John Ray
William Ray

In **Full Color**

Figures and code appear as they do in Xcode 4.3+

Covers iOS and OS X development topics—including MVC, Interface Builder, Storyboards, Core Data, and much more!

Additional files and updates available online

Sams **Teach Yourself**

Xcode® 4 in 24 Hours

SAMS

John Ray
William Ray

Sams Teach Yourself Xcode® 4 in 24 Hours

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

Sams Teach Yourself Xcode® 4 in 24 Hours

Copyright © 2012 by Pearson Education, Inc.

ISBN-13: 978-0-672-33587-7
ISBN-10: 0-672-33587-5

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

Printed in the United States of America

First Printing June 2012

Editor-in-Chief
Greg Wiegand

Acquisitions Editor
Laura Norman

Development Editor
Keith Cline

Managing Editor
Kristy Hart

Project Editor
Andy Beaster

Copy Editor
Keith Cline

Indexer
Tim Wright

Proofreader
Chrissy White

Technical Editor
Greg Kettell

Publishing Coordinator
Cindy Teeters

Book Designer
Gary Adair

Compositor
Nonie Ratcliff

Contents at a Glance

Table of Contents

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.

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 comfort of your desktop Mac or iPad.

The material in this book specifically targets Xcode 4.3 and later. If you are running an earlier 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:

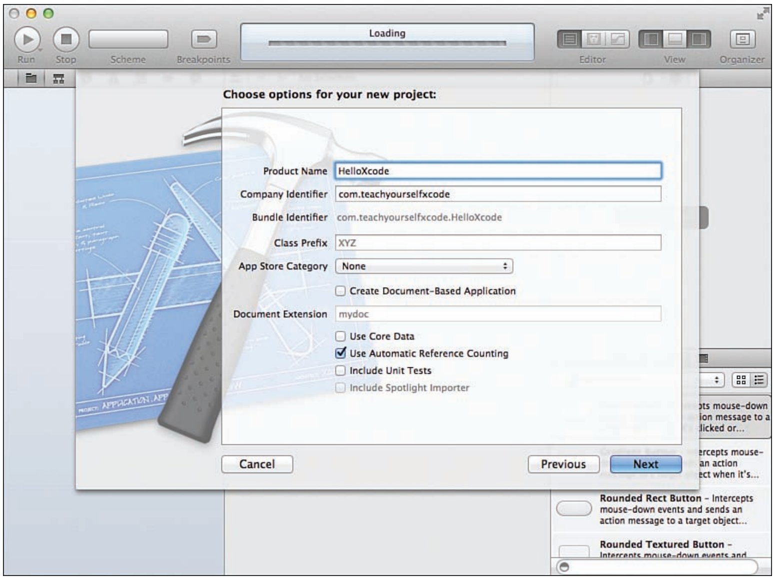
- Xcode Source Editor basics
- How to use autoformatting features
- The use of pragma marks to identify different pieces of code
- Ways to find and correct errors before your app even runs
- How to refactor existing code

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 you 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.



Choose options for your new project:

Product Name HelloXcode
Company Identifier com.teachyourselfxcode
Bundle Identifier com.teachyourselfxcode.HelloXcode
Class Prefix XYZ
App Store Category None
☐ Create Document-Based Application
Document Extension mydoc
☐ Use Core Data
☑ Use Automatic Reference Counting
☐ Include Unit Tests
☐ Include Spotlight Importer

Rounded Rect Button – Intercepts mouse-down events and sends an action message to a target object...

Rounded Textured Button –

FIGURE 6.1
Create a new Mac OS X Cocoa application.

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

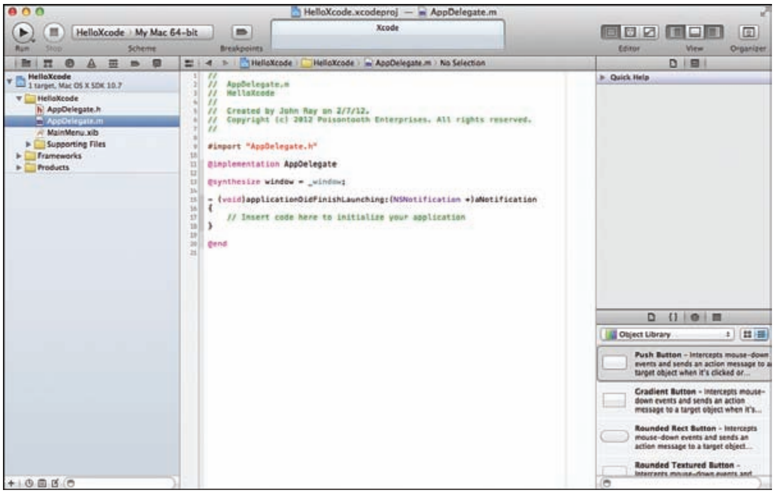
Using the Source Editor, start entering the following text to implement the `applicationDidFinishLaunching` method. Start a new line immediately following the existing comment "Insert code here to initialize your application." Update the method as shown in Listing 6.1.

LISTING 6.1 A Short Sample Mac OS X Application

```
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
{
    // Insert code here to initialize your application
    NSTextField *myMessage;
    NSTextField *myUnusedMessage;
    myMessage=[[NSTextField alloc] init];
```

LISTING 6.1 Continued

```
    myMessage.font=[NSFont systemFontOfSize:72.0];
    myMessage.stringValue=@"Hello Xcode";
    myMessage.textColor=[NSColor blueColor];
    myMessage.editable=NO;
    [self.window setContentView:myMessage];
}
```



```
//
//  AppDelegate.m
//  HelloXcode
//
//  Created by John Ray on 2/7/12.
//  Copyright (c) 2012 Poisontooth Enterprises. All rights reserved.
//

#import "AppDelegate.h"

@implementation AppDelegate

@synthesize window = _window;

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
{
    // Insert code here to initialize your application
}

@end
```

Push Button - Intercepts mouse-down events and sends an action message to a target object when it's clicked or...

Gradient Button - Intercepts mouse-down events and sends an action message to a target object when it's...

Rounded Rect Button - Intercepts mouse-down events and sends an action message to a target object...

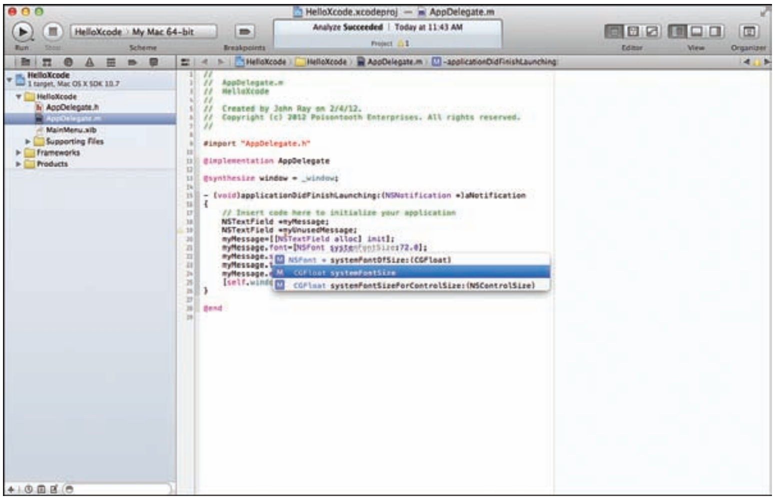
Rounded Textured Button - Intercepts mouse-down events and

FIGURE 6.2 Choose a file to edit.

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.


HelloXcode.xcodeproj — AppDelegate.m
Analyze Succeeded | Today at 11:43 AM
HelloXcode > My Mac 64-bit
Run
Stop
Scheme
Breakpoints
Editor
View
Organizer
HelloXcode > HelloXcode > AppDelegate.m > -applicationDidFinishLaunching:
HelloXcode
1 target, Mac OS X SDK 10.7
HelloXcode
AppDelegate.h
AppDelegate.m
MainMenu.xib
Supporting Files
Frameworks
Products

```
//
//  AppDelegate.m
//  HelloXcode
//
//  Created by John Ray on 2/4/12.
//  Copyright (c) 2012 Poisontooth Enterprises. All rights reserved.
//

#import "AppDelegate.h"

@implementation AppDelegate

@synthesize window = _window;

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
{
    // Insert code here to initialize your application
    NSTextField *myMessage;
    NSTextField *myUnusedMessage;
    myMessage=[[NSTextField alloc] init];
    myMessage.font=[NSFont systemFontSize:72.0];
    myMessage.s
    myMessage.t
    myMessage.e
    [self.wind
}

@end
```

```
M  NSFont * systemFontOfSize:(CGFloat)
M  CGFloat systemFontSize
M  CGFloat systemFontSizeForControlSize:(NSControlSize)
```

FIGURE 6.3
Xcode autocompletes recognized methods, classes, and symbols.

By the Way

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.

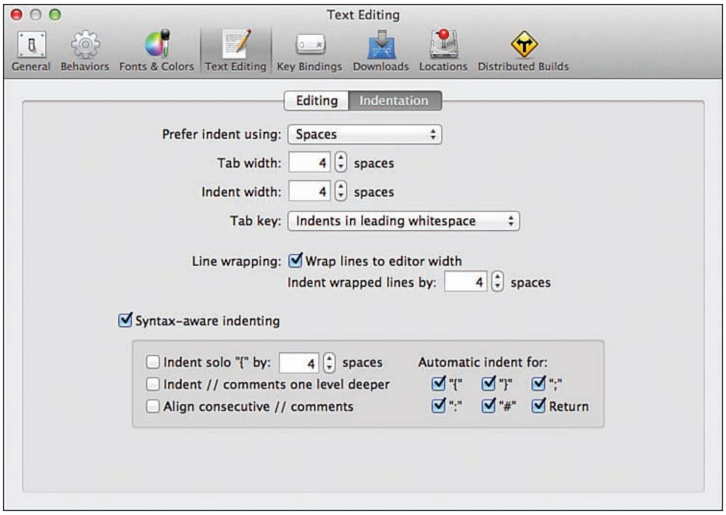
You can adjust the code completion options in the Xcode preferences within the Text Editing section.

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-indention. 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.



Prefer indent using: Spaces

Tab width: 4 spaces

Indent width: 4 spaces

Tab key: Indents in leading whitespace

Line wrapping: ✓ Wrap lines to editor width

Indent wrapped lines by: 4 spaces

✓ Syntax-aware indenting

☐ Indent solo "{" by: 4 spaces

☐ Indent // comments one level deeper

☐ Align consecutive // comments

Automatic indent for: ✓ "{" ✓ "}" ✓ ";" ✓ ":" ✓ "#" ✓ Return

FIGURE 6.4 Configure the Xcode indentation logic.

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.

By the Way

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.

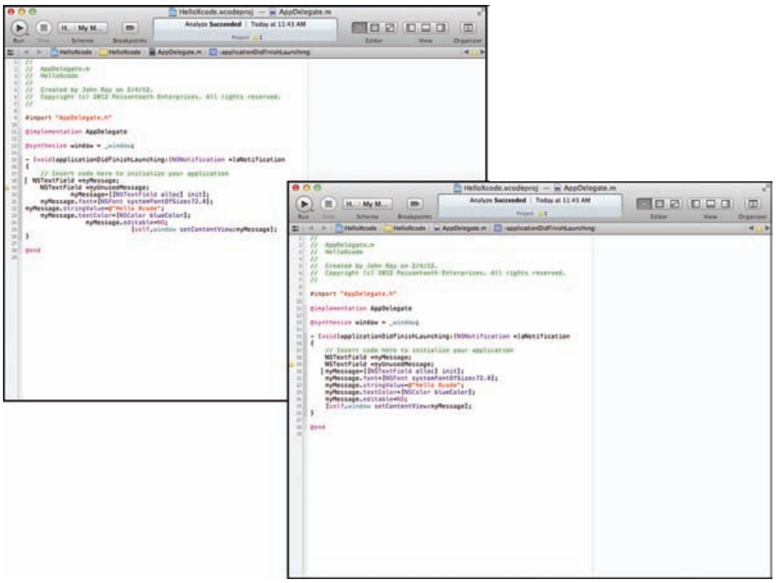


FIGURE 6.5
Before and after applying the indentation logic.

```
//
//  AppDelegate.m
//  HelloXcode
//
//  Created by John Ray on 2/4/12.
//  Copyright (c) 2012 Poisontooth Enterprises. All rights reserved.
//

#import "AppDelegate.h"

@implementation AppDelegate

@synthesize window = _window;

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
{
    // Insert code here to initialize your application
| NSTextField *myMessage;
    NSTextField *myUnusedMessage;
            myMessage=[[NSTextField alloc] init];
    myMessage.font=[NSFont systemFontOfSize:72.0];
myMessage.stringValue=@"Hello Xcode";
    myMessage.textColor=[NSColor blueColor];
                myMessage.editable=NO;
                        [self.window setContentView:myMessage];
}

@end
```

```
//
//  AppDelegate.m
//  HelloXcode
//
//  Created by John Ray on 2/4/12.
//  Copyright (c) 2012 Poisontooth Enterprises. All rights reserved.
//

#import "AppDelegate.h"

@implementation AppDelegate

@synthesize window = _window;

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
{
    // Insert code here to initialize your application
    NSTextField *myMessage;
    NSTextField *myUnusedMessage;
   |myMessage=[[NSTextField alloc] init];
    myMessage.font=[NSFont systemFontOfSize:72.0];
    myMessage.stringValue=@"Hello Xcode";
    myMessage.textColor=[NSColor blueColor];
    myMessage.editable=NO;
    [self.window setContentView:myMessage];
}

@end
```

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:

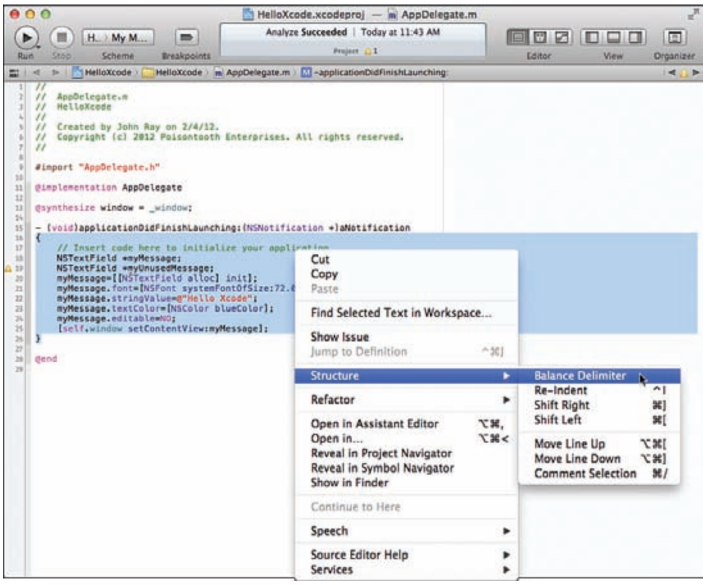
()	Parentheses for function calls
[]	Square brackets for Objective-C messaging
{}	Curly brackets for logical programming blocks

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.



```
//
//  AppDelegate.m
//  HelloXcode
//
//  Created by John Ray on 2/4/12.
//  Copyright (c) 2012 Poisontooth Enterprises. All rights reserved.
//

#import "AppDelegate.h"

@implementation AppDelegate

@synthesize window = _window;

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
{
    // Insert code here to initialize your applicat
    NSTextField *myMessage;
    NSTextField *myUnusedMessage;
    myMessage=[[NSTextField alloc] init];
    myMessage.font=[NSFont systemFontOfSize:72.
    myMessage.stringValue=@"Hello Xcode";
    myMessage.textColor=[NSColor blueColor];
    myMessage.editable=NO;
    [self.window setContentView:myMessage];
}

@end
```

FIGURE 6.6
Select the code between two delimiters.

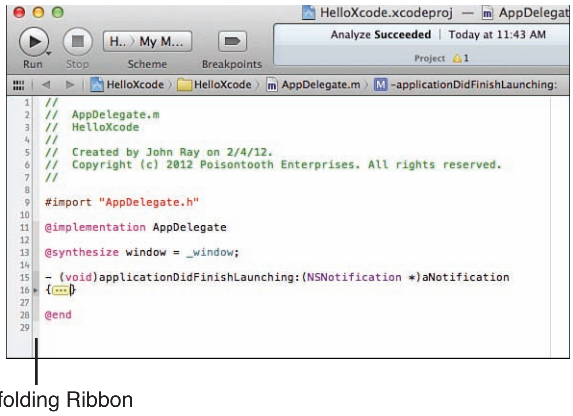
By the Way

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).

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.


HelloXcode.xcodeproj — AppDelegat
Run
Stop
H.. 〉 My M...
Scheme
Breakpoints
Analyze Succeeded | Today at 11:43 AM
Project 1
HelloXcode 〉 HelloXcode 〉 AppDelegate.m 〉 -applicationDidFinishLaunching:
Code-folding Ribbon

```
//
//  AppDelegate.m
//  HelloXcode
//
//  Created by John Ray on 2/4/12.
//  Copyright (c) 2012 Poisontooth Enterprises. All rights reserved.
//

#import "AppDelegate.h"

@implementation AppDelegate

@synthesize window = _window;

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
{...}

@end
```

FIGURE 6.7
Collapse your code to improve your focus.

By the Way

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.

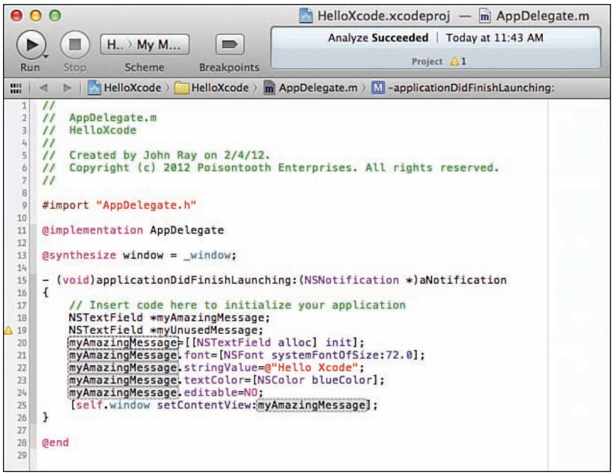
Line Numbers

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.


HelloXcode.xcodeproj — AppDelegate.m
Run
Stop
H.. › My M...
Scheme
Breakpoints
Analyze Succeeded | Today at 11:43 AM
Project ⚠1
HelloXcode › HelloXcode › AppDelegate.m › -applicationDidFinishLaunching:

```
//
//  AppDelegate.m
//  HelloXcode
//
//  Created by John Ray on 2/4/12.
//  Copyright (c) 2012 Poisontooth Enterprises. All rights reserved.
//

#import "AppDelegate.h"

@implementation AppDelegate

@synthesize window = _window;

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
{
    // Insert code here to initialize your application
    NSTextField *myAmazingMessage;
    NSTextField *myUnusedMessage;
    myAmazingMessage=[[NSTextField alloc] init];
    myAmazingMessage.font=[NSFont systemFontOfSize:72.0];
    myAmazingMessage.stringValue=@"Hello Xcode";
    myAmazingMessage.textColor=[NSColor blueColor];
    myAmazingMessage.editable=NO;
    [self.window setContentView:myAmazingMessage];
}

@end
```

FIGURE 6.8
Quickly change symbol names using Edit All in Scope.

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.

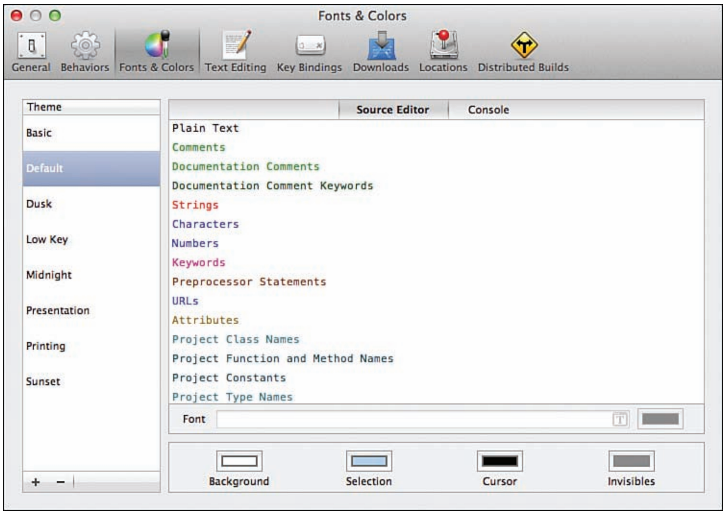


FIGURE 6.9
Change the syntax color rules to suit your sensibilities.

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 to want, as shown in Figure 6.10.

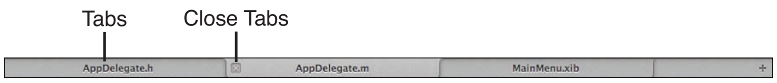


FIGURE 6.10 Keep multiple editors open simultaneously with tabs.

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.

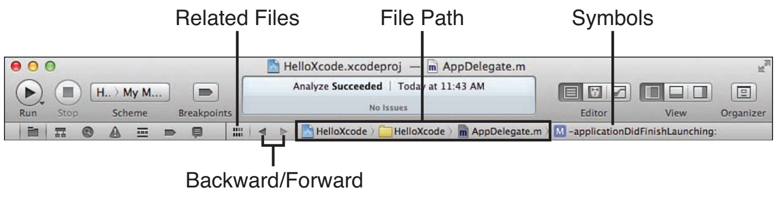
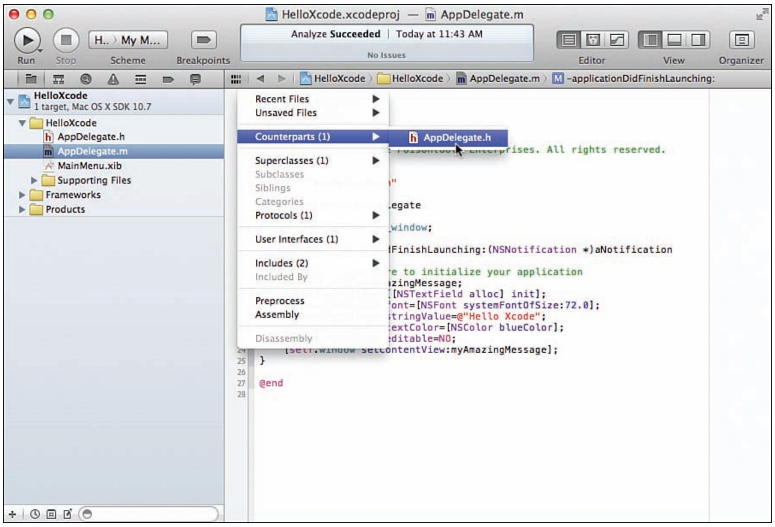


FIGURE 6.11 Quickly navigate your project hierarchy.

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.

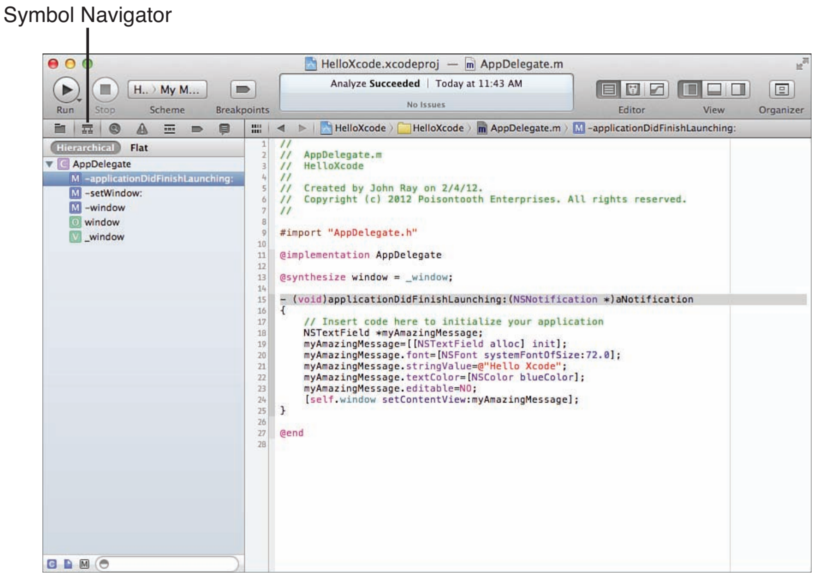


```
prises. All rights reserved.
"
.egate
_window;
dFinishLaunching:(NSNotification *)aNotification
e to initialize your application
zingMessage;
[NSTextField alloc] init];
font=[NSFont systemFontOfSize:72.0];
stringValue=@"Hello Xcode";
extColor=[NSColor blueColor];
editable=NO;
    [self.window setContentView:myAmazingMessage];
}

@end
```

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

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.


Symbol Navigator
HelloXcode.xcodeproj — AppDelegate.m
Run
Stop
H.. › My M...
Scheme
Breakpoints
Analyze Succeeded | Today at 11:43 AM
No Issues
Editor
View
Organizer
HelloXcode › HelloXcode › AppDelegate.m › -applicationDidFinishLaunching:
Hierarchical
Flat
AppDelegate
-applicationDidFinishLaunching:
-setWindow:
-window
window
_window

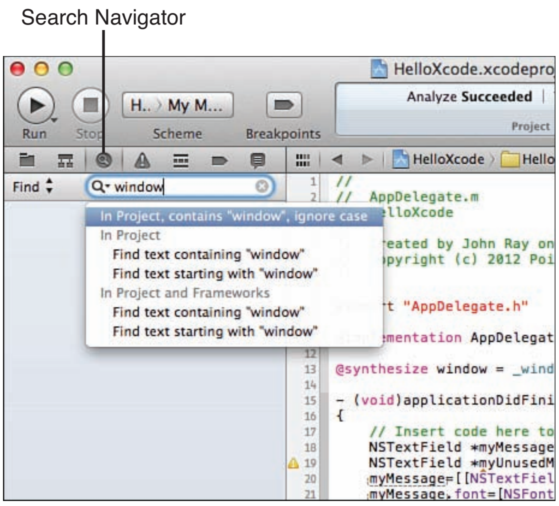
```
//
//  AppDelegate.m
//  HelloXcode
//
//  Created by John Ray on 2/4/12.
//  Copyright (c) 2012 Poisontooth Enterprises. All rights reserved.
//

#import "AppDelegate.h"

@implementation AppDelegate

@synthesize window = _window;

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
{
    // Insert code here to initialize your application
    NSTextField *myAmazingMessage;
    myAmazingMessage=[[NSTextField alloc] init];
    myAmazingMessage.font=[NSFont systemFontOfSize:72.0];
    myAmazingMessage.stringValue=@"Hello Xcode";
    myAmazingMessage.textColor=[NSColor blueColor];
    myAmazingMessage.editable=NO;
    [self.window setContentView:myAmazingMessage];
}

@end
```

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

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.

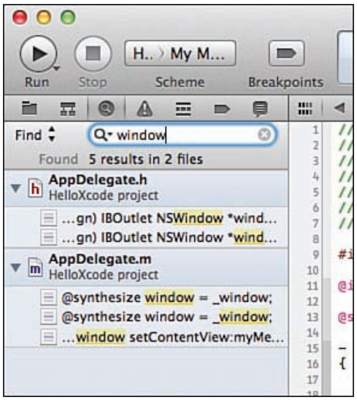

Search Navigator
HelloXcode.xcodepro
Run
Stop
H.. 〉My M...
Scheme
Breakpoints
Analyze Succeeded
Project
HelloXcode 〉Hello
Find
window
In Project, contains "window", ignore case
In Project
Find text containing "window"
Find text starting with "window"
In Project and Frameworks
Find text containing "window"
Find text starting with "window"

```
//
//  AppDelegate.m
   lloXcode

   eated by John Ray on
   pyright (c) 2012 Poi

   t "AppDelegate.h"

   mentation AppDelegat

@synthesize window = _wind

- (void)applicationDidFini
{
    // Insert code here to
    NSTextField *myMessage
    NSTextField *myUnusedM
    myMessage=[[NSTextFiel
    myMessage.font=[NSFont
```

FIGURE 6.14 Use the Search Navigator to find text in your project.

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.



window
Found 5 results in 2 files
AppDelegate.h
HelloXcode project
...gn) IBOutlet NSWindow *wind...
...gn) IBOutlet NSWindow *wind...
AppDelegate.m
HelloXcode project
@synthesize window = _window;
@synthesize window = _window;
...window setContentView:myMe...

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 -`

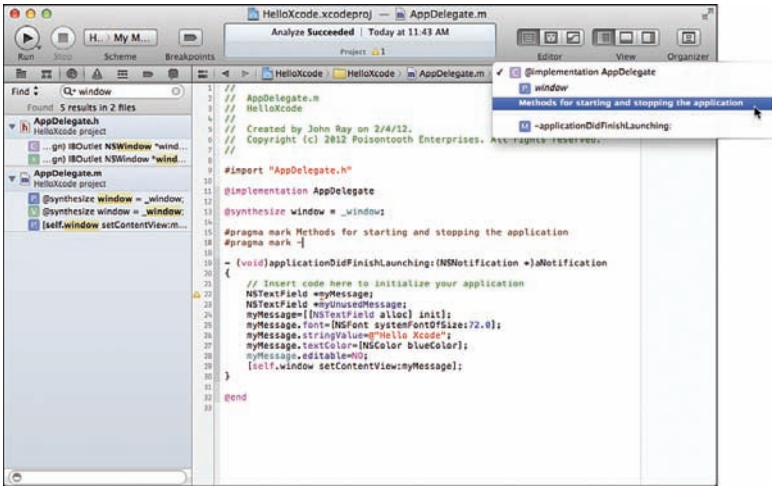
and

`#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.


HelloXcode.xcodeproj — AppDelegate.m
Analyze Succeeded | Today at 11:43 AM
Run
Stop
Scheme
Breakpoints
Editor
View
Organizer
HelloXcode
AppDelegate.m
✓ @implementation AppDelegate
window
Methods for starting and stopping the application
-applicationDidFinishLaunching:
Find
window
Found 5 results in 2 files
AppDelegate.h
AppDelegate.m

```
//
//  AppDelegate.m
//  HelloXcode
//
//  Created by John Ray on 2/4/12.
//  Copyright (c) 2012 Poisontooth Enterprises. All rights reserved.
//

#import "AppDelegate.h"

@implementation AppDelegate

@synthesize window = _window;

#pragma mark Methods for starting and stopping the application
#pragma mark -

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
{
    // Insert code here to initialize your application
    NSTextField *myMessage;
    NSTextField *myUnusedMessage;
    myMessage=[[NSTextField alloc] init];
    myMessage.font=[NSFont systemFontOfSize:72.0];
    myMessage.stringValue=@"Hello Xcode";
    myMessage.textColor=[NSColor blueColor];
    myMessage.editable=NO;
    [self.window setContentView:myMessage];
}

@end
```

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

Using the Assistant Editor

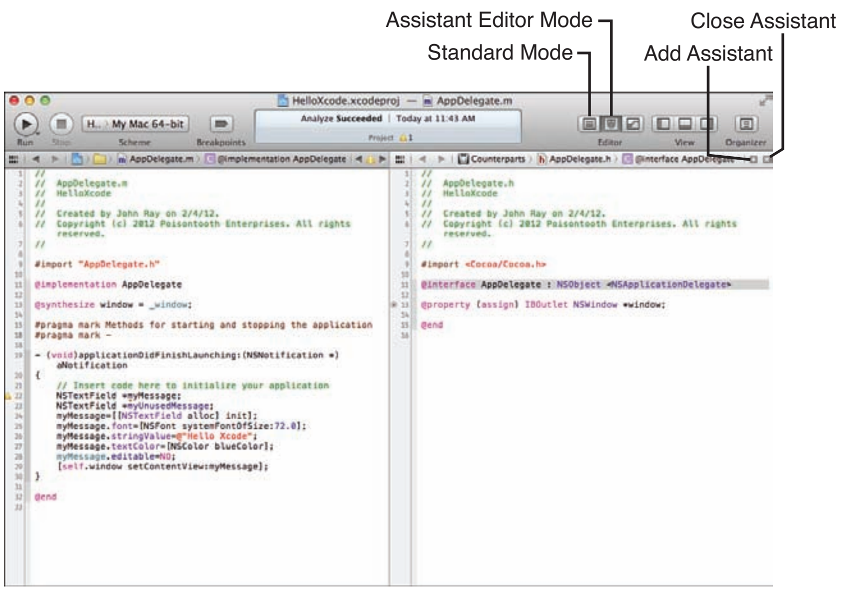
As you work with Xcode projects, you will quickly realize that most program functionality comes from editing one of two related files: an implementation file (.m extension) and an interface file (.h extension). You'll also learn that when you make changes to one of these two files, you'll often need 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.

By the Way

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.



```
//
//  AppDelegate.m
//  HelloXcode
//
//  Created by John Ray on 2/4/12.
//  Copyright (c) 2012 Poisontooth Enterprises. All rights
    reserved.
//

#import "AppDelegate.h"

@implementation AppDelegate

@synthesize window = _window;

#pragma mark Methods for starting and stopping the application
#pragma mark -

- (void)applicationDidFinishLaunching:(NSNotification *)
    aNotification
{
    // Insert code here to initialize your application
    NSTextField *myMessage;
    NSTextField *myUnusedMessage;
    myMessage=[[NSTextField alloc] init];
    myMessage.font=[NSFont systemFontOfSize:72.0];
    myMessage.stringValue=@"Hello Xcode";
    myMessage.textColor=[NSColor blueColor];
    myMessage.editable=NO;
    [self.window setContentView:myMessage];
}

@end
```

```
//
//  AppDelegate.h
//  HelloXcode
//
//  Created by John Ray on 2/4/12.
//  Copyright (c) 2012 Poisontooth Enterprises. All rights
    reserved.
//

#import <Cocoa/Cocoa.h>

@interface AppDelegate : NSObject <NSApplicationDelegate>

@property (assign) IBOutlet NSWindow *window;

@end
```

FIGURE 6.17
Xcode opens the file related to what you are working on.

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.

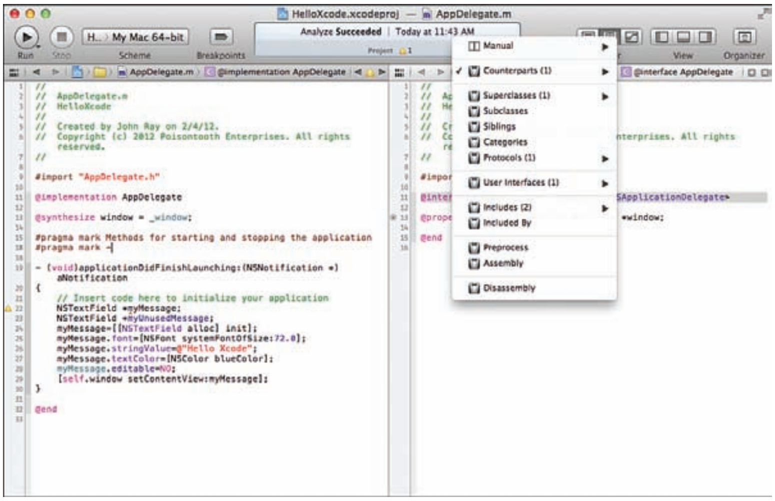
Changing the Assistant Editor Layout

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.



```
//
//  AppDelegate.m
//  HelloXcode
//
//  Created by John Ray on 2/4/12.
//  Copyright (c) 2012 Poisontooth Enterprises. All rights
    reserved.
//

#import "AppDelegate.h"

@implementation AppDelegate

@synthesize window = _window;

#pragma mark Methods for starting and stopping the application
#pragma mark -

- (void)applicationDidFinishLaunching:(NSNotification *)
    aNotification
{
    // Insert code here to initialize your application
    NSTextField *myMessage;
    NSTextField *myUnusedMessage;
    myMessage=[[NSTextField alloc] init];
    myMessage.font=[NSFont systemFontOfSize:72.0];
    myMessage.stringValue=@"Hello Xcode";
    myMessage.textColor=[NSColor blueColor];
    myMessage.editable=NO;
    [self.window setContentView:myMessage];
}

@end
```

FIGURE 6.18 Choose a different file to edit.

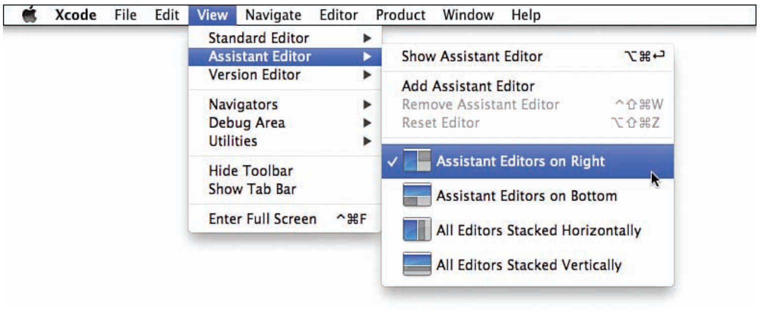


FIGURE 6.19 Configure how the Assistant Editors will appear in the display.

Correcting Errors and Warnings in the Issue Navigator

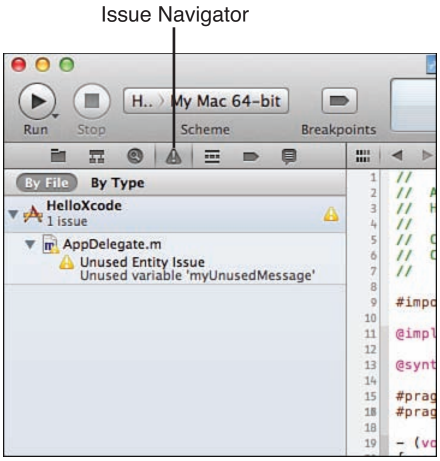
As you write your code, Xcode is sitting in the background judging you, but do not take it personally—it only wants to help. Xcode analyzes your code and detects issues that might be (or are) problems and displays warnings and errors before you ever click the Run button.

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.



Unused Entity Issue
Unused variable 'myUnusedMessage'

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

By the Way

Logic problems must be detected by choosing Product, Analyze from the menu bar. 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.

By the Way

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.

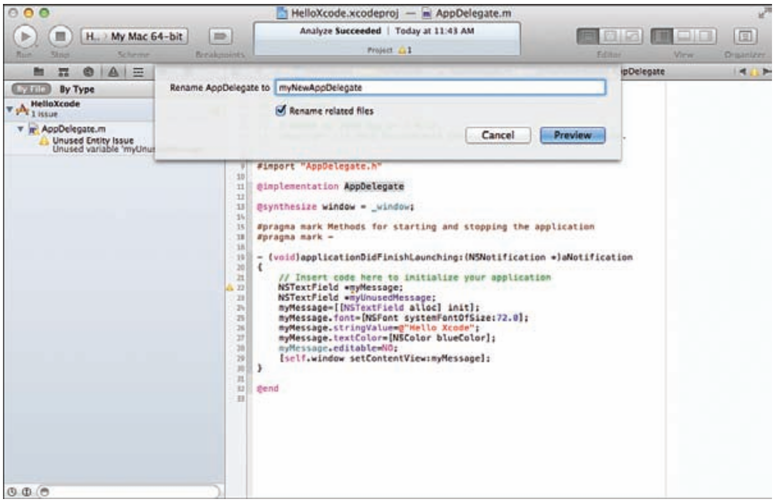
To access the refactoring tools, choose Edit, Refactor from the menu bar or right-click in your code to use the Refactor contextual menu item.

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.



Rename AppDelegate to myNewAppDelegate

✓ Rename related files

```
#import "AppDelegate.h"

@implementation AppDelegate

@synthesize window = _window;

#pragma mark Methods for starting and stopping the application
#pragma mark -

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
{
    // Insert code here to initialize your application
    NSTextField *myMessage;
    NSTextField *myUnusedMessage;
    myMessage=[[NSTextField alloc] init];
    myMessage.font=[NSFont systemFontOfSize:72.0];
    myMessage.stringValue=@"Hello Xcode";
    myMessage.textColor=[NSColor blueColor];
    myMessage.editable=NO;
    [self.window setContentView:myMessage];
}

@end
```

FIGURE 6.21
Rename a symbol (variable, class, method, etc. across your project).

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.

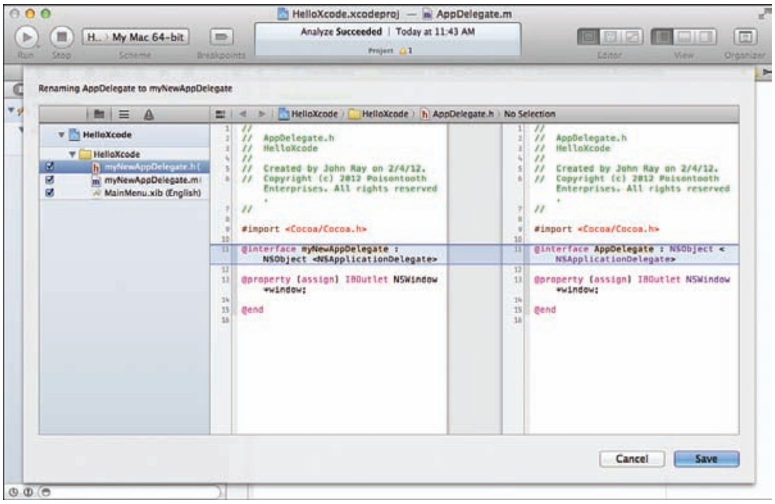
   From this window, you can choose which files to change (check or uncheck the boxes in front of their names) and even edit the code directly in the comparison views.

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.



Renaming AppDelegate to myNewAppDelegate

```
//
//  AppDelegate.h
//  HelloXcode
//
//  Created by John Ray on 2/4/12.
//  Copyright (c) 2012 Poisontooth
    Enterprises. All rights reserved
    .
//

#import <Cocoa/Cocoa.h>

@interface myNewAppDelegate :
    NSObject <NSApplicationDelegate>

@property (assign) IBOutlet NSWindow
    *window;

@end
```

```
//
//  AppDelegate.h
//  HelloXcode
//
//  Created by John Ray on 2/4/12.
//  Copyright (c) 2012 Poisontooth
    Enterprises. All rights reserved
    .
//

#import <Cocoa/Cocoa.h>

@interface AppDelegate : NSObject <
    NSApplicationDelegate>

@property (assign) IBOutlet NSWindow
    *window;

@end
```

HelloXcode.xcodeproj — AppDelegate.m
Analyze Succeeded | Today at 11:43 AM
H.. My Mac 64-bit
HelloXcode
myNewAppDelegate.h
myNewAppDelegate.m
MainMenu.xib (English)
HelloXcode > HelloXcode > AppDelegate.h > No Selection
Cancel
Save

FIGURE 6.22 Confirm the changes that will be made.

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.

Watch Out!

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.

By the Way

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

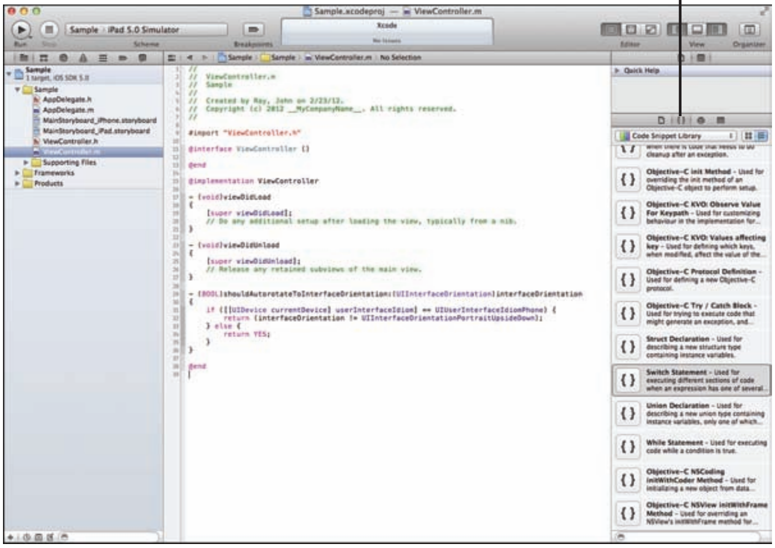
When you are writing code, you will often find yourself typing statements that are *very* similar to one another: `catch-try` blocks, `if-then` statements, `switch` statements, and so on. Instead of having to remember the syntax of each of these common code sequences, you can make use of the built-in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


Code Snippet Library
Sample.xcodeproj — ViewController.m
Sample › iPad 5.0 Simulator
Xcode
Sample › Sample › ViewController.m › No Selection
Quick Help
Code Snippet Library

```
//
//  ViewController.m
//  Sample
//
//  Created by Ray, John on 2/23/12.
//  Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//

#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
}

- (void)viewDidUnload
{
    [super viewDidUnload];
    // Release any retained subviews of the main view.
}

- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation
{
    if ([[UIDevice currentDevice] userInterfaceIdiom] == UIUserInterfaceIdiomPhone) {
        return (interfaceOrientation != UIInterfaceOrientationPortraitUpsideDown);
    } else {
        return YES;
    }
}

@end
```

cleanup after an exception.

Objective-C init Method - Used for overriding the init method of an Objective-C object to perform setup.

Objective-C KVO: Observe Value For Keypath - Used for customizing behaviour in the implementation for...

Objective-C KVO: Values affecting key - Used for defining which keys, when modified, affect the value of the...

Objective-C Protocol Definition - Used for defining a new Objective-C protocol.

Objective-C Try / Catch Block - Used for trying to execute code that might generate an exception, and...

Struct Declaration - Used for describing a new structure type containing instance variables.

Switch Statement - Used for executing different sections of code when an expression has one of several...

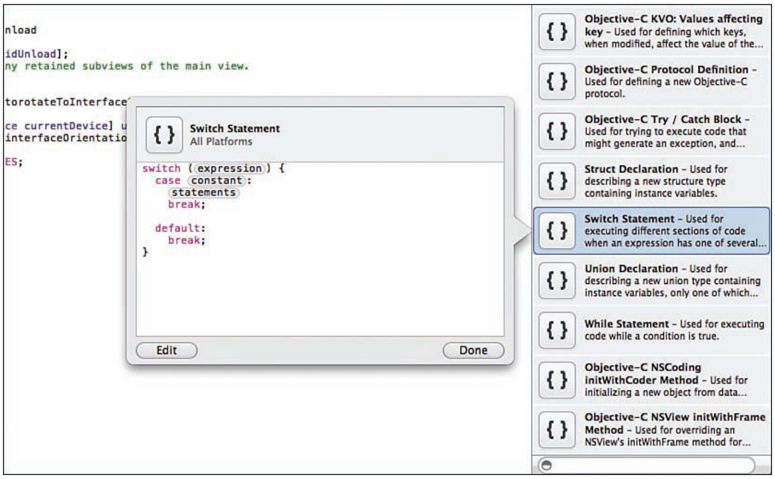
Union Declaration - Used for describing a new union type containing instance variables, only one of which...

While Statement - Used for executing code while a condition is true.

Objective-C NSCoding initWithCoder Method - Used for initializing a new object from data...

Objective-C NSView initWithFrame Method - Used for overriding an NSView's initWithFrame method for...

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



Switch Statement
All Platforms

```
switch (expression) {
    case constant:
        statements
        break;

    default:
        break;
}
```

Objective-C KVO: Values affecting key - Used for defining which keys, when modified, affect the value of the...

Objective-C Protocol Definition - Used for defining a new Objective-C protocol.

Objective-C Try / Catch Block - Used for trying to execute code that might generate an exception, and...

Struct Declaration - Used for describing a new structure type containing instance variables.

Switch Statement - Used for executing different sections of code when an expression has one of several...

Union Declaration - Used for describing a new union type containing instance variables, only one of which...

While Statement - Used for executing code while a condition is true.

Objective-C NSCoding initWithCoder Method - Used for initializing a new object from data...

Objective-C NSView initWithFrame Method - Used for overriding an NSView's initWithFrame method for...

FIGURE 6.24
View the contents of a snippet.

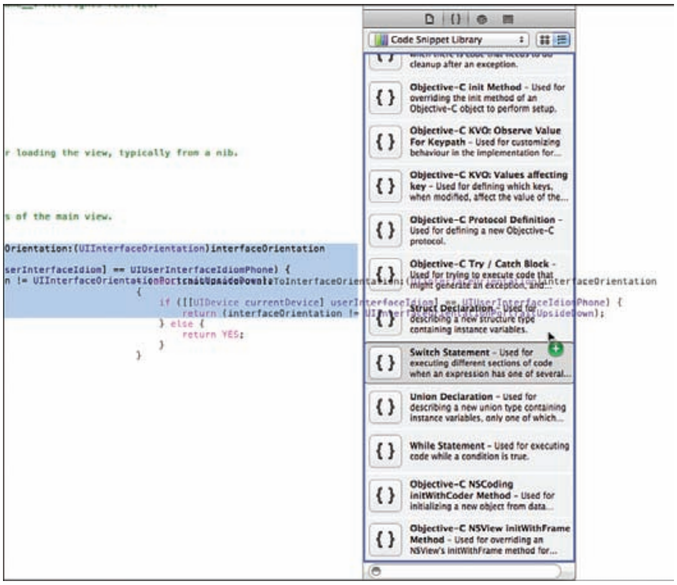
Inserting Snippets

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.

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.



```
r loading the view, typically from a nib.

s of the main view.

Orientation:(UIInterfaceOrientation)interfaceOrientation

serInterfaceIdiom] == UIUserInterfaceIdiomPhone) {
n != UIInterfaceOrientationPortraitUpsideDown);
                    {
                       if ([[UIDevice currentDevice] userInterfaceIdiom] == UIUserInterfaceIdiomPhone) {
                          return (interfaceOrientation != UIInterfaceOrientationPortraitUpsideDown);
                       } else {
                          return YES;
                       }
                    }
```

Code Snippet Library

cleanup after an exception.

Objective-C init Method – Used for overriding the init method of an Objective-C object to perform setup.

Objective-C KVO: Observe Value For Keypath – Used for customizing behaviour in the implementation for...

Objective-C KVO: Values affecting key – Used for defining which keys, when modified, affect the value of the...

Objective-C Protocol Definition – Used for defining a new Objective-C protocol.

Objective-C Try / Catch Block – Used for trying to execute code that might generate an exception, and...

Struct Declaration – Used for describing a new structure type containing instance variables.

Switch Statement – Used for executing different sections of code when an expression has one of several...

Union Declaration – Used for describing a new union type containing instance variables, only one of which...

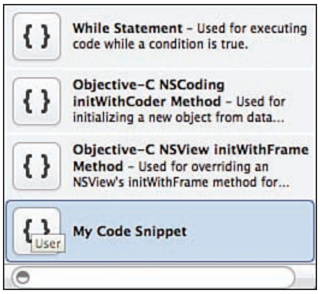
While Statement – Used for executing code while a condition is true.

Objective-C NSCoding initWithCoder Method – Used for initializing a new object from data...

Objective-C NSView initWithFrame Method – Used for overriding an NSView's initWithFrame method for...

FIGURE 6.25
Drag the code to the Code Snippet Library.

The text is added to the library under the title My Code Snippet, as shown in Figure 6.26.



While Statement – Used for executing code while a condition is true.

Objective-C NSCoding initWithCoder Method – Used for initializing a new object from data...

Objective-C NSView initWithFrame Method – Used for overriding an NSView's initWithFrame method for...

My Code Snippet

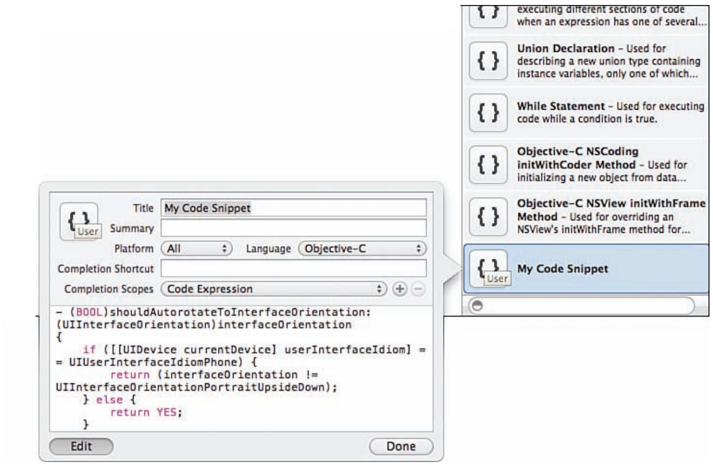
FIGURE 6.26
The new snippet is visible in the library.

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.



executing different sections of code when an expression has one of several...
Union Declaration – Used for describing a new union type containing instance variables, only one of which...
While Statement – Used for executing code while a condition is true.
Objective-C NSCoding initWithCoder Method – Used for initializing a new object from data...
Objective-C NSView initWithFrame Method – Used for overriding an NSView's initWithFrame method for...
My Code Snippet
User
Edit
Done

Title: My Code Snippet
Summary:
Platform: All Language: Objective-C
Completion Shortcut:
Completion Scopes: Code Expression

```
- (BOOL)shouldAutorotateToInterfaceOrientation:
(UIInterfaceOrientation)interfaceOrientation
{
    if ([[UIDevice currentDevice] userInterfaceIdiom] =
= UIUserInterfaceIdiomPhone) {
        return (interfaceOrientation !=
UIInterfaceOrientationPortraitUpsideDown);
    } else {
        return YES;
    }
```

FIGURE 6.27
Edit the snippet and the information stored about the snippet.

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.

By the Way

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.

Index