Fritz Anderson

Completely Revised

Covers Xcode 4 for iOS and Mac Development

Xcode 4

UNLEASHED

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Praise for *Xcode 4 Unleashed*

"There are many great resources out there for learning iOS and Mac development that cover Objective-C and Cocoa. Xcode is an extremely important part of iOS and Mac development that often gets overlooked. You owe it to yourself to understand Xcode and all of its quirks and power user features to achieve maximum efficiency as a developer. *Xcode 4 Unleashed* can help you do just that."

—Tony Hillerson,
Member and Software Architect, Tackmobile.com

"Fritz Anderson's *Xcode Unleashed* series is the definitive guide to using Xcode. *Xcode 4 Unleashed* has been rewritten to cover the sweeping changes in recent versions of the product. I highly recommend this book to anyone who uses Xcode—newbies and grizzled veterans alike."

—Duncan Champney,
Director of Software Development, WareTo

Praise for *Xcode 3 Unleashed*

"I would recommend this book to anyone that is serious about programming on the Mac. It is an excellent resource; I plan to refer to it often."

—Cortis Clark

"I've been doing Mac OS X development for seven years, so I was surprised at how much new information I learned in this book. The details on building and the overview of Instruments were invaluable."

—Dan Wood, Karelia Software

"There isn't a better book on the market to understand Apple's powerful—yet free integrated development environment, Xcode. Fritz Anderson stands among the most literate programmers I know, simultaneously able to provide a high-level development narrative while delving into the countless crucial details that make up modern development. I recommend *Xcode 3 Unleashed* to both novices as an introduction and professionals as a reference."

—Jonathan 'Wolf' Rentzsch, http://rentzsch.com

"Whether you are new to programming on Mac OS X or a seasoned veteran, *Xcode 3 Unleashed* has something for you. The book is full of examples and practical information. I recommend this book for anyone doing serious development on Mac OS X 10.5."

—Dave Dribin

Fritz Anderson

Xcode® 4

UNLEASHED

SAMS | 800 East 96th Street, Indianapolis, Indiana 46240 USA

Xcode® 4 Unleashed

ISBN-13: 978-0-672-33327-9
ISBN-10: 0-672-33327-9

The Library of Congress cataloging-in-publication data is on file.

Printed in the United States of America

First Printing May 2012 with corrections November 2012

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or programs accompanying it.

Bulk Sales

Sams Publishing offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales
1-800-382-3419
corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact

International Sales
international@pearson.com

Editor-in-Chief
Mark Taub

Acquisitions Editors
Trina MacDonald
Chuck Toporek

Managing Editor
Kristy Hart

Project Editor
Jovana San Nicolas-Shirley

Copy Editor
Apostrophe Editing Services

Indexer
Erika Millen

Proofreaders
Jess DeGabriele
Chrissy White

Technical Editors
Duncan Champney
Tony Hillerson
George Sealy
Rob Wittner

Publishing Coordinator
Olivia Basegio

Cover Designer
Gary Adair

Senior Compositor
Gloria Schurick

Contents at a Glance

Table of Contents

About the Author

Fritz Anderson has been writing software, books, and articles for Apple platforms since 1984. He has worked for research and development firms, consulting practices, and freelance. He was admitted to the Indiana bar, but thought better of it. He is now an iOS and Mac programmer for the Scholarly Technology department at the University of Chicago. He has two daughters.

Dedication

For Steve Jobs, who made my life's work possible,

and

for Kate and Bess, who made my life.

Acknowledgments

Only part of the effort that went into putting *Xcode 4 Unleashed* into your hands was spent at a text editor. I owe a debt of thanks to those without whom this book could not have been made.

Chuck Toporek, my editor, showed patience, good humor, and good sense in support of our second effort together. *Xcode 3.2 Unleashed* had reached nearly 400 pages when Apple announced Xcode 4, which would not see its first release for 8 months; major revisions spanned another 8. He encouraged a dispirited author through the long process of dumping everything, waiting, and starting again.

Chuck's assistant, Olivia Basegio, made sure the contracts, correspondence, (and advance payments!) all got through.

Trina MacDonald took over after Chuck left for Apple. She, and Jovana San Nicolas-Shirley, who oversaw production, were extremely accommodating with an author whose opinions on how to write a book are… exacting.

I'm especially grateful for the advice of the technical reviewers Chuck found for me. You did yeoman service in the face of a subject in rapid flux and saved me many embarrassments. Of course, any errors that remain are my own.

Quinn Dombrowski, boss and friend, took a kind interest in my book. She was generous with vacation time so I could get it done and patient when I began to fray under the strain of 60-hour weeks.

Bess and Kate bore more than daughters should of my doubts and frustrations, and were simply confident that I would do fine—which was all they needed to do.

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

You can email or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books stronger.

Please note that I cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail I receive, I might not be able to reply to every message.

When you write, please be sure to include this book's title and author as well as your name and phone number or email address. I will carefully review your comments and share them with the author and editors who worked on the book.

Email: feedback@samspublishing.com

Mail: Sams Publishing

800 East 96th Street

Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

This page intentionally left blank

Introduction

IN THIS CHAPTER

- Introducing Xcode
- Understanding the goals of *Xcode 4 Unleashed*
- Considering organization
- Typographic conventions

Welcome to *Xcode 4 Unleashed*! This book shows you how to use Apple's integrated development environment to make great products with the least effort.

Xcode 4 is the descendant of a family of development tools dating back nearly 20 years to NeXT's ProjectBuilder. It started as a text editor, a user-interface designer, and a front end for UNIX development tools. It has become a sophisticated system for building applications and system software, with a multitude of features that leverage a comprehensive indexing system and subtle incremental parser to help you assemble the right code for your project—and get it right the first time.

That much power can be intimidating. My aim in *Xcode 4 Unleashed* is to demystify Xcode, giving you a gradual tour through examples that show you how you can use it day to day. If you come to Xcode 4 from previous versions, the changes may be overwhelming; Chapter 24, "Xcode 4 for Xcode 3 Veterans," is just for you, an introduction to where you can find the facilities you're accustomed to—even newcomers wanting a quick overview may find it useful.

How This Book Is Organized

First, a word on my overall plan for *Xcode 4 Unleashed*. This is a book about developer tools. If it teaches you something about how to use the Cocoa frameworks, or something about programming, that's fine, but that's incidental to showing you what Xcode can do.

Every tour needs a pathway, and every lesson needs a story. The first three parts of this book demonstrate Xcode through three applications—a command-line tool, an iOS

app, and a Mac OS X application—that calculate and display some statistics in American football. None of the apps are useful; the graphical apps run almost entirely on sample data, but they demand enough of the development tools to give you a solid insight into how to use them.

The full code for the example programs is available online from `www.informit.com/title/9780672333279`. In the interest of space, I'll be showing only excerpts.

I'm using applications for iOS and a Mac OS X as examples, but read both Part II, "The Life Cycle of an iOS Application," and Part III, "Xcode for Mac OS X," even if you're interested only in one platform. The applications are only *stories*; the techniques apply to both platforms.

First Steps

In Part I, I'll take you from installing Xcode and running your first project, through basic debugging skills. You'll work through a small command-line application. The application may be simple, but you'll learn foundational skills you'll need before adding the complexity of graphical apps.

- **Chapter 1, "Getting Xcode":** Some things to consider before you download Xcode 4; two ways to download and install it.
- **Chapter 2, "Kicking the Tires":** Your first look at Xcode, setting a trivial project up and running it.
- **Chapter 3, "Simple Workflow and Passive Debugging":** You'll write, build, and run a simple application and respond to a crash.
- **Chapter 4, "Active Debugging":** You'll take charge of debugging by setting breakpoints and tracing through the program. I'll show you how to organize your workspace.
- **Chapter 5, "Compilation":** A pause for a description of the process of building an application.
- **Chapter 6, "Adding a Library Target":** Add a library target to a project and learn how to build a product from multiple targets.
- **Chapter 7, "Version Control":** Why source control is important and how to take advantage of Xcode's built-in support for versioning through Git and Subversion.

The Life Cycle of an iOS Application

Part II tells the story of a small iPhone app and how to use Apple's developer tools to build it. It introduces you to the graphical editor for user interfaces and shows how to profile an app to optimize its speed and memory burden.

- **Chapter 8, "Starting an iOS Application":** You'll start by creating an iOS project and learn the Model/View/Controller design at the core of Cocoa on iOS and Mac OS X alike.
- **Chapter 9, "An iOS Application: Model":** Design a Core Data schema and supplement it with your own code.
- **Chapter 10, "An iOS Controller":** Create a controller to link your model to the onscreen views. On the way, I'll tell you about refactoring and Xcode's continual error-checking.
- **Chapter 11, "Building a New View":** You'll design the user-interface views for your app with the integrated Interface Builder and take advantage of source-code completion.
- **Chapter 12, "Adding Table Cells":** While adding an onscreen component to your app, you debug memory management and control how Xcode builds, runs, and tests your apps through the Scheme editor.
- **Chapter 13, "Unit Testing":** Unit testing can speed development and make your apps more reliable. I'll show you how Xcode supports it as a first-class part of the development process.
- **Chapter 14, "Measurement and Analysis":** Use Instruments to track down performance and memory bugs.
- **Chapter 15, "Storyboard":** Simplifying iOS development with Storyboard, which enables you to specify the whole structure of your app in just one file.
- **Chapter 16, "Provisioning":** The three things you must understand to put your app on a device, from testing to the App Store.

Xcode for Mac OS X

Part III shifts focus to Mac OS X development. Some concepts are more important to Mac OS X than iOS, but you'll be learning techniques you can use whatever your platform.

- **Chapter 17, "Starting a Mac OS X Application":** Carrying iOS components over to Mac OS X, Automatic Reference Counting (ARC), and menus and how to link them to the responder chain.
- **Chapter 18, "Wiring a Mac Application with Bindings":** As you build a popover window, you'll use Mac OS X bindings to simplify the link between your data and the screen. With autoresizing, you can set up views to resize themselves.
- **Chapter 19, "A Custom View for Mac OS X":** Add a custom view to your app to see how Interface Builder can lay it out and configure it, even though it's not a standard Cocoa component.

- **Chapter 20, "Localization and Autolayout":** How you can translate your Mac and iOS apps into other languages. I'll show you how to use Lion's autolayout feature to give your views the right size and layout no matter how their contents change.
- **Chapter 21, "Bundles and Packages":** You'll master the fundamental structure of most Mac and iOS products and how both platforms use the `Info.plist` file to fit apps into the operating system.
- **Chapter 22, "Frameworks":** Package and share a complete subprogram you can incorporate into any Mac OS X application.
- **Chapter 23, "Property Lists":** Learn the basic JSON-like file type for storing data in both Mac OS X and iOS.

Xcode Tasks

Part IV moves on to topics that deserve a more concentrated treatment than Parts II and III.

- **Chapter 24, "Xcode 4 for Xcode 3 Veterans":** Written from the point of view of developers moving from Xcode 3's document-based approach to development to Xcode 4's "browser'' model, this chapter can help you find what you need from Apple's developer tools.
- **Chapter 25, "Documentation in Xcode":** How Xcode gives you both immediate help on API and browsable details on the concepts of Cocoa development. Find out how you can add your own documentation to the system.
- **Chapter 26, "The Xcode Build System":** I'll show you the fundamental rules and tools behind how Xcode goes from source files to executable products.
- **Chapter 27, "Instruments":** Using Apple's timeline profiler, you can go beyond basic performance and memory diagnostics to a comprehensive look at how your program uses its resources.
- **Chapter 28, "Snippets":** A roundup of tips, traps, and features to help you get the most from the developer tools.

Appendixes

The appendixes contain references to help you get your bearings on Objective-C, master the build system, choose the right starting points for your project, and find help and support.

- **Appendix A, "Objective-C":** A summary of the core programming language for development on Apple platforms, progressing from the simple additions it makes to C, to the specialized features that support Apple frameworks.
- **Appendix B, "Some Build Variables":** The most important configuration and environment variables from Xcode's build system.

- **Appendix C, "Project and Target Templates":** Where to start your new projects and files.
- **Appendix D, "Resources":** A compendium of books, tools, and Internet resources to support your development efforts.

About Versions

This book was written over most of a year, through the Fall of 2011. In that period Apple issued three major revisions of Xcode, as well as major revisions to both iOS and Mac OS X. *Xcode 4 Unleashed* covers Xcode 4.2, the first version with full support for iOS 5 and Mac OS X 10.7 Lion.

To demonstrate legacy techniques, Part II targets iOS 4.3, with an excursion into iOS 5 for Storyboard. Part III covers a Lion application. Where possible, I've noted changes introduced in Xcode 4.3.

Since the first printing of this book, more versions of Xcode have come out, with new features for Mac OS X 10.8 and iOS 6. We're not leaving you high and dry: Xcode 4 Unleashed: A Supplement is now available and it's free. To access this material, please register this book at informit.com/register and enter the ISBN for the print edition: 9780672333279.

Conventions

This book observes a number of typographic and verbal conventions.

- Human-interface elements, such as menu items and button labels, are shown **like this**.
- Filenames and programming constructs are shown `like this`. This will sometimes get tricky, as when I refer to the product of the "Hello World" *project* (plain text, because it's just a noun) as the *file* `Hello World`.
- Text that you type in will be shown **`like this`**.
- A new term is called out *like this*.

You'll do some command-line work in the terminal. Some of the content is wider than this page, so input lines break with backslashes (\) at the ends. Long output lines break simply by splitting them and indenting the continuations. When that output includes long file paths, ellipses (…) replace components, leaving the significant parts.

About the sample code in this book: A significant portion of the code I'll have you write is wrong. As an experienced programmer, you will be in pain and want to correct it. Leave it alone; the errors are intentional.

I'll refer to the location of various development resources as the `/Developer` directory. Xcode 4.3 moved that directory into the Xcode app itself. You can find the additional developer applications in the **Xcode** menu.

For many, many years the Macintosh had a one-button mouse (don't laugh—most purchasers didn't know what a mouse was; try pushing the wrong button on an old Mac mouse). Now it has four ways to effect an alternative mouse click: You can use the right button on an actual mouse; you can hold down the Control key and make an ordinary click; you can hold down two fingers while clicking a multitouch trackpad (increasingly common even on desktop Macs); or you can tap at a designated corner of a multitouch trackpad. And there are more variations as you run through the configurations. Unless the distinction actually matters, the text simply calls for a "right-click," and you can sort it out.

CHAPTER 8

Starting an iOS Application

IN THIS CHAPTER

- Planning with the Model-View-Controller pattern
- Creating an iOS project
- Using Project Search and Replace

Now that you have the basic skills down, let's move on to a real project. You'll build an iPhone application that manages a list of quarterbacks and displays their game and career statistics.

Planning the App

Before coding, it's best (though not customary) to know what you're doing. Specifically, what are you going to present to the app's user, what data do you need to keep to make that presentation, and how do you translate between the data and the presentation?

Model-View-Controller

The Model-View-Controller (MVC) design pattern formalizes those questions into an architecture for graphical applications. The Cocoa Touch application framework is designed to implement applications that follow the MVC pattern. If you don't follow it, you will find yourself "fighting the framework": Winning through to a finished application would be difficult, and maintaining it would be miraculous. The Xcode development tools are designed to support Cocoa programming and therefore the MVC pattern.

MVC divides the functionality of an application into three parts, and each class in the application must fall into one of them:

- *Model objects* embody the data and logic of a particular problem domain. Models tend to be unique to each application. You can create your own subclasses of `NSObject` or `NSManagedObject` to give life to your models.

- *View objects* handle user interaction, presenting information and enabling the user to manipulate data or otherwise influence the behavior of the program. Views are usually drawn from a repertoire of standard elements, such as buttons, tables, scrollers, and text fields. Views ideally know nothing about any problem domain: A button can display itself and report taps without needing to know what tapping means to your application. In iOS, views are instances of `UIView` or its many subclasses.
- *Controller objects* mediate between the pure logic of the model and the pure mechanics of the views. A controller object decides how views display and how user actions translate into model events. In iOS, controllers are almost always instances of subclasses of `UIViewController`.

NOTE

Okay, in practice some classes won't fall exactly into model, view, or controller. If you have a view custom-built to display your particular data, making that view completely independent of your data model is foolhardy. Still, MVC is an important discipline: If you fudge on it, you should be aware that you're fudging and consider whether you can restore the MVC separation.

The Model

From the nature of a passer rating, all you need is one model class: a `Passer` to carry one passer's name, and the total attempts, completions, yards, touchdowns, and interceptions. Let's make this a little more interesting: Ratings can be calculated over as many attempts as you like and are usually calculated per-game as well as in a career aggregate. So `Passer` should "own" any number of `Game` objects, with details of the game (who played, what date, and so on) as well as the passing statistics for that game.

The model then looks like the diagram presented in Figure 8.1.

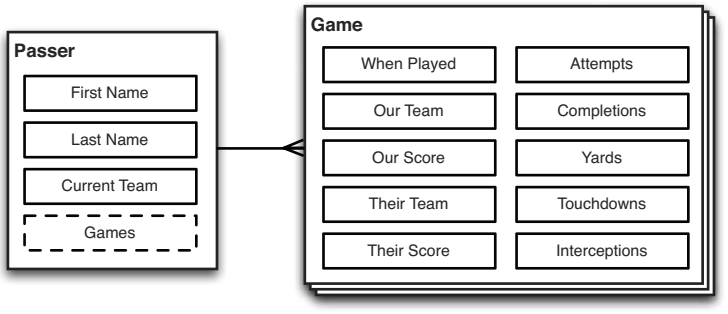

FIGURE 8.1 The summary description of what data a passer-rating app would need leads to the plan shown in this diagram: A `Passer` object serves only to identify a single player; his career statistics are in a set of Game objects that `Passer` "owns."

What about the `Passer`'s aggregate statistics—the career yards, touchdowns, and rating? Those can be pulled out of his `Games`—it turns out not to be hard at all.

The Views

iPhone applications don't usually have a concept of documents, but even simple ones acquire many screens' worth of views. You'll deal in passers and their games, and you need to view and edit both.

So you need a list of passers, who can be created or edited in a separate view; and a view devoted to a selected passer, with a list of games that need a view of their own to create or edit them. A sketch of the flow appears in Figure 8.2.

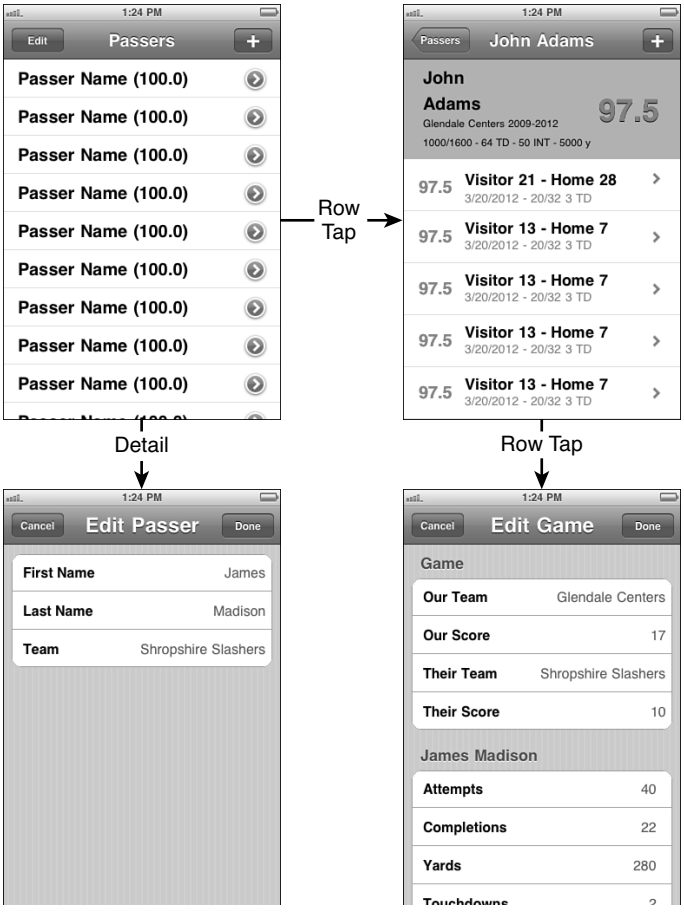

FIGURE 8.2 A rough sketch of the Passer Rating application. The initial view (top left) simply lists all the passers known to the app. Touching the **+** button creates a passer; touching the detail-disclosure button (>) on a passer opens it for editing. Either way, you use the view at the bottom left to edit it. Touching a passer opens his career record and a list of his games (top right). Games, too, can be added (**+**) or edited (touch the game), and the app needs an editor for them.

NOTE

That's what a full version of Passer Rating should look like, and getting it down on paper—I used the excellent Blueprint app for the iPad—is an essential step. Alas, this book will run out of Xcode examples before we complete the app.

Typically, each phase of an iPhone application displays a view object—a `UIView`—that fills the screen. That main view usually contains a hierarchy of other views. For instance, the `Passer` editor at the lower-left corner of the sketch (refer to Figure 8.2) consists of a wrapper `UIView`; it contains a navigation bar (`UINavigationItem`, at the top) which in turn contains two buttons (`UIBarButtonItem`, at either end). It also contains a table (`UITableView`) with three rows (`UITableViewCell`), each containing a label (`UILabel`) and a text-entry field (`UITextField`).

The Controllers

iPhone applications are organized around a sequence of *view controllers*, objects derived from `UIViewController`. Each view controller mediates between model objects and the views that fill the iPhone's screen. For each full-screen view you see in the sketch, you must provide a `UIViewController` subclass to link the data in the model to the views on the screen. In the life cycle of a view, the controller comes first; when it is initialized, it creates or loads the view objects and sets them up to reflect the model.

Now, even a simple application like this slides four main views onto and off of the screen, according to a precise hierarchy. Managing the relationships between them—which to slide in, which to slide back to—would seem to be an involved task—and is. But, thankfully, it is not a task you need to worry much about. UIKit, the user-facing part of iOS, provides umbrella view controllers (such as `UINavigationController`) that manage the navigation tasks for you by taking ownership of your controller objects. All you need to do is request a transition between the views, and the umbrella takes care of the rest.

NOTE

If you can restrict your project to iOS 5.0 or later, Xcode makes this process even easier. See Chapter 15, "Storyboard."

Starting a New iPhone Project

Start by creating a new Xcode project, selecting **File→New→Project…** (⇧⌘**N**). Select Application under iOS, and from the array of application types, select **Master-Detail Application**. Passer Rating follows the common pattern of presenting a progression of lists and detail views, under a navigation bar that provides a "breadcrumb" trail back up the tree. The Master-Detail Application template is a skeleton for such an app. Click **Next**. (For more on Xcode's project templates, see Appendix C, "Project and Target Templates.")

The next panel in the New Project assistant lets you name the project **`Passer Rating`**. That much is obvious. The next item is **Company Identifier**. Every application in the iOS

universe has a string that uniquely identifies it. The app needs one. The way to ensure uniqueness is to select a name that's unique for you (ordinarily the product name) and then to prefix an inverse-dns string. I own the `wt9t.com` domain, so I'd fill in **`com.wt9t`**.

NOTE

You don't have a domain name of your own? Next you'll be telling me you don't have a T-shirt for the project. Get one (a domain name). They're cheap, and you don't have to do anything else with them.

Xcode generates an identifier for you and displays it just under the company ID: `com.wt9t.Passer-Rating`.

Next is **Class Prefix**. This is a string that will be prepended to the class names and associated files that the project template creates for you. If this isn't brief, you'll end up with unwieldy names, so try **`PR`**.

Device Family determines whether the template should include UI setups for iPhone, iPad, or both. Select **iPhone**.

There are four more options. Take two of them:

- **Use Core Data:** Core Data is Cocoa's object-persistence and relational framework, which is handy for keeping the database organized. The project template will add a number of convenient housekeeping methods for getting a Core Data-based application running.
- **Include Unit Tests:** Unit testing, the practice of automatically exercising your code to assure yourself that it works, is an important technique in modern software development. Chapter 13, "Unit Testing," covers unit testing in detail, but if Xcode's going to set it up from the start, you're going to take it up.

The other two, **Use Storyboard** and **Use Automatic Reference Counting**, take advantage of features introduced in iOS 5. Storyboard enables you to design not only the user interface for your app, but also how it transitions from view to view as the user navigates through it (see Chapter 15). Automatic Reference Counting (ARC) takes care of memory management in a transparent and reliable way. (It's available in iOS 4.3 but not fully functional.) Both are great features, but you need to target iOS 4.3.

When I get to a Mac version of Passer Rating in Part III, "Xcode for Mac OS X," I'll talk about ARC. For details on reference-counted memory management, see Appendix A, "Objective-C."

Click **Next**. You get a get-file sheet to select a directory to receive the project. Pick something convenient, such as your `Desktop` directory; that's where Xcode will put the new `Passer Rating` project directory. And because Chapter 7, "Version Control," sold you on it, check **Create local git repository for this project**. Click **Create** and look at the project Xcode has set up for you.

Target Editor

The Passer Rating project consists of two targets: "Passer Rating," which produces the app, and "Passer RatingTests," which will contain the application test suite. Xcode now shows you the Target editor. (Click the Passer Rating project item at the top of the Project navigator to bring it up yourself.) It provides an interface for the basic settings that identify your project—its identifier, target environment, orientations, and the front-end images (icons, startup screen) that are the face UIKit puts on your app.

The **Summary** tab comes in three sections:

- **iOS Application Target** shows the application identifier; Xcode lets you edit the prefix, but you're stuck with `Passer-Rating` until you change the product name. You can set a version number (the release version, without markers for alpha or beta status; those confuse Xcode's packaging process) and a build number (which is where you'd mark prerelease status). And you designate a target (iPhone, iPad, or both) and the "deployment target" (the lowest verson of iOS the app can run on). Xcode obligingly sets this to the latest version Apple sells, but for Passer Rating, set it to **`4.3`**.

  Selecting the software development kit (SDK) is as important as the deployment target, even though the setting is in the **Build Settings** tab and not **Info**. This selects the libraries and headers for the highest version of the OS for which you are developing. If the SDK is set for a later version than the deployment target (it must never be set earlier), the features of the later OS are available, but linked "weakly," so your code can determine at run time whether the features are actually available before trying to use them.

- **iPhone/iPod Deployment Info** enables you to set the app icons and the launch images (the static PNG the OS shows the user while the app starts up). Also, you have your choice of the orientations you support, presented as a toggle button for the ones you can handle. This sets the *most* orientations the app will tolerate; your individual views may be more restrictive. You can ignore the **Main Interface** combo box; it tells the OS the NIB file (I'll get to them soon) that specifies your initial UI. For the master-detail app template, you're set up with code that creates the initial UI, and there's no need for a NIB to specify the details. Leave this (and **Main Storyboard**) blank.

  If you select **iPad** with the **Devices** pop-up, you get an "iPad Deployment Info" section, laid out for iPad resources, instead. And if you select **Universal**, you get both sections.

- **Linked Frameworks and Libraries** is an editable list of objects that Passer Rating links to: UIKit, Foundation, and Core Data. No need to change it.

- **Entitlements** sets up options for compatibility with iCloud and shared access to the cryptographic keychain. You probably don't need those things, and by the time you do, you'll know what these settings mean. I won't be covering entitlements.

The **Summary** tab is just a front end for the more comprehensive settings under the **Info** tab, which in turn is a specialized editor for the `Info.plist` file. Chapter 21, "Bundles and Packages," covers the contents of `Info.plist` exhaustively, but it will be quite some time before you need to change anything.

Copyright, Again

If you browse through the source files, you can see that Xcode has once again used `__MyCompanyName__` as the copyright holder (unless you set a company name in your card in the Address Book application). This has already been covered in the "`__MyCompanyName__`" sidebar in Chapter 7. Again, you have two tasks:

- Make sure Xcode doesn't repeat the mistake in this project. Do this by exposing the Utility area (click the right segment of the **View** control) and its File (first tab) inspector, selecting the project (top line) in the Project navigator, and filling in the **Organization** field.
- Correct the existing files with a projectwide search and replace. This will be your first look at the Search navigator.

    1. Click the third tab (a magnifying glass) in the Navigator area. You see a text field labeled **Find.**
    2. Type **`__MyCompanyName__`** in the field; a menu pops up offering you options for the search, but the defaults (in-project, containing, match case) will do fine. See Figure 8.3, top.
    3. Press Return. The list under the search field fills with references to every file in which `__MyCompanyName__` appears, with subentries showing the found text in context. If you click one, the editor area displays the file, highlighting the match. Refer to Figure 8.3, middle.
    4. The Search navigator's **Find** label is a pop-up menu; switch it to **Replace** to expose the replacement field.
    5. The magnifying glass icon in the search field is a drop-down menu containing your search history. The first item, **Show Find Options**, lets you customize the search: literal or regular-expression; restrictions on partial-or whole-word searches; whether it's case-sensitive; and whether it extends to included frameworks.
    6. Type **`Fritz Anderson`** in the replacement field and click **Replace All**. You have to do the projectwide search first, or the replace buttons won't be enabled—Xcode forces you to look at what you're changing.

Figure 8.3 shows the whole process.

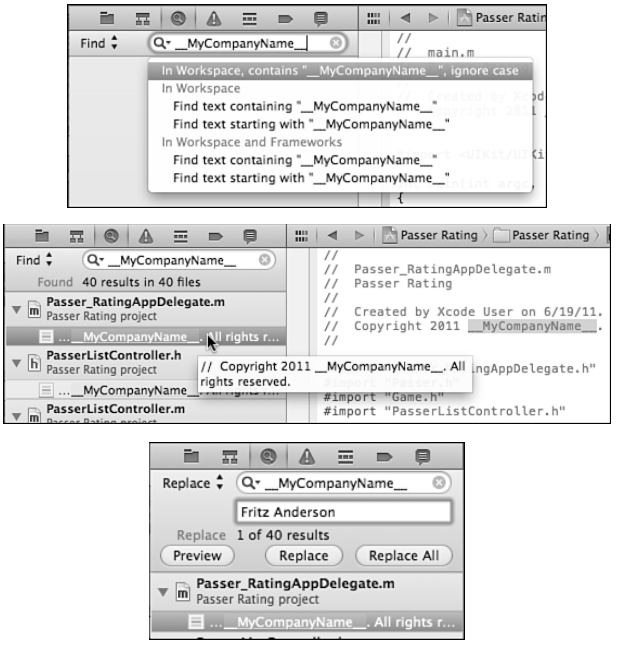

FIGURE 8.3 Expose the Search navigator to start a projectwide search and replace. (Top) Type the search string in the field; Xcode offers some common search options, but simply typing the string and pressing Return usually works. (Middle) When you press Return the navigator fills with every file that contains the string, showing each match in context. Selecting a match puts it into the Editor area. (Bottom) Use the Find/Replace pop-up to expose the replacement field. Type in your replacement string and click **Replace All** to make the change in all files.

At this point, Xcode offers to take a *snapshot* of your project. Snapshots are a supplement to version control; they copy the state of the project into a disk image before you make wholesale changes. Snapshots aren't a replacement for version control or backups—think of them as providing a mass "undo" that doesn't break your workflow as much as a commit would. Look for your snapshot history in the **Projects** panel of the Organizer (**Window→Organizer**, ⇧⌘**2**). From there, you can "Export" the contents of a snapshot to a new directory.

The first snapshot takes a few moments, but after that, they're cheap, so click **Enable**.

At last, you are rewarded with all instances of `__MyCompanyName__` replaced with "Fritz Anderson." If you switch back to the Project navigator, you see that all the changed files' icons are dark, meaning they have unsaved changes. If you save them all (**File→Save All**, ⌥⌘**S**—you must hold the Option key down to see it in the menu), the "M" version-control badge will appear next to all of them. Now would be a good time to commit a revision to the repository.

The project template you chose for Passer Rating includes a lot:

- **Frameworks:** The Frameworks group contains UIKit, Foundation, CoreData, and SenTestingKit (which is used only by the unit-test target). They provide links into the iOS system software, and Passer Rating won't run without them. If you open the **Build Phases** tab of the Project editor, for the Passer Rating target, you'll find the frameworks in the Link Binary with Libraries phase.
- **Class `PRAppDelegate`:** The application is represented by an object of class `UIApplication`, which you should never need to subclass or replace. True to the delegation pattern used throughout Cocoa, all the unique behavior of the application comes through the methods of a delegate object, which the application object calls into. `PRAppDelegate` is declared as a subclass of `NSObject` and an implementor of the `UIApplicationDelegate` protocol. The template for the implementation (`.m`) file contains a good starter for managing the application life cycle, including setting up the Core Data database.
- **Class `PRMasterViewController`:** This is, as the name says, the controller for the bottom-level view of Passer Rating, which the design says is a table of passer names and ratings. Because navigation-based applications almost always start with a table, the template makes `PRMasterViewController` a subclass of `UITableViewController`, which is suited for running a table view. The implementation file includes skeletons of the methods you need to fill the table in. It also provides an instance of `NSFetchedResultsController`, which does a lot to help link tables to Core Data data stores.
- **Class `PRDetailViewController`:** The controller for the next layer of Passer Rating, the one that is seen when the user taps a passer's name. The template can't be sure what you'll be doing with `PRDetailViewController`, so it declares it to be an instance of the more-generic `UIViewController`.
- **`Passer_Rating.xcdatamodeld`:** Core Data isn't a full-service relational database (although it uses the SQLite database library internally), but if it were, the Data Model file would be the equivalent of an SQL schema. It defines the entities (think "tables") that hold the data, and the attributes (think "columns") those entities have. Xcode provides a graphical editor for data models.
- In the Supporting Files group, **`Passer_Rating-Info.plist`:** This is the source file that yields an `Info.plist` file to be embedded in the application. It provides basic information on what the application can do, what data it can handle, and how it is presented to the user in the Home screen. Some of that information is presented to the user as text, so its content is merged with the application's `InfoPlist.strings` for the user's language. (Chapter 20, "Localization and Autolayout," covers localization in Mac OS X, but most of the concepts apply to iOS, as well.)
- **Miscellaneous source files:** `main.m` is the standard container for the `main()` function where the program starts; you usually won't change it. `Passer Rating-Prefix.pch` contains common initialization for the compiler.

- **XIB files:** XIB files are editable files that will be compiled into NIB files and installed in the application. They are archives, mainly of user-interface objects. `PRMasterViewController.xib` contains the `UITableView` for the root view and links it to the `PRMasterViewController` that runs the table. `PRDetailViewController.xib` is a more generic layout that does the same for `PRDetailViewController`.
- The "Passer RatingTests" target has a class file and `Info.plist` support similar to the app target's.

Xcode's template for the project also includes a panoply of build settings, specifying how Passer Rating is to be compiled, linked, and organized.

The project is fully functional, as far as it goes—run it: **Product→Run**(⌘**R**). Xcode builds the app, and in a few seconds, the iOS Simulator starts and launches Passer Rating. Out-of-the-box, the app is the iOS/Core Data equivalent to "Hello World." It shows an empty table under a navigation bar with **Edit** and + buttons. Tapping the + button adds a row with the current date and time; tapping the new entry pushes the "detail" view into view; the **Edit** button in the root list (or swiping across a row) lets you delete rows. You can close and reopen the app to find that the rows you added are still there. See Figure 8.4.

FIGURE 8.4 The skeletal code that comes with the Core Data + Master-Detail Application project template is enough to produce an app that can run in the iOS Simulator. It can add rows to its table, and, as shown here, respond to the **Edit** button by offering to delete rows.

NOTE

As to whether Passer Rating is actually saving those dates and times, make sure there's nothing up your sleeve. Closing an iOS app usually doesn't stop it: The OS just puts it to sleep in the background, and its data won't be disturbed. To see a real, fresh restart, go to Xcode, click the **Stop** button in the project toolbar, and then click **Run**.

One More Thing

Passer Rating is supposed to calculate passer ratings, which is a problem you already solved. Let's add your solution to the project. Select **File →Add Files to "Passer Rating"…** (⌥⌘A). Xcode opens a modified get-file sheet (Figure 8.5). Track down `rating.h` and `rating.c` from the old passer-rating project and command-click them to select both. Don't click **Add** yet because there's more to do.

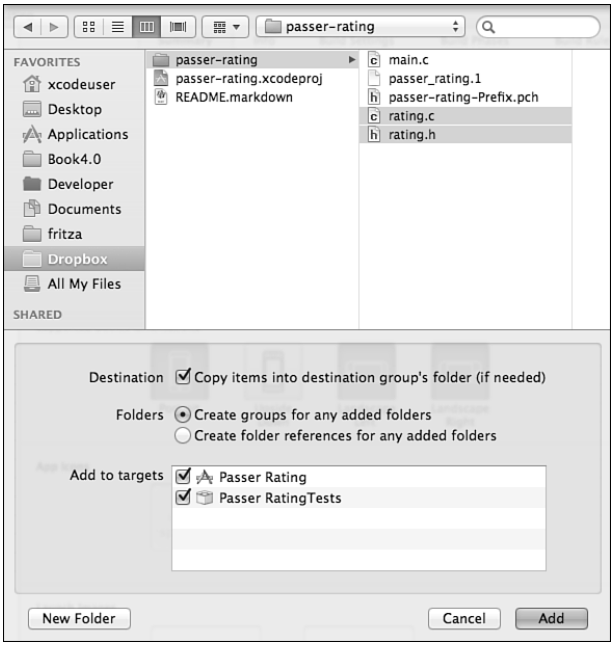

FIGURE 8.5 When you add files to a project, you have some options controlling whether to use the files in-place and how to use them.

Check the box labeled **Copy items into destination group's folder (if needed)**. That adds copies of the files to Passer Rating's project directory. It sometimes makes sense not to do this, as when you want to share a common copy of files across projects, but for now, this is the least complicated way to go. You'll see a better way to do the same thing later.

Under **Folders**, you can decide how Xcode treats any directories you add. It can represent them in the Project navigator by group folders, as an organizational aid, or it can have the project refer to the directories themselves as the objects the project contains. The latter makes sense if you want to include a whole directory of data files in your product. You're not adding any directories, so this isn't an issue, but it's a good idea to make sure the setting is **Create groups for any added folders** because it's usually the safer action to take.

Now you can click **Add**.

Summary

In this chapter, you began work on an iOS application. Before doing anything in Xcode, we decided what the app would do and what it would look like.

When that was done, you knew enough to have Xcode create the project. You explored the Project editor and saw how it managed the configuration of the application. Then you explored the files Xcode's template provided and used the Search navigator to track down and simultaneously repair a bad copyright notice in seven files.

The "empty" application Xcode provided was runnable; you found that running it launched the iOS Simulator, which allowed you to use the app.

Finally, you prepared the app for your own work by adding the `passer_rating` function from the `passer-rating` project.

Now you're ready to make the empty app your own to start implementing the design you put together in this chapter. We'll start with the model.

Index

Symbols

A

B

C

D

E

F

G

H

I

J–K

L

M

N

O

P

Q

R

S

T

U

V

W

X–Z