Sams Teach Yourself

Xcode® 4

in 24 Hours
# Contents at a Glance

<table>
<thead>
<tr>
<th>Hour</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>HOUR 1</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Xcode 4</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>Just Enough Objective-C and Cocoa</td>
<td>23</td>
</tr>
<tr>
<td>4</td>
<td>Understanding the MVC Design Pattern</td>
<td>57</td>
</tr>
<tr>
<td>5</td>
<td>Using Xcode Templates to Create Projects</td>
<td>71</td>
</tr>
<tr>
<td>6</td>
<td>Managing Projects and Resources</td>
<td>95</td>
</tr>
<tr>
<td>7</td>
<td>Using the Xcode Code Source Editor</td>
<td>117</td>
</tr>
<tr>
<td>8</td>
<td>Working with the Xcode 4 Documentation</td>
<td>145</td>
</tr>
<tr>
<td>9</td>
<td>Creating User Interfaces</td>
<td>165</td>
</tr>
<tr>
<td>10</td>
<td>Connecting a GUI to Code</td>
<td>195</td>
</tr>
<tr>
<td>11</td>
<td>Creating iOS Application Workflows with Storyboarding</td>
<td>215</td>
</tr>
<tr>
<td>12</td>
<td>Building and Executing Applications</td>
<td>251</td>
</tr>
<tr>
<td>13</td>
<td>Using Source Control</td>
<td>279</td>
</tr>
<tr>
<td>14</td>
<td>Xcode-Supported Languages</td>
<td>311</td>
</tr>
<tr>
<td>15</td>
<td>Planning for Re-use: Frameworks and Libraries</td>
<td>329</td>
</tr>
<tr>
<td>16</td>
<td>Building an iOS Application</td>
<td>355</td>
</tr>
<tr>
<td>17</td>
<td>Building an OS X Application</td>
<td>399</td>
</tr>
<tr>
<td>18</td>
<td>Attaching Big Data: Using Core Data in Your Applications</td>
<td>435</td>
</tr>
<tr>
<td>19</td>
<td>Test Early, Test Often</td>
<td>465</td>
</tr>
<tr>
<td>20</td>
<td>Getting the Bugs Out</td>
<td>487</td>
</tr>
<tr>
<td>21</td>
<td>Keeping things Organized: Shared Workspaces</td>
<td>509</td>
</tr>
<tr>
<td>22</td>
<td>Advanced: Analyzing Code with Instruments</td>
<td>533</td>
</tr>
<tr>
<td>23</td>
<td>Managing and Provisioning iOS Devices</td>
<td>549</td>
</tr>
<tr>
<td>24</td>
<td>Distributing Your Applications</td>
<td>567</td>
</tr>
<tr>
<td>25</td>
<td>Xcode CLI Utilities</td>
<td>587</td>
</tr>
<tr>
<td>Index</td>
<td>603</td>
<td></td>
</tr>
</tbody>
</table>
# Table of Contents

## Introduction

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
</tbody>
</table>

## HOUR 1: Xcode 4

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Welcome to Xcode</td>
<td>3</td>
</tr>
<tr>
<td>The Apple Developer Programs</td>
<td>8</td>
</tr>
<tr>
<td>Installing the Xcode Developer Tools</td>
<td>11</td>
</tr>
<tr>
<td>The Nickel Tour</td>
<td>13</td>
</tr>
<tr>
<td>Preparing Your iOS Device (Optional)</td>
<td>17</td>
</tr>
<tr>
<td>Summary</td>
<td>20</td>
</tr>
<tr>
<td>Q&amp;A</td>
<td>20</td>
</tr>
<tr>
<td>Workshop</td>
<td>21</td>
</tr>
</tbody>
</table>

## HOUR 2: Just Enough Objective-C and Cocoa

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object-Oriented Programming and Objective-C</td>
<td>23</td>
</tr>
<tr>
<td>What Is Objective-C?</td>
<td>25</td>
</tr>
<tr>
<td>Objective-C Programming Basics</td>
<td>38</td>
</tr>
<tr>
<td>Memory Management and Automatic Reference Counting</td>
<td>48</td>
</tr>
<tr>
<td>What Is Cocoa?</td>
<td>50</td>
</tr>
<tr>
<td>Cocoa Versus Cocoa Touch</td>
<td>51</td>
</tr>
<tr>
<td>Summary</td>
<td>54</td>
</tr>
<tr>
<td>Q&amp;A</td>
<td>55</td>
</tr>
<tr>
<td>Workshop</td>
<td>55</td>
</tr>
</tbody>
</table>

## HOUR 3: Understanding the MVC Design Pattern

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development, Design Patterns, and MVC</td>
<td>57</td>
</tr>
<tr>
<td>How Xcode Implements MVC</td>
<td>60</td>
</tr>
<tr>
<td>An MVC Walkthrough</td>
<td>64</td>
</tr>
<tr>
<td>Summary</td>
<td>68</td>
</tr>
<tr>
<td>Q&amp;A</td>
<td>68</td>
</tr>
<tr>
<td>Workshop</td>
<td>69</td>
</tr>
<tr>
<td>HOUR 4: Using Xcode Templates to Create Projects</td>
<td>71</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>--</td>
</tr>
<tr>
<td>Available Project Types</td>
<td>71</td>
</tr>
<tr>
<td>The Project-Creation Process</td>
<td>83</td>
</tr>
<tr>
<td>You've Got a Project</td>
<td>88</td>
</tr>
<tr>
<td>Summary</td>
<td>91</td>
</tr>
<tr>
<td>Q&amp;A</td>
<td>91</td>
</tr>
<tr>
<td>Workshop</td>
<td>92</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HOUR 5: Managing Projects and Resources</th>
<th>95</th>
</tr>
</thead>
<tbody>
<tr>
<td>Getting Your Bearings</td>
<td>95</td>
</tr>
<tr>
<td>Managing Project Files</td>
<td>99</td>
</tr>
<tr>
<td>Managing Frameworks and Libraries</td>
<td>108</td>
</tr>
<tr>
<td>Managing Groups</td>
<td>110</td>
</tr>
<tr>
<td>Managing Target Properties</td>
<td>111</td>
</tr>
<tr>
<td>Summary</td>
<td>114</td>
</tr>
<tr>
<td>Q&amp;A</td>
<td>115</td>
</tr>
<tr>
<td>Workshop</td>
<td>115</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HOUR 6: Using the Xcode Source Editor</th>
<th>117</th>
</tr>
</thead>
<tbody>
<tr>
<td>Understanding Editor Basics</td>
<td>117</td>
</tr>
<tr>
<td>Navigating Within and Between Files</td>
<td>126</td>
</tr>
<tr>
<td>Using the Assistant Editor</td>
<td>131</td>
</tr>
<tr>
<td>Correcting Errors and Warnings in the Issue Navigator</td>
<td>133</td>
</tr>
<tr>
<td>Refactoring Code</td>
<td>135</td>
</tr>
<tr>
<td>Using Code Snippets</td>
<td>138</td>
</tr>
<tr>
<td>Summary</td>
<td>142</td>
</tr>
<tr>
<td>Q&amp;A</td>
<td>142</td>
</tr>
<tr>
<td>Workshop</td>
<td>142</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HOUR 7: Working with the Xcode 4 Documentation</th>
<th>145</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview of Documentation Resources</td>
<td>145</td>
</tr>
<tr>
<td>Configuring the Xcode Documentation Downloads</td>
<td>148</td>
</tr>
<tr>
<td>Understanding the Documentation Resources</td>
<td>149</td>
</tr>
<tr>
<td>Using the Xcode Help Viewer</td>
<td>155</td>
</tr>
</tbody>
</table>
### HOUR 8: Creating User Interfaces

- **What Is Interface Builder?**
- **The Anatomy of an Interface Builder File**
- **Creating User Interfaces**
- **Working with the IB Layout Tools**
- **Customizing Interface Appearance**
- **Setting Object Identities**
- **Adding Custom Objects to Interface Builder**

### HOUR 9: Connecting a GUI to Code

- **Outlet, Actions, and Properties: A Review**
- **Making Connections to Outlets and Actions**
- **Writing Connection Code with Interface Builder**

### HOUR 10: Creating iOS Application Workflows with Storyboards

- **The Power of Storyboards**
- **Storyboard Terminology**
- **The Anatomy of a Multiscene Project**
- **Making Advanced Segues**
- **A Navigation Storyboard Example**

### Q&A

### Workshop
## Table of Contents

**HOUR 11: Building and Executing Applications**
- The Language of the Build ................................................................. 251
- Managing Targets .............................................................................. 257
- Managing Schemes ........................................................................... 264
- Using the iOS Simulator .................................................................... 271
- Summary .......................................................................................... 276
- Q&A .............................................................................................. 277
- Workshop ........................................................................................ 277

**HOUR 12: Using Source Control**
- Using Xcode Snapshots ..................................................................... 279
- A Brief Introduction to Source Control Systems ................................. 283
- Working with Subversion and Git Repositories ................................. 285
- Managing a Project in Source Control .............................................. 296
- Summary .......................................................................................... 307
- Q&A .............................................................................................. 308
- Workshop ........................................................................................ 308

**HOUR 13: Xcode-Supported Languages**
- Choosing the Right Language ............................................................ 311
- Built-In Languages ........................................................................... 313
- Adding Support for Other Languages .............................................. 322
- Summary .......................................................................................... 325
- Q&A .............................................................................................. 326
- Workshop ........................................................................................ 326

**HOUR 14: Planning for Reuse: Frameworks and Libraries**
- Understanding Frameworks ............................................................... 329
- Deploying a Framework .................................................................... 348
- Reusing Code from Existing C/C++ Libraries .................................... 349
- Summary .......................................................................................... 352
- Q&A .............................................................................................. 352
- Workshop ........................................................................................ 353
HOUR 15: Putting It All Together: Building an OS X Application

Getting Started ................................................................. 356
Creating the Interface ..................................................... 359
Attaching Code ............................................................... 364
Inserting Interface Object References into the Code .......... 369
Tying Things Together ..................................................... 373
Increasing Functionality with a Framework ..................... 376
Summary ........................................................................... 395
Q&A ................................................................................ 396
Workshop ........................................................................ 396

HOUR 16: Building an iOS Application

Assessing What You Already Have ...................................... 400
Building from the Template .............................................. 403
Adding a Static Library Target .......................................... 404
Updating Application Logic and Library Calls for iOS ........ 414
Summary ........................................................................... 430
Q&A ................................................................................ 431
Workshop ........................................................................ 432

HOUR 17: Attaching Big Data: Using Core Data in Your Applications

Introducing Core Data ....................................................... 435
Using the Xcode Core Data Model Editor ......................... 438
Binding a Data Model to a User Interface ......................... 446
Accessing Data Through Code ........................................ 458
Summary ........................................................................... 462
Q&A ................................................................................ 463
Workshop ........................................................................ 463

HOUR 18: Test Early, Test Often

Adding Unit Tests to an Existing Application ....................... 466
Implementing Tests for Existing Code ................................. 472
Accessing the Rest of an Application Through the Bundle Loader 480
# Table of Contents

Summary ........................................................................................ 483  
Q&A .............................................................................................. 484  
Workshop ........................................................................................ 485  

**HOUR 19: Getting the Bugs Out** .............................................................................. 487  
Getting Started with the Debugger .............................................................................. 488  
Proactive Debugging ................................................................................................. 493  
Working with Breakpoints ......................................................................................... 497  
Summary .................................................................................................................. 506  
Q&A ......................................................................................................................... 506  
Workshop .................................................................................................................. 507  

**HOUR 20: Keeping Things Organized: Shared Workspaces** ................................................. 509  
Using Workspaces ...................................................................................................... 510  
Creating a Workspace ............................................................................................... 515  
Adding Projects to the Workspace .............................................................................. 517  
Configuring the OS X Project to Work in the Workspace .............................................. 519  
Configuring the iOS Project to Work in the Workspace ................................................. 523  
Summary .................................................................................................................... 530  
Q&A ........................................................................................................................... 531  
Workshop .................................................................................................................. 531  

**HOUR 21: Advanced: Analyzing Code with Instruments** ...................................................... 533  
The Instruments Interface ............................................................................................. 534  
Using Instruments ...................................................................................................... 536  
Additional Runs ......................................................................................................... 541  
Collecting Data from Additional Instruments Simultaneously ........................................ 543  
Summary .................................................................................................................... 545  
Q&A ........................................................................................................................... 546  
Workshop .................................................................................................................. 546  

**HOUR 22: Managing and Provisioning iOS Devices** ............................................................. 549  
Creating an iOS Distribution Certificate ...................................................................... 550  
Creating an App ID ....................................................................................................... 556
Creating a Distribution Provisioning Profile .......................................... 560
Summary ........................................................................................ 563
Q&A .............................................................................................. 564
Workshop ........................................................................................ 565

**HOUR 23: Distributing Your Applications** 567
Finalizing Your Distribution Build ........................................................ 568
Configuring an iTunes Connect Application Record ............................. 576
Distributing Your Archived Application ................................................ 578
Summary ........................................................................................ 583
Q&A .............................................................................................. 583
Workshop ........................................................................................ 584

**HOUR 24: Xcode CLI Utilities** 587
Using xcode-select ............................................................................ 588
Using xcodebuildid ........................................................................ 590
Using xcrun .................................................................................... 595
Other Xcode Command-Line Tools ...................................................... 598
Bigger and Better Command-Line Uses ................................................ 599
Summary ........................................................................................ 600
Q&A .............................................................................................. 600
Workshop ........................................................................................ 602

Index 603
About the Authors

John Ray is currently serving as a Senior Business Analyst and Development Team Manager for the Ohio State University Office of Research. He has written numerous books for Macmillan/Sams/Que, including Using TCP/IP: Special Edition, Teach Yourself Dreamweaver MX in 21 Days, Mac OS X Unleashed, and Teach Yourself iOS 5 Development in 24 Hours. As a Macintosh user since 1984, he strives to ensure that each project presents the Macintosh with the equality and depth it deserves. Even technical titles such as Using TCP/IP contain extensive information about the Macintosh and its applications and have garnered numerous positive reviews for their straightforward approach and accessibility to beginner and intermediate audiences.

Will Ray is an assistant professor of pediatrics in the Battelle Center for Mathematical Medicine at Nationwide Children’s Hospital. Trained as a biophysicist in computational biology and scientific visualization, Dr. Ray’s group is working to bring cutting-edge computational technology to end users, through simplified user interfaces. He has been developing training materials and teaching users and programmers to live at the intersection of Macintosh and UNIX technologies since 1989.

You can visit their Xcode book website at http://teachyourselfxcode.com or follow their book-related tweets on Twitter at #XcodeIn24.
Dedication

Since Will and I couldn’t agree on dedicating this to his parents or mine, we hereby dedicate this book to the game #Starhawk. Come play the authors and discuss Xcode in the regularly appearing “Old-N-Slow” server.

Acknowledgments

Thank you to the group at Sams Publishing—Laura Norman, Keith Cline, Greg Kettell—for working through the table of content changes, schedule conflicts, and on-the-fly revisions. You’ve made this book a reality and deciphered many 2 a.m. sentences that were barely more than random keyboard mashing.

Thanks to everyone around us—family, friends, distant relations, strangers, and pets—for providing food, ibuprofen, and paying the bills.
We Want to Hear from You!

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

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

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

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

E-mail: consumer@samspublishing.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 informit.com/register for convenient access to any updates, downloads, and errata that might be available for this book.
This page intentionally left blank
Introduction

So you've decided to write applications for OS X or iOS. You sit down at your Macintosh, start up Xcode, and... what? Create a project? Create a file? Make a storyboard? Build a Core Data model? What?

For an operating system that prides itself on being accessible to many, Xcode can appear as an insurmountable obstacle to an unprepared developer. With an iTunes-like interface, and more panels, palettes, menus, and buttons than you can count, even a simple Hello World application can seem daunting. Apple, while diligent in providing documentation, provides very few resources for developers who understand programming fundamentals but not their OS X/iOS implementation. That's where this book comes in.

Xcode offers a range of integrated tools for everything from data modeling to performance analysis and optimization. *Teach Yourself Xcode in 24 Hours* takes 24 of the most important aspects of Xcode development and condenses them down into easily understandable chunks. To help convey some of the core concepts, you work with real projects for both iOS and OS X that demonstrate important features such as shared libraries/frameworks, storyboards, Core Data models, and even hands-on debugger practice.

Xcode 4 represents an entirely redesigned version of Apple’s development suite. Despite reaching version 4.4 (in beta) during this writing, it has only been in developer’s hands for slightly more than a year. Unfortunately, this means it is a still a bit rough around the edges. We point out the issues where we encounter them, but don’t be shy about filing bug reports with Apple if features don’t quite work as anticipated. With the help of the OS X/iOS community, Xcode is being improved and enhanced rapidly. Each new release brings more consistency and reliability to the product.

Our goal for this book is to open Xcode development to programmers who may have previously eyed the platform with trepidation. A learning curve applies to becoming an Xcode developer, but once you begin to understand how Apple intends the tools to be used, you’ll find that OS X and iOS development can be fast and, most important, fun.

Who Should Use This Book?

This book targets individuals who have used programming tools but who are new to the Xcode development platform. Although no previous development experience is required to complete the book, an understanding of programming fundamentals is helpful. To be clear,
even though we provide code samples and an introduction to Objective-C, we do not have
the space in 24 hours to teach the concepts of loops, arrays, and other foundation topics.

In addition, to be successful, we recommend that you spend time reading the Apple developer
documentation and researching the topics presented in this book. A vast amount of
information about OS X/iOS development is available, and it is constantly changing.
Apple’s integrated online documentation system makes it possible to stay up-to-date on
your development knowledge and learn the details of available technologies from the com-
fort of your desktop Mac or iPad.

The material in this book specifically targets Xcode 4.3 and later. If you are running an ear-
lier version, you definitely want to upgrade before moving too far along. In addition, many
lessons are accompanied by project files that include sample code. While opening a project
and clicking Run can be fun, we prefer that you follow along, when possible, and build the
application yourself.

Be sure to download the project files from the book’s website at
http://teachyourselfxcode.com. If you have issues with any projects, view the posts on this
site to see whether a solution has been posted.

In addition to the support website, you can follow along on Twitter. Search for #XcodeIn24
on Twitter to receive official updates and tweets from other readers. Use the hashtag
#XcodeIn24 in your tweets to join the conversation. To send me messages via Twitter, begin
each tweet with @johnemeryray.

---

By the way

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/9780672335877.
HOUR 6

Using the Xcode Source Editor

What You’ll Learn in This Hour:

- Xcode Source Editor basics
- How to use autoformatting features
- The use of pragma marks to identify different pieces of code
- Ways to find and correct errors before your app even runs
- How to refactor existing code

Over the past few hours, you have learned how to create projects, add files, add frameworks, and do much of the work necessary to successfully build your own application projects. What we have not touched on, however, is the Xcode Source Editor itself, which you use to edit, unsurprisingly, source code. You’ve got your project and files; now how about editing them?

This hour walks through the different Source Editor features—from automatic code completion to code refactoring. You learn how the Source Editor can help you write clean, well-formatted code and even identify problems before you even try to run your application. Even if you have played around with editing files already, you’re still likely to find a few undiscovered tricks in this hour.

Understanding Editor Basics

Let’s be serious: If you’re learning how to program in Xcode, you know how to edit a text file. I am not going to bore you with details on how to move your cursor or copy and paste. The Xcode Source Editor works just like any text editor, with several additions that may make your life easier. To follow along with this hour’s lesson, create a new project.
called **HelloXcode** using the Mac OS X Cocoa Application template and the configuration shown in Figure 6.1. We’ll edit this so that it displays a simple message (Hello Xcode) in the application’s window. Nothing earth shattering, but you’ll find it handy to keep this open so that you can test the tools as you read.

To edit code in Xcode, use the Project Navigator to find the file you want to work on, and then click the filename. The editable contents of the file are shown in the Editor area of the Xcode interface. For our sample application, click the AppDelegate.m file, as shown in Figure 6.2.

### Code Completion

Using the Source Editor, start entering the following text to implement the applicationDidFinishLaunching method. Start a new line immediately following the existing comment “Insert code here to initialize your application.” Update the method as shown in Listing 6.1.

```objective-c
1: - (void)applicationDidFinishLaunching:(NSNotification *)aNotification
2: {
3: // Insert code here to initialize your application
4:   NSTextField *myMessage;
5:   NSTextField *myUnusedMessage;
6:   myMessage=[[NSTextField alloc] init];
```
As you enter the code, notice that when you are typing a method or class name that Xcode recognizes, a pop-up dialog appears near your cursor, as shown in Figure 6.3. Here, the systemFontOfSize method is being typed, and Xcode is presenting potential options for autocompletion as I type.

To choose an autocompletion value, use the arrow keys to highlight the value you want to use, and then press Return to insert it into your code. You can also press Escape to make the pop-up disappear.

If you are completing a method name, chances are that you need to provide parameters as well. (In the case of systemFontOfSize, it is a floating-point value that describes the size of the text.) You can again use the arrow keys to move between the parameter fields and enter the values you want, or you can just press Tab to skip from parameter to parameter.
If you have already finished typing (or autocompleted) a line in your code and discover that it is not what you want, you can click anywhere within the line and then choose Editor, Show Completions (Control+Spacebar) or press Escape. Doing so displays the autocomplete pop-up dialog with all the potential matches wherever your cursor is located—as if the rest of the line does not exist.

You can adjust the code completion options in the Xcode preferences within the Text Editing section.

**Auto-Indentation**

Clean code is easier to read and easier to maintain, and Xcode works behind the scenes to keep your code nicely formatted through auto-indentation. As you program, Xcode automatically indents lines so that they fall either directly under the previous line or as appropriate to the structure of the statements you are writing.

Code within conditional blocks and loops, for example, are indented farther than surrounding code to visually show that they are a cohesive block. This has no effect on execution, but it can make reading through a source code file much easier than if each line’s code starts at the first character.

You can control the logic for the Xcode auto-indentation system using the Text Editing panel of the application preferences (Xcode, Preferences) and the Indentation button within that, as shown in Figure 6.4.
A nice feature of Xcode is that indentation isn’t just applied while you’re typing and then lost. Although you can certainly move your code around and make it into a mess, you can always apply the indentation rules again, as follows:

1. Within the sample project, add tabs and spaces in front of some of the lines in `applicationDidFinishLaunching`.
2. Delete the indentation from some of the lines, as well.
3. After you have made the method sufficiently ugly in appearance, select the lines of code.
4. Choose Editor, Structure, Re-Indent (Control+I).

Xcode reformats the code, and all is well again, as shown in Figure 6.5.

You can access the Structure menu by right-clicking directly in the Xcode Source Editor. This can be a useful menu to access while editing, so the faster you can get to it, the better.
Balancing Delimiters

Indentation may be “just for looks,” but delimiters are not. Properly balanced delimiters are the difference between code that works the way you want and code that seems to have a mind of its own. When coding in Objective-C, you work with three primary types of block delimiters:

- Parentheses for function calls
- Square brackets for Objective-C messaging
- Curly brackets for logical programming blocks

Each time you type one of these delimiters, you need another matching one added to your code. If you happen to miss one or put one in the wrong place, chances are your code won’t run. Worse yet, it might run and not do what you expect.

To help you keep track of where your delimiters are (or aren’t) balanced, Xcode automatically highlights the first delimiter when you type the second (or move the cursor to the immediate right of the second delimiter).

For example, return to the sample method that you wrote earlier (applicationDidFinishLaunching), position your text entry cursor immediately before the right curly bracket, }, at the end of the method. Next, press the right arrow to move the text cursor to the right of the curly bracket. As you do this, notice
that the left curly bracket, {, at the start of the method is briefly highlighted. This same behavior occurs with parentheses and square brackets, too.

To select all of the code that is contained within a set of delimiters, right-click within the code, and then choose Structure, Balance Delimiter (also available from the Editor menu), as shown in Figure 6.6.

![Select the code between two delimiters.](image)

When you are entering your code, Xcode automatically inserts matching curly and square brackets when you type the first bracket. In other words, if you type {, Xcode automatically adds a corresponding } for you. You can disable this behavior in the Xcode Text Editing preferences (Xcode, Preferences, Text Editing, Editing).

**By the Way**

**Code Folding**

Working in conjunction with your code delimiters and the logical blocks/methods you define in your code, Xcode’s code-folding features let you focus on editing a specific piece of code and hide the rest. To the immediate right of your Editor is the gutter, which typically holds line numbers (enabled in Xcode’s Text Editing preferences) and error/warning icons. The very right side of the gutter is the code-folding ribbon. By clicking in the ribbon, you can fold (collapse) your methods and functions and comment blocks.
For example, view the code in the `applicationDidFinishLaunching` method. Click immediately to the left of the method in the code-folding ribbon. The code collapses and is replaced with an ellipsis (…), as shown in Figure 6.7. You can expand the code again by double-clicking the ellipsis or by using the disclosure arrow that appears in the ribbon.

To help identify what blocks of your code can be collapsed, Xcode lightly shades the code folding ribbon. Blocks within blocks are successively darker in color. Hovering your mouse over a shaded portion of the code-folding ribbon highlights the relevant block within Xcode.

You can quickly collapse all comment blocks (text between /* and */) and function/methods using the Editor, Code Folding menu.

### The Other Bits

As with any application, some settings and options (although useful to some) do not warrant a full discussion. The Xcode Source Editor has a number of other features that might come in handy but that do not necessarily pertain to the editing process itself.

What follows are a few configuration tweaks and editing functions that you might want to use in your projects.

#### Line Numbers

To enable or disable the display of line numbers beside your code, use the Line Numbers check box accessed within the Text Editing section of Xcode preferences.
Line numbers are a great help when debugging code or describing your code to other developers.

**Edit All in Scope**

If you’ve ever written a function or method only to smack yourself in the head and say, “I should have named that variable something else,” you will love the Edit All in Scope feature. To simultaneously change all the instances of a variable name within a method, select the variable name, and then choose Editor, Edit All in Scope (Control+Command+E). Each instance of the variable highlights, and all update as you edit one, as shown in Figure 6.8.

![Figure 6.8: Quickly change symbol names using Edit All in Scope.](image)

**Shift, Move, or Comment**

The Structure contextual menu (also accessible from the Editor menu) contains commands for shifting (indenting/outdenting) your code left or right, moving it up or down, or commenting it out. Use these functions to quickly act on a selection of lines, rather than changing each one individually.

**Hide/Show Invisibles**

There’s more to your code than you can see. To get a glimpse at how the code is formatted (tabs/spaces/return characters) or check for any gremlins that may have crept into your code from pasting from other applications, use the Editor, Show Invisibles command. This displays all normally invisible characters directly within your code listing. To hide the characters, choose Editor, Hide Invisibles.
Syntax Coloring

Normally, Xcode colors your code based on the type of file you are editing. You can override the chosen syntax-highlighting scheme by choosing Editor, Syntax Coloring from the menu. You can also change the syntax colors entirely using the Xcode Fonts & Colors preferences, shown in Figure 6.9.

Navigating Within and Between Files

Now that you know the basic code-editing features provided by Xcode, we can turn our attention to some of the features that simplify working with multiple files. Except in the most rudimentary development, your projects will consist of multiple source files with multiple methods spread between them. Becoming efficient and jumping between these files is a skill that becomes increasingly valuable as your applications increase in scale.

This section examines some of the tools you can use when working with multiple files (or very large individual files).

Tabbed Editing

Tabbed editing is just like tabbed browsing in your favorite web browser. Using tabbed editing, you can have many files open simultaneously and switch between them by clicking tabs at the top of the Editor area.
To create a new tab, choose File, New, New Tab (Command+T). A new tab appears with the contents of the file you are currently editing. You can switch the contents of the tab to whatever you want by clicking a file in the Project Navigator. You can repeat this process to create as many tabs as you want, with whatever file contents you want, as shown in Figure 6.10.

To close a tab, click the X that is displayed on the left side of the tab when hovering over it with your mouse. As with all files in Xcode, the files you edit in tabs are automatically saved when you close them; you do not have to explicitly use the Save command.

The Jump Bar

When editing a file, you might have noticed that above the Editor area is a visual path to the file you are editing and several buttons. This is collectively known as the jump bar. Clicking any portion of the path reveals a pop-up menu for quickly jumping to other files in same location, as shown in Figure 6.11. The last segment (on the right) of the jump bar is special: You can click it to view and jump between the symbols (methods, properties, and so on) within the file you are currently editing.

To the left of the path are forward and back arrows. The arrows move back and forth between files that you have been editing—just like pages you visit in a web browser. Finally, to the left of the arrows is the Related Files button.

Use the Related Files button to show a pop-up menu of categorized files such as recently opened or unsaved. This menu even displays files that are just referenced (even if just included or imported) in the file currently open in the Source Editor. Figure 6.12 shows the contents of the menu when editing the application delegate file for an empty iOS application.
Using the Xcode Source Editor

The Symbol Navigator

The easiest way to find a method or property within a source code file is to use the Symbol Navigator, opened by clicking the icon to the immediate right of the Project Navigator. This view, shown in Figure 6.13, enables you to expand your project classes to show all the methods, properties, and variables that are defined. Choosing an item from the list jumps to and highlights the relevant line in your source code.

FIGURE 6.12
Find files that are referenced by, or directly/indirectly related to the file you are editing.

FIGURE 6.13
The Symbol Navigator is a quick way to jump between methods and properties.
Navigating Within and Between Files

For example, with the HelloXcode project open, switch to the Symbol Navigator and expand the AppDelegate item. This is the only object used in this application. Next, find and select `applicationDidFinishLaunching` from the list that is displayed. Xcode jumps to the finds and select the line where the method begins.

The Search Navigator

Searching for text anywhere in your project is trivial using the Search Navigator. To access this search feature, click the magnifying glass icon in the icon bar above the Navigator. A Search field displays at the top of the Navigator area, into which you can enter whatever you want to find. As you type, a drop-down menu appears, as shown in Figure 6.14, that shows potential options for refining your search. Choose one of the options or press Return to perform a non-case-sensitive search of the text you enter.

The search results display below the Search field, along with a snippet of the file containing the text you were looking for, as shown in Figure 6.15. Clicking a search result opens the corresponding file in the Source Editor and jumps to the line containing your search string.

To make things even more interesting, you can use the Filter field at the bottom of the Search Navigator to filter your search results by a secondary term. You can also click the Find label at the top of the Search Navigator to switch to a Replace mode, enabling you to perform projectwide find and replace.
FIGURE 6.15
Search results are displayed along with the context of the match.

By the Way
If you’re looking for a string within a file you are actively editing, choose Edit, Find, Find (Command+F) to open a more traditional Find field at the top of the Source Editor. This gives you quick access to find (or find/replace) within a given file, rather than across the entire project.

Pragma Marks
Sometimes navigating code by symbols or with a search is not very efficient. To help denote important pieces of code in plain English, you can insert a #pragma mark directive.Pragma marks do not add any features to your application; instead, they create logical sections within your code. These sections are then are displayed, with the rest of the code symbols, when you click the last item in the visual path above the Editor area.

There are two common types of pragma marks:

```
#pragma mark -
and
#pragma mark <label name>
```

The first inserts a horizontal line in the symbol menu; the second inserts an arbitrary label name. You can use both together to add a section heading to your code. For example, to add a section called Methods for starting and stopping the application followed by a horizontal line, you can enter the following:

```
#pragma mark Methods for starting and stopping the application
#pragma mark -
```
After the pragma mark has been added to your code and saved, the symbol menu updates accordingly, as shown in Figure 6.16. Choosing a pragma mark from the Symbol menu jumps to that portion of the code.

FIGURE 6.16
Use pragma marks to add logical delimiters to your code.

Using the Assistant Editor

As you work with Xcode projects, you will quickly realize that most program functionality comes from editing one of two related files: an implementation file (.m extension) and an interface file (.h extension). You’ll also learn that when you make changes to one of these two files, you’ll often need to make changes to the other. You will see a similar pattern emerge when you start editing GUIs in Xcode.

Xcode simplifies this back-and-forth editing with the Assistant Editor mode. The Assistant Editor (or just called Assistant in Apple documentation) automatically looks at the file you have opened for editing and opens, right beside it (or under it, if you prefer), the related file that you also need to work on, as shown in Figure 6.17.

To switch between Standard and Assistant Editor modes, you use the first and second buttons, respectively, in the Editor section of the Xcode toolbar.

In the upper-right corner of the Assistant Editor’s jump bar, notice a + icon and an X icon. The + icon adds another Assistant Editor to your screen; the X icon closes the current Assistant Editor.

By the Way
Choosing the Assistant File

When you are using the Assistant Editor, it sometimes chooses a file that it thinks you want to edit but that really is not what you had in mind. To view all the Assistant’s recommended options, click the Assistant Editor icon (a little suit and bowtie) displayed in the jump bar located at the top of the Assistant Editor view. This displays a menu of all the related files that the Assistant Editor has identified, sorted by type, as shown in Figure 6.18.

Changing the Assistant Editor Layout

As you can see, the Assistant Editor occupies more than a little bit of screen space. If you are coding on a MacBook Air (which I love), you’ll find that you need to hide the Navigator/Utility areas to make the full Assistant Editor work.

To change how the Assistant Editor displays in the Xcode interface, choose View, Assistant Editor, as shown in Figure 6.19. From there, you can choose how the Assistant Editor is added to the display and what will happen if multiple Assistant Editors are in use.

By the Way

To set up shortcuts for when the Assistant Editor (or tabs) are used, open the Xcode general preferences (Xcode, Preferences, General). You have several options to fine-tune how you invoke the special editing modes of Xcode.
Correcting Errors and Warnings in the Issue Navigator

As you write your code, Xcode is sitting in the background judging you, but do not take it personally—it only wants to help. Xcode analyzes your code and detects issues that might be (or are) problems and displays warnings and errors before you ever click the Run button.

**Error Types**

You can receive three general types of feedback from Xcode when it analyzes your application: errors, warnings, and logic problems. Warnings are potential problems that might cause your application to misbehave; a yellow caution sign warns you of
these. Errors, however, are complete showstoppers. You cannot run your application if you have an error. The symbol for an error, appropriately enough, is a red stop sign.

Logic problems, found by the Xcode analyze process, are shown as a blue badge. All of these bugs, across all your files, are consolidated in the Issue Navigator. The Issue Navigator displays automatically if problems are found during a build or analyze process. You can also open it directly by clicking the exclamation point icon (on the toolbar above the Navigator area).

For example, the method `applicationDidFinishLaunching` that you wrote earlier contains an unused variable (`myUnusedMessage`). This is highlighted with a yellow exclamation point in the Issue Navigator (Unused Entity Issue), as shown in Figure 6.20.

![Issue Navigator](image)

**FIGURE 6.20**
Use the Issue Navigator to browse all your project’s potential problems.

Warning and errors are shown immediately as you code.

**By the Way**

Warnings and errors are shown immediately as you code.

### Jumping to an Error

To jump to an error in your code, just click it in the Issue Navigator. The corresponding code is opened, and the error message is visible directly after the line that caused the problem. To remove the warning from the sample method, just delete the line `NSTextField *myUnusedMessage;` to empty the Issue Navigator.
By the Way

If you are in the middle of editing a file that contains errors, you’ll see the errors immediately displayed onscreen—so no need to jump back and forth to the Issue Navigator. You can also quickly cycle through your errors using the forward and backward arrows found at the rightmost side of the window, directly above the Editor area. These controls are visible only if errors are present, however.

Fixing Errors to Find Others

With the warning message still visible in the applicationDidFinishLaunching method (add the code line back in, if you need to), try adding a line of complete garbage into the code. Pay attention to what happens when you click off of the line.

What you will see is that the new code is flagged as an error (red stop sign), but the original warning has disappeared. This brings up an important point: Sometimes not all errors or warnings are detected and displayed in the Issue Navigator or in a source file. You might find, after fixing a line or two, that new and previously undetected errors appear. Conversely, you’ll also occasionally see false errors that disappear when you correct others.

To control which issues are visible (hiding warnings, for example), choose Editor, Issues from the menu bar.

Refactoring Code

Earlier in this hour, you learned how to use Edit All in Scope to change the name of a variable within a method. But what happens if you need to make a more massive change, such as changing the name of a class? Making changes like this is called refactoring and can involve a ridiculous amount of work given the amount of code that needs to be touched, filenames that need to change, and so on. If you find yourself in this situation, however, Xcode offers refactoring features to help get you out of a jam.

To access the refactoring tools, choose Edit, Refactor from the menu bar or right-click in your code to use the Refactor contextual menu item.

Renaming

To rename a symbol across your entire project, including any files named after that symbol, follow these steps:
1. Select the symbol in your project.
2. Choose Edit, Refactor, Rename from the menu bar.
   You are prompted for the new name, and whether to rename any associated files, as shown in Figure 6.21.

3. After you have made your naming choice, click Preview to show the files that will be changed and the differences between the original and new versions, as shown in Figure 6.22.
   From this window, you can choose which files to change (check or uncheck the boxes in front of their names) and even edit the code directly in the comparison views.
4. Click Save to make the changes across the entire project.

The remaining refactoring options work in a similar way, but with different effects.

### Extracting
If you find that you have written code for a method but that code would be better suited in its own method or function that is called by the current method, you can use the Extract function. Simply select the code you want to move out of the current method, and then choose Edit, Refactor, Extract.
You are prompted for the new function or method name, then presented with a preview of the change. Xcode automatically analyzes the code to determine what parameters are necessary for the new method, and even includes the method/function call in your original method.

**Double-Check Xcode’s Work!**
Be sure to double-check the methods that Xcode creates when using the Extract feature. I have had mixed results with its ability to correctly parse the method parameters.

**Creating Superclasses**
Sometimes, when writing a class, you might discover that it would have made more sense for the class you’re building to be a subclass of a larger superclass. To quickly create a superclass, select the class name, and then choose Edit, Refactor, Create Superclass, and, again, follow the onscreen prompts to name and save the new class file.

**Moving Up/Down**
After creating a superclass (or if you already have one), it might make sense to move some of the variables or methods from your current class into the superclass. Conversely, you might have a superclass symbol that make more sense being
contained in one of its subclasses. Use the Edit, Refactor, Move Up/Down functions to move a variable/method into a class's superclass or a superclass's variable/method into a subclass.

**Encapsulating**

When you use the Edit, Refactor, Encapsulate action on an instance variable, it creates explicit getters and setters for that variable. In most cases, however, the @property/@synthesize combination should be enough to handle setting up your accessors.

If you need a refresher on instance variables, classes, setters, getters, and so on, refer back to Hour 2, “Just Enough Objective-C and Cocoa.” These are important concepts to understand, so be sure you have got a good grip on the basics before moving on.

**Using Code Snippets**

When you are writing code, you will often find yourself typing statements that are very similar to one another: catch-try blocks, if-then statements, switch statements, and so on. Instead of having to remember the syntax of each of these common code sequences, you can make use of the built-in Xcode Code Snippet Library.

You can access the Code Snippet Library from the Xcode Source Editor by choosing View, Utilities, Show Code Snippet Library, or by pressing Control+Option+Command+2. The library appears in the Utility area and is represented by the {} icon, as shown in Figure 6.23.

**Viewing Snippets**

To view what a snippet contains, click its entry in the library. A popover window appears displaying the contents of the snippet, as shown in Figure 6.24.

Notice that certain parts of the snippet are highlighted in gray. These represent portions of the code that you need to provide. When a snippet is inserted into your code, you can use the Tab key to jump between these areas. Click the Done button when viewing the snippet code to hide the popover.
Using Code Snippets

Inserting Snippets
To insert a code snippet into your class files, just drag and drop from the Code Snippet Library into your code. Wherever your cursor points, the snippet is inserted, exactly as if you had typed the code yourself.

**FIGURE 6.23**
The Code Snippet Library contains snippets of useful code.

**FIGURE 6.24**
View the contents of a snippet.
Adding Snippets

Not only can you use the existing snippets to help code, but when you write useful code sequences of your own, you can add them to the Code Snippet Library as user-defined snippets. To do this, follow these steps:

1. Select the code in the Source Editor that you want to turn into a snippet.
2. Make sure the Code Snippet Library is visible (Control+Option+Command+2).
3. Click and drag from the selected text to the Code Snippet Library, as shown in Figure 6.25.

The text is added to the library under the title My Code Snippet, as shown in Figure 6.26.
Because the default name is not very descriptive, you will probably want to edit it to reflect its purpose.

**Editing Snippets**

To edit a snippet you have defined, follow these steps:

1. Click it in the library so that the popover appears displaying its code.

2. Click the Edit button in the lower-left corner. Figure 6.27 shows a snippet in edit mode.

3. Use the fields in the Source Editor to set a title and summary for the snippet and define the platform and development language the snippet applies to.

4. Enter a shortcut that can be typed to automatically insert the snippet and the different areas (scopes) within a typical source file where the completion shortcut should be active.

5. Click Done to save the snippet.

To remove a snippet from the library, just click once to select it in the Code Snippet Library and then press the Delete key on your keyboard.
Summary

In this hour, you learned how to use the Xcode Source Editor for more than just typing text. We looked at special features like automatic code completion, auto-indentation, and code folding. You also learned how the Assistant Editor mode makes working on multiple related files a pain-free process.

In addition to the Source Editor itself, you explored the tools for identifying and correcting errors in your code, for searching and replacing text across your entire project, and for refactoring existing code. Although it is technically true that you can develop an application using any text editor you want, using Xcode gets you best-of-breed tools that are specific to the language and methodology of the Apple development platform.

Q&A

Q. I really hate editing files in a huge window. Can I break out into just a single window per file?
A. Yes. Just double-click the file in the Navigator to open a new window with all features disabled except the Source Editor. You can turn this into a single-click in the Xcode preferences if you prefer.

Q. I’m confused. What is a symbol?
A. Think of a symbol as anything you’ve named within your code. Typically, this means variables/properties—but it can also be method names, class names, and so on.

Q. Xcode keeps autocompleting methods that I don’t want. What do I do?
A. Unless you press Tab or the arrow keys, Xcode will not autocomplete at all. You can either just keep typing and enter the full name yourself, or you can disable autocompletion in the Text Editing area of the Xcode preferences.

Workshop

Quiz

1. I use a MacBook for my development. How will I ever get the Assistant Editor to fit on my screen?
2. Besides the Assistant Editor, you can only have a single file open at once. True or false?

3. It is impossible to easily rename a class after creating it. True or false?

**Answers**

1. Learn to use the View buttons on the Xcode toolbar. If you disable the Project Navigator and Utility area, you’ll find that the Assistant Editor fits quite nicely on your screen.

2. False. Using the tabbed editing feature, you can open as many files simultaneously as you want, using the tabs below the Xcode toolbar to switch between them.

3. False. The Rename refactoring tool simplifies the process of changing a class name after the fact.

**Activities**

1. Using the sample project you created this hour, walk through the different features of the Source Editor and Assistant Editor. Try your hand at Edit All in Scope and refactoring.

2. Create errors your HelloXcode application by duplicating lines or methods in the AppDelegate class files. Add a few lines with arbitrary text for good measure. Note where and when Xcode detects the errors. Does it find all the erroneous lines at once, or does it present one error, which, when fixed, shows another, and so on? Get used to this behavior; it is rare you’ll write perfect code, and knowing how Xcode presents errors and warnings is a big help when things do not work right.
This page intentionally left blank
Symbols
@2x naming convention, 114
@class directive, 222
@implementation directive, 34
#import directive, 29, 34
@interface directive, 29-30
@property directive, 31-33
@synthesize directive, 35

A
accessing
Core Data model data, 458-460
Quick Help, 158
actions, 196
connections, creating, 198, 201-203
creating, 208-211
prebuilt, 203
activating Quick Help Inspector, 160

ad hoc distribution, iOS applications, 580
adding
attributes in Core Data model editor, 441-442
browser component to OS X interface, 382-387
code snippets, 140-141
custom objects to IB, 192-193
embedded frameworks to OS X applications, 388-395
entities in Core Data model editor, 440-441
files to projects, 105-106
frameworks to projects, 108-109
groups to projects, 110-111
icons to applications, 113-114
programming language support
Python, 322-323
Ruby, 324-325
projects to workspaces, 517-519
scenes to storyboard, 218-219
schemes, 270-271
shared frameworks to OS X applications, 376-382
supporting view controller subclasses to scenes, 220-223
templates, 99-105
unit tests to applications, 466-471
Xcode Help viewer bookmarks, 156
Address Book Actions, 81
advanced segues, creating navigation controllers, 230-234		tab bar controllers, 234-239
agvtool, 598
alignment (IB), 179
allocating objects, 40-41
Analyze action, 255, 268
API (application programming interface), 145
App IDs, creating, 556-560
App Store
applications, distributing, 583
iTunes Connect records, configuring, 576-578
AppleScript, 317-319
application access, comparing Mac OS X and iOS, 7
application logic, updating iOS applications, 414-430
Application templates, 72
Master-Detail template, 73
OpenGL Game template, 74
applications
accessing through Bundle Loader, 480-483
distributing via App Store, 583
icons, adding, 113-114
iOS
ad hoc distribution, 580
distribution build, finalizing, 572-576
enterprise distribution, 581-582
launching in iOS simulator, 272-273
OS X
browser component, adding to interface, 382-387
building, 356-358
code, attaching to interface, 364-369
distribution build, finalizing, 568-572
embedded frameworks, adding, 388-395
interface, building, 374-375
interface, creating, 359-364
object references, inserting in code, 369-373
shared frameworks, adding, 376-382
signed applications, distributing, 580
unsigned applications, distributing, 579
unit tests, adding, 466-471
ARC (Automatic Reference Counting), 49
archive
building applications
OS X applications, 571-572
iOS applications, 575-576
settings, confirming for OS X applications, 569-570
Archive action, 255, 268
arrays, Cocoa, 52
articles, 150
Assistant Editor, 131-132
attaching code to OS X interface, 364-369
attributes, 436
adding in Core Data model editor, 441-442
of properties, 33
Attributes Inspector (IB), 188-189
Auto Layout system (IB), 183-188
auto snapshots, 282
auto-indentation feature (Source Editor), 120-121
autocompletion feature (Source Editor), 118-120
Automator Action template, 81
balancing delimiters (Source Editor), 122-123
binding data models to user interfaces, 446-458
bindings, 438
blocks, 43
bookmarks, adding Xcode Help viewer, 156
branching, 284, 304-306
breakpoints
  conditional, 502-506
  setting, 497-498
broken distribution certificates, repairing, 551-552
browser component, adding to OS X interface, 382-387
browsing Xcode Help viewer library, 156
build actions, 254, 266
build configurations, 252, 255
  build actions, 254
  schemes
    adding, 270-271
    editing, 266-268
  targets, 252
  case study, 260-262
  creating, 257-258
  products, 253
  verifying, 256
Build Phases tab (Project Navigator), 263
Build Rules tab (Project Navigator), 264
Build Settings tab (Project Navigator), 263
building
  iOS applications
    archive, 575-576
    from template, 403-404
    static library target, adding, 404-413
  OS X applications, 356-358
    archive, 571-572
    interface, 359-375, 382-387
built-in languages
  AppleScript, 317-319
  C, 313-315
  C++, 315-316
  Java, 319-320
  Objective-C, 316-317
  Perl, 321-322
Bundle Loader, 480-483
bundles, 80

C
  C, 313-315
  C++, 315-316
  reusing code, 349-352
categories, defining, 37
certificates, 550
changes, committing, 284
checkouts, 283
CIS (continuous integration server), 600
class methods, 27
classes, 26, 216
cleaning up final code
  iOS applications, 574
  OS X applications, 570
Cocoa, 50-51
  arrays, 52
  dates, 53
  dictionaries, 53
  numbers, 53
  strings, 52
  URLs, 54
  versus Cocoa Touch, 51
Cocoa Application template, 79
Cocoa Framework template, 80
Cocoa Touch, 51
Cocoa Touch Static Library, 77
Cocoa-AppleScript Application template, 79
code
  attaching to OS X interface, 364-369
  auto-indentation (Source Editor), 120-121
  autocompletion (Source Editor), 118-120
  C/C++, reusing, 349-352
  connection code, writing with Interface Builder, 205-211
design, 599
  extracting, 137
  interface object references, inserting, 369-373
  refactoring instance variables, encapsulating, 138
  symbols, renaming, 135-136

How can we make this index more useful? Email us at indexes@samspublishing.com
code

snippets
  adding, 140-141
  editing, 141
  viewing, 138
spaghetti code, 59
code folding (Source Editor), 123-124
coding how-tos, 150
collecting
data simultaneously with Instruments, 543-545
multiple runs in Instruments, 541-542
Command Line Tool template, 80
command-line
  agvttool, 598
  xcode-select tool, 588-589
  xcodebuild tool, 590-591
  available SDKs, displaying, 592-595
  workspaces, 591
  xcrun, 595-597
command-line tools, 587
commits, 298-299
committing changes, 284
comparing
dynamic libraries and frameworks, 330
Mac OS X and iOS development, 6
  application access, 7
  frameworks, 6
  interface development, 6
Xcode 3.2 and Xcode 4.2, 4
compilers, LLVM, 49
conditional breakpoints, 502-506
configurations, 437
configuring
documentation downloads, 148-149
iTunes Connect records, 576-578
segues, 225-227
template for projects, 85-87
workspaces
  iOS projects, 524-529
  OS X projects, 519-524
connecting to remote repositories, 287-290
connection code, writing with Interface Builder, 205-211
connections, 166
  creating, 198
  to actions, 201-203
  to outlets, 199-200
  editing with Quick Inspector, 204
continuing program execution in debuggers, 499
controllers, 59, 62, 67-68
  IBAction directive, 63-64
  IBOutlet directive, 63
convenience methods, 41
converting OS X applications to iOS, 400-402
Core Data Editor, 64-65, 438-440
attributes, adding, 441-442
entities, adding, 440-441
  graph style, 445
relationships, defining, 443-444
Core Data model, 60, 436
  binding to user interface, 446-458
  data, accessing through code, 458-460
data, reading, 461-462
data, writing to Core Data store, 460-461
Core Image, 82
creating
  actions, 208, 210-211
  App IDs, 556-560
  connections, 198
    to actions, 201-203
    to outlets, 199-200
distribution certificates, 552-554
distribution profiles, 560-563
frameworks, 341-347
ICNS files, 571
interfaces (OS X), 359-375, 382-387
iOS distribution certificates, 550-551
outlets, 206-208
projects, 13, 83
  template, configuring, 85-87
  template, saving, 87
  template, selecting, 84
protocols, 38
provisioning profiles, 552-554
segues, 223-225
navigation controllers, 230-234
push segues, 243
tab bar controllers, 234-239
snapshots, 280
static libraries, 333-342
superclasses, 137
targets, 257-258
user interfaces, 174
working copies, 295-296
workspaces, 515-517
custom objects, adding to IB, 192-193
customizing templates, 91
data models, binding to user interfaces, 446-458
data type classes, Cocoa
arrays, 52
dates, 53
dictionaries, 52-53
numbers, 53
strings, 52
URLs, 54
dates, Cocoa, 53
debuggers, 488
breakpoints, 497-498
conditional breakpoints, 502-506
gdb, 489-492
lldb, 490
paused applications, maneuvering, 498-499
proactive debugging, 493-496
program execution, continuing, 499
stepping forward, 500-502
declaring variables, 38
object data types, 40
primitive data types, 39
defining
categories, 37
methods, 30-31
relationships in Core Data model editor, 443-444
delimiters (Source Editor), 122-123
deploying frameworks, 349
derived data interference, troubleshooting, 575
design patterns, 58
MVC, 59
collectors, 62-64, 67-68
Data model, 64-65
models, 60
views, 61, 66
developers, paid developer programs, 8
joining, 9-11
registration, 9-10
development paradigms
comparing Mac OS X and iOS, 6
imperative development, 24
OOP, 24
inheritance, 25
Objective-C, 26-49
development profiles, installing, 17-19
dictionaries, Cocoa, 53
distributing, applications
iOS applications
ad hoc distribution, 580
App Store, 583
enterprise distribution, 581-582
signed applications,
OS X, 580
unsigned applications,
OS X, 579
distribution builds, finalizing
iOS applications, 572-576
OS X applications, 568-572
distribution certificates
creating, 550-554
repairing, 551-552
distribution profiles, 17
creating, 560-563
documentation
articles, 150
coding how-tos, 150
downloads, configuring, 148-149
Getting Started, 151
Quick Help, 147
accessing, 158
Quick Help Inspector, activating, 160
results, interpreting, 159-160
reference documents, 152
sample code, 153
technical notes, 154
technical Q&As, 154

How can we make this index more useful? Email us at indexes@samspublishing.com
video, 155
web-based, 146
Xcode Help viewer, 146
bookmarks, adding, 156
library, browsing, 156
library, searching, 157
Xcode tasks, 154
downloads
documentation, configuring, 148-149
updates, 284
Xcode developer program, 11
dynamic libraries, 330

E
editing
code snippets, 141
connections with Quick Inspector, 204
schemes
Analyze action, 268
Archive action, 268
Build action, 266
Profile action, 268
Run action, 267
Test action, 267
embedded frameworks, adding to OS X applications, 388-395
class stub files, 138
ending interface files, 33
evaluating project toolsets, 16
enterprise distribution for iOS applications, 581-582
entities, 436
adding in Core Data model editor, 440-441
errors, 133
fixing, 135
jumping to in Issue Navigator, 134
events, actions, 196
events, actions, 196
examples of navigation controllers, 239-248
exporting snapshots, 282
expressions, 44-45
condition-based loops, 47
if-then-else statements, 45
loops, 46
switch statements, 45
syntax, 44
Extended Detail pane (Instruments), troubleshooting
default code, compactor, 540
extracting code, 137
for loops, 46
Framework & Library templates, 72
frameworks, 6
adding to projects, 108-109
Core Data, 436
Core Image, 82
creating, 341-347
deleting, 349
embedded frameworks, adding to OS X applications, 388-395
headers, 109
removing from projects, 110
selecting, 331-333
shared frameworks, adding to OS X applications, 376-382
files
adding to projects, 105-106
locating, 107
removing, 106-107
renaming, 106
filters (project Navigator), 98
final code, cleaning up in iOS applications, 574
finalizing distribution build
iOS applications, 572-576
OS X applications, 568-571
archive settings, confirming, 569-570
archive, building, 571-572
final code, cleaning up, 570
fixing
distribution certificates, 551-552
errors in Issue Navigator, 135
for loops, 46
File Template Library, 103-105
File Template Wizard, adding template-based files, 99-103
features
iOS simulator, 275-276
Source Editor, 125-126
fetch requests, 437
fetched properties, 437
headers, 109
removing from projects, 110
selecting, 331-333
shared frameworks, adding to OS X applications, 376-382
G

gdb (GNU debugger), 489-492

Generating multitouch events in iOS simulator, 274

**Getting Started documents, 151**

Git, 283

- branching, 304-306
- commits, 298-299
- merging, 306-307
- pulls, 299-300
- pushes, 298-299

repositories

- creating, 286-287
- loading projects into, 293-294
- status codes, 297
- updates, 299-300
- working copies, creating, 295-296

**graph style (Core Data model editor), 445**

groups

- adding to projects, 110-111
- removing from projects, 111

guides (IB), 177-178

H

**headers (frameworks), 109**

Hellegrass, Aaron, 317

**how-to videos, 155**

I

**IBAction directive, 63-64**

**IBOutlet directive, 63**

ibtool, 599

ICNS files, constructing, 571

icons, adding to application, 113-114

IDE (interface development environment), 16

**if-then-else statements, 45**

Image Unit plug-in, 82

imperative development, 24

implementation files

- #import directive, 34
- @implementation directive, 34
- @synthesize directive, 35
- methods, implementing, 35-36

implementing

- methods, 35-36
- tests for existing code, 472-480

inheritance, 25

initializing objects, 40

- convenience methods, 41

Installer plug-in, 82

installing

- development profiles, 17-19
- Xcode developer suite, 11

instance methods, 27

instance variables, 27

encapsulating, 138

instantiation, 26-27

**Instruments**

- interface, 534-536
- memory leaks
- additional information, retrieving, 540
- checking for, 537-539
- multiple runs, collecting, 541-542
- simultaneous data collection, 543-545

**interface, 96**

- Instruments, 534-536
- Navigator, 97-98

OS X

- browser component, adding, 382-387
- code, attaching, 364-369
- creating, 359-364
- external code, enabling communication with QuartzGrough instance, 374-375
- object references, inserting in code, 369-373
- Xcode, 13, 88

**Interface Builder, 166**

- connection code, writing, 205-211
- custom objects, adding, 192-193
- interfaces, creating, 174

layout tools

- alignment, 179
- Auto Layout system, 183-188

How can we make this index more useful? Email us at indexes@samspublishing.com
Interface Builder

guides, 177-178
selection handles, 178
Size Inspector, 179-182
object identities, setting, 191-192
OS X interface, 167
creating, 359-364
storyboards, 172-174
loading, 168
XIB files, 168, 170
interface objects, 170-171
placeholder objects, 169-170
Interface Builder Editor, 62
interface development, comparing Mac OS X and iOS, 6
interface files, 28
#import directive, 29
@interface directive, 29-30
@property directive, 31-33
ending, 33
methods, defining, 30-31
Interface Inspector
Attributes Inspector, 188-189
Simulate Document command, 189-190
interface objects, 170-171
interpreting Quick Help results, 159-160
IOKit Driver template, 82
iOS applications
ad hoc distribution, 580
application logic, updating, 414-430
building from template, 403-404
converting from OS X applications, 400-402
distribution build, finalizing, 572-576
enterprise distribution, 581-582
projects, configuring for workspaces, 524-529
static target library, adding, 404-413
iOS distribution certificates
creating, 550-554
requesting through Provisioning Portal, 554-556
iOS Empty template, 83
iOS project templates, 72
Application templates
Master-Detail template, 73
OpenGL Game template, 74
iOS simulator, 271-272
applications, launching, 272-273
features, 275-276
multitouch events, generating, 274
simulated device, rotating, 274
Issue Navigator, errors, 133
fixing, 135
jumping to, 134
iTunes Connect Records, configuring, 576-578

J-K
Java, 319-320
joining paid developer program, 9-11
jump bar (Source Editor), 127
jumping to errors, Issue Navigator, 134
JVM (Java Virtual Machine), 320
Keychain Access, 552-554

L
languages
built-in languages
AppleScript, 317-319
C, 313-315
C++, 315-316
Java, 319-320
Objective-C, 316-317
Perl, 321-322
Objective-C, 311
Python, adding support, 322-323
Ruby, adding support, 324-325
selecting, 312-313
unsupported, 325
launching applications in iOS simulator, 272-273
layout of Assistant Editor, modifying, 132
layout tools (IB)
  alignment, 179
  Auto Layout system, 183-188
  guides, 177-178
  selection handles, 178
  Size Inspector, 179-182

libraries
  C/C++, reusing code, 349-352
  dynamic, versus frameworks, 330
  static, 330
  creating, 333-342

library calls for iOS applications, updating, 414-430
lldb, 490
LLVM, 49

loading
  projects into repositories, 291-294
  storyboards, 168
local Git repositories, creating, 286-287
locating files, 107

M
Mac OS X
  application access, comparing with iOS, 7
  development, 6
  frameworks, 6
  interface development, 6
  project templates, 78
  Automator Action template, 81
  Cocoa Application template, 79
  Cocoa Framework template, 80
  Cocoa-AppleScript Application template, 79
  Command Line Tool template, 80
  XPC Services template, 81
  macros, testing code functionality with STAssert, 474-480
  man pages, xcode select tool, 589
  managed objects, 437
  managing
    projects
      commits, 298-299
      pulls, 299-300
      pushes, 298-299
      revisions, viewing, 301-304
      status codes, 297
      updates, 299-300
      snapshots, 282
    manoeuvring paused applications, 498-499
  Master-Detail application template, 73
  memory leaks
    additional information, retrieving, 540
    checking for with Instruments, 537-539
    multiple runs, performing with Instruments, 541-542
  memory management
    ARC, 49
    objects, releasing, 48-49
    merging, 284-285, 306-307
    messaging, 27, 41-42
      blocks, 43
      nested messaging, 42-43
    methods
      convenience methods, 41
      defing, 30-31
      implementing, 35-36
      prepareForSegue:sender method, 228-229
    modal segues, dismissing, 227
    modal views, 217
    models, 59-60
      Data model, 64-65
    modifying Assistant Editor
      layout, 132
      multiple runs, collecting in Instruments, 541-542
    multiscene projects, 216
      scenes
        adding to storyboard, 218-219
        naming, 219-220
    multitouch events, generating in iOS simulator, 274
  MVC (model-view-controller), 57-59
    controllers, 62, 67-68
    IBAction directive, 63-64
    IBOutlet directive, 63
    Data model, 64-65

How can we make this index more useful? Email us at indexes@samspublishing.com
MVC (model-view-controller)

design patterns, 58
models, 60
views, 61, 66

N
naming scenes, 219-220
navigating Xcode workspace, 15-17
navigation controllers
example, 239-248
segues, creating, 230-234
Navigator, 97-98
nested messaging, 42-43
NeXTSTEP/OpenStep
development tools, 4
NSManagedObject subclasses,
creating, 458-460
numbers, Cocoa, 53

O
object data types, 40
object identities, setting in IB, 191-192
Object Library, 174
object references, inserting in OS
X interface code, 369-373
Objective-C, 25-28, 311, 316-317
categories, defining, 37
class structure, 36
expressions, 44-45
if-then-else statements, 45
messaging, 41-42
blocks, 43
nested messaging, 42-43
objects
initializing, 40-41
releasing, 48-49
protocols, creating, 38
switch statements, 45
variables, declaring, 38-40
implementation files
#import directive, 34
@interface directive, 29-30
@property directive, 31-33
ending, 33
methods, defining, 30-31

objects
convenience methods, 41
initializing, 40
releasing, 48-49
OCUnit framework, 465
online documentation, 146
OOP (object-oriented
programming), 24
inheritance, 25
Objective-C, 25-28
ARC, 49
categories, defining, 37
class structure, 36
expressions, 44-45
if-then-else statements, 45
implementation files, 34-36
interface files, 28-33
messaging, 41-43
objects, initializing, 40-41
protcols, creating, 38
releasing objects, 49
switch statements, 45
variables, declaring, 45

opendiff, 599
OpenGL Game template, 74
OS X applications
building, 356-358
converting to iOS applica-
tions, 400-402
distribution build, finalizing, 568-572
embedded frameworks,
adding, 388-395
interface
browser component,
adding, 382-387
building, 359-364
code, attaching, 364-369
external code, enabling
communication with
QuartzGrough instance, 374-375
object references,
inserting in code, 369-373
Interface Builder, 167
projects, configuring for work-
spaces, 519-524
shared frameworks, adding, 376-382
signed applications, distributing, 580
unsigned applications, distributing, 579
outlets, 195-196
connections, creating, 198-200
creating, 206-208

P
PackageMaker, 599
paid developer programs, 8
joining, 9-11
registration, 9-10
panes, Instruments
application, 534
parameters, 27
parents, 26
paused applications
continuing, 499
maneuvering, 498-499
stepping forward, 500-502
Perl, 321-322
placeholder objects, 169-170
pointers, 40
pop, 230
pragma marks, 131
prebuilt actions, 203
predicates, 437
prepareForSegue:sender method, 228-229
preparing iOS device for Xcode
development, 17
primitive data types, 39
proactive debugging, 493-496
procedural programming, 24
productbuider, 599
products, 253
productsign, 599
Profile action, 255, 268
program execution
continuing, 499
stepping forward, 500-502
programming
Cocoa, 50
arrays, 52
dates, 53
dictionaries, 53
numbers, 53
strings, 52
URLs, 54
versus Cocoa Touch, 51
imperative development, 24
language, selecting, 312-313
Objective-C, 25-28
ARC, 49
categories, defining, 37
class structure, 36
expressions, 44-45
if-then-else statements, 45
implementation files, 34-36
interface files, 28-33
messaging, 41-43
objects, initializing, 40-41
protocols, creating, 38
releasing objects, 49
switch statements, 49
variables, declaring, 38-40
project files, 88
Project Navigator, 97-98
Build Phases tab, 263
Build Rules tab, 264
Build Settings tab, 263
files, renaming, 106
filters, 98
templates, adding, 99-103
projects
adding to workspaces, 517-519
commits, 298-299
configuration, reviewing, 89
creating, 13, 83
template, configuring, 85-87
template, saving, 87
template, selecting, 84
files
adding, 105-106
locating, 107
removing, 106-107
frameworks
adding, 108-109
headers, 109
removing, 110
groups
adding, 110-111
removing, 111
loading into repositories
Git, 293-294
Subversion, 291-293

How can we make this index more useful? Email us at indexes@samspublishing.com
products, 253
pulls, 299-300
pushes, 298-299
revisions, viewing, 301-304
schemes, adding, 270-271
status codes, 297
targets, 252
  build phases, 263
  build rules, 264
  build settings, 263
  case study, 260-262
  creating, 257-258
  properties, 111-113
templates, building, 90
updating, 299-300
properties, 27, 195-196, 437
  attributes, 33
  of targets, 111-113
protocols, 27
  creating, 38
Provisioning Portal
  App IDs, creating, 556-560
  distribution certificates, requesting, 554-556
provisioning profiles, 17, 550
  creating, 552-554
  distribution profiles, creating, 560-563
  installing, 17-19
  repairing, 551-552
pulls, 299-300
push segues, creating, 243
pushes, 230, 298-299
Python, adding support for, 322-323

Q
Quartz Composer plug-in, 82
Quick Help, 147
  accessing, 158
  results, interpreting, 159-160
Quick Help Inspector
  activating, 160
  connections, editing, 204
Quick Look plug-in, 83

R
reading Core Data model data, 461-462
refactoring
  code, extracting, 137
  instance variables, encapsulating, 138
  symbols, renaming, 135-136
reference documents, 152
registration, Apple Developer Registration Portal, 9-10
relationships, 217, 437
  defining in Core Data model editor, 443-444
release configurations, testing, 470
releasing objects, 48-49
remote repositories
  connecting to, 287-290
  Git, loading projects, 293-294
  Subversion, loading projects, 291-293
removing
  files, 106-107
  frameworks from projects, 110
  groups from projects, 111
renaming
  files, 106
  symbols, 135-136
repairing distribution certificates, 551-552
repositories, 283
  connecting to, 287-290
  Git
    creating, 286-287
    projects, loading into, 293-294
    Subversion, loading projects into, 291-293
requesting distribution certificates through Provisioning Portal, 554-556
restoring snapshots, 280-282
retaining objects, 48-49
reusability
  C/C++ code, 349-352
  code, selecting between frameworks and static libraries, 331-333
reviewing project configuration, 89
revisions (source control), viewing, 301-304
rotating simulated devices in iOS simulator, 274
Ruby, adding support for, 324-325
Run action, 267
Run build action, 254
sample code, 153
saving templates, 87
scenes, 217
   adding to storyboard, 218-219
   multiscene projects, 216
   naming, 219-220
   segues
      advanced segues, creating, 230-239
      configuring, 225-227
      creating, 223-225
      dismissing, 227
      push segues, creating, 243
   creating, 243
   sharing information between, 228-230
   supporting view controller subclasses, adding, 220-223
schema, 437
Scheme Editor
   Analyze action, 268
   Archive action, 268
   Build action, 266
   Profile action, 268
   Run action, 267
   Test action, 267
schemes, 256
   adding, 270-271
SDKs, displaying with xcodebuild tool, 592-595
Search Navigator, 129-130
searching Xcode Help viewer library, 157
segmented controls, 198
segues, 217
   advanced segues, creating navigation controllers, 230-234
   tab bar controllers, 234-239
   configuring, 225-227
   creating, 223-225
   dismissing, 227
   push segues, creating, 243
   creating, 243
   sharing information between, 228-230
   preparingForSegue:sender method, 228-230
   supporting view controller subclasses, adding, 220-223
settings
   breakpoints, 497-498
   object identities (IB), 191-192
shared frameworks, adding to OS X applications, 376-382
sharing information between scenes, 228-230
signed applications (OS X), distributing, 580
Simulate Document command (IB), 189-190
simultaneous data collection, performing with Instruments, 543-545
singletons, 26
Size Inspector (IB), 179-182
snapshots, 279
   auto snapshots, 282
   creating, 280
   exporting, 282
   managing, 282
   restoring, 280-282
   viewing, 280-282
snippets
   adding, 140-141
   editing, 141
   viewing, 138
source control
   branching, 284, 304-306
   changes, 284
   commits, 298-299
   merging, 284-285, 306-307
   pulls, 299-300
   pushes, 298-299
   repositories, 283
      connecting to, 287-290
      Git, creating, 286-287
      projects, loading into, 291-294
      revisions, viewing, 301-304
      snapshots, 279
      auto snapshots, 282
      creating, 280
      exporting, 282
      managing, 282
      restoring, 280-282
      viewing, 280-282
status codes, 297
trunks, 284
updates, 299-300
source control

version control systems, 283
working copies, 283
creating, 295-296
Source Editor, 117
auto-indentation, 120-121
balancing delimiters, 122-123
code completion, 118-120
code folding, 123-124
features, 125-126
jump bar, 127
pragma marks, 131
Search Navigator, 129-130
Symbol Navigator, 128
tabbed editing, 126-127
spaghetti code, 59
Spotlight Importer, 83
STAssert macros, testing code functionality, 474-480
static libraries, 330
creating, 333-342
selecting, 331-333
target, adding to iOS application, 404-413
versus dynamic libraries, 330
status codes, 297
stepping forward (debuggers), 500-502
STL C++ Library, 81
storyboards, 167-174, 215-217
navigation controllers
advanced segues, creating, 230-234
example, 239-248
scenes
adding, 218-219
multiscene projects, 216
naming, 219-220
segues, configuring, 225-227
segues, creating, 223-225
segues, dismissing, 227
sharing information between, 228-230
supporting view controller subclasses, adding, 220-223
tab bar controllers, creating advanced segues, 234-239
strings, Cocoa, 52
Stroustrup, Bjarne, 315
subclasses, 26
NSManagedObject, creating, 458-460
view controller subclasses, adding to scenes, 220-223
Subversion, 283
branching, 304-306
commits, 298-299
merging, 306-307
pulls, 299-300
pushes, 298-299
repositories, loading projects into, 291-293
status codes, 297
updates, 299-300
working copies, creating, 295-296
superclasses, 26
creating, 137
switch statements, 45
Symbol Navigator, 128
symbols, renaming, 135-136
syntax
expressions, 44
messaging, 41-42
tab bar controllers, creating segues, 234-239
tabbed editing, 126-127
targets, 252
build phases, 263
build rules, 264
build settings, 263
case study, 260-262
creating, 257-258
products, 253
properties, 111-113
technical notes, 154
technical Q&As, 154
templates
adding, 99-103
building, 90
configuring, 85-87
customizing, 91
File Template Library, 103-105
iOS applications, building, 403-404
iOS project templates, 72
Master-Detail application template, 73
OpenGL Game template, 73-74
Mac OS X project templates, 78
Automator Action template, 81
Cocoa Application template, 79
Cocoa Framework template, 80
Cocoa0-AppleScript Application template, 79
Command Line Tool template, 80
XPC Services template, 81
saving, 87
selecting, 84
Test action, 267
Test build action, 255
test-driven development, 465
unit testing, 465-471
Bundle Loader, 480-483
tests, implementing for existing code, 472-480
unpaid developer programs, 8
unsigned applications (OS X), 579
unsupported languages, 325
updates (source control), 299-300
updating iOS application logic, 414-430
URLs, Cocoa, 54
user interfaces
creating, 174
data models, binding, 446-458
Instruments, 534-536
Navigator, 97-98
variables, 27
declaring, 38
object data types, 40
primitive data types, 39
verifying build configurations, 256
version control systems, 283
branching, 284, 304-306
changes, committing, 284
Git, creating repositories, 286-287
merging, 284-285, 306-307
repositories, 283
connecting to, 287-290
loading projects to, 291-294
revisions, viewing, 301-304
trunks, 284
working copies, 283
creating, 295-296
Version editor, viewing revisions, 301-304
videos, how-to, 155
view controllers, 216
navigation controllers
advanced segues, 239-248
subclasses, adding to scenes, 220-223
tab bar controllers, creating advanced segues, 234-239
viewing
code snippets, 138
revisions, 301-304
snapshots, 280-282
views, 59-61, 66, 216
modal views, 217
web-based documentation, 146
welcome screen (Xcode), 13
working copies, 283
creating, 295-296

How can we make this index more useful? Email us at indexes@samspublishing.com
workspaces, 509-514
creating, 515-517
iOS projects, configuring, 524-529
navigating, 15-17
OS X projects, configuring, 519-524
projects, adding, 517-519
xcodebuild tool, 591
writing
connection code with Interface Builder, 205-211
data to Core Data store, 460-461

X-Y-Z
Xcode developer suite, installing, 11
Xcode Help viewer, 146
bookmarks, adding, 156
library
browsing, 156
searching, 157
Xcode tasks, 154
xcode-select tool, 588-589
xcodebuild, 590-591
available SDKs, displaying, 592-595
workspaces, 591
xcrun, 595-597
XIB files, 168-170
interface objects, 170-171
loading, 168
placeholder objects, 169-170
XPC services template, 81