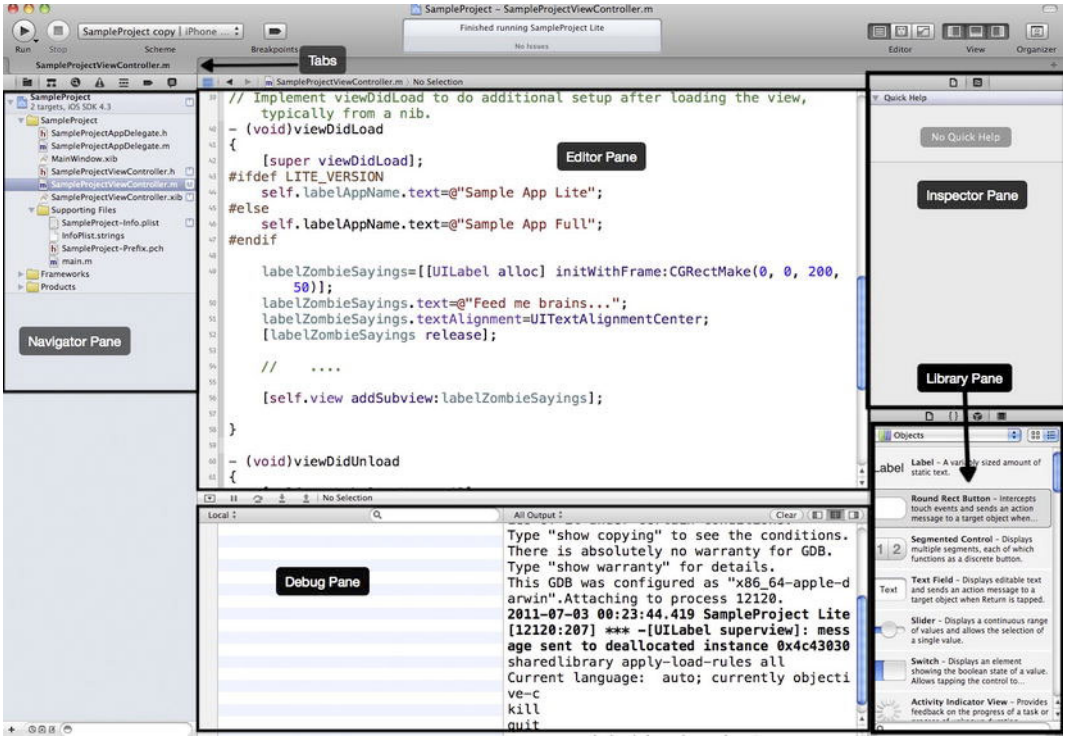


Figure 1–1. *The Xcode interface*

Even Interface Builder has been included in the single window interface of Xcode 4. With the inclusion of Interface Builder, Apple has built some swift functionality to help you go from visual interface to functioning code. Figure 1–2 shows Interface Builder being used to construct an application's user interface. This will be covered more in Chapter 2.

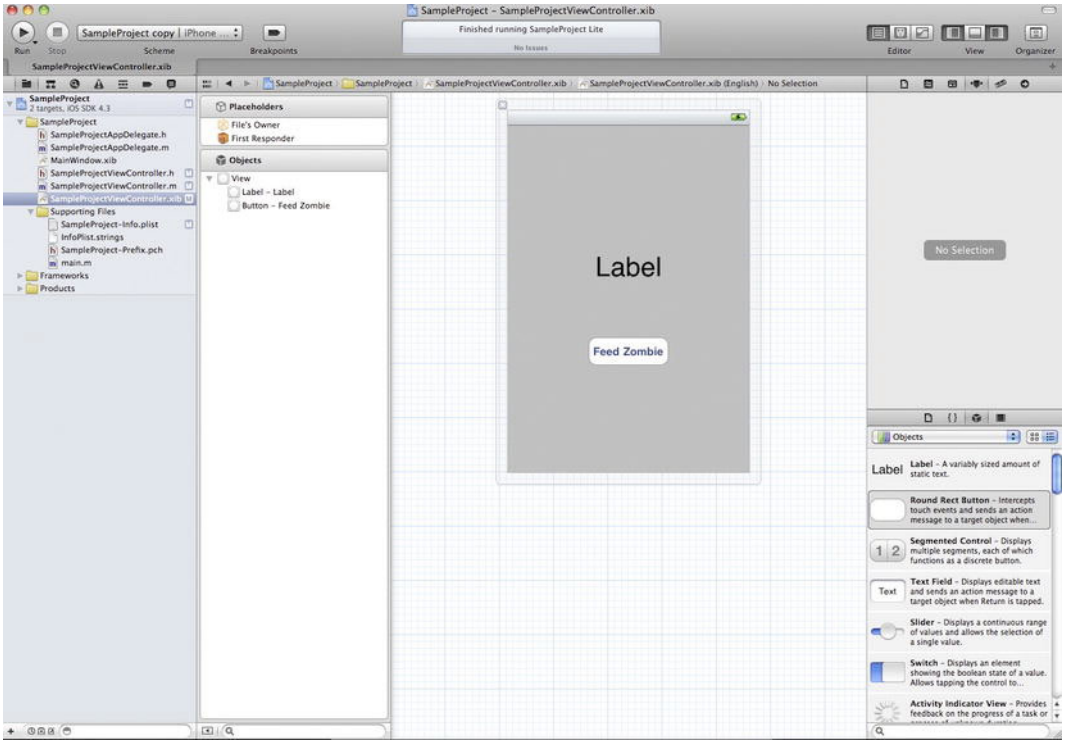


Figure 1–2. *Interface Builder*

With the Assistant Editor, you can easily see two related files side by side. This is very useful when working with class headers and implementation files because you can easily modify both files in a single view. By using the small navigation area at the top of each pane, you can either select specific files to show together, or specify "Counterparts" to automatically show the related header or implementation file, as shown in Figure 1–3.

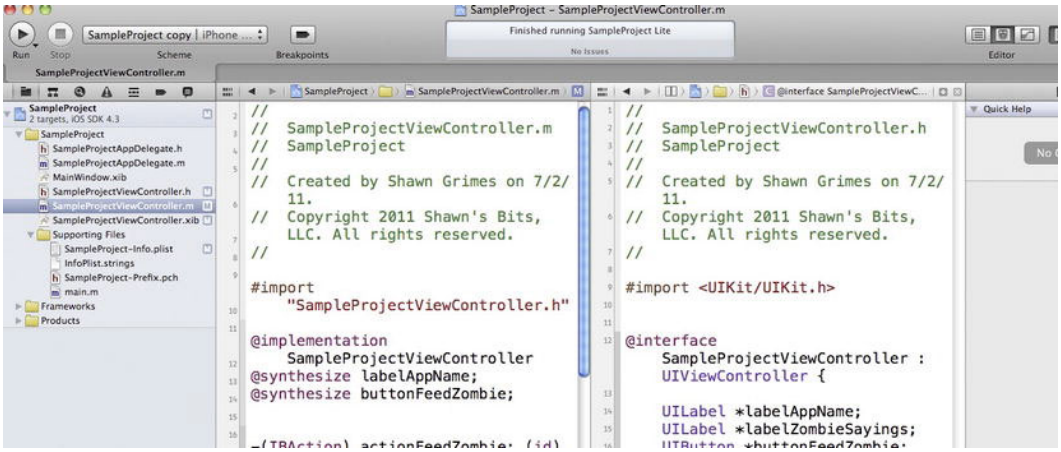


Figure 1–3. *The Assistant Editor*

A feature of Xcode 4 that is sure to save you time is Fix-It. This feature tries to detect common programming mistakes and offers suggestions on how to fix them. It does this while you are writing the code rather than waiting for you to run a build command. This makes it a great time saver for common mistakes.

Xcode 4 also features better source control integration with Git. You are now given the option to create a local Git repository every time you start a new project, and modified files are clearly marked in the Navigator pane. The Timeline Editor view will even show you changes that you've made since the last check-in or compare your current file to any past file version in the repository. This view back in time is very similar to the Time Machine backup interface in Snow Leopard, as shown in Figure 1–4.

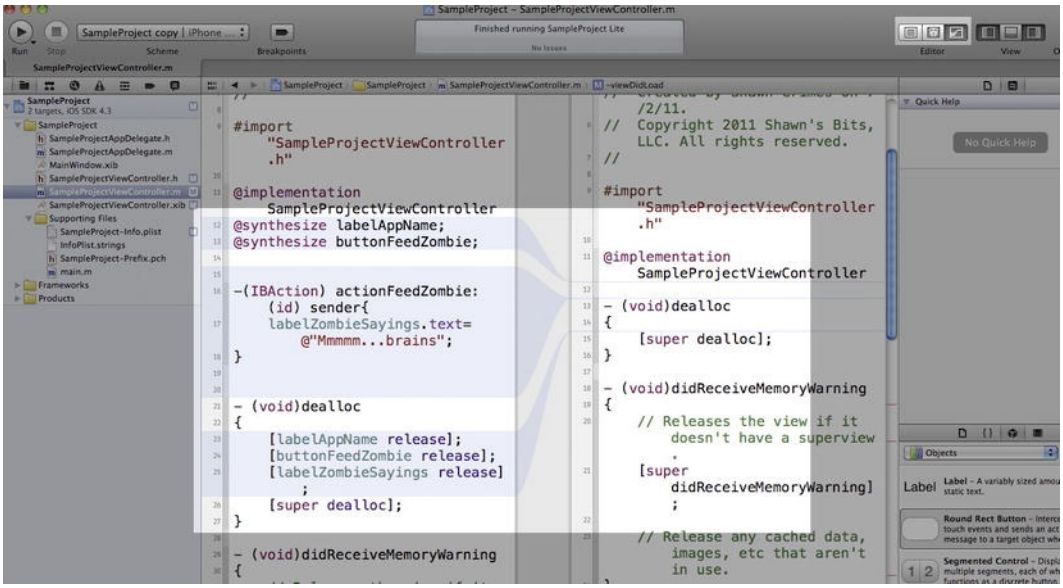


Figure 1–4. *Timeline Editor displaying recent revisions*

Build a Lite and Full Version in One Xcode Project

Offering a lite version of your app is a great way to give customers a chance to try your app before buying it. Maintaining two code bases, however, can be quite tiresome and get out of hand as you implement new features into your app. While the ability to maintain two build targets was available in Xcode 3, Xcode 4 has made it even easier.

Select your project file in the Navigator area, and then select the build target for your project. Now press D to duplicate the target. You will be prompted to "Duplicate Only" or "Duplicate and Transition to iPad." Click Duplicate Only to create a new target that will be used for your Lite build, as shown in Figure 1–5. This will result in a separate build target with which you can implement a second version.

Figure 1–5. *Project duplication options*

Rename the new target with an appending "Lite". You will also want to go to the Build Settings tab and find the Product Name attribute under the Packaging heading in order to append "Lite" to the app name. Now that the build name is set, you need a way to differentiate between the two builds in your source code. For that, scroll down and find the Preprocessor Macros, and add a new macro named `LITE_VERSION`. Make sure to add the new macro for the Debug and Release build settings. Figure 1–6 shows an example of these changes.

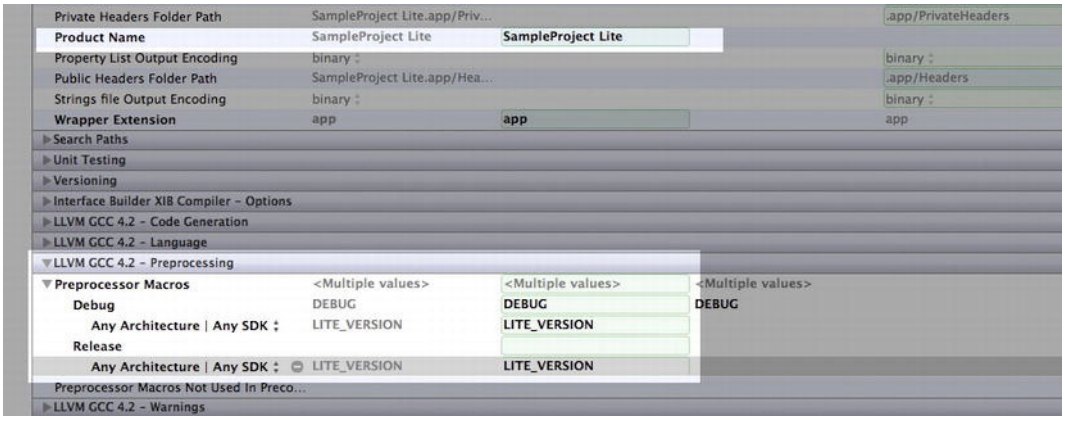


Figure 1–6. *"Lite" application configuration*

If you build and run that now, you will end up with a second app on your device with the name “SampleApp Lite” as the title, but it runs the same code as the regular version of the app, as demonstrated in Figure 1–7. Keep in mind that the two targets must have separate bundle identifiers in order to show up as separate apps. This is the default setting, but be careful when making changes.

Figure 1–7. *Two versions of the same application*

To build different features into your app, you will need to utilize that preprocessor macro you created. Anywhere in your code that you want to specify different code for your lite version vs. the full version, use the following `#ifdef` directive:

```
#ifdef LITE_VERSION
//Stuff for Lite version
Self.labelAppName.text=@”Sample App Lite”;
#else
//Stuff for Full version
Self.labelAppName.text=@”Sample App Full”;
#endif
```

Build and compile the two apps on the simulator, and you will see that the apps change the code they compile and run based on the preprocessor macro and the power of the `#ifdef` directive. Figure 1–8 demonstrates the result of this configuration.

Figure 1–8. *Full and "lite" applications*

> **NOTE:** You can also control what files are included in each build. For instance, you may not need to include the full version artwork in the lite version. Click your Lite project target and go to the Build Phases tab. Now expand the Copy Bundle Resources ribbon, and remove or add any files that are specific to the lite version.

Zombie Hunter

Occasionally, you will run into an error described only as "`EXC_BAD_ACCESS`," and unfortunately, it doesn't tell you in which line the bad access is occurring. This is caused when you have released a variable and then tried to access that freed object. When an

object is no longer there and you try to access it, the term is a zombie object. Enter the zombie hunter, the `NSZombieEnabled` flag. This is not new to Xcode 4, but where you set the flag can be difficult to find in Xcode 4. Go to the Product menu and select "Edit Scheme…". Now select the Run step and click the Arguments tab. Under the Environment Variables section, add NSZombieEnabled and set the value equal to YES, as shown in Figure 1–9.

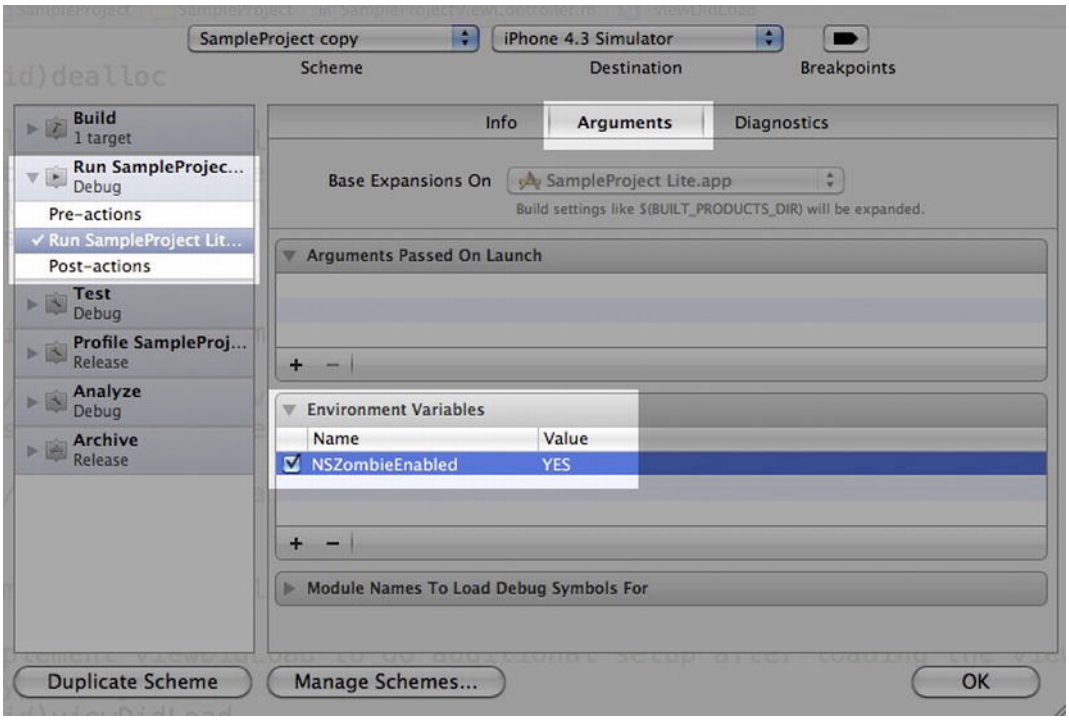


Figure 1–9. *Enabling* `NSZombieEnabled`

The next time you run your code, the zombies will be identified in the Debug window. Figure 1–10 displays an example of a zombie caught by Xcode.

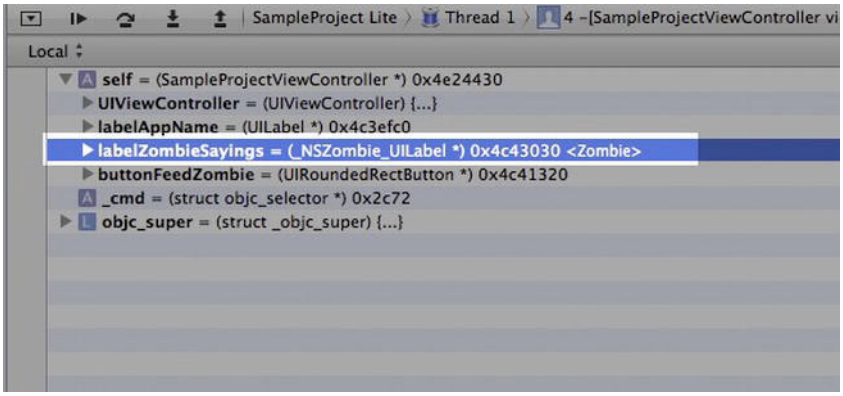


Figure 1–10. *A zombie identified*

Version Control with Xcode 4

Version control can be a daunting concept to new developers, but it is something worth learning. Once you have started using version control, you will wonder how you ever got along without it. Its benefits for teams of developers are fairly obvious. Individual team members can work on different parts of an app without stepping on each other's code.

Single developers can benefit from version control as well. With multiple branches, you can add features to your app without disturbing the previous released version. If a bug is discovered in your released version, you can switch branches and fix the bug without impacting the future version of your app. Then you can merge the two versions and have a new version that contains the bug fixes and the new features. All the while, you can reach back to any point in time and see changes that were made to your code.

Xcode 4 introduced version control into the Xcode environment. Initially, it supported only local Git repositories, but Xcode 4.2 has brought remote repositories to the environment. This is great news if you are part of a development team or if you work on multiple machines. Remote repositories also provide a safe place for your code in case of computer failure or loss.

Creating a Local Repository

Whenever you start a new project in Xcode 4, you are given the option to create a local Git repository, as shown in Figure 1–11. If you select this box, Xcode will create the local repository and automatically add the project files it thinks are necessary.

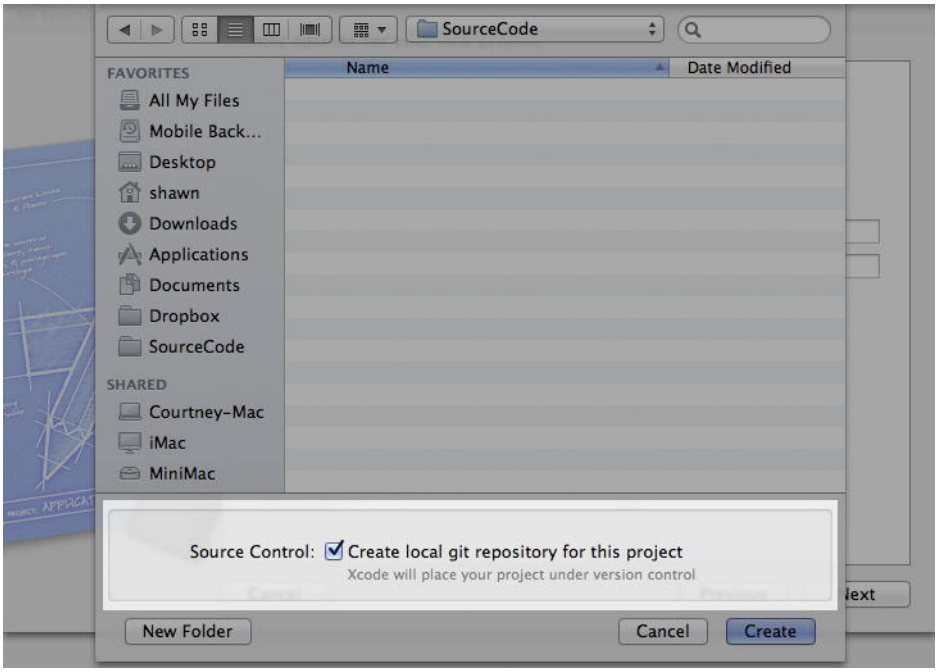


Figure 1–11. *Creating a Git repository*

As you make changes to your project and its files, their source control status will be displayed in the navigator window. “A” is for when a file has been added to your project, and “M” is for when it has been modified since the last check-in. Figure 1–12 shows a navigation pane with multiple files with these statuses.

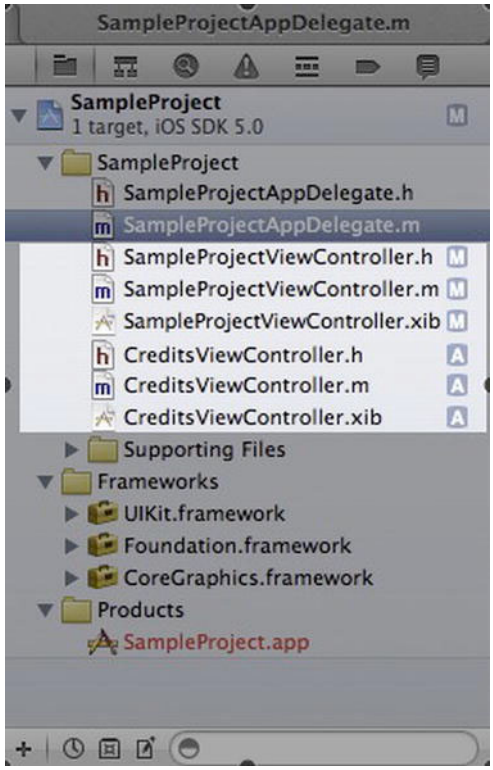


Figure 1–12. *Modified and added project files*

You can filter the navigator contents so that you see only the files that are pending changes to the source control repository by clicking the middle icon in the bottom of the navigator pane, as shown in Figure 1–13.

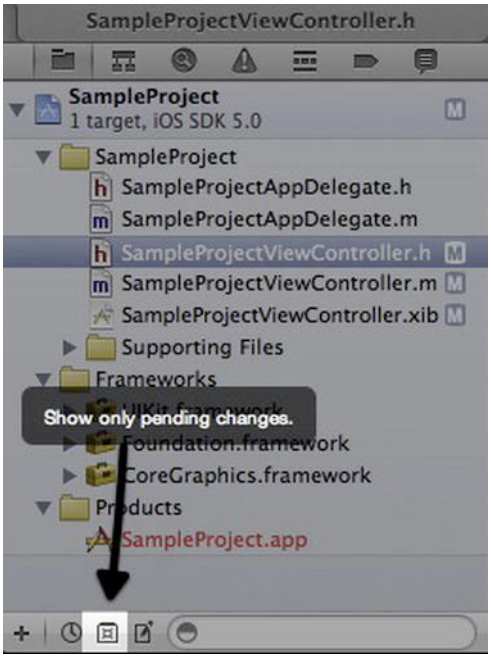


Figure 1–13. *Filtered modified files*

To commit your changes back into your local repository, go to **File ➤ Source Control ➤ Commit** or C. The Commit window will be presented. By clicking a modified file, you will see your edited version in the left pane and the current version in the repository in the right pane. All of your changes will be highlighted so that you can easily see the differences between the two files. Figure 1–14 displays such a window with highlighted changes.

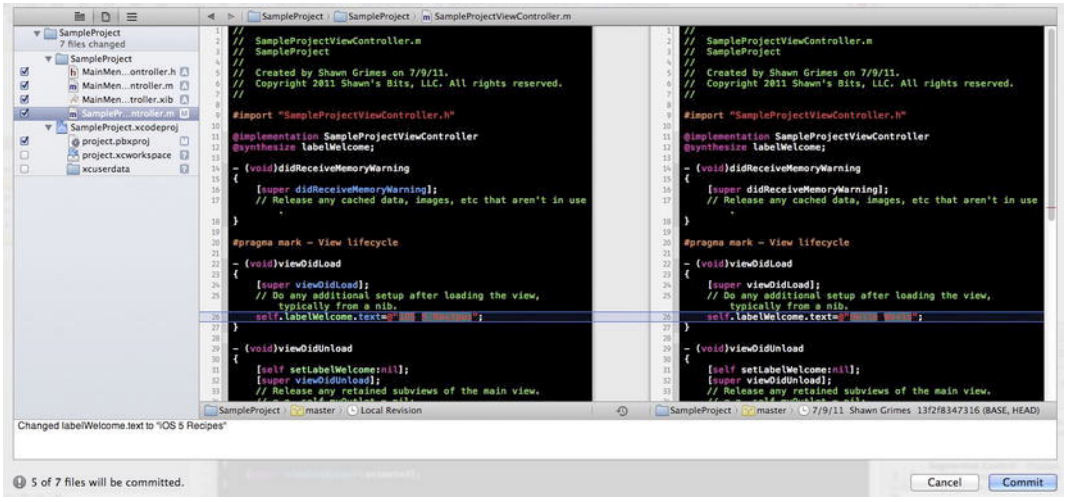


Figure 1–14. *Viewing file changes for committal*

Worth mentioning is the fact that the left pane is a live editor, so if you see something that should not be committed, such as an `NSLog` statement, this is your chance to comment it out or make the necessary changes.

Xcode does a good job of suggesting which files should be committed. You do not want to version control your workspace file (`*.xcworkspace`) or your userdata directory (`xcuserdata`). Generally Xcode will not check those files, and you will note the "?" mark icon next to the files. This means they are not currently under version control, nor should they be. Figure 1–15 shows these non-version-controlled files/directories.

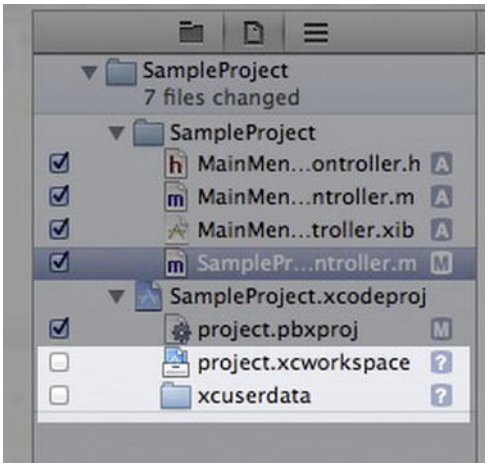


Figure 1–15. *Disabled version control for certain directories*

At the bottom of the commit window, as shown in Figure 1–16, is where you need to enter a message about the changes you have made before committing. Your commit message should be a descriptive summary of the changes you have made to your code, such as "added such and such feature."

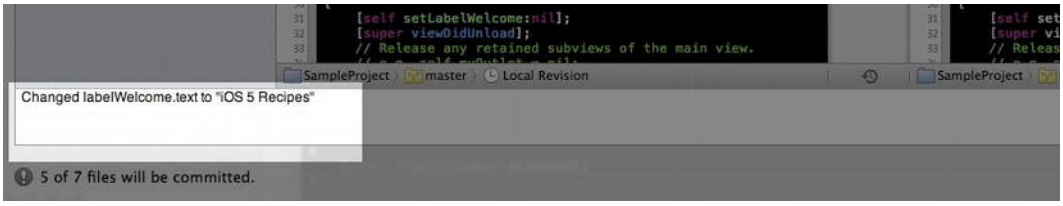


Figure 1–16. *Commit message*

Branching and Merging

Branches are copies of your project that you can work on without disturbing the main branch, also known as the master branch. They allow you to add features and fixes without affecting the main build.

To manage your repository, you can go to **Window Organizer** or 2 and click the Repositories tab. In this view, you will see a list of repositories that Xcode knows about.

As shown in Figure 1–17, you can click the `Branches` folder under a repository to see a list of branches available to Xcode for this repository.

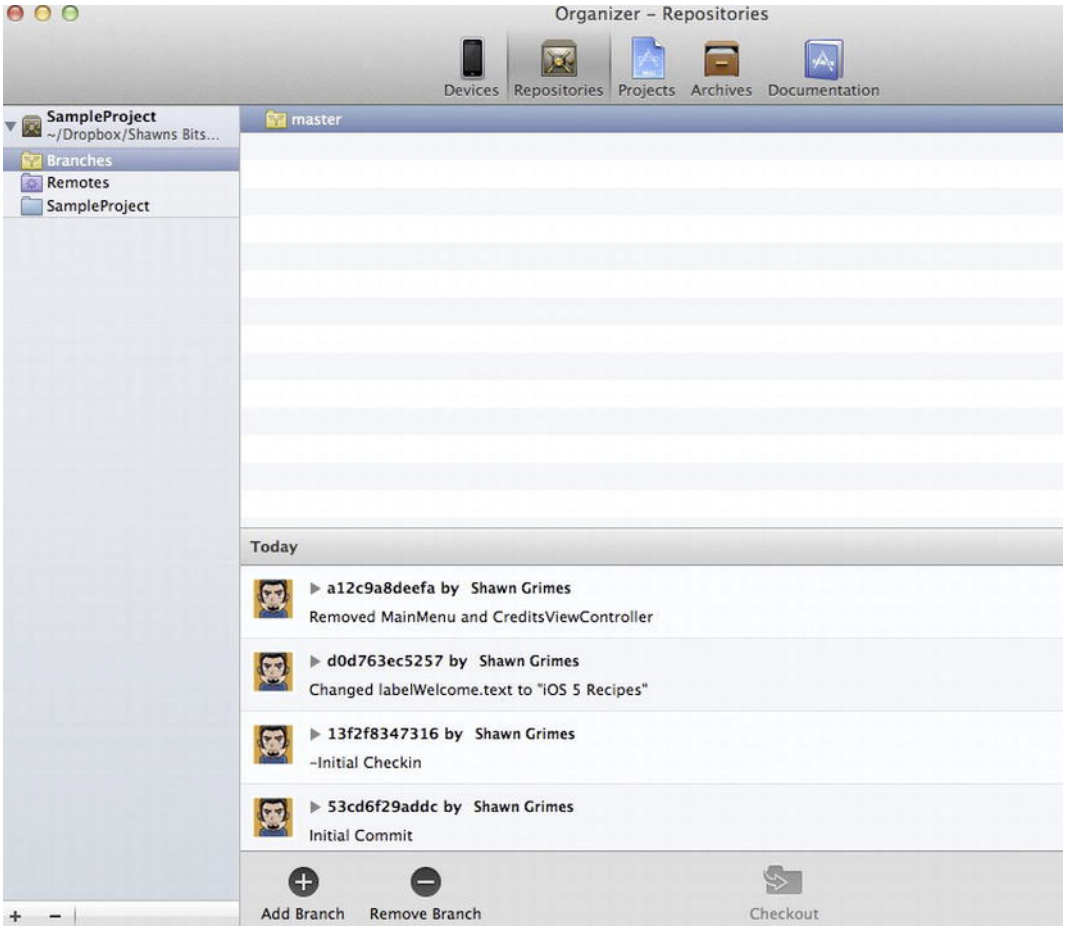


Figure 1–17. *Repositories tab in Organizer*

When you select a branch, you will see a list of the latest commits to that branch. The information includes who made the commit and their commit message.

Create a new branch to start adding a main menu to your app. Click the Add Branch button at the bottom of the Organizer window. In the window shown in Figure 1–18, type a branch name and click the check box next to “Automatically switch to this branch”. This will duplicate the code in the master branch into a new branch called “MainMenu,” and then it will switch you to that branch.

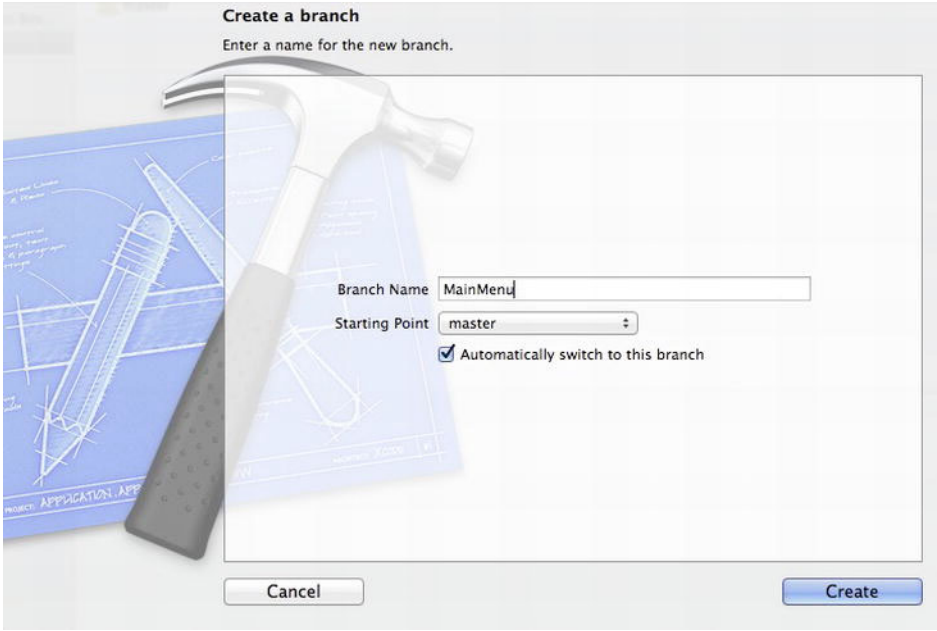


Figure 1–18. *Creating a branch*

Now that you are working in the MainMenu branch, you can add a new view controller for the main menu without affecting the rest of the app source code. After adding the view controller and coding it up, you can commit this back to the source code repository. Again, this will affect only the MainMenu branch and not make any changes to the master branch.

To merge the two branches, you want to switch to the branch that you want to merge the changes into. You are done coding up the MainMenu, and you want to put it into the master branch, so you are going to switch to the master branch. This is done from the Organizer window, so press 2.

Click the project folder, and then click the Switch Branch button on the bottom right. Select the branch you want to switch to—master in this case—and click OK. This will switch your active branch back to the master branch, and now you can merge the two branches together. Figure 1–19 highlights these steps.