

Jesse Feiler

**SECOND
EDITION**

**Updated for
Xcode 4.3**

Sams **Teach Yourself**

Core Data for **Mac[®] and iOS**

in **24**
Hours

SAMS



Jesse Feiler

Sams **Teach Yourself**

Core Data for Mac[®] and iOS

in **24**
Hours

Second Edition

SAMS

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

Sams Teach Yourself Core Data for Mac® and iOS in 24 Hours, Second Edition

Copyright © 2012 by Pearson Education, Inc.

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-0-672-33619-5

ISBN-10: 0-672-33619-7

Library of Congress Cataloging-in-Publication data is on file.

Printed in the United States of America

First Printing: June 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.

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@pearsoned.com

Editor-in-Chief

Greg Wiegand

Executive Editor

Loretta Yates

Development Editor

Sandra Scott

Managing Editor

Sandra Schroeder

Project Editor

Mandie Frank

Indexer

Brad Herriman

Proofreader

Megan Wade

Technical Editor

Robert McGovern

Publishing Coordinator

Cindy Teeters

Designer

Gary Adair

Compositor

Mark Shirar

Contents at a Glance

Introduction	1
Part I: Getting Started with Core Data	
HOURL 1: Introducing Xcode 4	7
2: Creating a Simple App	49
3: Understanding the Basic Code Structure	63
Part II: Using Core Data	
HOURL 4: Getting the Big Core Data Picture	85
5: Working with Data Models	101
6: Working with the Core Data Model Editor	117
7: What Managed Objects Can Do	133
8: Controllers: Integrating the Data Model with Your Code	143
9: Fetching Data	153
10: Working with Predicates and Sorting	171
Part III: Developing the Core Data Interface	
HOURL 11: Finding Your Way Around the Interface Builder Editor: The Graphics Story	189
12: Finding Your Way Around the Interface Builder Editor: The Code Story	209
13: Control-Dragging Your Way to Code	223
14: Working with Storyboards and Swapping Views	239
Part IV: Building the Core Data Code	
HOURL 15: Saving Data with a Navigation Interface	257
16: Using Split Views on iPad	279
17: Structuring Apps for Core Data, Documents, and Shoeboxes	289
18: Validating Data	317

Part V: Managing Data and Interfaces

HOURL 19: Using UITableView on iOS	337
20: Using NSTableView on Mac OS	363
21: Rearranging Table Rows on iOS	375
22: Managing Validation	393
23: Interacting with Users	409
24: Migrating Data Models	423

Appendix

A What's Old in Core Data, Cocoa, Xcode, and Objective-C	441
Index	443

Table of Contents

Introduction	1
Who Should Read This Book	1
Some Points to Keep in Mind	2
How This Book Is Organized	3
 Part I: Getting Started with Core Data	
 HOURL 1: Introducing Xcode 4	7
Getting to Know Xcode	8
Goodbye “Hello, World”	8
Hello, App Development for Mac OS X and iOS	11
Getting Started with Xcode	13
Using the Navigator	15
Using Editors	25
Working with Assistant	29
Getting Help in an Editor Window	31
Using Utilities—Inspectors	31
Using Utilities—Libraries	35
Using the Text Editor	40
Using the Organizer Window	45
Summary	47
Workshop	48
Activities	48
 HOURL 2: Creating a Simple App	49
Starting to Build an App	49
Building the Project	52
Exploring the App	58
Summary	60
Workshop	60
Activities	61

HOURL 3: Understanding the Basic Code Structure	63
Working with the Code	63
Looking at Object-Oriented Programming in the Context of Objective-C	66
Using Declared Properties	68
Messaging in Objective-C	73
Using Protocols and Delegates	75
Using the Model/View/Controller Concepts	81
Importing and Using Declarations in Files	82
Summary	83
Workshop	84
Activities	84

Part II: Using Core Data

HOURL 4: Getting the Big Core Data Picture	85
Starting Out with Core Data	85
Examining Core Data at Runtime: The Core Data Stack	90
Working with Fetched Results	96
Summary	99
Workshop	99
Activities	99
HOURL 5: Working with Data Models	101
Making the Abstract Concrete	101
Working with Entities	103
Adding Attributes to Entities	105
Linking Entities with Relationships	107
Keeping Track of Your Data in Files and Documents	108
Summary	116
Workshop	116
Activities	116

HOURL 6: Working with the Core Data Model Editor	117
Moving the Data Model from Paper to Xcode and the Core Data Model Editor	117
Adding Entities to the Data Model	119
Choosing the Editor Style	125
Adding Relationships to a Data Model	126
Summary	132
Workshop	132
Activities	132
HOURL 7: What Managed Objects Can Do	133
Using Managed Objects	133
Deciding Whether to Override <code>NSManagedObject</code>	134
Overriding <code>NSManagedObject</code>	136
Implementing Transformation in an <code>NSManagedObject</code> Subclass	140
Summary	142
Workshop	142
Activities	142
HOURL 8: Controllers: Integrating the Data Model with Your Code	143
Looking Inside Model/View/Controller	143
Integrating Views and Data on Mac OS	147
Integrating Views and Data on iOS	151
Summary	152
Workshop	152
Activities	152
HOURL 9: Fetching Data	153
Choosing the Core Data Architecture	153
Exploring the Core Data Fetching Process	154
Using Managed Object Contexts	158
Creating and Using a Fetch Request	159
Stopping the Action to Add New Data	161
Optimizing Interfaces for Core Data	162

Summary	168
Workshop	168
Activities	169
HOURL 10: Working with Predicates and Sorting	171
Understanding Predicates	171
Constructing Predicates	177
Creating a Fetch Request and Predicate with Xcode	178
Sorting Data	185
Summary	187
Workshop	187
Activities	187
Part III: Developing the Core Data Interface	
HOURL 11: Finding Your Way Around the Interface Builder Editor:	
The Graphics Story	189
Starting to Work with the Interface Builder Editor in Xcode	189
Working with the Canvas	197
Summary	206
Workshop	206
Activities	207
HOURL 12: Finding Your Way Around the Interface Builder Editor:	
The Code Story	209
Using the Connections Inspector	209
Using IBOutletlets for Data Elements	215
Summary	222
Workshop	222
Activities	222
HOURL 13: Control-Dragging Your Way to Code	223
Repurposing the Master-Detail Application Template	223
Adding New Fields as IBOutletlets	230
Summary	237

Workshop	237
Activities	238
HOURL 14: Working with Storyboards and Swapping Views	239
Creating a Project with a Storyboard	239
Swapping Views on iOS Devices	241
Swapping Detail Views (the Old Way)	244
Understanding the Storyboard Concept	246
Looking at the Estimator Storyboard and Code	248
Creating a Storyboard	251
Summary	254
Workshop	255
Activities	255
 Part IV: Building the Core Data Code	
 HOURL 15: Saving Data with a Navigation Interface	257
Using a Navigation Interface to Edit and Save Data	257
Starting from the Master-Detail Template	263
Using the Debugger to Watch the Action	267
Adding a Managed Object	272
Moving and Saving Data	273
Cleaning Up the Interface	275
Summary	277
Workshop	278
Activities	278
 HOURL 16: Using Split Views on iPad	279
Moving to the iPad	279
Implementing the Second Interface	281
Changing the Data Update and Saving Code	284
Summary	287
Workshop	287
Activities	288

HOURL 17: Structuring Apps for Core Data, Documents, and Shoeboxes	289
Looking at Apps from the Core Data Point of View:	
The Role of Documents	289
Exploring App Structure for Documents, Mac OS, and iOS	292
Moving Data Models	311
Moving a Data Model from One Project to Another	312
Summary	315
Workshop	316
Activities	316
HOURL 18: Validating Data	317
Using Validation Rules in the Data Model	317
Setting Up Rules in Your Data Model	320
Entering Data into the Interface and Moving It to the Data Model (and Vice Versa)	327
Creating Subclasses of <code>NSObject</code> for Your Entities	331
Summary	335
Workshop	336
Activities	336
 Part V: Managing Data and Interfaces	
 HOURL 19: Using UITableView on iOS	337
Working with Table Views and iOS, Mac OS, and Core Data	337
Comparing Interfaces: Settings on iOS and System Preferences on Mac OS	339
Using UITableView Without Core Data	344
Using UITableView with Core Data	357
Summary	360
Workshop	361
Activities	361
 HOURL 20: Using NSTableView on Mac OS	363
Exploring the New NSTableView Features	363
Building an NSTableView App	366

Summary	373
Workshop	374
Activities	374
HOURL 21: Rearranging Table Rows on iOS	375
Handling the Ordering of Table Rows	375
Allowing a Table Row to Be Moved	380
Doing the Move	382
Summary	391
Workshop	392
Activities	392
HOURL 22: Managing Validation	393
Validation for Free	393
Validation on Mac OS	394
Programming Validation for iOS or Mac OS	402
Summary	407
Workshop	407
Activities	408
HOURL 23: Interacting with Users	409
Choosing an Editing Interface	409
Communicating with Users	413
Using Sheets and Modal Windows on Mac OS	419
Summary	422
Workshop	422
Activities	422
HOURL 24: Migrating Data Models	423
Introducing the Core Data Migration Continuum	423
Managing Data Model Migration	424
Working with Data Model Versions	426
Using Automatic Lightweight Migration	432
Looking at a Mapping Model Overview	434

Summary	438
Workshop	438
Activities	439
APPENDIX A: What's Old in Core Data, Cocoa, Xcode, and Objective-C	441
Declared Properties	441
Required and Optional Methods in Protocols	442
Storyboards in Interface Builder	442
Ordered Relationships	442
Index	443

About the Author

Jesse Feiler is a developer, web designer, trainer, and author. He has been an Apple developer since 1985 and has worked with mobile devices starting with Apple's Newton and continuing with the iOS products such as the iPhone, iPod touch, and iPad. Feiler's database expertise includes mainframe databases such as DMS II (on Burroughs), DB2 (on IBM), and Oracle (on various platforms), as well as personal computer databases from dBase to the first versions of FileMaker. His database clients have included Federal Reserve Bank of New York; Young & Rubicam (advertising); and many small and nonprofit organizations, primarily in publishing, production, and management.

Feiler's books include the following:

- ▶ *Sams Teach Yourself Objective-C in 24 Hours* (Sams/Pearson)
- ▶ *Data-Driven iOS Apps for iPad and iPhone with FileMaker Pro, Bento by FileMaker, and FileMaker Go* (Sams/Pearson)
- ▶ *FileMaker 12 in Depth* (Sams/Pearson)
- ▶ *Using FileMaker Bento* (Sams/Pearson)
- ▶ *iWork for Dummies* (Wiley)
- ▶ *Sams Teach Yourself Drupal in 24 Hours* (Sams/Pearson)
- ▶ *Get Rich with Apps! Your Guide to Reaching More Customers and Making Money NOW* (McGraw-Hill)
- ▶ *Database-Driven Web Sites* (Harcourt)
- ▶ *How to Do Everything with Web 2.0 Mashups* (McGraw-Hill)
- ▶ *The Bento Book* (Sams/Pearson)

He is the author of MinutesMachine, the meeting management software for iPad—get more details at champlainarts.com.

A native of Washington, D.C., Feiler has lived in New York City and currently lives in Plattsburgh, NY. He can be reached at northcountryconsulting.com.

Acknowledgments

Thanks go most of all to the people at Apple, along with the developers and users who have helped to build the platform and imagine possibilities together to make the world better.

At Pearson, Loretta Yates, Executive Editor, has taken a concept and moved it from an idea through the adventures along the way to printed books and eBooks in a variety of formats. She is always a pleasure to work with.

Mandie Frank, Project Editor, has done a terrific job of keeping things on track with a complex book full of code snippets, figures, and cross references in addition to the text. Technical Editor Robert McGovern caught numerous technical typos and added comments and perspectives that have clarified and enhanced the book.

As always, Carole Jelen at Waterside Productions has provided help and guidance in bringing this book to fruition.

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.

As an Editor-in-Chief for Sams Publishing, I welcome your comments. 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 better.

Please note that I cannot help you with technical problems related to the topic of this book. We do have a User Services group, however, where I will forward specific technical questions related to the book.

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

Email: feedback@amspublishing.com

Mail: Greg Wiegand
Editor-in-Chief
Sams Publishing
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

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

This page intentionally left blank

Introduction

Organizing things is an important human activity. Whether it is a child organizing toys in some way (by size, color, favorites, and so forth) or an adult piecing together a thousand-piece jigsaw puzzle, the desire to “make order out of chaos” (as one inveterate puzzler put it) reflects a sense that somehow if we try hard enough or just have enough information, we can find or create an understandable view of the world. Or at least an understandable view of the left overs in the refrigerator or the photos in an album.

Core Data is a powerful tool that you can use with the Cocoa and Cocoa Touch frameworks on iOS and Mac OS to help you make order out of the chaos of the hundreds, thousands, and even billions of data elements that you now can store on your computer or mobile device.

Who Should Read This Book

This book is geared toward developers who need to understand Core Data and its capabilities. It’s also aimed at developers who aren’t certain they need the combination of Core Data and Cocoa. It places the technologies in perspective so that you can see where you and your project fit in. Part of that is simply analytical, but for everyone, the hands-on examples provide background as well as the beginnings of applications (apps) that you can create with these two technologies.

If you are new to databases or SQL, you will find a basic introduction here. If you are familiar with them, you will find a refresher as well as details on how the concepts you know already map to Core Data terminology.

Likewise, if you are new to development on Mac OS, iOS, or Cocoa and Cocoa Touch, you will find a fairly detailed introduction. If you are already familiar with them, you will see how some of the basic concepts have been expanded and rearranged to work with Core Data.

There is a theme that recurs in this book: links and connections between interface and code as well the connections between your app and the database. Much of what you find in this book helps you develop the separate components (interface, database, and code) and find simple ways to link them.

Some Points to Keep in Mind

Not everyone starts from the same place in learning about Core Data (or, indeed, any technology). Learning and developing with new technologies is rarely a linear process. It is important to remember that you are not the first person to try to learn these fairly complex interlocking technologies. This book and the code that you experiment with try to lead you toward the moment when it all clicks together. If you do not understand something the first time through, give it a rest, and come back to it another time. For some people, alternating between the graphical design of the interface, the logical design of the code processes, and the organizational structure of the database can actually make things seem to move faster.

Here are some additional points to consider.

Acronyms

In many books, it is a convention to provide the full name of an acronym on its first use—for example, HyperText Markup Language (HTML). It is time to recognize that with wikipedia.org, dictionaries built into ebooks and computers, and so many other tools, it is now safe to bring a number of acronyms in from the cold and use them without elaboration. Acronyms specific to the topic of this book are, indeed, explained on their first use in any chapter.

There is one term that does merit its own little section. In this book, as in much usage today, SQL is treated as a name and not as an acronym. If you look it up on Wikipedia, you will see the evolution of the term and its pronunciation.

Development Platforms

It is not surprising that the development of Mac OS X apps takes place on the Mac itself. What may surprise some people, though, is that iOS apps that can run on iPad, iPod touch, and iPhone must be developed on the Mac. There are many reasons for this, not the least of which is that the development tool, Xcode, takes advantage of many dynamic features of Objective-C that are not available on other platforms. Also, Xcode has always served as a test bed for new ideas about development, coding, and interfaces for the Apple engineers. Registered Apple developers have access to preview versions of the developer tools. As a result, the Apple developers had access to features of Lion such as full-screen apps nine months before the general public. In fact, Xcode 4 is optimized for Lion in both speed and interface design.

Assumptions

Certain things are assumed in this book. (You might want to refer to this section as you read.) They are as follows:

- ▶ *Cocoa*, as used in this book, refers to the Cocoa framework on Mac OS and, unless otherwise specified, also to the Cocoa Touch framework on iOS.
- ▶ *iPhone* refers to iPhone and iPod touch unless otherwise noted.

Formatting

In addition to the text of this book, you will find code samples illustrating various points. When a word is used in a sentence as computer code (such as `NSTableView`), it appears like this. Code snippets appear set off from the surrounding text. Sometimes they appear as a few lines of code; longer excerpts are identified with listing numbers so they can be cross-referenced.

Downloading the Sample Files

Sample files can be downloaded from the author's website at northcountryconsulting.com or from the publisher's site at www.informit.com/9780672335778.

How This Book Is Organized

There are five parts to this book. You can focus on whichever one addresses an immediate problem, or you can get a good overview by reading the book straight through. Like all of the *Teach Yourself* books, as much as possible, each chapter (or hour) is made to stand on its own so that you can jump around to learn in your own way. Cross-references throughout the book help you find related material.

Part I, “Getting Started with Core Data”

This part introduces the basic issues of the book and shows you principles and techniques that apply to all of the products discussed:

- ▶ Chapter 1, “Introducing Xcode 4”—Xcode is the tool you use to build Mac OS and iOS apps. It includes graphical editors for designing your interface and data model. The current version, Xcode 4, represents a significant step forward from previous development environments. You'll get started by learning the ins and outs of Xcode 4. After you use it, you'll never look back.

- ▶ Chapter 2, “Creating a Simple App”—This hour walks you through the process of creating an app from one of the built-in Xcode templates. It’s very little work for a basic app that runs.
- ▶ Chapter 3, “Understanding the Basic Code Structure”—This hour introduces design patterns used in Objective-C as well as some of the features (such as delegates and protocols) that distinguish it from other object-oriented programming languages.

Part II, “Using Core Data”

Here you will find the basics of Core Data and its development tools in Xcode:

- ▶ Chapter 4, “Getting the Big Core Data Picture”—Here you’ll find an overview of Core Data and a high-level introduction to its main components.
- ▶ Chapter 5, “Working with Data Models”—Data models have been around since the beginning of databases (and, in fact, since long before, if you want to include data models such as the classifications of plants and animals). This hour lets you learn the language of Core Data.
- ▶ Chapter 6, “Working with the Core Data Model Editor”—In this hour, you will learn how to build your data model graphically with Xcode’s table and grid styles.
- ▶ Chapter 7, “What Managed Objects Can Do”—In this hour, you’ll discover the functionality of managed objects and what you can do to take advantage of it and to expand it.
- ▶ Chapter 8, “Controllers: Integrating the Data Model with Your Code”—The key point of this book is to show you how to link your database and data model to interface elements and your code. This hour provides the basics for Mac OS and for Cocoa.
- ▶ Chapter 9, “Fetching Data”—Just as the SQL `SELECT` statement is the heart of data retrieval for SQL databases, fetching data is the heart of data retrieval for Core Data. Here you’ll learn the techniques and terminology.
- ▶ Chapter 10, “Working with Predicates and Sorting”—When you fetch data, you often need to specify exactly what data is to be fetched—that is the role of predicates. In addition, you will see how to build in sorting to your fetch requests so that the data is already in the order you need.

Part III, “Developing the Core Data Interface”

Now that you understand the basics of Core Data, you can use it to drive the commands, controls, and interfaces of your apps:

- ▶ Chapter 11, “Finding Your Way Around Interface Builder: The Graphics Story”—The Interface Builder editor in Xcode 4 (a separate program until now) provides powerful tools and a compact workspace to help you develop your interface and app functionality.
- ▶ Chapter 12, “Finding Your Way Around Interface Builder: The Code Story”—This hour shows you the graphical tools to link the code to the interface.
- ▶ Chapter 13, “Control-Dragging Your Way to Code”—A special aspect of linking your interface to your code is using the tools in Xcode 4 to actually write the interface code for you.
- ▶ Chapter 14, “Working with Storyboards and Swapping Views”—One of the major advances in Xcode 4, storyboards not only create and manage the views and controllers that make up your interface, but also let you manage the sequences in which they are presented (segues). You will find that storyboards can replace a good deal of code that you would otherwise have to write for each view you display.

Part IV, “Building the Core Data Code”

Yet another aspect of the connections between Core Data, your code, and your interface consists of the data source protocol and table views. This part explains them:

- ▶ Chapter 15, “Saving Data with a Navigation Interface”—Originally designed for iPhone, navigation interfaces are an efficient use of screen space for organized data. This hour shows you how to use them.
- ▶ Chapter 16, “Using Split Views on iPad”—Split views on iPad provide a larger-screen approach to data presentation than navigation interfaces. As you see in this hour, you can combine navigation interfaces with a split view on iPad. Data sources provide your Core Data data to the table view. This hour shows how that happens and moves on to how you can work with tables and their rows and sections. You’ll also see how to format cells in various ways.
- ▶ Chapter 17, “Structuring Apps for Core Data, Documents, and Shoeboxes”—This hour goes into detail about how and where your data can actually be stored.
- ▶ Chapter 18, “Validating Data”—When you use Xcode and Core Data to specify what data is valid, you do not have to perform the validation yourself. This hour shows you how to set up the rules

Part V, “Managing Data and Interfaces”

- ▶ Chapter 19, “Using UITableView on iOS”—Table views let you manage and present data easily. The UITableView structure on iOS is designed for seamless integration with Core Data.
- ▶ Chapter 20, “Using NSTableView on Mac OS”—NSTableView on Mac OS is revised in Lion. The older versions of table views still work, but as you see in this hour, some of the new features of UITableView have been backported to Mac OS.
- ▶ Chapter 21, “Rearranging Table Rows on iOS”—The ability to rearrange table rows by dragging them on the screen is one of the best features of iOS. It is remarkably simple once you know the table view basics.
- ▶ Chapter 22, “Managing Validation”—This hour shows you how to build on the validation rules from Hour 18 to actually implement them and let users know when there are problems.
- ▶ Chapter 23, “Interacting with Users”—On both iOS and Mac OS, it is important to let users know when they are able to modify data and when it is only being displayed.
- ▶ Chapter 24, “Migrating Data Models”—You can have Core Data automatically migrate your data model to a new version. This hour shows you how to do that, as well as how to use model metadata and alternative types of data stores.

Appendixes

- ▶ Appendix A, “What’s Old in Core Data, Cocoa, Xcode, and Objective-C”—There are some legacy features in the sample code you’ll find on developer.apple.com and in apps you might be working with. This appendix helps you understand what you’re looking at and how to modernize it.

NOTE

Due to the complexity of the topics discussed, some figures in this book are very detailed and are intended only to provide a high-level view of concepts. Those figures are representational and not intended to be read in detail. If you prefer to view these figures on your computer, you can download them at informit.com/title/9780672336195.

HOOR 1

Introducing Xcode 4

What You'll Learn in This Hour:

- ▶ Understanding the new development paradigms
- ▶ Exploring the Xcode workspace window
- ▶ Defining projects and workspaces
- ▶ Debugging with breakpoints
- ▶ Caring for your source code with repositories and versions

The Origins of Xcode 4

Xcode 4 has its roots in Project Builder and Interface Builder, the two development tools created for NeXTSTEP. The NeXTSTEP operating system ran on the NeXT computer, which was manufactured by NeXT, the company Steve Jobs founded when he left Apple in 1985. The hardware side of the business was not successful, and NeXTSTEP morphed into OPENSTEP, which ran on Sun's Solaris operating system, and later on Windows. After Apple purchased NeXT in 1996, the software became Rhapsody and, later, Mac OS X. A branch of the software became the iPhone operating system which, after the introduction of iPad, became iOS.

Project Builder and Interface Builder remained the developer tools through all this time. Project Builder was the tool you used to write code, and Interface Builder was the graphically oriented tool you used to draw the interface. Project Builder was renamed Xcode in 2003; it contained significant changes to its user interface at that time.

At Apple's 2010 Worldwide Developer Conference, Xcode 4 was given its debut. It was released as the official version in spring 2011. One of its most significant features was the integration of Project Builder and Interface Builder in a single tool.

This book is based on Xcode 4. If you are using an earlier version, it is time for you to update to the latest software because by the time this book is published, Xcode 4 will be more than a year old (depending on whether you start counting from the demonstrations or from the official release). Now that you know the history and origins of Xcode 4, there is no reason to distinguish it from its predecessors: From this point on, it is simply referred to as *Xcode*.

Getting to Know Xcode

Everything you do in the development of Mac and iOS apps is done in the context of Xcode. First demonstrated at Apple's Worldwide Developers Conference in June 2010, it was released in several preview versions until the final release in the spring of 2011. Xcode 4 is not just a revision to the interface of Xcode 3; it is a rethinking of the way in which developers work on their apps.

This hour helps you understand this new way of working and why it is so relevant to apps written for Mac and iOS in today's world. Not only will you find out how to use Xcode 4, but you will see why it is structured the way it is and how you can best take advantage of its new features.

As you use Xcode 4, try to use the new features and new ways of working so that you understand what the people at Apple think a productive development process can look like today. And bear in mind one important point about Apple's developer tools: for many years, these tools have been testing and proving grounds for new ideas about interface design. What you see in Xcode 4 includes some novel approaches to interface design that you may consider using for your own apps both on Mac and iOS.

- One of the most important features of Xcode is its simulator: software that lets you test iOS apps on your Mac. You'll find out more about the simulator in Part II of this book, "Using Core Data."

Goodbye "Hello, World"

For many people, their first program was something along the lines of the well-known Hello World program shown in Listing 1.1. It is from the classic *The C Programming Language* by Brian Kernighan and Dennis Ritchie (1978).

LISTING 1.1 Hello, World

```
main( ) {  
    printf("hello, world");  
}
```

Many people still think of this as a model of what programming is all about: You start with a blank piece of paper or a blank screen, you type in your code, you run it, you make revisions, and you continue on to the next program on its own blank piece of paper or blank screen.

Today's programming is based on several commonly used paradigms. Two of the most important have to do with how programs function—declarative and imperative paradigms. A third, object-oriented programming, has to do with the structure of programs.

Working with Imperative and Declarative Programming Paradigms

Today's apps are much more complex than just printing or displaying a line of text. How do you get from Hello, World to an app such as iTunes? Even an app that appears to be text-based such as Pages in the iWork suite is a far cry from Hello, World. And when you consider that Mac OS X and iOS are basically just very large apps, it is hard to see how they evolved from Hello, World.

When Hello, World first was written, the programming world was already moving away from this linear do this/do that paradigm (called *imperative* or *procedural* programming) to a new paradigm called *declarative* programming, in which the mechanics of *how* something is done are less important than *what* is done.

Procedural programming is used in the code you write; most of that is Objective-C when you are writing for Mac OS X and iOS. For most people, writing procedural code “feels” like programming. (In addition to its procedural programming concepts, Objective-C uses object-oriented programming, hence its name.)

Languages that are declarative (that is, focusing on what is done) are particularly common on the Web. Most people consider Cascading Style Sheets (CSS), regular expressions, and the basics of SQL (SELECT statements, for example) to be examples of declarative languages. Markup languages in general—including HTML itself—are declarative rather than procedural because they describe what the end result should look like. For many people, designing databases and web pages doesn't “feel” like programming (and many people do not think that it is).

The distinction between these two programming paradigms is not a matter of good versus bad or old versus new: It is simply a contrast between two ways of developing software. As you approach Xcode, Mac OS X, and iOS, you do not have to make a choice because both paradigms are supported in Xcode. Most of the time, a specific editing function is implemented only in procedural or declarative styles because one or the other is the natural way of editing that particular set of instructions.

NOTE

In at least one case—the creation of interface views—you can choose between procedural and declarative styles. In those cases, this book will point out some of the differences that affect your finished app.

If you are starting building apps for Mac OS X or iOS that use Core Data, you will use descriptive editors for the Core Data side of things just as you do with many SQL-based development environments, and you will use procedural editors for the text-based code that you write to manipulate the interface and the database.

Working with Object-Oriented Programming

Object-oriented programming is now so pervasive that for many people, it is the only kind of programming they do. Instead of the simple and relatively unstructured code shown in Listing 1.1, *objects* are created that encapsulate data and functionality. These objects interact with one another to get the work of the program done.

When people first started using object-oriented programming techniques, some critics pointed out that it took much more code and programming time to use object-oriented techniques and languages than to use traditional techniques and languages. The idea of writing a program with the three lines shown in Listing 1.1 is unthinkable in the object-oriented programming world.

However, the arguments made by proponents of object-oriented programming and borne out by decades of experience are that

- ▶ Object-oriented programming is easier to maintain and modify over time in part because of its inherent structures.
- ▶ It might take many more lines to write a very simple program using object-oriented programming techniques, but as the complexity of the program increases, the incremental effort to build each new feature can be significantly less than with traditional techniques.

When you put these points together, you can see that there is a significant difference between simple and complex programs no matter whether you are using object-oriented programming or traditional programming. The benefits of object-oriented programming really only appear in complex programs, whereas the limitations of traditional programming methods do not appear in short programs.

In practical terms, this means that to learn how to use the tools of Mac OS X and iOS along with Xcode, you have to work with hefty examples. And if you try to use a

simplified example, you might wind up thinking that these tools are overly complex. That is true in one sense: Using these tools to write something very simple is overkill. But not using tools like this to write complex software is frequently self-defeating.

As you begin to work with Xcode, Core Data, Mac OS X, and iOS, you will find yourself at the helm of a sophisticated and powerful development environment. In this book, you will see how to start small and build up to very complex apps. In the initial hours, because the examples are small, you may be tempted to worry about the complexity, but just remember that the complexity will pay off as the examples become more complex.

- With that overview, you might be interested in the Tutorial “Using Xcode to Write ‘Hello, World’” in Hour 1 of *Apple’s Xcode Quick Start Guide*. It is 20 pages long and demonstrates precisely these points.

TIP

If you have not done so already, register as a developer with Apple at developer.apple.com. A variety of developer programs are available, but the most common are the Mac OS X developer program (\$99/year), the iOS developer program (\$99/year), and the Safari developer program (free). All these programs are built on your registration as a developer with Apple, which is free.

Without even registering, you have access to libraries of documentation. All Apple documentation referred to in this book is available through developer.apple.com. Any documentation that is not available through developer.apple.com will be identified.

You can visit <http://developer.apple.com/programs/which-program/> to compare the various developer programs and to choose the one that makes sense for you.

Hello, App Development for Mac OS X and iOS

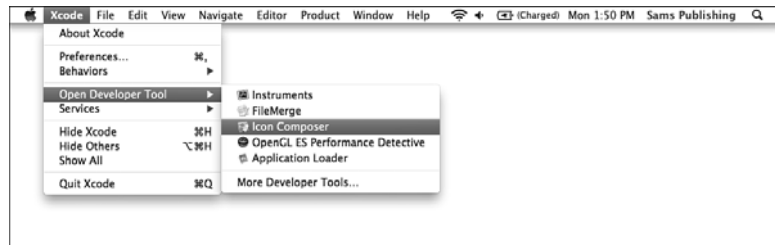
To get started, register and sign up for a developer program so you can download Xcode from developer.apple.com. If you are not certain that you want to register as a developer, you can purchase Xcode alone from the Mac App store. It is currently free. Starting with Xcode 4.3.1, it is an app just like any other you download from the App Store. It comes with a variety of tools as shown in Figure 1.1. (Prior to Xcode 4.3.1, it and the tools were installed in a special Developer folder.)

Launch Xcode to open the window shown in Figure 1.2. (While you are at it, you might want to set the option to keep it in the Dock. Some people like to launch

Xcode directly; others launch it by opening the Xcode project document they are currently working on.)

FIGURE 1.1

Xcode comes with a variety of developer apps.

**FIGURE 1.2**

Launch Xcode.

**TIP**

If you have run Xcode before, preferences might have changed and you may see a different welcome screen—or none at all.

As you can see in Figure 1.2, from this point you can create a new project, get help, and generally get started with Xcode.

- At this point, you can get started using Xcode by creating a simple app as described in Hour 2, “Creating a Simple App,” p. 49.

This hour continues with an exploration of the Xcode window and how to use it.

Getting Started with Xcode

Whether you are creating a new Xcode project or reopening an old one, you see the Xcode workspace window shown in Figure 1.3. Note that depending on your project and your Xcode preferences, the details of the window (not to mention the code) will very likely be different.

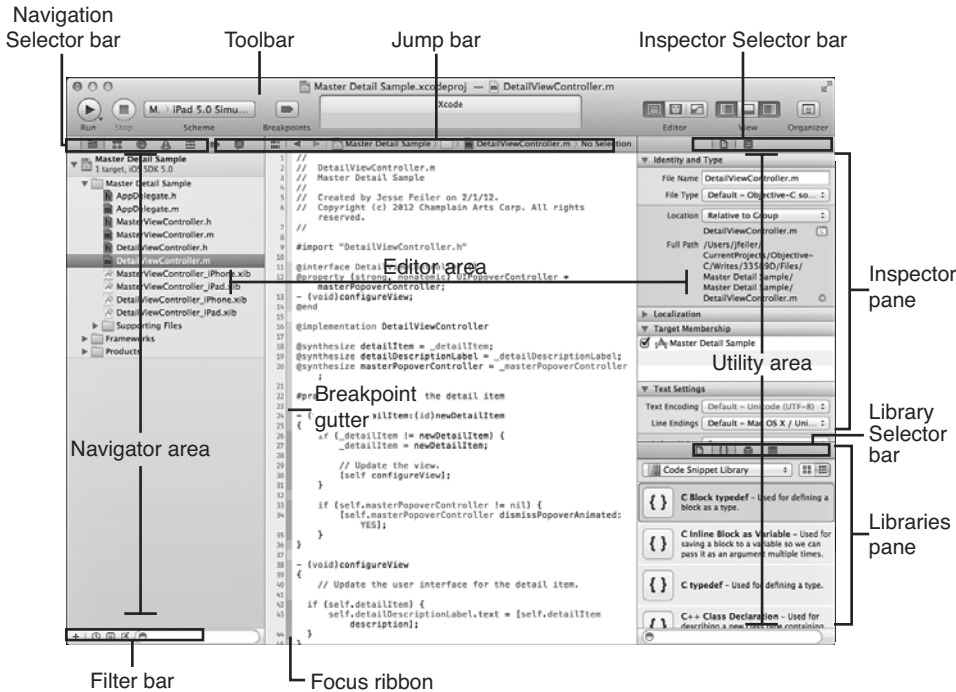


FIGURE 1.3

You work inside the Xcode workspace window.

Using the Workspace Window

As noted previously, Apple developer tools often provide a test bed for new interface features (and, under the hood, performance advances such as advanced threading). In its first demonstration of Mac OS X 10.7 (Lion), Apple showed how full-screen apps could take over the screen in much the same way that all apps do on mobile iOS devices. As Apple has moved forward, Xcode has provided an example of how a full-screen app can work. It was compelling and relatively simple to demonstrate

full-screen implementations of existing apps such as Preview, iCal, iPhoto, and Mail, which Apple did as long ago as fall of 2010.

But how would full-screen apps work with data that is not visual the way that photos, calendars, and the documents shown in Preview are? The answer was under developers' eyes right at the first preview: They just had to download a beta version of Xcode 4.

The window is a combination of panes and panes-within-panes that can be shown or hidden as well as resized. At first glance, Figure 1.3 can be daunting. But when you look at it a second time, you will see that it is actually fairly simple. It uses and reuses three components. Each component exhibits the same behavior wherever it appears. In addition, you can show or hide almost all the components, rearrange them, and resize them.

These are the main components of the workspace window:

- ▶ **Areas**—There are three areas shown in Figure 1.3. At the left is the navigator area, at the right is the utility area, and hidden at the bottom is the debug area. Each of these can be shown or hidden by using the three View buttons at the upper right of the workspace window. The editor area, in the center of the workspace window, is always visible.
- ▶ **Bars**—At the top of the navigator, editor, and debug areas, you will find a bar you can use to select different views for the area. The bar above the editor area is the jump bar, but the others are the navigator selector bar and the debug bar.
- ▶ **Panes**—The utility area is divided into two panes, each of which can be resized. The combined height of the utility area remains constant within the window size, so if you enlarge the height of the library pane, you automatically reduce the height of the inspector pane. Selector bars appear at the top of the panes in the utility area.

There are three lesser components in the workspace window:

- ▶ **Filter bar**—At the bottom of the navigator area, this lets you filter the lists in the navigator to include or exclude certain types of items, such as class symbols, files with unsaved changes, and so forth.
- ▶ **Breakpoint gutter**—This appears in the editor area and lets you insert and delete breakpoints for debugging.
- ▶ **Focus ribbon**—This lets you expand or collapse sections of code in the editor.

TIP

The best way to explore the workspace window is to open or create a project and then explore the menu bar. This hour can only provide a high-level summary of the workspace window.

There you have it: The workspace window is a compact and powerful environment to let you manage your development process. The same interface elements are used over and over, which means you do not have to learn a multitude of interfaces and functions. This is the result of the consolidation of Project Builder and Interface Builder along with a great deal of hard work and imagination.

Xcode is designed to be customizable with all kinds of preferences; these, together with the basic interface components, allow you to work the way you want to work on the projects you want to work on. (An iPhone app? A Mac OS app? And if you work for Apple, Mac OS X itself?) For these reasons, there is no sequential way to start working with Xcode. The sections that follow highlight some of the main components: Feel free to skip around.

NOTE

This overview of Xcode walks through the workspace window. There is an Xcode menu bar, as you would expect in a Mac app, but menus today are not nearly so important as they were many years ago. If this book had been written 10 years ago, it is quite likely that the overview would have walked you through each menu and each command in that menu. Now, however, we are in a world of direct manipulation where buttons, commands, and hot items are located throughout the interface—they are placed where you want to use them. This means that that lengthy mouse trip up to the menubar is often not necessary because the interface element that does what you want to have done is right on the window itself. (Hmmm, just like on an iOS device.) The menu commands are more often than ever available with keyboard equivalents. For many people, the menubar and its commands serve largely as a place to go to find the keyboard equivalent for a command. For these reasons, you will find the menu commands scattered through this hour; they are dealt with in the interface elements they affect.

Using the Navigator

The starting point for this exploration is the navigator pane at the left of the workspace window. You show or hide it with the leftmost View button, as pointed out in Figure 1.3. At the top of the navigator is a selector bar. The seven items in it control which navigator is displayed. You can use commands in the Navigators submenu of the View menu or keyboard equivalents instead of the selector bar if you want.

TIP

If the navigator is not visible, the menu command will automatically open it.

If you want to hide the navigator, use the leftmost View button or the View > Navigators > Hide Navigator command (⌘-0).

The next sections explain the navigators, their keyboard equivalents, and what they do.

Project ⌘-1

Figure 1.4 shows the project navigator. When you have first created a new project, it will very likely look like this. At the top of the navigator is a single item with a disclosure triangle to its left.

Click the disclosure triangle, and the single project item opens revealing its files and groups, as you see in Figure 1.5.

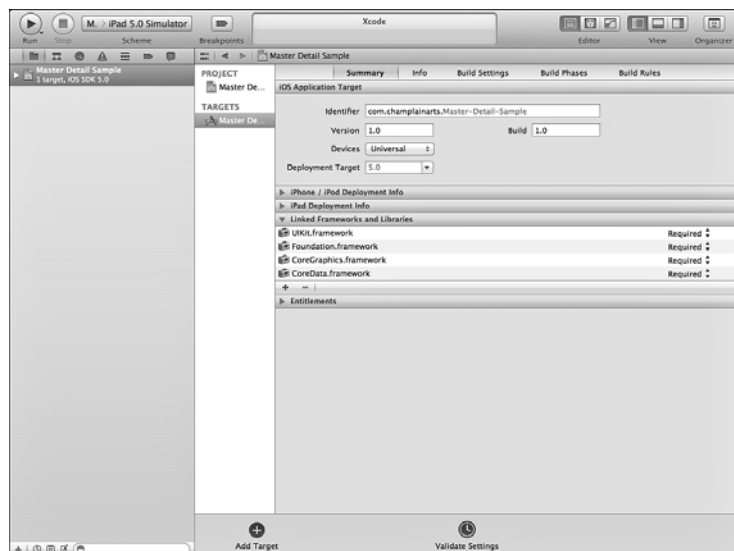
NOTE

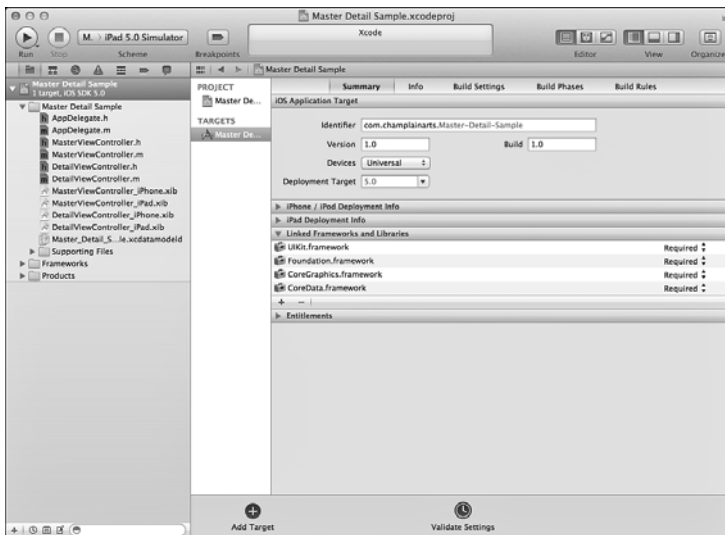
Groups are shown with folder icons, but they are not file system folders. The groups into which you organize your project's files are a construct within Xcode. The files can be anywhere you want.

Figure 1.5 also demonstrates another feature of Xcode: the parts of the workspace window know about one another. When you click the project icon at the top of the navigator, the editor area of the workspace window shows information about the project, as you can see in Figures 1.4 and 1.5.

FIGURE 1.4

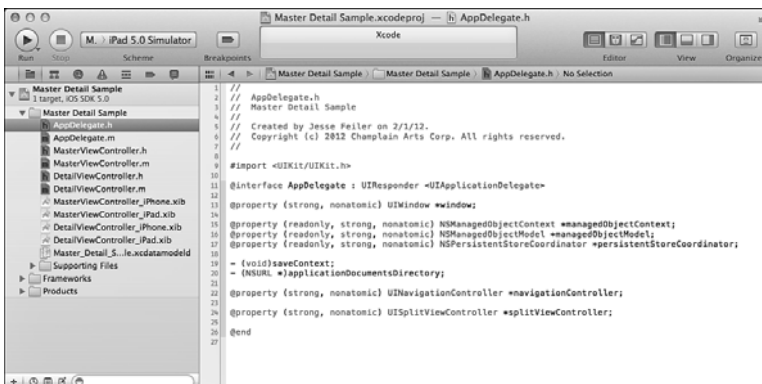
The project is shown in a collapsed form in the navigator right after you have created it.



**FIGURE 1.5**

You can expand groups in the project navigator.

Click one of the files in the project, and it appears in the editor area shown in Figure 1.6.

**FIGURE 1.6**

Click a file to edit it.

Clicking a file opens it in the editor area no matter what kind of file it is. Figure 1.7 shows an interface file (a *nib* file) in the editor area. Note that new projects for iOS have the option to use storyboards instead of nib files; for older projects and on Mac OS, nibs remain the standards.

- Learn more about storyboards in Hour 14, “Working with Storyboards and Swapping Views.”

Figure 1.8 shows a Core Data data model file in the editor area.

In Figure 1.9, you see that if you have added an image file to your project, clicking it opens the image in the editor area.

FIGURE 1.7
Edit a nib file in Xcode.

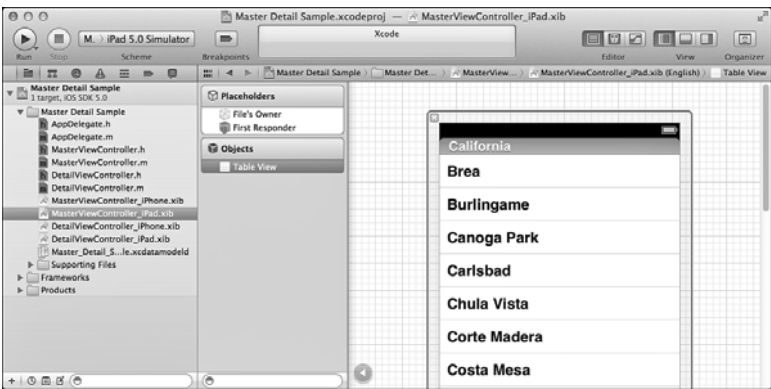


FIGURE 1.8
Edit your data model in Xcode.

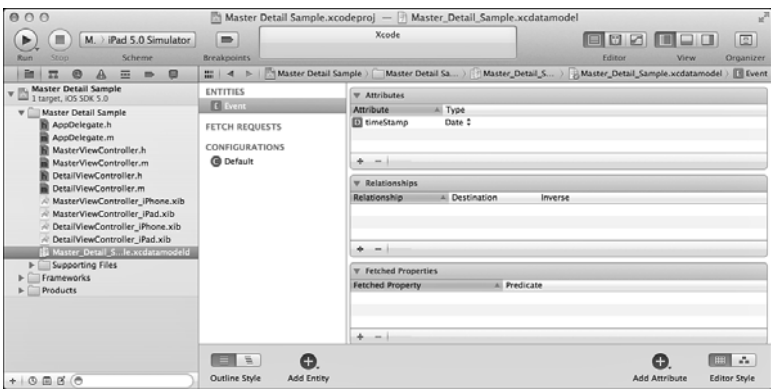
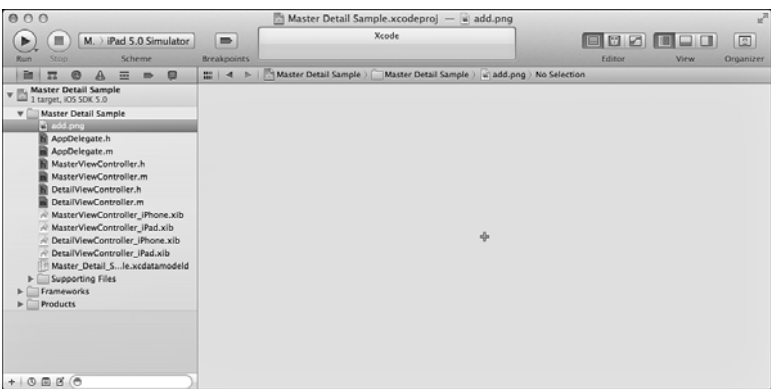


FIGURE 1.9
Open resource files.



In other words, no matter what kind of file it is, select it in the project navigator and edit it in the editor (for the file types that Xcode supports).

You have seen how to use the navigator to explore your project and its files, but how do you manage the files themselves? When you create a project, as you will see in Hour 2, the files are automatically created for you. In your own projects, you might need to add files to it. Control-click in the project navigator to bring up the shortcut menu shown in Figure 1.10. For many people, right-clicking the mouse will have the same effect. You can add the new file anywhere you want and move it to the right position in the navigator just by dragging it. If you control- or right-click in a group, the file will be added to that group and you might not have to move it.

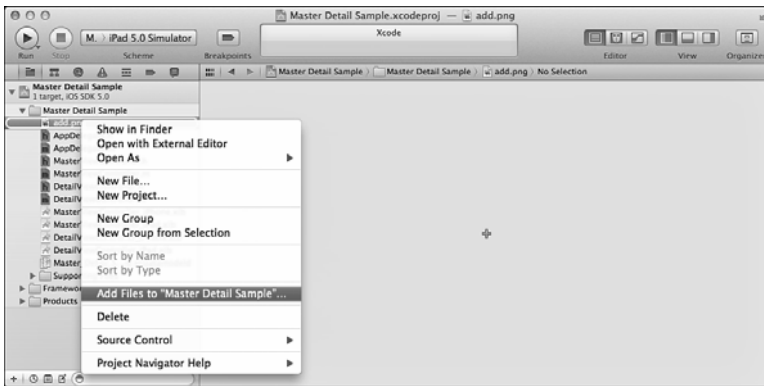


FIGURE 1.10
Use the shortcut menu to add files to the project.

Once you have selected a file to add, the sheet shown in Figure 1.11 opens.

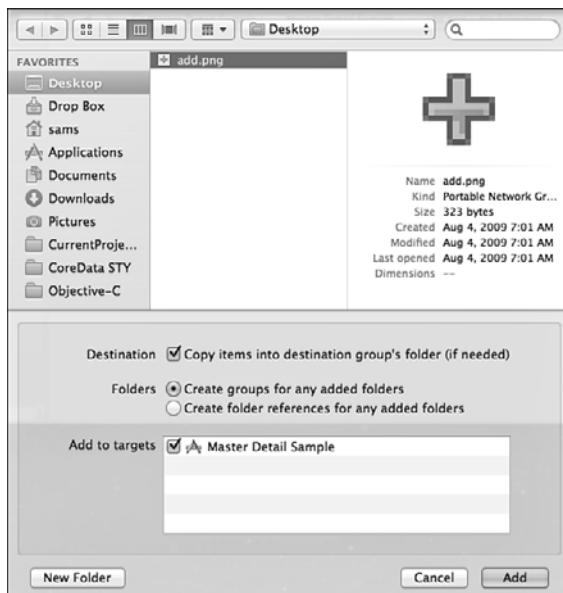


FIGURE 1.11
Specify a file to add.

The most important part of this dialog other than the filename is the Destination checkbox. This determines whether the project will use the file that may be somewhere else on your disk or network or whether it will copy it into your project. Normally, you do want to copy the file into the project so that you can then move the entire project folder to another computer if necessary.

TIP

Sometimes, a filename will appear in red. This indicates that it is part of the project but that it is missing. For example, before you have built your project, the file named <MyProjectName>.app appears in red. After you have successfully built your project, the name appears in black.

The filter bar at the bottom of the project navigator lets you filter by filename (or part thereof). The + in the bottom-left lets you add a new file with a template (it is not the same as the add file to project command shown in the shortcut menu in Figure 1.10). Three symbols to the right of the + limit your navigation. From left, here are their effects:

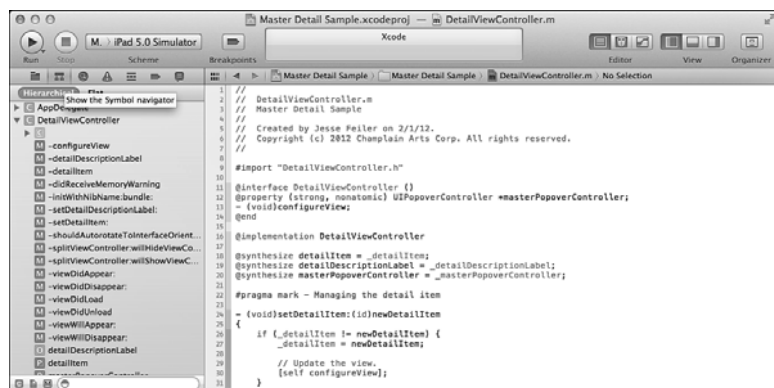
- Show only recently edited files
- Show only files with source-control status such as modified
- Show only files with unsaved changes

Symbol ☿-2

The symbol navigator, shown in Figure 1.12, shows you the symbols in your project: the classes (indicated with C), methods (M), and properties (P). Interface Builder actions (A) and outlets (O) manage the interactions between your code and your interface.

FIGURE 1.12

Use the symbol navigator.



Properties are identified by P unless they are Interface Builder outlets—a special kind of property. The `synthesize` directive that is the companion to a property directive is flagged with a V (for variable).

- ▶ You will find out more about the property and `synthesize` directives in Hour 3, “Understanding the Basic Code Structure,” p. 63.

At the bottom of the symbol navigator, you can filter the display. Use the search box to type text to search for in symbol names. To the left of the filter bar, symbols let you choose what to display and hide. From left, the following effects are available:

- ▶ Show only class symbols—that is, no globals
- ▶ Show only symbols defined in the project
- ▶ Show only containers such as classes and categories; do not show members

Search ☞-3

The search navigator packs a lot of searching into a small space. You can use it by simply typing a search term into the box; Xcode will search for it through the project. The list of results (if any) is shown in the search navigator. You will see the relevant filename, a symbol such as the ones shown previously in Figure 1.12, and the beginning of the line of code. The search term is highlighted in yellow in each line. Sometimes this means that you do not see the beginning of the line, but never fear—a click on the line will display it in the editor area, or you can hover the pointer over it to see a tooltip with the full text.

You can switch between searching and replacing text at the upper-left, as shown in Figure 1.13. In addition, at the bottom of the search navigator, the filter bar lets you search within the results. In Figure 1.13, for example, the find was executed on “detail.” (You can see this because “detail” is highlighted in all of the search results.) The filter bar is used to filter on “item.” If you look at the search results, you will see that “detail” is always found, but each of those results also contains “item,” which is not highlighted because it was not part of the original search. You can duplicate these results for yourself. Conduct a search without a filter, and then add a filter. You’ll see that the number of results is reduced.

Just to the right of the magnifying glass in the search field, a disclosure triangle lets you show or hide the Find Options shortcut, as shown in Figure 1.14. It also lets you repeat recent searches.

FIGURE 1.13
Specify a
search.
search.

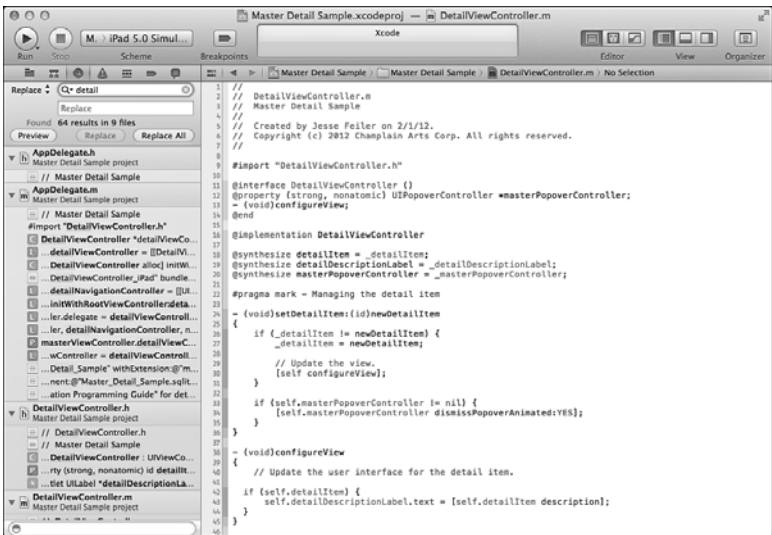
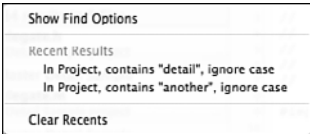
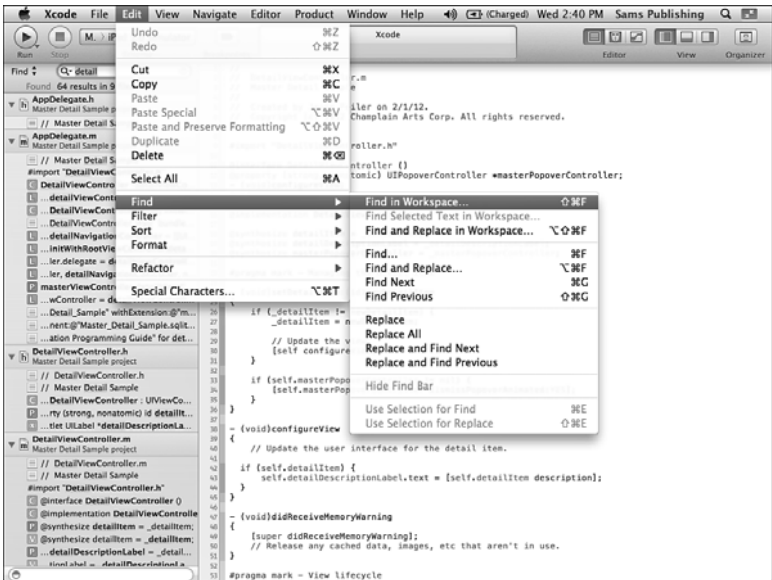


FIGURE 1.14
Show or hide
Find Options
shortcuts.



The search navigator searches throughout the project. The Edit menu has traditional single-file Find commands, as shown in Figure 1.15.

FIGURE 1.15
The Edit menu
provides a mul-
titude of search
and replace
options.



Issue ¶-4

The issue navigator lets you view the issues with your project. In the old days, these used to be called *compile errors*, but with Xcode, you will have many fewer compile errors. Do not get your hopes up, though. That is because Xcode has a powerful parser that checks your code as you type. It is as lively as a spell-checker, but it looks for syntax errors as well as ordinary misspellings. This means that compile errors now show up much earlier—just as you are typing them in many cases. The issue navigator lets you see them. You can display them by file (the traditional way of showing compile errors), but you can also display them by type so that like errors are grouped together. Sometimes that can make fixing the errors faster, particularly if you are consistently mistyping a variable name.

Figure 1.16 shows the issue navigator. In addition, note that, in the breakpoint gutter at the left of the editor area, symbols show up as soon as you have made the offending keystroke. (An extra *s* has just been added to *synthesize*—*synthesize*.)

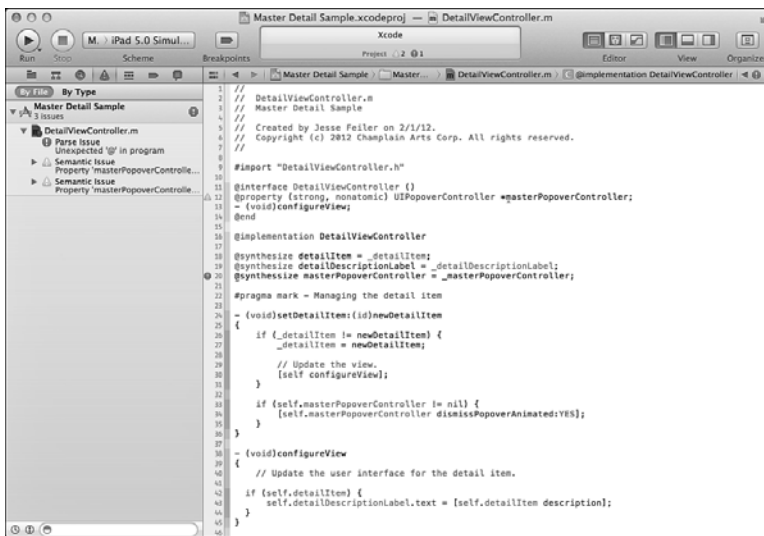


FIGURE 1.16

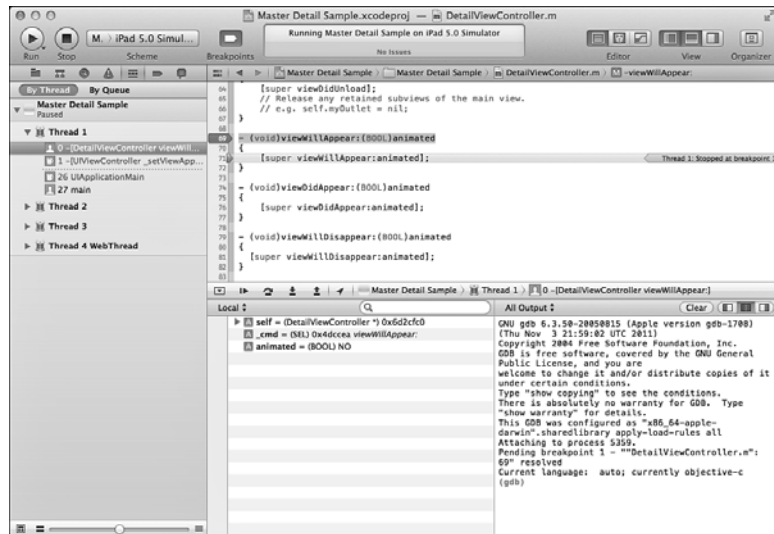
The issue navigator helps you correct errors as you type.

Debug ¶-5

Debug shows you the calling sequence for each of your app's threads (in the simplest case, there is only one). For example, Figure 1.17 shows the app stopped in `DetailViewController viewWillAppear`. That was called from `UISplitViewController viewWillAppear`, and so on back to the bottom of the calling sequence—`main`, which starts the program running.

FIGURE 1.17

Use the debug navigator to track a calling sequence.



Breakpoint ¶-6

Breakpoints let you stop program execution at specific lines of code. You place a breakpoint in the breakpoint gutter to the left of the editor area and, when the program is about to execute that line of code, it stops. You can then inspect the variables in the debug area. In Figure 1.17, a breakpoint was set in the editor area at `[super viewWillAppear:animated;]`. The program stopped just before executing that line of code. The calling sequence is visible in the debug navigator. In the editor area, you see the breakpoint, and, to its right, a small green arrow that points to the line of code about to be executed. If you have several breakpoints, you need to know which one has just stopped the app.

Beneath the editor area, the debug area shows you information about the breakpoint. On the left is a view of the variables at this moment. On the right are console messages. Buttons at the upper-right of the debug area let you choose which—or both—views to display. In the view of variables, you can expand and collapse containers as you examine exactly what data is where.

TIP

Breakpoints can be useful even if they do not trip. When you cannot figure out why a line of code does not work properly, set a breakpoint on it to examine the data. If the breakpoint is not tripped, work backwards to see where the app goes off the rails. Command-click on a breakpoint to edit it. For example, you can stop only after the *n*th pass through the breakpoint and only if a certain data condition is true. You can add actions to the breakpoint such as a sound; a log message; a shell command; or even that trustworthy and powerful tool, an AppleScript script.

To remove a breakpoint, drag it out of the breakpoint gutter. You can also use the breakpoint navigator to list the breakpoints. Clicking one will take you to the line of code. You can drag breakpoints out of the breakpoint navigator to remove them if you prefer not to drag them out of the breakpoint gutter.

TIP

Note that there is a global breakpoint control in the toolbar. Use it to turn all breakpoints on or off. This is helpful in debugging when you are done with the breakpoints but might want to turn the breakpoint back on the next time a bug appears.

Log ☒-7

Finally, the log navigator keeps track of what you've been doing with this app. Figure 1.18 shows the log navigator. The events are in reverse chronological order (latest first). As always in the navigator, a filter bar lets you filter the entries so you can easily find builds or other specific types of entries; you can also use the control at the bottom-left to see the most recent log entries. Clicking a log entry shows you the console results for that compile, build, or other action.

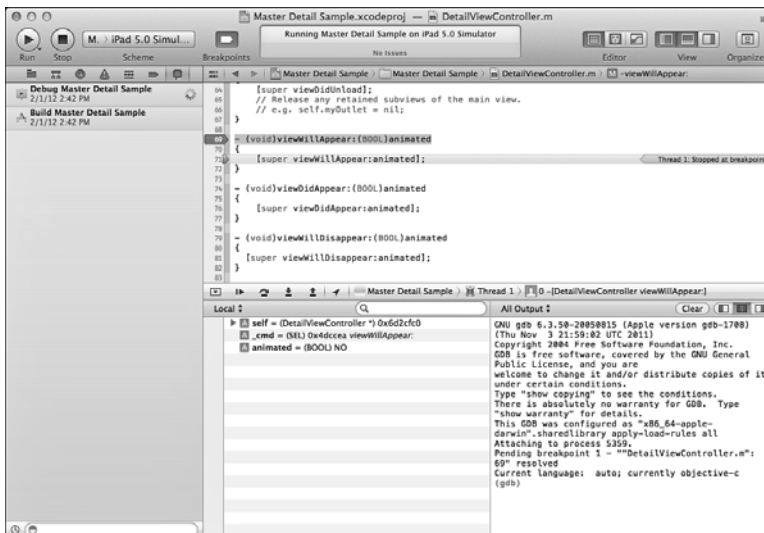


FIGURE 1.18
The log navigator keeps track of your work.

Using Editors

The center of the workspace window is reserved for editing your project and its files. As you have seen, different editors are automatically opened for the different types of files in your project.

- This section focuses on text editors; other editors are discussed in Hour 6, “Working with the Core Data Model Editor,” p. 117, and Hour 11, “Finding Your Way Around Interface Builder: The Graphics Story,” p. 189.

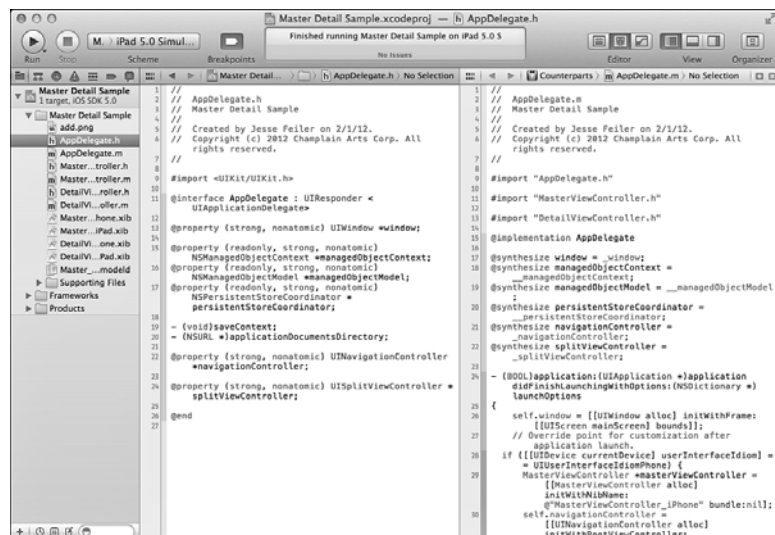
Using Editing Modes

Three editing modes are available in Xcode:

- ▶ **Standard**—This displays a single file in the edit area.
- ▶ **Assistant**—This displays two or more related files in the edit area.
- ▶ **Version**—If you are using source control, you can compare a file with its previous version or versions.
- ▶ Refer to “Working with Assistant” on p. 29 of this hour for details about the Assistant mode.

You select the editing mode with the trio of buttons marked Editor at the right of the top of the Xcode window, as shown in Figure 1.19. You can also use View, Editor to choose among them.

FIGURE 1.19
Select the
assistant you
want to use.



Using the Jump Bar

The *jump bar* appears at the top of the editor area no matter what mode you are in. As you can see in Figure 1.20, the jump bar above the editor area shows the path to the file you are working on relative to the project and lets you quickly navigate to a file, method, property, or class in the file. If you have several files open (as is often the case in Assistant and Version editor panes), each has its own jump bar.

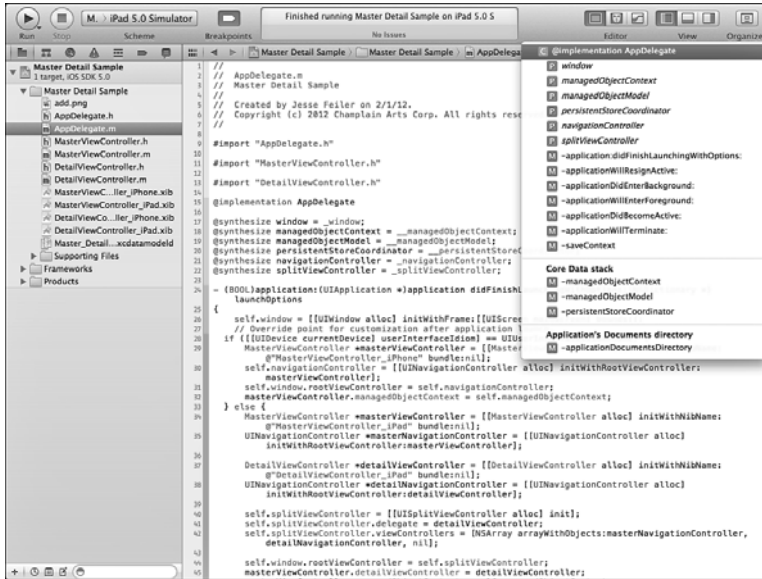


FIGURE 1.20
Jump bar in action.

You can use it to quickly navigate to a file or to a method, property, or class in the open file.

Thus, at the left, you see the icon for the project (Master Detail Sample); within that, you see a group (Master Detail Sample—shown with a Finder folder icon), and within that, the filename is shown (Hour1_AppDelegate.m). The next level down is a list of the methods, properties, and classes in that file.

TIP

It is important to note that this is the logical structure of the project, files, and groups. If you move the project to another folder, drive, or computer, this structure will remain the same.

Organizing Your File's Pop-Up Menu List

In addition to the names of the methods, properties, and classes, titles appear in the pop-up list. You put titles into the file using a pragma directive:

```
#pragma mark - headingName
```


At the top of the menu, submenus show you unsaved files and recent files. Submenus show you these types of related files when you are looking at source code. Other types of files, such as nib files and Core Data model editor files, have different submenus:

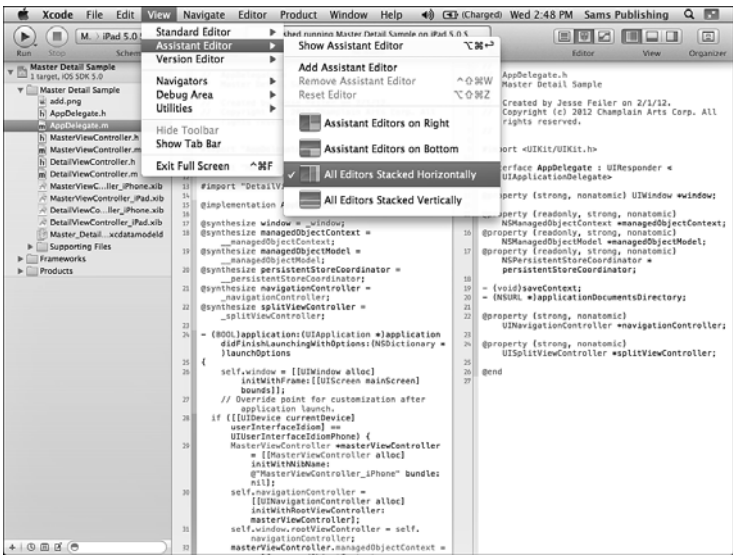
- ▶ **Counterparts**—This means the .h files for .m files, and vice versa.
- ▶ **Superclasses**—There is always a superclass (except for NSObject). This list is organized in order so that the last item at the bottom is always NSObject.
- ▶ **Subclasses**—If any.
- ▶ **Siblings**—These are classes that share the same immediate superclass.
- ▶ **Categories**—This is an Objective-C construct that allows you to add methods to an existing class.
- ▶ **Protocols**—This Objective-C features lets you declare a set of methods that can be implemented by several classes in their own ways and with their own data structures. Protocols provide functionality similar to multiple inheritance in some other object-oriented languages.
 - ▶ Both categories and protocols are discussed in Hour 3, p. 63.
- ▶ **Includes**—These are the files that are included in the file you are looking at.
- ▶ **Included By**—From an included file, you can return easily to this file; you can also see the other files in your project that may include this file.

With these various navigational tools available and updated by Xcode, you might want to use the adjacent forward and back arrows. They function just as forward and back arrows do in a browser. This means that you can use the related items menu to explore the rest of your project and get back to where you started from with just a few mouse clicks.

Working with Assistant

Assistant lets you see several files in the same pane of the window, and it can take advantage of the fact that Xcode keeps track of the relationships among files that you have already seen in the related items menu. As soon as you think about displaying several files in the same pane, the question arises as to how to display them. Xcode gives you a variety of choices, as shown in Figure 1.22.

FIGURE 1.22
Control the lay-
out of assistant
panes.

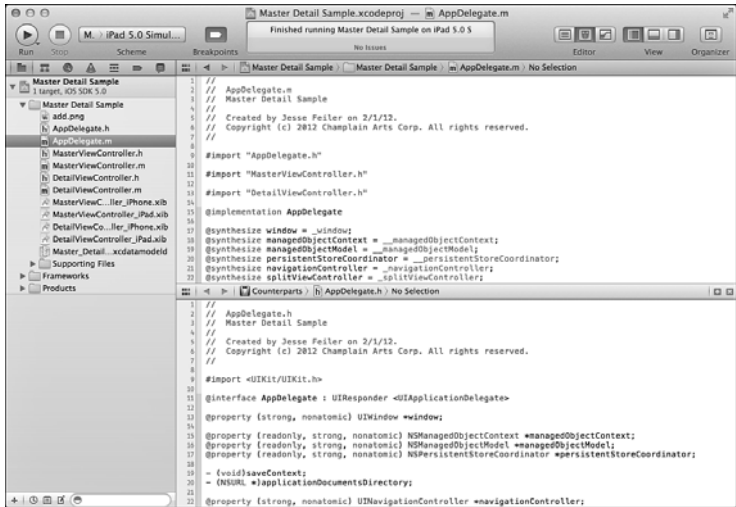


Experiment with the various layouts. Most people switch back and forth among them, depending on the size of their display and the files that they are working with. Sometimes, you are dealing with short lines of code that look good side-by-side, but in other cases, you have large chunks of code that need the width of your computer display.

Once you are using an assistant, you might be able to open additional panes in the assistant. Figure 1.23 shows two panes displayed, one above the other. When you have several panes in the assistant window, each has its own jump bar.

Also, note that small widget at the right of a jump bar let you close that pane or add another pane.

FIGURE 1.23
You can open
additional
assistant
panes.



Getting Help in an Editor Window

You can option-click on a word in an editor window to bring up help and documentation, as shown in Figure 1.24.

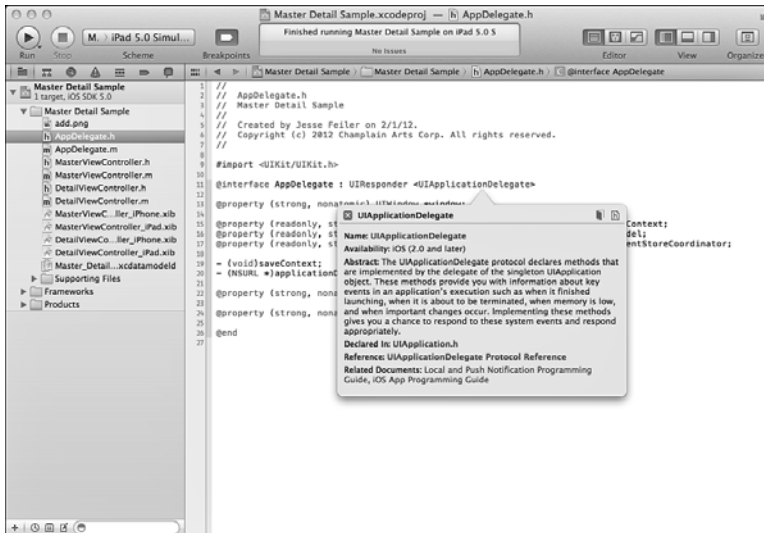


FIGURE 1.24
Use option-click to get more information about code syntax.

Where possible, there will be two links to the documentation—the filename is a link, as is the file icon with .h in the upper-right of the window. The book in the upper-right opens the reference in the Organizer window, which is described later in this hour.

- Find out more about help and documentation in the “Using the Organizer Window” section on p. 45 of this hour.

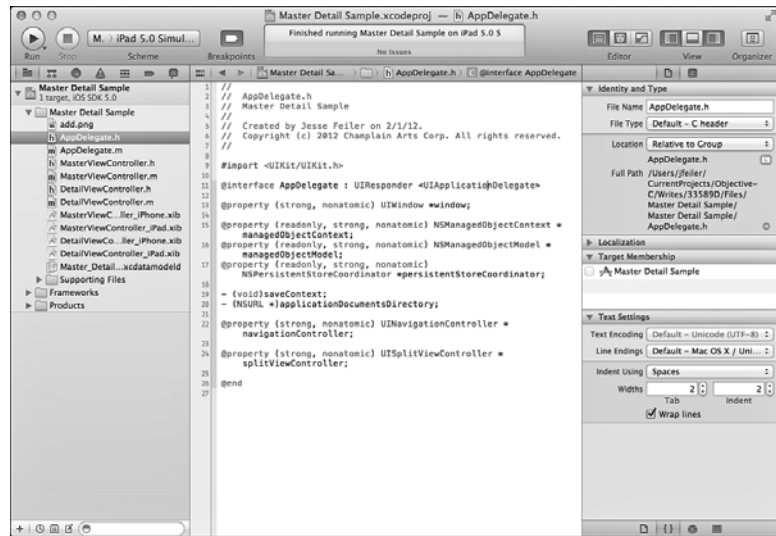
Using Utilities—Inspectors

At the right of the workspace window is the utility area. This consists of two panes stacked one above the other. You can drag the divider between them to change their sizes, but they always fill the utility pane.

At the top of utilities are the inspectors. They change as you select objects in the editor window. The content of the pane depends on what is selected in the editor, as well as on which of the buttons at the top of the inspector is selected. However, as you will see, a consistent framework applies to all selected objects.

In Figure 1.25, you see the file inspector as it appears when a file is selected in the project navigator; if a line of text within a file is selected in the text editor, the display may look the same.

FIGURE 1.25
Use the file
inspector.



At the left of the top of the inspector, the small icon lets you view the information about the file you have selected. Information about the filename and file type is available. Each section of the file inspector has its own heading; you can expand or collapse each one.

These settings are self-explanatory, but one of them needs careful attention if you want to avoid problems. The location of each file can be set to one of six settings:

- ▶ Absolute Path
- ▶ Relative to Group
- ▶ Relative to Project
- ▶ Relative to Build Products
- ▶ Relative to Developer Directory
- ▶ Relative to SDK

Relative to project means that if you move the project to another computer, folder, or disk, all the files within the project move together and the internal file structure stays intact. An absolute path is great if the path is to a location on a shared server that a number of people will be using. In that case, the project files stay in one place, but the developers can move from computer to computer.

Relative to Group can be a good structure for a multiperson project where components are being developed by different people at different times. Each person can structure a group without worrying about how they will be arranged together. The remaining choices are useful in specific cases that typically are involved with large projects or special conditions.

TIP

Of course, by using a source code repository, you can handle the issues of sharing and version control easily.

To the immediate right of the file inspector button is a Quick Help button. If an element in the editor is selected and help is available, it will be displayed as shown in Figure 1.26.

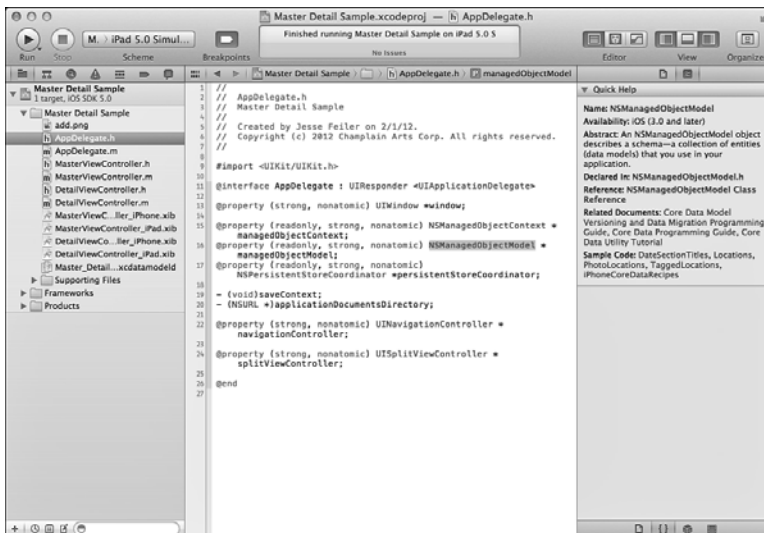


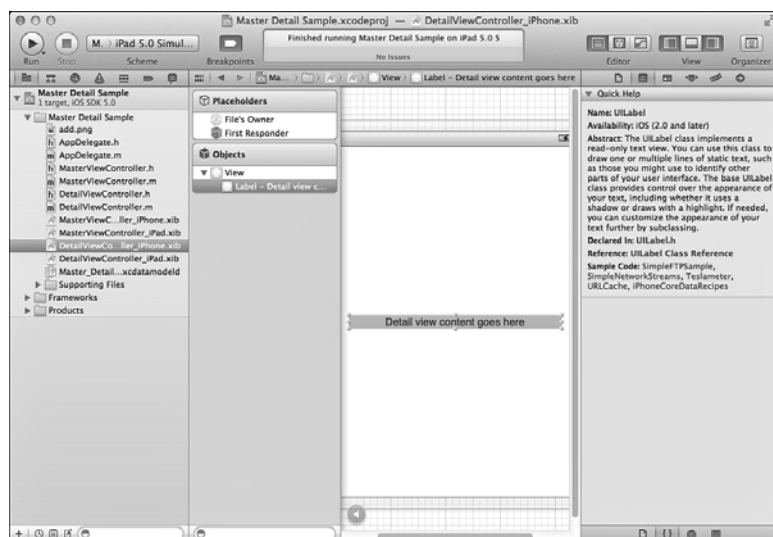
FIGURE 1.26 Quick Help is available wherever possible in the inspector pane.

In Figure 1.27, an Interface Builder document is open. The file inspector is still the left-most button at the top of the window, and its data is much the same.

Immediately to its right, a help inspector will reflect information about the selected item in the editor. However, new inspectors are available to let you inspect items in the interface.

FIGURE 1.27

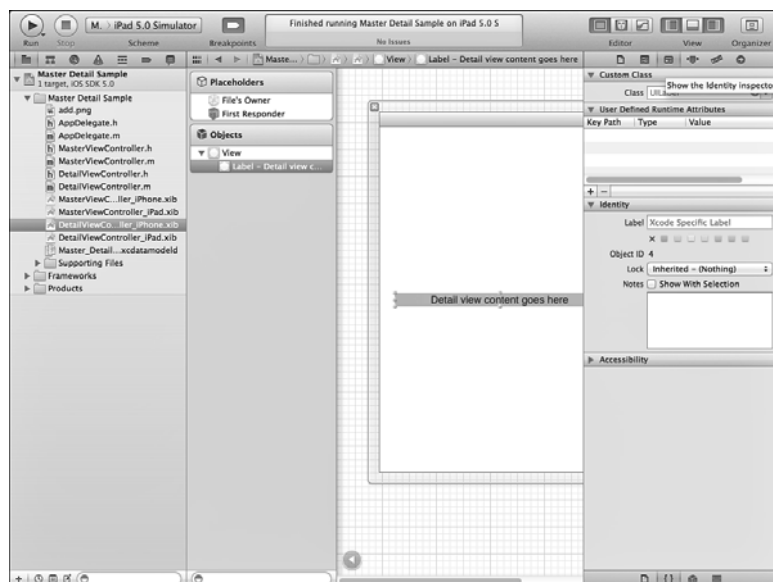
Inspectors change depending on what is selected in the editor.



For example, in Figure 1.28, you see the Identity inspector in action. It identifies a selected object in the interface.

FIGURE 1.28

Use the Identity inspector.



- Refer to Hour 11, p. 189, to find out more about how you can use these inspectors to set everything from an object's location to its behavior as people type in it.

Using Utilities—Libraries

The bottom pane of the utility area is for libraries. These are collections of items that you can add to your apps just by dragging them to the appropriate place in an editor.

- ▶ More information on libraries is included in Hour 11, p. 189.

The selector bar at the top of the library pane lets you choose from four libraries:

- ▶ **File templates**—[ctrl][option][command]1
- ▶ **Code snippets**—[ctrl][option][command]2
- ▶ **Objects**—[ctrl][option][command]3
- ▶ **Media**—[ctrl][option][command]4

You can also use the View, Utilities submenu to select the library you are interested in. If the Utilities submenu is hidden, use the View menu or the rightmost of the three View buttons at the upper-right of the workspace window to show it.

TIP

Alternatively, if the utility area is hidden, choosing View, Utilities, File Template Library or any of the other commands in the View, Utilities menu will show utilities and select the appropriate library with one command (or one keyboard shortcut).

Figure 1.29 shows the general components of the library pane. At the top of the library pane, a pop-up menu lets you navigate to sections within that library. To its right, buttons let you display the contents of that library as icons or in a list. The icon view can make finding images or objects such as graphic elements very fast; for other items such as code snippets, the list view is better.

At the bottom of the library pane, a search field lets you filter the library shown above it.

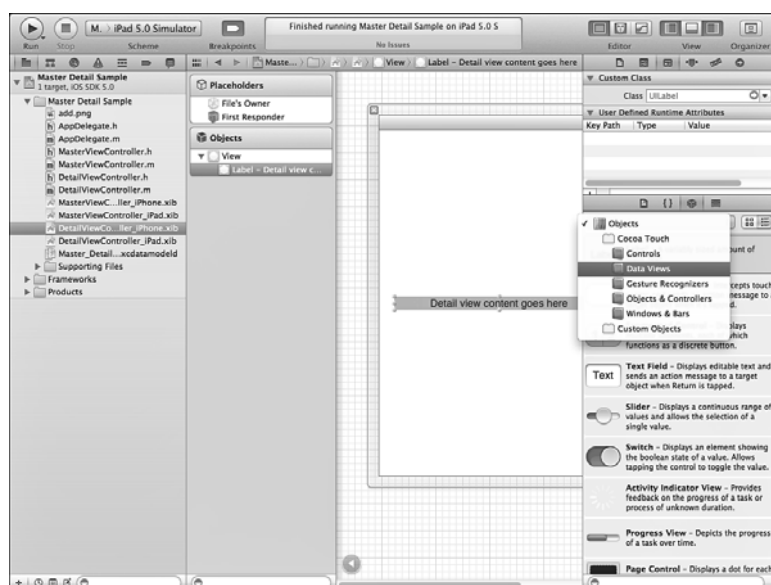
When you select an item in the library, a description appears floating over the editor area, as shown in Figure 1.30. This description is generally somewhat lengthier than the summary in the library list.

File Templates Library

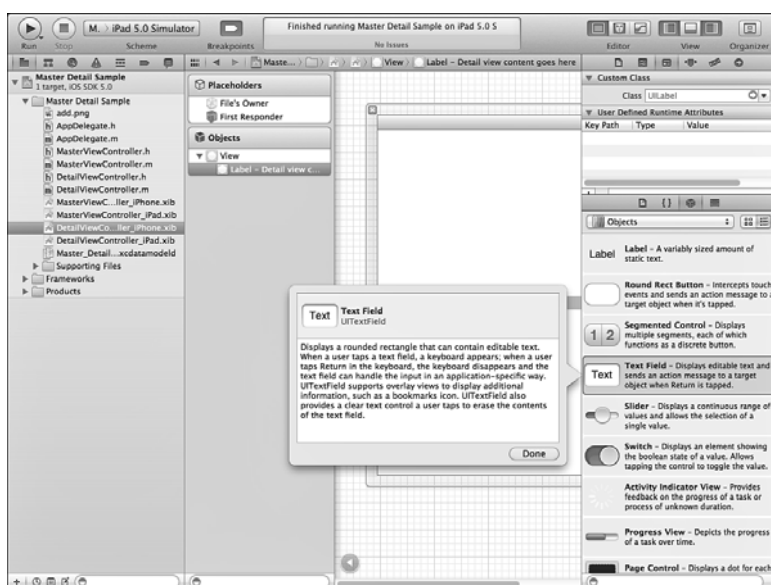
These file templates give you a headstart for whatever type of code you want to write. The pop-up menu at the top of the library pane lets you choose between iOS and Mac OS X file templates.

FIGURE 1.29

Use the library pane to take advantage of existing code, objects, and media.

**FIGURE 1.30**

Select an item in the library to see its description.



You select the appropriate file template from the file template library and drag it into the project navigator, as shown in Figure 1.31.

TIP

Remember, that for this to work, you need to have both the project navigator and the file template library visible.

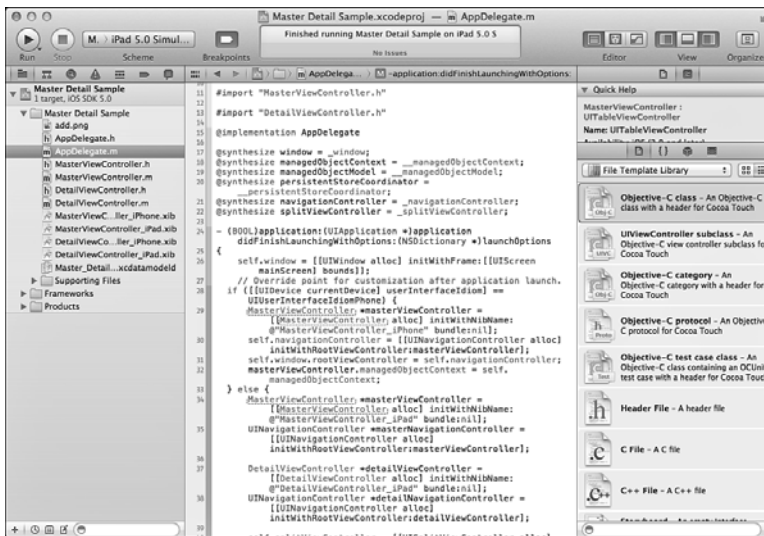


FIGURE 1.31
Use a file template.

You can also use file templates by choosing File, New, New File, as shown in Figure 1.32. The templates in this interface are shown grouped by their SDK and area of functionality. However, as you can see in Figure 1.32, by comparing the descriptions of `NSObject` subclass in the library and in the sheet, they are the same. The menu command gives you the organization by SDK and area, while the library provides you with the ability to search with the filter at the bottom of the pane. The choice is yours.

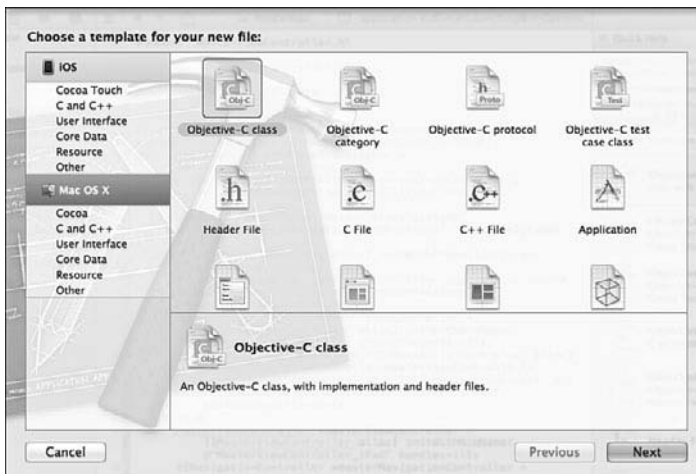


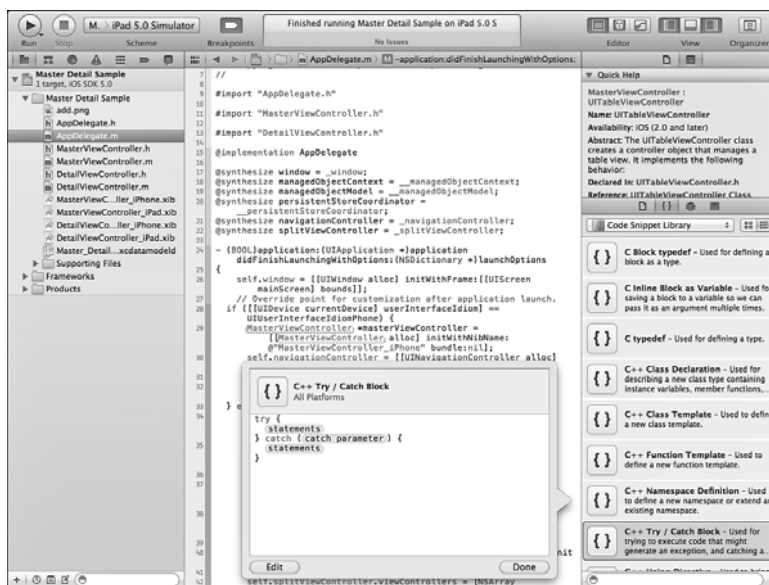
FIGURE 1.32
Use either the library or the menu to access a template.

Code Snippet Library

Code snippets can only be dragged into text editing files. They provide common examples and templates. The pop-up menu lets you choose from iOS, Mac OS X, and your own snippets (which you can add).

If you select a code snippet, its code appears as shown in Figure 1.33. Sometimes, if you have just forgotten a small piece of syntax, this refresher is enough and you do not have to worry about actually dragging the snippet into your file. Other times, the snippet gets you started with your own programming.

FIGURE 1.33
Select a snippet to see its contents.



Try It Yourself

Add Your Own Code to the Code Snippet Library

Add your own snippets to the library to save time or to enforce standards on yourself or your colleagues in a multiperson project. (A particularly useful snippet would be the copyright notice you place at the beginning of each file if you want to protect your work.) Here's how:

1. Show the User section of the Code Snippet library. If necessary, show utilities and choose User from the pop-up menu at the top of the library pane.
2. Select the code you want to make into a snippet.
3. Drag the code into the Users pane of the code snippet library. It will appear in the list, as shown in Figure 1.34.

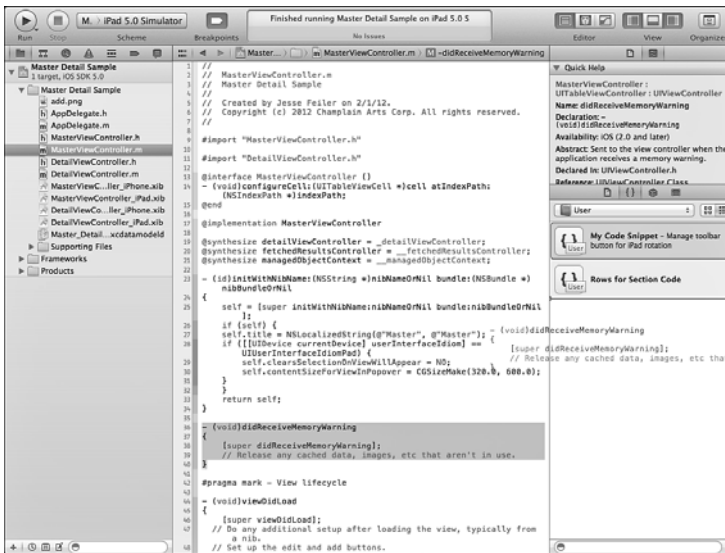


FIGURE 1.34
Drag the code into the library.

4. Provide a title and summary. Also, check that the code is complete (check the first and last characters in case of sloppy mousing).
5. Provide the other information (optional). The more information you provide, the more useful your snippet will be. In particular, specifying the language as shown in Figure 1.35 will remove it from the code snippet library for files that cannot use it. And, of course, a title other than My Code Snippet will increase the usability of the code.

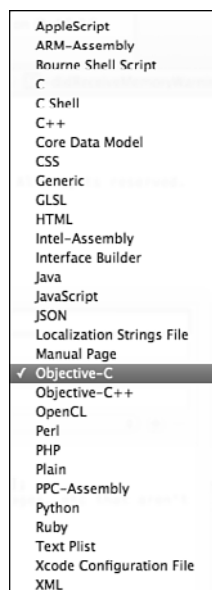


FIGURE 1.35
Identify the snippet language.

6. Click Done, and your snippet is added to the library.

▲ To change the snippet's name, summary, or other data, select it and then click the Edit button, as shown previously in Figure 1.33. Click Done to save the changes.

Object Library

The Object library contains objects you use in building interfaces. This includes visible interface elements, such as views and buttons, as well as objects that work behind the scenes, such as view controllers.

- ▶ See Hour 11, p. 189, and Hour 12, “Finding Your Way Around Interface Builder: The Code Story,” p. 209, for more details on developing your app's interface.

Media Library

The Media library brings together media files (icons, sounds, and images) from your workspace or from the system. Particularly when you have large projects, this helps you keep things organized. It also means that in creating your file groups, you can organize them functionally rather than putting all media files in one group and all code files in another.

Using the Text Editor

The text editor in Xcode is similar to many text editors that you have probably used already. Two areas deserve your attention even if you are used to using text editors:

- ▶ **Editing preferences**—Xcode provides extensive preferences for displaying and auto-completing code. Even if you have used other text editors, take a quick look at these preferences so that you can find out what's new in Xcode and, if you are used to another text editor, how to customize colors and behaviors to what you are used to.
- ▶ **Fix-it and Live Issues**—The LLVM compiler in Xcode 4 is not just for formal compiles. Its engine runs in the background checking syntax as you type so that errant keystrokes are caught in many cases as soon as you make them. Not only is the LLVM engine looking for misspellings, but it is aware of common syntax errors that can take a long time to track down, even though they are absurdly simple (once you know what the error is). One

such error is demonstrated in this line of code that almost every developer has typed more than once:

```
if ( x = 3 ) {...
```

That is a replacement statement, not a logical comparison. Fix-It would most likely suggest the following:

```
If ( x == 3 ) {...
```

Setting Editing Preferences

As in most Mac apps, preferences are set from the application menu (that is, the Xcode menu in this case). Tabs at the top let you set different collections of preferences, and, as in the case of text editing, further tabs let you set more details such as the editing and indentation preferences.

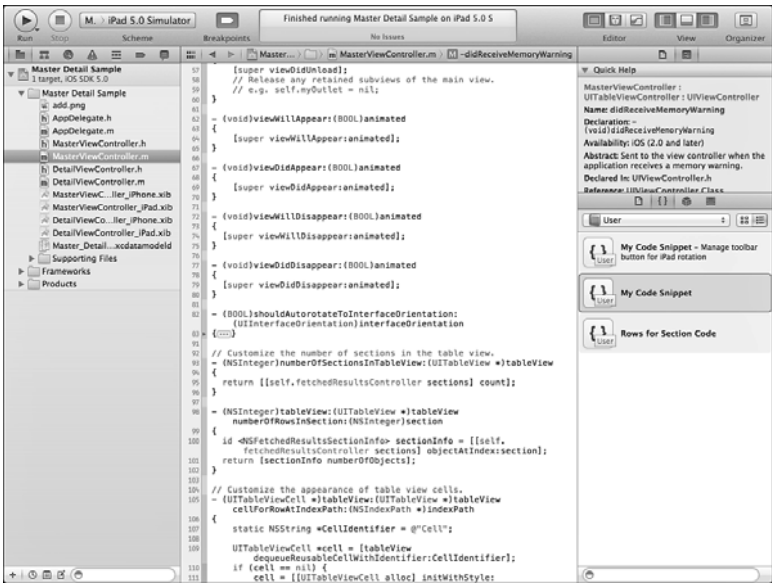
Figure 1.36 shows the editing preferences. Most are familiar to users of other code text editors, but two may be new to you. The code folding ribbon appears to the right of the gutter and the left of the main text editing area. It lets you collapse blocks of code so you can focus on other areas. If the code folding ribbon is shown, you have a further option—to focus on code as you hover the pointer over it.



FIGURE 1.36
Set editing preferences.

Figure 1.37 shows this behavior in action. Note the folded code in `shouldAutorotateToInterfaceOrientation`.

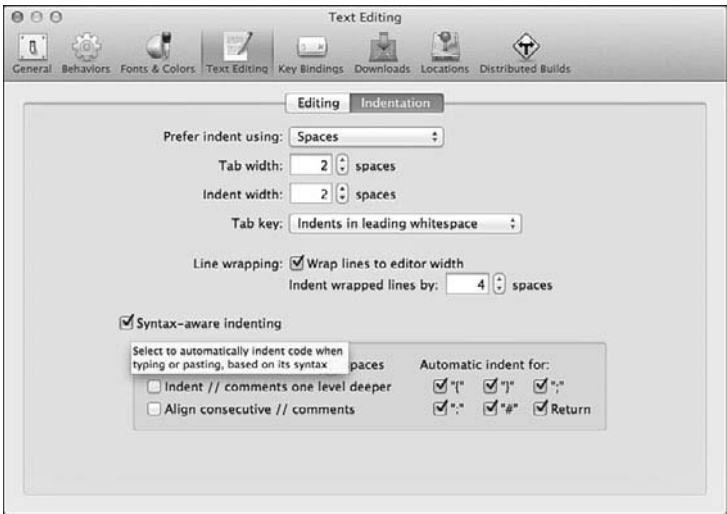
FIGURE 1.37
Highlight blocks of code by hovering over them.



Many people have this option on at all times. As you move the mouse over code, you will quickly spot unmatched brackets or quotation marks because the highlighted block of code will be illogical.

Syntax-aware indenting can be set, as shown in Figure 1.38. Just as with the highlighting of code in the code ribbon, this can provide an early warning of unbalanced punctuation.

FIGURE 1.38
Syntax-aware indentation makes your code neater and catches some keystroke errors as well.



The final preference you should look at is Fonts & Colors, as shown in Figure 1.39.

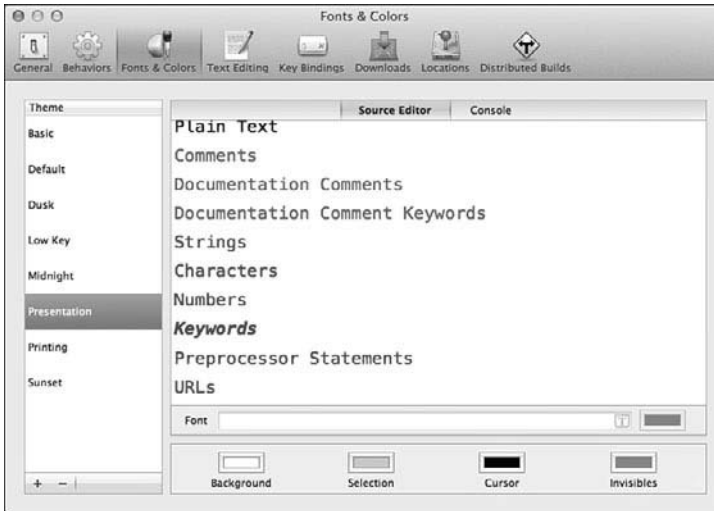


FIGURE 1.39
Set Fonts & Colors.

A variety of predefined styles is available, and you can switch back and forth among them as you wish. The color wells at the bottom of the window bring up a color picker for you to use to replace any of the colors in the theme for the syntax element that you have highlighted in the main body of the window. The font for the highlighted syntax element is identified in the font field; click the T at the right of the field to bring up the font panel and change the size, style, or font.

Among the provided themes is one called Presentation. For some people, this is one of the most frequently used themes. Whereas the other themes ship with fonts that are 11 points, the Presentation theme ships with an 18-point font. Not only is Xcode used to build Mac OS X and iOS as well as Apple apps, it is also often used to prepare slides for conferences such as the Worldwide Developer Conference—and that is where the presentation theme comes in handy. Even if you are not presenting at WWDC, the Presentation theme can be useful for code reviews and documentation in your own organization.

Using Fix-It and Code Completion

Xcode is constantly indexing your project and its files in the background. As it does so, it can provide code completion (type-ahead) tips for you. Figure 1.40 shows this feature in action. As you type each character in a symbol name, a list of the possible completions appears. You can select one of them or continue typing to narrow down your search. In many cases (such as your own variables), there is no list; there is just a grayed-out completion displayed. Pressing Return accepts the completion.

NOTE

With both code completion and Fix-It, do pay attention to what you are accepting. Certain types of errors can generate incorrect corrections or completions. The main benefit may come simply from stopping you to let you know there is an error. If you automatically accept any suggested correction, you are likely to make the same type of mistake that can result in using the wrong word in English.

Using the Organizer Window

The companion to the Xcode workspace window is the Organizer window, shown in Figure 1.42.

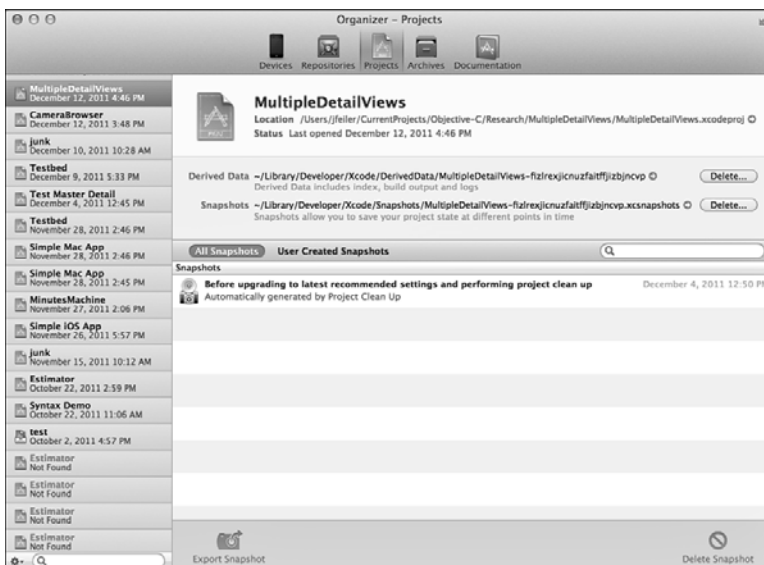


FIGURE 1.42
The Organizer window keeps track of files in repositories and archives, projects, devices, and documentation.

The five tabs at the top let you switch from one view to another:

- ▶ **Devices**—Primarily for iOS, this is a list of devices you have provisioned through Apple's developer program. This process is described on developer.apple.com. It is the process whereby you present your developer credentials to Apple and receive a digitally signed signature that lets your app run on specific devices that are listed here.
- ▶ **Repositories**—Xcode supports industry-standards Git and Subversion as source code repositories for version control. Both are widely used

open-source projects. Xcode puts a graphical user interface onto them. The functionality of both is the same in Xcode as it is in other environments.

- **Projects**—This tab lets you organize snapshots of your project created while working on your project. You can create them manually from File, Create Snapshot, but it is easier to have Xcode create them automatically at critical moments. As you can see in Figure 1.43, you can use File, Project Settings and the Snapshots tab in your workspace window to turn on these automatic snapshots and set the locations for their storage. (You can also set these locations in the Locations tab of Xcode preferences.) Snapshots require Git to be installed. (That is an option in the Xcode 4 install.)

FIGURE 1.43
Use snapshot
at critical
moments in
restructuring
your project.



- **Archives**—Archives can be used to create installable archives of your app for deployment. There is more information at developer.apple.com.
- **Documentation**—For many people, the most commonly used tab is for documentation. When you click on a link in Quick Help, the detailed documentation opens in the documentation tab of the Organizer window. You can use the jump bar at the top of the Organizer to select the appropriate area in which to search for your topic.

Summary

Xcode 4 is not just a cosmetic change to previous versions; it is a new development environment complete with a new compiler (LLVM) that includes an engine that runs in the background to catch basic errors as you (occasionally) make them. This hour helps you get ready to write code using the latest and greatest technologies for software development.

The workspace window gives you all of a project's data and controls in a single multi-paned window. You can control which panes are shown, and, to a certain extent, you can even rearrange their positions as they are shown. For most people, this is not a matter of setting up a preferred workspace and sticking with it: Depending on what you are doing, you often show and hide parts of the workspace window so you can focus on the task at hand.

Xcode 4 includes interfaces to source code management tools such as Git and Subversion (Git is preferred). As a result, you can manage your code—even on a multiperson project—and keep track of revisions. In addition, Xcode provides a snapshot feature that can capture your entire project at specific moments, such as when you ask and when you are about to perform project-wide automated changes.

Q&A

Q. *What is the best way to get started with Xcode?*

A. Use it. Open it and create a test project based on one of the built-in templates. Explore and experiment, and then throw it away. If you start working with it on a real project, your beginning mistakes will be around to haunt you for a long time.

Q. *What is the best way to handle the periodic updates to operating systems and SDKs?*

A. Registered developers are notified in advance of these updates; you can download the pre-release versions of both OSs and SDKs as well as new releases of Xcode. This enables you to test your apps with the new environmental software and prepare to use new features. Typically, you are warned not to use this software for production use. Apps developed with the new OS and SDK cannot be submitted to either App Store until a few weeks before the release of the software to the public. This process allows developers to get up to speed with the new technologies. The period of a few weeks before the public release of the new software allows the App Store to be stocked when the software is in final versions.

Workshop

Quiz

1. *How do you get a copy of Xcode?*
2. *If you are used to another development environment, can you use it to develop software for Mac OS X and iOS?*

Quiz Answers

1. Register at developer.apple.com. Various registration categories are designed for different types of developers. The paid levels of developer registration include technical support assistance (two incidents for the basic programs). There are also free registrations that provide no support but do allow you to download Xcode. You can also buy Xcode through the Mac App Store. It is currently free.
2. Only with great difficulty, and you will not be able to submit your apps to the App Store.

Activities

As you start to work with Xcode, take advantage of its productivity features such as code snippets. In particular, add your own snippets as you think of them. It generally is best not to sit down and make a list of snippets that you think you will need—that is often a waste of time. Instead, keep alert and, whenever you find yourself typing something that might be useful as a snippet, add it right then and there. These snippets are stored in your environment—not just in a single project.

Many people shy away from the debugging tools, but you will find that they can save you a large amount of time and effort. Some of them are for advanced developers but practice using breakpoints. This is a simple technology, and it is very easy to just click in the gutter of the editor to set a breakpoint. The debug area will let you examine local variables; it also will help you track the path of execution so you can see why a certain section of code is or is not being executed.

Index

A

- Abstract Entity entity setting
(Data Model inspector), 322
- abstractions, data models,
101-103
- Access the Persistent Store
Coordinator listing (4.6), 94-95
- Accessing the Fetched Results
Controller listing (4.8), 97-98
- Accessing the Managed Object
Model listing (4.7), 95
- Accessor attribute (declared
property), 72
- accessory view, 345
- ad hoc display order, table rows,
handling, 378-380
- adaptors, 156
- Add a Detail Disclosure
Accessory to Row listing (23.1),
414-415
- Add a New Field to insert
NewObject listing (12.3), 218
- Add buttons, inserting, 371
- Adopting the
UISplitViewControllerDelegate
Protocol listing (3.12), 79
- Advanced setting (Data Model
inspector), 325-327
- aggregate operators, 174
- ALL aggregate operator, 174
- ANY aggregate operator, 174
- AppDelegate.h for a Core Data
Project listing (4.3), 92
- Apple documentation, 73
- Apple's Xcode Quick Start
Guide*, 11
- applicationDocumentsDirectory
(iOS) listing (5.2), 114
- applicationFilesDirectory (Mac
OS) listing (5.1), 113
- apps
 - architectures, 154
 - building, 52-53
 - creating, 195-198
 - storyboards, 239-241
 - delegates, 293-299
 - document-based, 154
 - Mac OS, 305-311
 - iOS
 - creating, 53-56

apps

- exploring, 58-59
- integrating views and data, 147-151
- library/shoebox, 154, 291
- library/shoebox apps, creating, 292-305
- Mac
 - creating, 56-58
 - exploring, 58-59
- Master-Detail App, creating, 263-267
- navigation-based apps
 - finishing interface, 275-276
 - implementing saving, 267-272
- NSTableView, building, 366-372
- structures, 292
- universal, creating, 190-192, 279-281
- architectures, 153-154, 292
- Archives tab (Organizer window), 46
- areas, workspace window, 14
- Arranged setting (Data Model inspector), 327
- array controllers, 148
- array operators, 174
- arrays, predicates, 175-176
- assistant editing mode (Xcode), 26
- Assistant editor, 232-233
- Atomicity attribute (declared property), 72
- attribute settings, Data Model inspector, 324-325

- Attribute Type setting (Data Model inspector), 325
- attributes, 72, 87
 - data model, 216
 - declared properties, 72
 - displayOrder, 379-380
 - entities, adding to, 105-107, 123-125
 - renaming, 432-433
- Attributes inspector, 205
 - setting entity names, 368
- automatic lightweight migration, 423
 - data models, 432-434

B

- bars, workspace window, 14
- batteryLevel property (UIDevice), 190
- batteryMonitoringEnabled property (UIDevice), 190
- batteryState property (UIDevice), 190
- BEGINSWITH string, 174
- bidirectional relationships, 127
- binary data, entities, 106-107
- Binary Large Objects (BLOBs), 106
- bindings, 144, 148-149
 - examining, 150
 - NSTableView, 366
- Bindings inspector, 205
- BLOBs (Binary Large Objects), 106

- Boolean data, entities, 107
- breaking connections, 213-215
- breakpoint gutters, workspace window, 14
- breakpoint navigator, 24-25
- breakpoints, 24
 - debugger, 268-270
 - toggling, 25
- building data stacks, 91-96
- buttons, Add, inserting, 371

C

- C Programming Language, The*, 8
- canvas (Interface Builder), 197-205
- cardinality, 126, 327
 - relationships, 127
- cascade delete rule, 128
- Categories submenu (model editor files), 29
- cellForRowAtIndexPath listing (19.3), 352
- cells
 - table views, 345
 - tables
 - creating labels, 357
 - styled, 355-357
- Change setValue: forKey listing (13.4), 229
- Change the Attribute for the Sort Descriptor listing (13.2), 229
- Change the Entity for the Fetched Result Controller listing (13.1), 227

Change valueForKey in
configureCell listing (13.3), 229

changed views, 413

Character Large Objects
(CLOBs), 106

Class entity setting (Data Model
inspector), 322

Class from i.e
RootViewController.m listing
(3.13), 80

classes

NSSortDescriptor, 185
Objective-C, 66

Clauses, WHERE, 171, 173

CLOBs (Character Large Objects),
106

Cocoa

dictionaries, key-value pairs,
172-173
frameworks, 63-64

code

code snippet library, adding
to, 38-40

completing, 43-45

glue

Document.h, 396
MyDocument.m, 397-399
nib file, 399-401

Objective-C, 64-66

classes, 66
declarations, 82

declared properties,
68-73

delegates, 75-76, 81

instances, 66

messaging, 73-75

MVC (model/view/con-
troller) design pattern,
81-82

naming conventions,
74-75

object-oriented program-
ming, 66-68

objects, 66-68

protocols, 75-80

synthesizing properties,
70-72

Objective-C language, 63

saving, 284-286

code listings

Access the Persistent Store
Coordinator, 94-95

Accessing the Fetched
Results Controller, 97-98

Accessing the Managed
Object Model, 95

Add a Detail Disclosure
Accessory to the Row,
414-415

Add a New Field to
insertNewObject, 218

Adopting the
UISplitViewController
Delegate Protocol, 79

AppDelegate.h for a Core
Data Project, 92

applicationDocuments
Directory (iOS), 114

applicationFilesDirectory (Mac
OS), 113

cellForRowAtIndexPath, 352

Change setValue:
forKey, 229

Change the Attribute for the
Sort Descriptor, 229

Change the Entity for the
Fetched Result
Controller, 227

Change valueForKey in
configureCell, 229

Class from i.e
RootViewController.m, 80

configureView, 284

Create a Predicate with a
Format String, 184

Create a Predicate with a
Format String and Runtime
Data, 184

Creating a Fetch
Request, 160

Creating a Managed Object
Context, 159

Creating a Popover View
Controller, 417

Customer.h, 333

Customer.m, 334

Defining the Protocol, 77

didSelectRowAtIndexPath, 251

Executing a Fetch
Request, 161

Existing Private Declaration in
DetailViewController.m, 330

Getter for
managedObjectContext in
AppDelegate.h, 93

Getter for
numberFormatter, 330

Handle the Tap in the
Selected Row, 415

Handling the Move, 389

code listings

- Header for a Custom NSManagedObject Class, 384
- Header for a Document-based Mac OS App, 308
- Hello, World, 8
- Implementation for a Custom NSManagedObject Class, 385
- Implementation for a Document-based Mac OS App, 309-311
- Implementation of the Protocol with a Navigation Bar, 78
- Implementation of the Protocol with a Toolbar, 78
- Implementing the Mac OS App Delegate, 295-299
- insertNewObject As It Is in the Template, 216
- Interface for DetailViewController with Table View, 349
- iOS App Delegate Implementation, 301-305
- iOS Application Delegate, 300
- Legacy Class Declaration, 68-69
- Legacy Class Declaration with Accessors, 69
- Marking Protocol Methods Required or Optional, 77
- MasterViewController.h, 211
- Modern Class Declaration, 69
- Moving Related Objects into a Mutable Array, 388
- Moving the Top-Level Objects into a Mutable Array, 387
- MyDocument.h, 396
- MyDocument.m, 397-398
- numberOfRowsInSection, 351
- Opening a Persistent Store, 433-434
- Place.h, 88
- Place.m, 89
- prepareForSegue in MainViewController.m, 250
- Protocol Adoption with a Navigation Bar, 77
- Protocol Adoption with a Toolbar, 77
- saveNameData, 285
- Saving the Data, 390
- Set Section Header and Footer Titles, 354-355
- Set the New View Controller, 415
- setDetailItem, 276
- Setting Up the App Delegate, 294
- Setting Up the Fetch Request, 377-378
- Styling Cells, 356-357
- Swapping the View, 245
- Synthesize Directives to Match Listing 3.3, 70
- Synthesize the Core Data Stack Properties, 93
- Transforming an Image to and from NSData, 141
- Use a Predicate Template with Hard-coded Data, 183
- Use a Predicate Template with Runtime Data, 183
- Use More than One Section, 354
- Using a Private Variable in a Property, 71
- Using a Sort Descriptor, 186
- viewWillAppear, 273
- viewWillDisappear, 274
- code property (NSError), 404**
- code samples, 50-52**
- code snippet library, 38**
 - adding code to, 38-40
- columns, 87**
- comparison operators, predicates, 173-175**
- compatibility, data models versions, determining, 430-431**
- compound indexes, 323**
- configureView, 284**
- configureView listing (16.1), 284**
- connections**
 - creating, 213-215
 - trace, 149
- Connections inspector, 149, 205, 209-210**
 - connections, creating, 213-215
 - outlets, 210-212
 - referencing, 212-213
- CONTAINS string, 174**
- contexts, managed objects, 90-91, 148, 153, 158**
 - creating, 158-159
 - saving, 274
- continuum, migration, 423**

- control-drag, building interfaces, 232-236
 - controller concept (MVC (model/view/controller) design pattern), 82
 - controllers
 - array controllers, 148
 - dictionary controllers, 149
 - navigation, 151
 - object controllers, 148
 - page view, 151
 - split view, 151
 - tab bar, 151
 - table view, 151
 - terminology, 410
 - tree controllers, 149
 - user defaults controllers, 149
 - view, iOS, 148-151
 - converting dates to strings, 216
 - Core Data, 85**
 - documents, 291
 - examining at runtime, 90-96
 - origins, 85-87
 - UITableView, 357-359
 - user interface, 195
 - Core Data faulting, 155**
 - Core Data model editor, 86, 117-119**
 - Core Data Model editor**
 - data models
 - adding entities to, 119-123
 - adding relationships to, 126-127
 - styles, choosing, 125-126
 - “Core Data Programming Guide”, 403**
 - Core Data stack, implementing, 307-311
 - Count setting (Data Model inspector), 327
 - Counterparts submenu (model editor files), 29
 - Create a Predicate with a Format String and Runtime Data listing (10.4), 184
 - Create a Predicate with a Format String listing (10.3), 184
 - Create the Cell Labels, 358
 - Creating a Fetch Request listing (9.2), 160
 - Creating a Managed Object Context listing (9.1), 159
 - Creating a Popover View Controller listing (23.4), 417
 - Customer.h listing (18.3), 333
 - Customer.m listing (18.4), 334
- D**
- data
 - databases, adding, 161-162
 - flattening, 271-272
 - integrating
 - iOS, 151
 - Mac OS, 147-150
 - interfaces, entering into, 327-331
 - moving and saving, 273-274
 - normalizing, 106
 - sorting, sort descriptors, 185-186
 - data elements, IBOutlets, 215-216
 - data encapsulation, objects, 67
 - data fetching, 154
 - fetch requests, creating, 159-161
 - metrics, 156-158
 - paradigms, 155
 - performance, 156-158
 - representing results, 158
 - data fields, model, adding to, 217-221
 - Data Model inspector, 320-321**
 - Advanced setting, 325-327
 - Arranged setting, 327
 - Attribute setting, 325
 - attribute settings, 324-325
 - Count setting, 327
 - Default Value setting, 325
 - Delete Rule setting, 327
 - Destination setting, 326-327
 - entity settings, 321
 - Abstract Entity, 322
 - Class, 322
 - indexes, 323
 - Name, 321
 - Parent Entity, 323
 - Inverse setting, 326
 - Name setting, 324-326
 - Properties setting, 326
 - Property setting, 324
 - Regular Expression setting, 325

Data Model inspector

relationship settings,
325-327

Validation setting, 325

data models, 101

abstractions, 101-103

adjusting code, 226-229

attributes, 216

Core Data Model editor,
117-119

styles, 125-126

Core Data stack, 153

creating, 226-227, 426-427

Data Model inspector,
320-321

attribute settings,
324-325

entity settings, 321-323

relationship settings,
325-327

data quality rules, 318-319

deleting, 313

designing, 102-103

entities, 103-104

adding attributes to,
105-107

adding to, 119-123

binary data, 106-107

Boolean data, 107

dates, 106

linking with relationships,
107-108

external, 436

mapping models, 434-437

migration, 423-424

automatic lightweight
migration, 432-434

managing, 424-426

moving, 311-314

moving data into, 327-331

naming, 101-102

relational integrity rules,
318-319

relationships

adding to, 126-127,
129-131

cardinality, 127

delete rule, 128

rules, setting up, 320-327

validation rules, 317-319

versions, 426-430

creating, 426-430

determining compatibility,
430-431

forcing incompatibility, 432

data quality, 319

data quality rules, data model, 318-319

setting up, 320-327

data retrieval, predicates, 176

data stacks, 90-96

building, 91-96

CHANGE TO Core Data
stack, 153

data model, 153

initialization, 153

persistent stores, 153

data stores, 258

data types, choosing, 88

data updates, changing, 284-286

data validation, 319

free, 393-394

summarizing on Mac OS,
401-402

testing, 401-402

Mac OS, 394-402

managing, 393-394

programming, 402-406

rules

data model, 317-327

database management systems (DBMSs), 171

database manager, sorting

data, 186

databases

adding data, 161-162

Core Data faulting, 155

data retrieval, 154

fetch requests, 159-161

metrics, 156-158

paradigms, 155

performance, 156-158

representing results, 158

load-a-chunk design

pattern, 155

load-then-process design pat-
tern, 155

locating, 109-111

relational, 87

rules

cardinality, 127

delete, 128

schemas, 424

sorting data, 185-186

tables, 87

dates

converting to strings, 216

entities, 106

- DBMSs (database management systems), 171
- debug navigator, 23-24
- Debug pane, displaying, 270-272
- debugger, 267-268
 - breakpoints, 268-270
 - Debug pane, 270-272
- debugging connections, 213-215
- declarations, 82
- declarative programming paradigms, 9-10
- declared properties, 64, 441
 - attributes, 72
 - Objective-C, 68-73
- Default Value setting (Data Model inspector), 325
- Defining the Protocol listing (3.6), 77
- delegates, 293
 - apps, 295-299
 - Objective-C, 75-76, 81
- delete rule, relationships, 128
- Delete Rule setting (Data Model inspector), 327
- deleting
 - data models, 313
 - document types, 307
- deny delete rule, 128
- design patterns
 - Core Data faulting, 155
 - load-a-chunk, 155
 - load-then-process, 155
 - MVC (model/view/controller), 143-144
 - controlling data, 144
 - controlling views, 144-147
 - designing data models, 102-103
 - Destination setting (Data Model inspector), 326
 - detail disclosure accessories, rows, adding, 414-415
 - Detail views, swapping, 244-245
 - DetailViewController, 231, 266, 268
 - detaillItem instance variable, 272
 - outlets, 225-226
 - DetailViewController.m, 330
 - devices, iOS, swapping views, 241-243
 - Devices tab (Organizer window), 45
 - dictionaries, key-value pairs, 172-173
 - dictionary controllers, 149
 - didSelectRowAtIndexPath listing (14.3), 251
 - dismissing modal windows and sheets, 421
 - Disney, Walt, 246
 - display order, table rows, handling, 378-380
 - displayOrder attribute, 379-380, 387-390
 - document structure area, 199-201
 - objects, 204-205
 - placeholders, 201-204
 - document outline area (Xcode), 199
 - document types, 306
 - deleting, 307
 - document-based apps, 154
 - Mac OS, creating, 305-311
 - document-based Mac OS apps, creating, 292-299
 - Document.h, glue code, building in, 396
 - documentation, Apple, 73
 - Documentation tab (Organizer window), 46
 - documents, 110, 289-291
 - app structure, 292
 - Core Data, 291
 - tracking data in, 108-111
 - domain property (NSError), 404

E

 - editing data
 - navigation interfaces, 257-262
 - users, 409
 - editing interfaces, 409-412
 - communicating with users, 413-418
 - editing modes (Xcode), 25-30
 - editing preferences, 40-43
 - editing window (Xcode), 31
 - editing-in-place, 409-411
 - ENDSWITH string, 174
 - Enterprise Objects Framework (EOF), 85, 109, 156, 176
 - entires, 172
 - entities, 87
 - attributes, adding to, 123-125
 - data models, 103-104

entities

- adding attributes to, 105-107
- adding to, 119-123
- binary data, 106-107
- Boolean data, 107
- dates, 106
- linking with relationships, 107-108
- names, setting, 368
- NSManagedObject, subclasses, 331-334
- Place, 89
- relationships
 - moving, 389
 - rules, 126
- renaming, 432-433
- entity settings, Data Model inspector, 321**
 - Abstract Entity, 322
 - Class, 322
 - indexes, 323
 - Name, 321
 - Parent Entity, 323
- environments, multiuser, 312**
- EOF (Enterprise Objects Framework), 85**
- error messages, 413**
- Estimator interface, 342**
- Executing a Fetch Request listing (9.3), 161**
- Existing Private Declaration in DetailViewController.m listing (18.1), 330**
- expressions, regular, 319, 325**
- external data models, 436**
- external objects, iOS, 151**

F

- faulting, 155**
- fetch request controllers, 96**
- fetch requests, 96-98**
 - creating, 159-161, 178-183
 - setting up, 377
- fetches, 133**
- fetching data, 154**
 - metrics, 156-158
 - paradigms, 155
 - performance, 156-158
 - representing results, 158
- fields, 87**
 - IBOutlets, adding, 230-231
 - removing, table view, 345-349
 - second interface, adding to, 281-284
- file inspector, 32**
- file templates library, 35, 37**
- File's Owner object, 201-202**
 - outlets, 210-211
- FileMaker Pro, 157**
- FileMaker Server, 157**
- files**
 - declarations, 82
 - identifying, 52-53
 - rearranging, 120
 - renaming, 120
 - semi-hidden, 110-111
 - creating, 111-115
 - iOS, 114
 - Mac OS X, 110-115
 - tracking data in, 108-111
- filter bar, workspace window, 14**

- First Responder, 203, 212**
- Fix It, 40, 43-45**
- flattening data, 271-272**
- Focus ribbon, workspace window, 14**
- folders, Inside Applications, 193**
- footers, tables, setting, 354-355**
- format strings, predicates, 177, 184**
- formatters, 216, 329**
 - type conflict issue, solving, 329-331
- frameworks, Cocoa, 63-64**
- free validation, 393-394**
 - summarizing on Mac OS, 401-402
 - testing, 401-402
- full-screen view (Interface Builder), 197**

G

- generatesDeviceOrientation Notifications property (UIDevice), 190**
- Getter for managedObjectContext in AppDelegate.h listing (4.5), 93**
- Getter for numberFormatter listing (18.2), 330**
- Git repository, 55**
- Git source code repository, 49, 57**
- glue code**
 - Document.h, building in, 396

MyDocument.m
 building in, 397-399
 nib file, 399-401

Go menu, Libabry folder, adding to, 193

Gone with the Wind, 246

groups, rearranging, 120

H

Handle the Tap in the Selected Row listing (23.2), 415

Handling the Move listing (21.6), 389

Header for a Custom NSManagedObject Class listing (21.2), 384

Header for a Document-based Mac OS App listing (17.5), 308

headers, tables, setting, 354-355

Hello, World listing, 8

hidden primary keys, 162

I

IBOutlets
 data elements, 215-216
 new fields, adding, 230-231

iCloud, 107

identifiers, predicates, 173

Identity inspector, 34, 205

imperative programming paradigms, 9-10

Implementation for a Custom NSManagedObject Class listing (21.3), 385

Implementation for a Document-based Mac OS App listing (17.6), 309-311

Implementation of the Protocol with a Navigation Bar listing (3.11), 78

Implementation of the Protocol with a Toolbar listing (3.10), 78

Implementing the Mac OS App Delegate listing (17.2), 295-299

IN aggregate operator, 174

incompatibility, data models, forcing, 432

indexes, 323

insertNewObject As It Is in the Template listing, 216

Inside Applications folder, 193

inspectors, 31-34, 205

 Attributes, 205

 Bindings, 205

 Connections, 205, 209-210

 creating connections, 213-215

 outlets, 210-213

 file, 32

 Identity, 34, 205

 Size, 205

 View Effects, 205

instances

 adding, 259

 Objective-C, 66

Interface Builder editor,
 document structure area, 199-201

objects, 204-205

placeholders, 201-204

inter-property validation, 405-406

Interface Builder, 7

 Connections inspector, 209-210

 creating connections, 213-215

 outlets, 210-212

 referencing outlets, 212-213

 storyboards, 442

Interface Builder editor, 189-190, 198-200, 344

 apps, creating, 195-198

 canvas, 197-205

 full-screen view, 197

 iOS apps, locating sandbox, 192-194

 macros, 230-231

 Project navigator, 198

 storyboards, 192

 table views, 199-200

 type qualifiers, 230-231

 universal apps, creating, 190-191

Interface for DetailViewController with Table View listing (19.1), 349

interfaces

 building, control-drag, 232-236

 cleaning up, 275-276

 comparing, 339-344

 editing interfaces, 409-412

 communicating with users, 413-418

interfaces

- entering data into, 327-331
 - Estimator, 342
 - integrating views and data
 - iOS, 151
 - Mac OS, 147-150
 - iOS features, 165-167
 - iPhone, 343
 - Mac OS features, 163-165
 - navigation-based apps,
 - finishing, 275-276
 - optimizing, 162-167
 - removing, table view, 345-349
 - second, adding fields to,
 - 281-284
 - text fields, adding to,
 - 217-221
 - initialization, Core Data stack, 153**
 - Inverse setting (Data Model inspector), 326**
 - iOS**
 - apps
 - creating, 53-56
 - exploring, 58-59
 - integrating views and data, 151
 - locating sandbox,
 - 192-194
 - structure, 292
 - development process, 258
 - devices, swapping views,
 - 241-243
 - interfaces, 339-344
 - features, 165-167
 - library/shoebox apps,
 - creating, 299-305
 - popovers, 416-418
 - semi-hidden files, 114
 - settings, 339-344
 - swapping views, 413-415
 - table rows
 - allowing movement,
 - 380-382
 - moving, 382-390
 - ordering, 375-380
 - table views, comparing,
 - 337-338
 - UITableView, 337-345
 - accessory view, 345
 - cells, 345
 - implementing methods,
 - 350-357
 - interface removal, 345-349
 - removing fields, 345-349
 - sections, 345
 - using with Core Data,
 - 357-359
 - using without Core Data,
 - 344-357
 - user interaction, 338-339
 - validation, programming,
 - 402-406
 - versions, 190
 - iOS App Delegate Implementation listing (17.4), 301-305**
 - iOS Application Delegate listing (17.3), 300**
 - iPad, 279**
 - split view controllers,
 - 250, 311
 - storyboards, 247-248
 - universal apps, creating,
 - 279-281
 - iPhone**
 - interface, 343
 - storyboards, 246-247
 - iPhone apps**
 - Master-Detail apps, creating,
 - 263-267
 - navigation-based apps
 - adding managed objects,
 - 272-273
 - finishing interfaces,
 - 275-276
 - implementing saving,
 - 267-272
 - issue navigator, 23**
- ## J
- Jobs, Steve, 363**
 - join tables, 127**
 - jump bars (Xcode), 27, 294-295, 301**
- ## K
- Kernighan, Brian, 8**
 - key-value coding (KVC), 144**
 - key-value observing (KVO), 144**
 - key-value pairs, dictionaries, 172-173**
 - key-value validation, 403-404**
 - KVC (key-value coding), 144**
 - KVO (key-value observing), 144**

L

labels, cells, creating, 357

launching Xcode, 12

legacy class declaration, 68

Legacy Class Declaration listing
(3.1), 68-69

Legacy Class Declaration with
Accessors listing (3.2), 69

legacy versions, Objective-C, 64

libraries, 35-38

adding code snippets, 38-40

file templates, 35-37

Media, 40

Object, 40

SQLite, 156

Library folder, Go menu, adding
to, 193

library/shoebox apps, 154, 291

iOS, creating, 299-305

Mac OS, creating, 292-299

lightweight migration, 423

automatic, data models,
432-434

LIKE string, 174

linking entities with relationships,
107-108

list elements, moving, 389

listings

Access the Persistent Store
Coordinator, 94-95

Accessing the Fetched
Results Controller,
97-98

Accessing the Managed
Object Model, 95

Add a Detail Disclosure
Accessory to Row, 414-415

Add a New Field to
insertNewObject, 218

Adopting the
UISplitViewControllerDelegate
Protocol, 79

AppDelegate.h for a Core
Data Project, 92

applicationDocumentsDirectory
(iOS), 114

applicationFilesDirectory (Mac
OS), 113

cellForRowAtIndexPath, 352

Change setValue: forKey, 229

Change the Attribute for the
Sort Descriptor, 229

Change the Entity for the
Fetched Result
Controller, 227

Change valueForKey in
configureCell, 229

Class from i.e
RootViewController.m, 80

configureView, 284

Create a Predicate with a
Format String, 184

Create a Predicate with a
Format String and Runtime
Data, 184

Creating a Fetch Request, 160

Creating a Managed Object
Context, 159

Creating a Popover View
Controller, 417

Customer.h, 333

Customer.m, 334

Defining the Protocol, 77

didSelectRowAtIndexPath, 251

Executing a Fetch
Request, 161

Existing Private Declaration in
DetailViewController.m, 330

Getter for
managedObjectContext in
AppDelegate.h, 93

Getter for numberFormatter,
330

Handle the Tap in the
Selected Row, 415

Handling the Move, 389

Header for a Custom
NSManagedObject
Class, 384

Header for a Document-based
Mac OS App, 308

Hello, World, 8

Implementation for a Custom
NSManagedObject
Class, 385

Implementation for a
Document-based Mac OS
App, 309-311

Implementation of the
Protocol with a Navigation
Bar, 78

Implementation of the
Protocol with a Toolbar, 78

Implementing the Mac OS
App Delegate, 295-299

insertNewObject As It Is in
the Template, 216

Interface for
DetailViewController with
Table View, 349

listings

- iOS App Delegate
 - Implementation, 301-305
- iOS Application Delegate, 300
- Legacy Class Declaration, 68-69
- Legacy Class Declaration with Accessors, 69
- Marking Protocol Methods
 - Required or Optional, 77
- MasterViewController.h, 211
- Modern Class Declaration, 69
- Moving Related Objects into a Mutable Array, 388
- Moving the Top-Level Objects into a Mutable Array, 387
- MyDocument.h, 396
- MyDocument.m, 397-398
- numberOfRowsInSection, 351
- Opening a Persistent Store, 433-434
- Place.h, 88
- Place.m, 89
- prepareForSegue in
 - MainViewController.m, 250
- Protocol Adoption with a Navigation Bar, 77
- Protocol Adoption with a Toolbar, 77
- saveNameData, 285
- Saving the Data, 390
- Set Section Header and Footer Titles, 354-355
- Set the New View
 - Controller, 415
- setDetailItem, 276
- Setting Up the App
 - Delegate, 294

- Setting Up the Fetch Request, 377-378
- Styling Cells, 356-357
- Swapping the View, 245
- Synthesize Directives to
 - Match Listing 3.3, 70
- Synthesize the Core Data
 - Stack Properties, 93
- Transforming an Image to and from NSData, 141
- Use a Predicate Template
 - with Hard-coded Data, 183
- Use a Predicate Template
 - with Runtime Data, 183
- Use More than One
 - Section, 354
- Using a Private Variable in a
 - Property, 71
- Using a Sort Descriptor, 186
- viewWillAppear, 273
- viewWillDisappear, 274
- literals, predicates, 173**
- load-a-chunk design pattern, 155**
- load-then-process design pattern, 155**
- loading mutable arrays, 386-388**
- localizedModel property (UIDevice), 190**
- log navigator, 25**
- logical operators, predicates, 171-173, 176-177**
 - arrays, 175-176
 - comparison operators, 173-175
 - constructing, 177-183
 - format strings, 177, 184
 - identifiers, 173

- literals, 173
- syntax, 173-175

M**Mac OS**

- app structure, 292
- apps
 - creating, 56-58
 - exploring, 58-59
 - integrating views and data, 147-150
- development process, 258
- document-based applications,
 - creating, 305-311
- free validation, summarizing, 401-402
- interfaces, 339-344
 - features, 163-165
- library/shoebox apps,
 - creating, 292-299
- modal windows, 419-421
- NSTableView
 - building app, 366-372
 - new features, 363-365
- sheets, 419-421
- system preferences, 339-344
- table views, comparing, 337-338
- user interaction, 338-339
- validation, 394-402
 - programming, 402-406
- versions, 190
- Mac OS X, semi-hidden files, 110-115**

macros, Interface Builder editor,
230-231

managed objects, 91, 133

- adding, 272-273
- context, saving, 274
- contexts, 90-91, 148,
153, 158
- creating, 158-159
- NSManagedObject
 - creating subclasses of,
331-334
 - overriding, 134-140
 - transformations, 136,
140-141
 - validation, 136

managedObjectContext, 400

many-to-many relationships, 127

mapping

- migration, 424
- models, 434-437

Marking Protocol Methods

Required or Optional listing
(3.7), 77

master views, 258

Master-Detail App, creating,
263-267

Master-Detail Application tem-
plate, 242, 343-344, 409-410

- repurposing, 223-230

Master-Detail template,
166-167, 263

MasterViewController, 97

- outlets, 225-226

MasterViewController.h listing
(12.1), 211

MATCHES string, 174

Media library, 40

messaging, Objective-C, 73-75

methods

- NSDictionary, 172
- protocols, 442
- saveAction, 293
- saveNameData, 285
- table view, implementing,
350-357
- viewWillAppear, 269, 273
- viewWillDisappear, 269
- windowWillReturnUndo
Manager, 293

metrics, data retrieval,
156-158

migration, 423-424

- continuum, 423
- data models
 - automatic lightweight
migration, 432-434
 - managing, 424-426
- lightweight, 423
- mapping, 424

modal windows, 419-421

model concept (MVC
(model/view/controller) design
pattern), 82

model property (UIDevice), 190

model/view/controller (MVC)
design pattern. See MVC
(model/view/controller) design
pattern

models, data fields, adding to,
217-221

Modern Class Declaration listing
(3.3), 69

movement, table rows, allowing,
380-382

moving

- data, 273-274
- table rows, 382-390

Moving Related Objects into
a Mutable Array listing
(21.5), 388

Moving the Top-Level Objects
into a Mutable Array listing
(21.4), 387

multitaskingSupported property
(UIDevice), 190

multiuser environments, 312

mutable arrays, loading, 386-388

MVC (model/view/controller)
design pattern, 81-82, 143-144

- controlling data, 144
- controlling views, 144-147

MyDocument.h listing (22.1), 396

MyDocument.m, glue code,
building in, 397-399

MyDocument.m listing (22.2),
397-398

N

Name attribute setting (Data
Model inspector), 324

Name entity setting (Data Model
inspector), 321

name property (UIDevice), 190

Name relationship setting (Data
Model inspector), 326

names, entities, setting, 368

naming data models

naming data models, 101-102
 naming conventions, Objective-C, 74-75
 navigation bars, 241, 259, 271
 navigation controllers, 151
 navigation interfaces, 257-262
 navigation-based apps
 implementing saving, 267-272
 interface, finishing, 275-276
 managed objects, adding, 272-273
 navigator pane (Xcode), 15-25
 navigators
 breakpoint, 24-25
 debug, 23-24
 issue, 23
 log, 25
 project, 16-20
 search, 21-22
 symbol, 20-21
 NeXT, 85, 290
 NeXTSTEP, 7
 nib file, glue code, building in, 399-401
 no action delete rule, 128
 non-unique user identifiers, 162
 NONE aggregate operator, 174
 normalizing data, 106
 NSApplicationDelegate protocol, 300
 NSDictionary method, 172
 NSError, 404-405
 NSFormatter, 329
 NSKeyValueCoding protocol, 403-404

NSManagedObject, 133, 382-388
 creating override, 383
 creating subclasses, 331-334
 overriding, 134-140
 subclasses, matching, 140
 transformations, 136, 140-141
 using directly, 134
 validation, 136
 NSManagedObjectContext, 91
 NSPersistentDocument, 305
 NSPersistentStore, 91
 NSSortDescriptor class, 185
 NSTableView
 apps, building, 366-372
 bindings, 366
 new features, 363-365
 NSWindowDelegate protocol, 293
 nullify delete rule, 128
 numberFormatter, 330
 numberOfRowsInSection listing (19.2), 351

O

object controllers, 148
 Object library, 40
 Object library (iOS), 151
 object stores
 persistent, 90
 object-oriented databases, 86
 object-oriented programming
 Objective-C
 classes, 66

 instances, 66
 objects, 66-68

object-oriented programming (OOP), 10-11

Objective-C, 64-66

 classes, 66
 declarations, 82
 declared properties, 68-73
 delegates, 75-76, 81
 instances, 66
 legacy versions, 64
 messaging, 73-75
 MVC (model/view/controller)
 design pattern, 81-82
 naming conventions, 74-75
 object-oriented programming, 66-68
 objects, 66-67
 purposes, 67-68
 properties, synthesizing properties, 70-72
 protocols, 75-80

Objective-C language, 63

object-oriented programming, Objective-C, 66-68

objects

 data encapsulation, 67
 document structure area, 204-205
 external, iOS, 151
 File's Owner, 201-202
 iOS, 151
 Mac OS, 148
 managed, 91
 adding, 272-273

- contexts, 90-91, 148, 153, 158-159
 - saving context, 274
- managed objects,
 - NSManagedObject, 134-141
- Objective-C, 66-68
- persistent object stores, 91
- placeholders, 201-204
- receiving and sending messages, 67
- runtime, 153
- state, 67
- one-to-many relationships, 127
- OOP (object-oriented programming), 10-11
- opening persistent stores, 433-434
- Opening a Persistent Store listing (24.1), 433-434
- operating systems, versions, 190
- operators
 - aggregate, 174
 - array, 174
 - comparison, predicates, 173-175
 - logical, predicates, 171-183
- optimizing interfaces, 162-167
- ordered relationships, 442
- ordering table rows, 375-380
- Organizer window (Xcode), 45-46
- orientation property (UIDevice), 190
- outlets, 210-212
 - DetailViewController, 225-226
 - File's Owner, 210-211

- IBOutlets, adding fields, 230-231
- MasterViewController, 225-226
- referencing, 210, 212-213
- overriding NSManagedObject, 134-140

P

- page view controllers, 151
- panes, workspace window, 14
- Parent Entity entity setting (Data Model inspector), 323
- performance, data retrieval, 156-158
- persistent object stores, 90-91
- persistent stores, 86, 108, 133
 - Core Data stack, 153
 - opening, 433-434
 - types, 108-109
- Place entity, 89
- Place.h listing (4.1), 88
- Place.m listing (4.2), 89
- placeholders, 201-204
 - First Responder, 203
- Plural/Cardinality setting (Data Model inspector), 327
- pop-up menu lists, organizing, 27-28
- popovers, iOS, 416-418
- predicates, 171-173, 176-177
 - arrays, 175-176
 - comparison operators, 173-175
 - constructing, 177-183

- data retrieval, 176
- format strings, 177, 184
- identifiers, 173
- literals, 173
- syntax, 173-175
- templates, 177
 - hard-coded data, 182-183
 - runtime data, 183

- prepareForSegue, 250
- prepareForSegue in MainViewController.m listing (14.2), 250
- primary keys, hidden, 162
- programming validation, 402-406
- programming languages. *See* Objective-C
- Project Builder, 7, 189
- project navigator, 16-20
- Project navigator (Interface Builder), 198
- projects
 - building, 52-53
 - creating, 195-198
 - storyboards, 239-241
 - identifying, 52-53
 - iOS
 - creating, 53-56
 - exploring, 58-59
 - iOS library/shoebox-based apps, creating, 299-305
 - Mac
 - creating, 56-58
 - exploring, 58-59
 - Mac OS document-based apps, creating, 305-311

projects

- Mac OS library/shoebox-based apps, creating, 292-299
- Master-Detail App, creating, 263-267
- moving data models between, 312-314
- renaming, 120
- storyboards, setting, 251-252
- Projects tab (Organizer window), 46**
- properties**
 - declared, 441
 - declared properties, 64
 - attributes, 72
 - Objective-C, 68-73
 - synthesizing, 70-72
 - UIDevice, 190-191
- Properties setting (Data Model inspector), 326**
- Property attribute setting (Data Model inspector), 324**
- Protocol Adoption with a Navigation Bar listing (3.9), 77**
- Protocol Adoption with a Toolbar listing (3.8), 77**
- protocols**
 - methods, 442
 - Objective-C, 75-80
- Protocols submenu (model editor files), 29**
- proximityMonitoringEnabled property (UIDevice), 190**
- proximityState property (UIDevice), 190**
- proxy objects, 201-204**

Q

- quality edits, 319, 405-406**
- Quick Help, 33**
- records (tables), 87**
- referencing outlets, 210-213**
- referential integrity, preserving, 318**
- Regular Expression setting (Data Model inspector), 325**
- regular expressions, 319, 325**
- relational databases, 87**
- relational integrity, 128**
- relational integrity rules, data model, 318-319**
 - setting up, 320-327
- relationship entities, moving, 389**
- relationship settings, Data Model inspector, 325-327**
- relationships**
 - bidirectional, 127
 - data models
 - adding to, 126-131
 - cardinality, 127
 - delete rule, 128
 - entities
 - linking with, 107-108
 - rules, 126
 - many-to-many, 127
 - one-to-many, 127
 - ordered, 442
- renaming attributes entities, 432-433**
- renaming project files, 120**
- Repositories tab (Organizer window), 45**

- repurposing templates, 223-230**

- requests, fetch, 96-98**

- retrieving data, 154**

- metrics, 156-158

- paradigms, 155

- performance, 156-158

- Ritchie, Dennis, 8**

- RootViewController, 79**

- rows**

- detail disclosure accessories, adding, 414-415

- tables

- allowing movement, 380-382

- moving, 382-390

- ordering, 375-380

- taps, handling, 415

- rows (tables), 87**

- rules**

- data model, 318-319

- setting up, 320-327

- validation rules, 317-319

- runtime, Core Data, examining, 90-96**

- runtime objects, 153**

S

- sample code, 50-52**

- sandboxes, iOS apps, locating, 192-194**

- saveAction method, 293**

- saveNameData listing (16.2), 285**

- saveNameData method, 285**

systemVersion property (UIDevice)

- saving
 - code, 284-286
 - data, 273-274
 - managed object context, 274
 - navigation-based apps, implementing, 267-272
- Saving the Data listing (21.7), 390
- scenes, storyboards, 246
- schemas, databases, 424
- Seagull, The, 246
- search navigator, 21-22
- second interface
 - fields, adding to, 281-284
 - implementing, 281
- sections, table views, 345
- segues, storyboards, 246
- SELECT statement, 171
- semi-hidden files, 110-111
 - creating, 111-115
 - iOS, 114
 - Mac OS X, 110-115
- Set Section Header and Footer Titles listing (19.5), 354-355
- Set the New View Controller listing (23.3), 415
- setDetailItem listing (15.3), 276
- Setter attribute (declared property), 72
- Setting Up the App Delegate listing (17.1), 294
- Setting Up the Fetch Request listing (21.1), 377-378
- settings, iOS, 339-344
- sheets, 161
 - creating, 419-420
 - dismissing, 421
 - Mac OS, 419-421
- Siblings submenu (model editor files), 29
- simulator, iOS app sandboxes, locating, 192-194
- Size inspector, 205
- SOME aggregate operator, 174
- sort descriptors, 185-186
- split view controller, iPad, 250
- split view controllers, 151
- split view controllers (iPad), 311
- split views, 271-272
- SQLite, 90, 96
 - document types, 306
 - libraries, 156
- standard editing mode (Xcode), 26
- Stanislavski, Constantin, 246
- state, objects, 67
- statements, SELECT, 171
- storyboards, 87, 146, 192, 239-241, 246-251, 442
 - creating, 251-253
 - iPad, 247-248
 - iPhone, 246-247
 - scenes, 246
 - setting, 251-252
 - view controllers, adding and deleting, 252-253
- storyboards, segues, 246
- strings
 - BEGINSWITH, 174
 - CONTAINS, 174
 - converting to dates, 216
 - ENDSWITH, 174
 - format, predicates, 177, 184
 - LIKE, 174
 - MATCHES, 174
- structures, apps, 292
- styled cells, tables, creating, 355-357
- Styling Cells listing (19.6), 356-357
- subclasses, NSObject
 - creating from, 331-334
 - matching, 140
- Subclasses submenu (model editor files), 29
- summarizing free validation, Mac OS, 401-402
- Superclasses submenu (model editor files), 29
- Swapping the View listing (14.1), 245
- swapping views, 248-251
 - Detail views, 244-245
 - iOS, 413-415
 - devices, 241-243
- symbol navigator, 20-21
- syntax, predicates, 173-175
- Synthesize Directives to Match Listing 3.3 listing (3.4), 70
- Synthesize the Core Data Stack Properties listing (4.4), 93
- synthesizing properties, 70-72
- system preferences, Mac OS, 339-344
- systemVersion property (UIDevice), 190

tab bar controllers

T

tab bar controllers, 151

table view controllers (iOS), 151

table views, 345

accessory view, 345

adding, 369

cells, 345

fields, removing, 345-349

interface, removing, 345-349

methods, implementing,
350-357

sections, 345

table views (Interface Builder
editor), 199-200

tables, 87

cells

creating labels, 357

styled, 355-357

footer titles, setting, 354-355

header titles, setting,
354-355

multiple sections, 354

rows

allowing movement,
380-382

moving, 382-390

ordering, 375-380

templates, 52

Master-Detail Application,

166-167, 242, 263,

343-344, 409-410

predicates, 177

hard-coded data, 182-183

runtime data, 183

repurposing, 223-230

testing free validation, 401-402

text editor (Xcode), 40-45

code completion, 43-45

editing preferences, setting,
40-43

Fix It, 40, 43-45

text fields, interfaces, adding to,
217-221

"Three Little Pigs", 246

trace connections, 149

transformations,
NSManagedObject, 136,
140-141Transforming an Image to and
from NSData listing (7.1), 141

tree controllers, 149

type conflict issue, 328-329

solving, formatters, 329-331

type qualifiers, Interface Builder
editor, 230-231**U**UIApplicationDelegate
protocol, 300

UIDevice, properties, 190-191

UIResponder, 300

UISplitViewControllerDelegate, 79

UITableView

accessory view, 345

cells, 345

fields, removing, 345-349

interface, removing,
345-349

iOS, 337-345

using with Core Data,
357-359using without Core Data,
344-357methods, implementing,
350-357

sections, 345

UIUserInterfaceIdiom, 231

unique user-visible identifiers,
generating, 162universal apps, creating,
190-192, 279-281Use a Predicate Template with
Hard-coded Data listing
(10.1), 183Use a Predicate Template with
Runtime Data listing
(10.2), 183Use More than One Section
listing (19.4), 354

user defaults controllers, 149

user interaction, 338-339

user interface, Core Data, 195

user-visible identifiers,
generating, 162

userInfo property (NSError), 405

userInterfaceIdiom property
(UIDevice), 190

users

communicating with, 413-418
editing data, 409Using a Private Variable in a
Property listing (3.5), 71Using a Sort Descriptor listing
(10.5), 186

utilities

inspectors, 31-34

- libraries, 35-38
 - code snippet, 38-40
 - file templates, 35, 37

V

validation

- free, 393-394
 - summarizing on Mac OS, 401-402
 - testing, 401-402
- inter-property, 405-406
- key-value, 403-404
- Mac OS, 394-402
- managing, 393-394
- NSManagedObject, 136
- programming, 402-406

validation rules, data model, 317-319

- setting up, 320-327

Validation setting (Data Model inspector), 325

validity edits, 319

valueForKey, 134-136

version editing mode (Xcode), 26

versions, data models, 426-430

- creating, 426-430
- determining compatibility, 430-431
- forcing incompatibility, 432

view concept (MVC

(model/view/controller) design pattern), 82

view controllers

- creating, 244

- iOS, 151
- Mac OS, 148
- popover, 417
- setting, 415
- storyboards, adding and deleting, 252-253

View Effects inspector, 205

View menu commands, Welcome to Xcode, 50

views

- changed, 413
- controlling, 144-147
- Detail, swapping, 244-245
- integrating
 - iOS, 151
 - Mac OS, 147-150
- swapping, 248-251
 - iOS, 413-415
 - iOS devices, 241-243

viewWillAppear, 284

viewWillAppear listing (15.1), 273

viewWillAppear method, 269, 273

viewWillDisappear method, 284-285

viewWillDisappear listing (15.2), 274

viewWillDisappear method, 269

viewWillDisappearAndBeSaved, 284

W

WebObjects, 156

Welcome to Xcode command, 50

WHERE clauses, 171-173

windows (modal)

- creating, 421
- dismissing, 421
- Mac OS, 419-421

windowWillReturnUndoManager method, 293

workspace window (Xcode), 13-15

- areas, 14
- bars, 14
- breakpoint gutters, 14
- filter bar, 14
- Focus ribbon, 14
- navigator pane, 15-25
- panes, 14

Worldwide Developers Conference, 64

X

xcdatamodeld files, 313

Xcode, 8, 13, 49-50

- automatic installation, 12
- code samples, 50-52
- control-drag, building interfaces, 232-236
- Core Data model editor, 86
- declarative programming paradigms, 9-10
- document structure area, 199
- editing modes, 25-30
- editing window, 31
- fetch requests, creating, 178-183

Xcode

- files, identifying, 52-53
- imperative programming
 - paradigms, 9-10
- jump bar, 294-295, 301
- launching, 12
- Master-Detail template, 263
- navigator pane, 15-25
 - breakpoint navigator,
 - 24-25
 - debug navigator, 23-24
 - issue navigator, 23
 - log navigator, 25
 - project navigator, 16-20
 - search navigator, 21-22
 - symbol navigator,
 - 20-21
- organization tools, 28-29
- Organizer window, 45-46
- predicates, constructing,
 - 177-183
- projects
 - building, 52-53
 - identifying, 52-53
 - iOS, 53-56, 58-59
 - Mac, 56-59
- storyboards, 192
- templates, 52
- text editor, 40-45
 - code completion, 43-45
 - Fix It, 40, 43-45
 - setting editing
 - preferences, 40-43
- workspace window, 13-15

Xcode 4, 7