

John Ray



In **Full Color**

Figures and
code appear as
they do in Xcode 5.x

Covers **iOS 7, Xcode 5.x,
iPhone, iPad, and More!**

Additional files and
updates available
online

Sams **Teach Yourself**

iOS[®] 7

Application Development

in **24**
Hours

SAMS

FREE SAMPLE CHAPTER

SHARE WITH OTHERS



John Ray

Sams **Teach Yourself**

iOS 7 Application Development

in **24**
Hours

SAMS

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

Sams Teach Yourself iOS 7 Application Development in 24 Hours

Copyright © 2014 by Pearson Education, Inc.

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

ISBN-13: 978-0-672-33706-2

ISBN-10: 0-672-33706-1

Library of Congress Control Number: 2013953997

Printed in the United States of America

First Printing January 2014

Trademarks

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

Warning and Disclaimer

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

Special Sales

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact international@pearsoned.com.

Editor-in-Chief

Greg Wiegand

Acquisitions Editor

Laura Norman

Development Editor

Keith Cline

Technical Editor

Rosanne Groves

Managing Editor

Kristy Hart

Project Editors

Katie Matejka

Elaine Wiley

Indexer

Brad Herriman

Proofreader

Paula Lowell

Cover Designer

Mark Shirar

Compositor

Nonie Ratcliff

Contents at a Glance

	Introduction	xv
HOOR 1	Preparing your System and iDevice for Development	1
2	Introduction to Xcode and the iOS Simulator	23
3	Discovering Objective-C: The Language of Apple Platforms	67
4	Inside Cocoa Touch	101
5	Exploring Interface Builder	131
6	Model-View-Controller Application Design	167
7	Working with Text, Keyboards, and Buttons	195
8	Handling Images, Animation, Sliders, and Steppers	229
9	Using Advanced Interface Objects and Views	257
10	Getting the User's Attention	291
11	Implementing Multiple Scenes and Popovers	321
12	Making Choices with Toolbars and Pickers	373
13	Advanced Storyboards using Navigation and Tab Bar Controllers	421
14	Navigating Information Using Table Views and Split View Controllers	459
15	Reading and Writing Application Data	503
16	Building Responsive and Backward-Compatible User Interfaces	547
17	Using Advanced Touches and Gestures	593
18	Sensing Orientation and Motion	619
19	Working with Rich Media	647
20	Interacting with Other Applications	693
21	Implementing Location Services	727
22	Building Background-Ready Applications	757
23	Building Universal Applications	789
24	Application Tracing, Monitoring, and Debugging	809
	Index	833

Table of Contents

Introduction	xv
HOOR 1: Preparing Your System and iDevice for Development	1
Welcome to the iOS Platform	1
Becoming an iOS Developer	6
Creating and Installing a Development Provisioning Profile	12
Running Your First iOS App	17
Developer Technology Overview	20
Further Exploration	21
Summary	21
Q&A	21
Workshop	22
Activities	22
HOOR 2: Introduction to Xcode and the iOS Simulator	23
Using Xcode	23
Using the iOS Simulator	58
Further Exploration	63
Summary	64
Q&A	65
Workshop	65
Activities	66
HOOR 3: Discovering Objective-C: The Language of Apple Platforms	67
Object-Oriented Programming and Objective-C	67
Exploring the Objective-C File Structure	72
Objective-C Programming Basics	84
Memory Management and Automatic Reference Counting	95
Further Exploration	97
Summary	97

Q&A	98
Workshop	99
Activities	99
HOUR 4: Inside Cocoa Touch	101
What Is Cocoa Touch?	101
Exploring the iOS Technology Layers	103
Tracing the iOS Application Life Cycle	109
Cocoa Fundamentals	110
Exploring the iOS Frameworks with Xcode	120
Further Exploration	129
Summary	129
Q&A	129
Workshop	130
Activities	130
HOUR 5: Exploring Interface Builder	131
Understanding Interface Builder	131
Creating User Interfaces	138
Customizing the Interface Appearance	148
Connecting to Code	154
Further Exploration	164
Summary	165
Q&A	165
Workshop	166
Activities	166
HOUR 6: Model-View-Controller Application Design	167
Understanding the Model-View-Controller Design Pattern	167
How Xcode Implements MVC	169
Using the Single View Application Template	173
Further Exploration	191
Summary	192

Q&A	192
Workshop	192
Activities	193
HOOR 7: Working with Text, Keyboards, and Buttons	195
Basic User Input and Output	195
Using Text Fields, Text Views, and Buttons	197
Further Exploration	226
Summary	227
Q&A	227
Workshop	227
Activities	228
HOOR 8: Handling Images, Animation, Sliders, and Steppers	229
User Input and Output	229
Creating and Managing Image Animations, Sliders, and Steppers	231
Further Exploration	252
Summary	254
Q&A	254
Workshop	254
Activities	255
HOOR 9: Using Advanced Interface Objects and Views	257
User Input and Output (Continued)	257
Using Switches, Segmented Controls, and Web Views	262
Using Scrolling Views	279
Further Exploration	288
Summary	289
Q&A	289
Workshop	289
Activities	290
HOOR 10: Getting the User's Attention	291
Alerting the User	291
Exploring User Alert Methods	301
Further Exploration	318

Summary	319
Q&A	319
Workshop	319
Activities	320
HOUR 11: Implementing Multiple Scenes and Popovers	321
Introducing Multiscene Storyboards	322
Understanding the iPad Popover	341
Using a Modal Segue	352
Using a Popover	364
Further Exploration	370
Summary	371
Q&A	371
Workshop	372
Activities	372
HOUR 12: Making Choices with Toolbars and Pickers	373
Understanding the Role of Toolbars	373
Exploring Pickers	377
Using the Date Picker	385
Using a Custom Picker	401
Further Exploration	417
Summary	417
Q&A	418
Workshop	418
Activities	419
HOUR 13: Advanced Storyboards Using Navigation and Tab Bar Controllers	421
Advanced View Controllers	421
Exploring Navigation Controllers	423
Understanding Tab Bar Controllers	429
Using a Navigation Controller	434
Using a Tab Bar Controller	445
Further Exploration	456
Summary	457

Q&A	457
Workshop	458
Activities	458
HOUR 14: Navigating Information Using Table Views and Split View Controllers	459
Understanding Tables	459
Exploring the Split View Controller (iPad Only)	468
A Simple Table View Application	471
Creating a Master-Detail Application	481
Further Exploration	499
Summary	500
Q&A	500
Workshop	501
Activities	501
HOUR 15: Reading and Writing Application Data	503
iOS Applications and Data Storage	503
Data Storage Approaches	506
Creating Implicit Preferences	514
Implementing System Settings	522
Implementing File System Storage	535
Further Exploration	543
Summary	543
Q&A	544
Workshop	544
Activities	545
HOUR 16: Building Responsive and Backward-Compatible User Interfaces	547
Responsive Interfaces	547
Using Auto Layout	552
Programmatically Defined Interfaces	575
Swapping Views on Rotation	583
Further Exploration	590

Summary	590
Q&A	591
Workshop	591
Activities	592
HOUR 17: Using Advanced Touches and Gestures	593
Multitouch Gesture Recognition	593
Adding Gesture Recognizers	594
Using Gesture Recognizers	596
Further Exploration	617
Summary	617
Q&A	617
Workshop	618
Activities	618
HOUR 18: Sensing Orientation and Motion	619
Understanding Motion Hardware	619
Accessing Orientation and Motion Data	622
Sensing Orientation	626
Detecting Acceleration, Tilt, and Rotation	631
Further Exploration	643
Summary	644
Q&A	645
Workshop	645
Activities	645
HOUR 19: Working with Rich Media	647
Exploring Rich Media	647
The Media Playground Application	662
Further Exploration	689
Summary	690
Q&A	691
Workshop	691
Activities	692

HOUR 20: Interacting with Other iOS Services	693
Extending iOS Service Integration	693
Using the Address Book, Email, Social Networking, and Maps	708
Further Exploration	724
Summary	725
Q&A	725
Workshop	726
Activities	726
HOUR 21: Implementing Location Services	727
Understanding Core Location	727
Creating a Location-Aware Application	734
Using the Magnetic Compass	744
Further Exploration	754
Summary	755
Q&A	755
Workshop	756
Activities	756
HOUR 22: Building Background-Ready Applications	757
Understanding iOS Backgrounding	757
Disabling Backgrounding	763
Handling Background Suspension	764
Implementing Local Notifications	765
Using Task-Specific Background Processing	768
Completing a Long-Running Background Task	773
Performing a Background Fetch	780
Further Exploration	785
Summary	786
Q&A	786
Workshop	786
Activities	787

HOUR 23: Building Universal Applications	789
Universal Application Development	789
Creating a Universal Application (Take 1)	794
Creating a Universal Application (Take 2)	799
Using Multiple Targets	803
Further Exploration	806
Summary	806
Q&A	807
Workshop	807
Activities	807
HOUR 24: Application Tracing, Monitoring, and Debugging	809
Instant Feedback with <code>NSLog</code>	810
Using the Xcode Debugger	813
Further Exploration	828
Summary	830
Q&A	830
Workshop	830
Activities	831
Index	833

Dedication

This book is dedicated to being the best book that it can be. Give it a big hug.

About the Author

John Ray is currently serves as the Director of the Office of Research Information Systems at The Ohio State University. 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*, *My Mavericks MacBook*, and *Teach Yourself iOS 6 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 users.

You can visit his website at <http://teachyourselfios.com> or follow him on Twitter at @johnemeryray or #iOSIn24.

Acknowledgments

Thank you to the group at Sams Publishing—Laura Norman, Keith Cline, Mark Renfrow—and my Tech Editor, Anne Groves, for getting me through this edition. The abject terror that your team places within me drives me to finish. I hope someday to escape this small room and again see the light of day. Between typing sessions, I catch raindrops on the top of my MacBook and use them to quench my parched lips. My leather iPad case, half eaten, is my sole source of sustenance. I know what will happen if I miss another deadline, but the tears no longer flow, and the screams no longer come.

I hear them coming. I must go.

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.

We welcome your comments. You can email or write to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book.

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

Email: consumer@samspublishing.com

Mail: Sams Publishing
ATTN: Reader Feedback
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

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

Introduction

When you pick up an iOS device and use it, you feel connected. Whether it be an iPad Air, an iPhone, or an iPod, the interface acts as an extension to your fingers; it is smooth, comfortable, and invites exploration. Other competing devices offer similar features, and even sport gadgets such as styluses and trackpads, but they cannot match the user experience that is iOS.

The introduction of iOS 7 solidifies Apple's commitment to user-focused design. The new operating system places content front-and-center, eliminating the glossy edges and shadows that served as little more than a distraction. Now, depth and translucency help keep users connected to their content and aware of the context in which they are accessing it. iOS has taken a giant leap forward, and with it, the tools used for development.

When creating iOS and the Xcode development tools, Apple considered everything from interface to application performance and battery life. There is always the expectation that, no matter what, the device will remain responsive and usable. As a developer, does this mean that there are rules to follow? Absolutely. But, by following these rules, you can create applications that are interactive works of art for your users to love—not software they will load and forget.

Through the App Store, Apple has created the ultimate digital distribution system for iOS applications. Programmers of any age or affiliation can submit their applications to the App Store for just the cost of a modest yearly Developer Membership fee. Games, utilities, and full-feature applications have been built for everything from pre-K education to retirement living. No matter what the content, with a user base as large as the iPhone, iPod Touch, and iPad, an audience exists.

Each year, Apple introduces new devices—bringing larger, faster, and higher-resolution capabilities to the iOS family. With each new hardware refresh come new development opportunities and new ways to explore the boundaries between software and art.

My hope is that this book brings iOS development to a new generation of developers. *Teach Yourself iOS 7 Development in 24 Hours* provides a clear and natural progression of skills development, from installing developer tools and registering your device with Apple, to submitting an application to the App Store. It's everything you need to get started in 24 one-hour lessons.

Who Can Become an iOS Developer?

If you have an interest in learning, time to invest in exploring and practicing with Apple’s developer tools, and an Intel Macintosh computer running Mountain Lion, you have everything you need to begin creating software for iOS.

Developing an app won’t happen overnight, but with dedication and practice, you can be writing your first applications in a matter of days. The more time you spend working with the Apple developer tools, the more opportunities you’ll discover for creating new and exciting projects.

You should approach iOS application development as creating software that *you* want to use, not what you think others want. If you’re solely interested in getting rich quick, you’re likely to be disappointed. (The App Store is a crowded marketplace—albeit one with a lot of room—and competition for top sales is fierce.) However, if you focus on building useful and unique apps, you’re much more likely to find an appreciative audience.

Who Should Use This Book?

This book targets individuals who are new to development for iOS and have experience using the Macintosh platform. No previous experience with Objective-C, Cocoa, or the Apple developer tools is required. Of course, if you do have development experience, some of the tools and techniques may be easier to master, but the author does not assume that you’ve coded before.

That said, some things are expected of you, the reader. Specifically, you must be willing to invest in the learning process. If you just read each hour’s lesson without working through the tutorials, you will likely miss some fundamental concepts. In addition, you need to spend time reading the Apple developer documentation and researching the topics presented in this book. A vast amount of information on iOS development is available, but only limited space in this book. Therefore, this book covers what you need to forge your own path forward.

What Is (and Isn’t) in This Book?

The material in this book specifically targets iOS release 7 and later on Xcode 5 and later. Much of what you’ll learn is common to all the iOS releases, but this book also covers several important areas that have only come about in recent iOS releases, such as gesture recognizers, embedded video playback with AirPlay, Core Image, social networking, multitasking, universal (iPhone/iPad) applications, Auto Layout, and more!

Unfortunately, this is not a complete reference for the iOS application programming interfaces (APIs); some topics just require much more space than this book allows. Thankfully, the Apple developer documentation is available directly within the free tools you install in Hour 1, “Preparing Your System and iDevice for Development.” In many hours, you’ll find a section

titled “Further Exploration.” This identifies additional related topics of interest. Again, a willingness to explore is an important quality in becoming a successful developer.

Each coding lesson is accompanied by project files that include everything you need to compile and test an example or, preferably, follow along and build the application yourself. Be sure to download the project files from this book's website at <http://teachyourselfios.com>. If you have issues with any projects, view the posts on this site to see whether a solution has been identified.

In addition to the support website, you can follow along on Twitter! Search for #iOSIn24 on Twitter to receive official updates and tweets from other readers. Use the hashtag #iOSIn24 in your tweets to join the conversation. To send me messages via Twitter, begin each tweet with @johnemeryray.

This page intentionally left blank

This page intentionally left blank

HOUR 7

Working with Text, Keyboards, and Buttons

What You'll Learn in This Hour:

- ▶ How to use text fields
- ▶ Input and output in scrollable text views
- ▶ How to enable data detectors
- ▶ A way to spruce up the standard iOS buttons

In the preceding hour, you explored views and view controllers and created a simple application that accepted user input and generated output when a button was pushed. This hour expands on these basic building blocks. In this hour, we create an application that uses multiple different input and output techniques. You learn how to implement and use editable text fields, text views, and graphical buttons, and how to configure the onscreen keyboard.

This is quite a bit of material to cover in an hour, but the concepts are very similar, and you'll quickly get the hang of these new elements.

Basic User Input and Output

iOS gives us many different ways of displaying information to a user and collecting feedback. There are so many ways, in fact, that we're going to be spending the next several hours working through the tools that the iOS software development kit (SDK) provides for interacting with your users, starting with the basics.

Buttons

One of the most common interactions you'll have with your users is detecting and reacting to the touch of a button (`UIButton`). Buttons, as you may recall, are elements of a view that respond to an event that the user triggers in the interface, usually a `Touch Up Inside` event to indicate that the user's finger was on a button and then released it. Once an event is detected, it can trigger an action (`IBAction`) within a corresponding view controller.

Buttons are used for everything from providing preset answers to questions to triggering motions within a game. Although the default iOS button style is minimalist, buttons can take on many different forms through the use of images. Figure 7.1 shows an example of a fancy button with gradients.

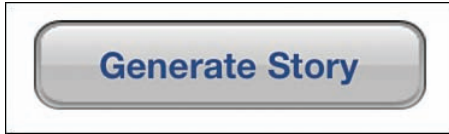


FIGURE 7.1

Buttons can be simple, fancy (like this one), or set to any arbitrary image.

Text Fields and Views

Another common input mechanism is a text field. Text fields (`UITextField`) give users space to enter any information they want into a single line in the application; these are similar to the form fields in a web form. When users enter data into a field, you can constrain their input to numbers or text by using different iOS keyboards, something we do later this hour. You can also enable editing of styles within the text, such as underlining and bold. Text fields, like buttons, can respond to events but are often implemented as passive interface elements, meaning that their contents (provided through the `text` property) can be read at any time by the view controller.

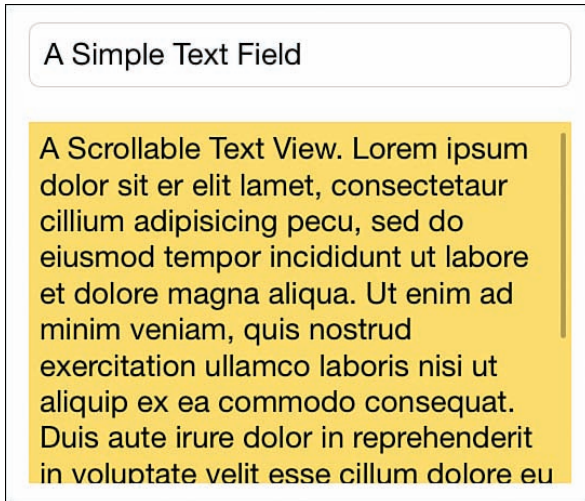
Similar to the text field is the text view (`UITextView`). The difference is a text view can present a scrollable and editable block of text for the user to either read or modify. These should be used in cases where more than a few words of input are required. Figure 7.2 shows examples of a text field and text view.

Labels

The final interface feature that we're going to be using here and throughout this book is the label (`UILabel`). Labels are used to display strings within a view by setting their `text` property.

The text within a label can be controlled via a wide range of label attributes, such as font and text size, alignment, and color. As you'll see, labels are useful both for static text in a view and for presenting dynamic output that you generate in your code.

Now that you have basic insight into the input and output tools we'll be using in this hour, let's go ahead and get started with our project: a simple substitution-style story generator.

**FIGURE 7.2**

Text fields and text views provide a means for entering text using a device's virtual keyboard.

Using Text Fields, Text Views, and Buttons

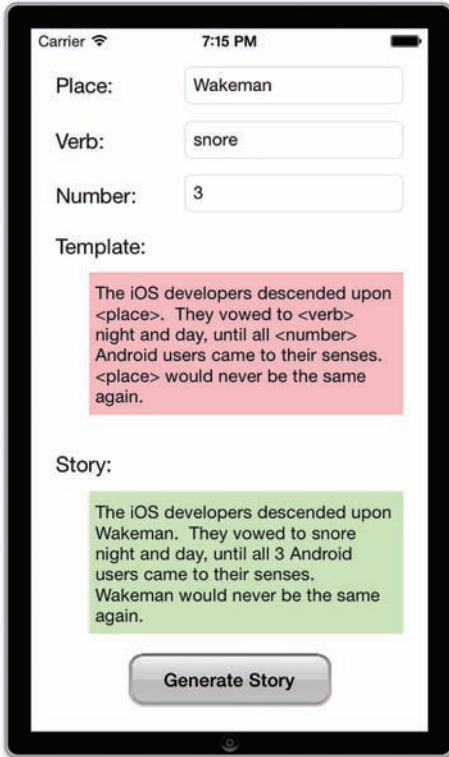
Although not everyone will agree with my sentiment, I enjoy entering text on my iPhone and iPad. The virtual keyboard is responsive and simple to navigate. What's more, the input process can be altered to constrain the user's input to only numbers, only letters, or other variations. (This varies depending on device.) You can have iOS automatically correct simple misspellings, allow text styling, or capitalize letters—all without a line of code. This project reviews many aspects of the text input process.

Implementation Overview

In this project, we create a Mad Libs-style story creator. Users enter a noun (place), verb, and number through three text fields (`UITextField`). They may also enter or modify a template that contains the outline of the story to be generated. Because the template can be several lines long, we use a text view (`UITextView`) to present this information. A button press (`UIButton`) triggers an action that generates the story and outputs the finished text in another text view, demonstrated in Figure 7.3.

Although not directly part of the input or output process, we also investigate how to implement the now-expected “touch the background to make the keyboard disappear” interface standard, along with a few other important points. In other words, pay attention!

We'll name this tutorial project **FieldButtonFun**. You may certainly use something more creative if you want.

**FIGURE 7.3**

The tutorial app in this hour uses two types of text input objects.

Setting Up the Project

This project uses the same Single View Application template as the preceding hour. If it isn't already running, launch Xcode, and then complete these steps:

1. Choose File, New, Project.
2. Select the iOS application project type.
3. Find and select the Single View Application option in the Template list, and then click Next to continue.
4. Enter the project name, **FieldButtonFun**, be sure that your device is chosen, and then click Next.
5. Choose your save location and click Create to set up the new project.

As before, we focus on the view, which has been created in `Main.storyboard`, and the view controller class `ViewController`.

Planning the Properties and Connections

This project contains a total of six input areas that must connect to our code via outlets. Three text fields are used to collect the place, verb, and number values. We'll access these through private properties named `thePlace`, `theVerb`, and `theNumber`, respectively. The project also requires two text views: one to hold the editable story template, `theTemplate`; and the other to contain the output, `theStory`.

NOTE

Yes, we'll use a text view for output as well as for input. Text views provide a built-in scrolling behavior and can be set to read-only, making them convenient for both collecting and displaying information.

Finally, a single button is used to trigger a method, `createStory`, which serves as an action and creates the story text, unlike the preceding hour's example.

TIP

If UI elements are used only to trigger actions; they do not need outlets. If your application needs to manipulate an object, however—such as setting its label, color, size, position, and so on—it needs an outlet and corresponding instance variable/property defined.

Now that you understand the objects we'll add and how we'll refer to them, let's turn our attention to building the user interface (UI) and creating our connections to code.

Preparing Button Templates with Slicing

In the preceding hour's lesson, you created a button (`UIButton`) and connected it to the implementation of an action (`IBAction`) within a view controller. Nothing to it, right? Working with buttons is relatively straightforward, but what you may have noticed is that, by default, the buttons you create in the Interface Builder (IB) are, well, not very button-like. To create visually appealing graphical buttons that don't require a new image for each button, we can prepare a button template using a technique called *slicing*.

The Xcode slicing tool is used to define areas of an image that can be resized when the image is stretched. You can choose to create both vertical and horizontal slices to accommodate both vertical and horizontal stretching. We can use this to create graphically rich buttons that take on any size and look great.

Adding the Images

Slices are added using a tool within the Xcode asset catalog. So, our first task is to add some images.

Inside this hour's Projects directory is an Images folder with two Apple-created button templates: `whiteButton.png` and `blueButton.png`. Within the Xcode project navigator, click the Images.xcassets assets catalog icon. Next, drag the Images folder from the OS X Finder into the left column of the asset catalog in Xcode. Your display should resemble Figure 7.4.

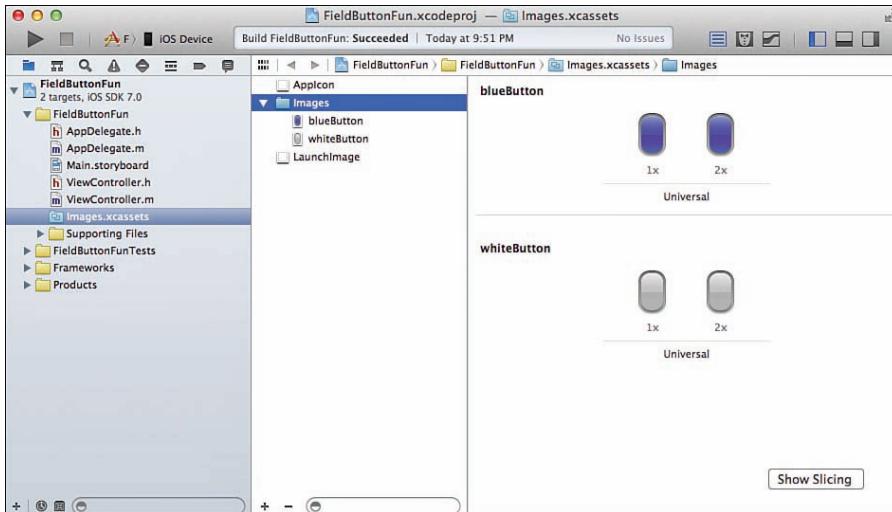


FIGURE 7.4

To use custom buttons, drag the project images folder into the Images.xcassets folder in Xcode.

Creating Slices

Slicing may sound complicated, but it is easy to understand, and even easier to perform. When creating a slice, you visually specify a horizontal or vertical (or both) stripe of pixels within an image. This is the “slice,” and it will be repeated to fill in space if the image needs to grow. You’ll also (optionally) get to choose a portion (also a stripe of pixels) that is replaced by the slice when the image resizes.

To create your slices, first make sure that you’ve opened the asset catalog by selecting the Images.xcassets icon in the project navigator, and then follow these steps:

1. Expand the Images folder within the asset catalog.
2. Select one of the button images to slice; I’m starting with `whiteButton`.

3. Click the Show Slicing button in the lower-right corner of the asset catalog content area, as shown in Figure 7.5.

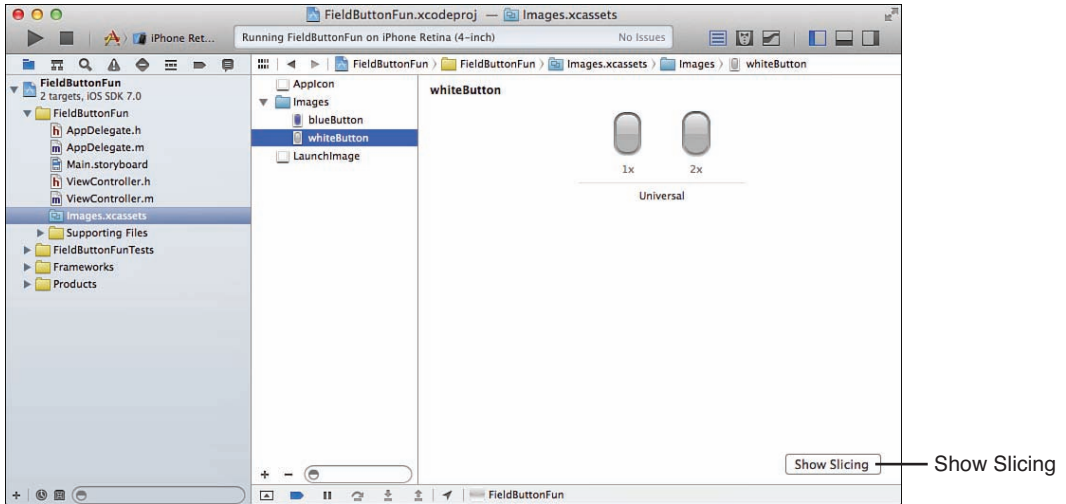


FIGURE 7.5

Use the Show Slicing button to start the slicing process.

4. The screen refreshes to show an enlarged copy of the graphic (both non-Retina and Retina, if available). Click the Start Slicing button on the non-Retina image to begin.
5. Xcode prompts for the type of slicing. Use the buttons, as demonstrated in Figure 7.6, to choose between Horizontal, Horizontal & Vertical, or just Vertical slicing. For our button, we want both horizontal and vertical.
6. The slicing editor displays three dragging lines in the horizontal/vertical directions. The horizontal lines determine the vertical slicing, and the vertical lines determine the horizontal slicing.

The first and second lines (going left to right/top to bottom) determine the slice that can grow. The second and third lines define the area that will be replaced by copies of the slicing when the image resizes, as shown in Figure 7.7. The second and third lines can be positioned right next to one another if you simply want the stripe to stretch without replacing any other parts of the image.

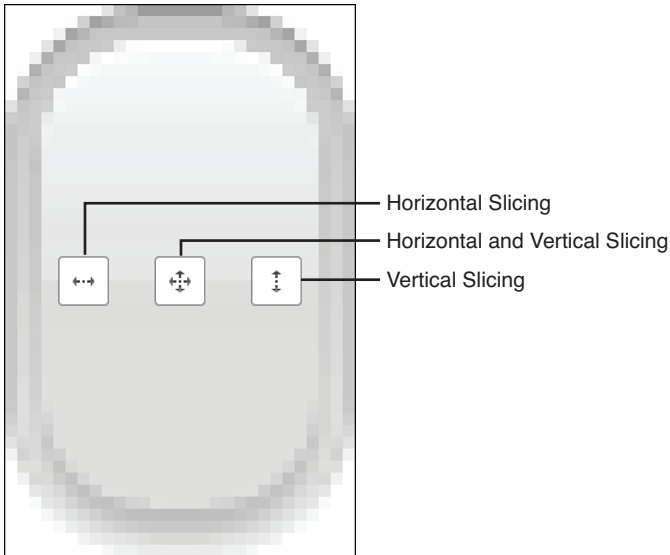


FIGURE 7.6
Choose the type of slicing you want to perform.

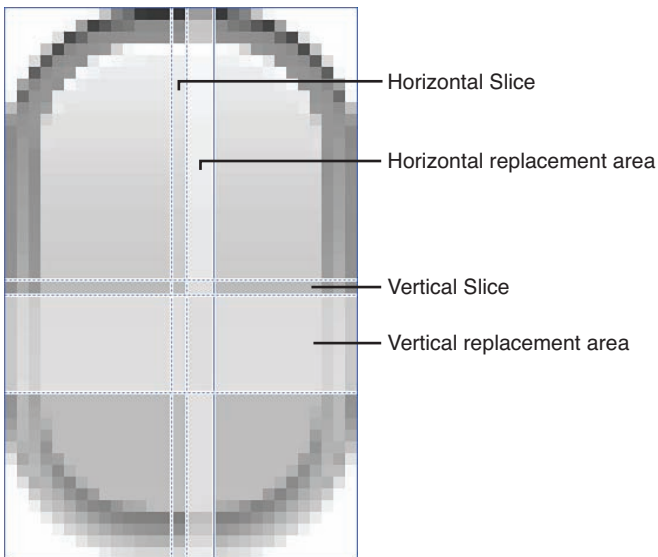


FIGURE 7.7
Use the lines to choose your slices.

7. For the button template, we want the pretty, curvy corners to always stay the same; so they aren't resized. The portion we want to grow is a stripe about 12 pixels in (horizontally and vertically), and can just be a single pixel wide. Drag the first vertical line about 12 pixels in.
8. Drag the second vertical line so that creates a 1-pixel-wide stripe (that is, about 13 pixels in).
9. Drag the third vertical line so that it is right next to the second line; there's no reason to replace any portion of the image with the repeated stripe. You've just completed the horizontal slice.
10. Repeat steps 7–9 for the horizontal lines, creating the vertical slice. Your finished slicing layout should look almost identical to Figure 7.8.

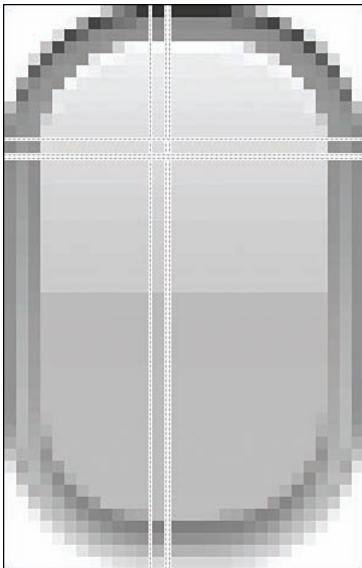


FIGURE 7.8
The finished slicing.

TIP

To fine-tune your slices, open the Attributes Inspector while slicing. This shows the position of the slice in pixels and enables you to manually enter X and Y values.

After you've finished the slicing for the non-Retina white button, do the same for the Retina version. You'll need to position your slices about 24 pixels in (versus 12 on the non-Retina). Next, repeat the process for the blue button assets.

When you finish, you've created images that can be resized to create attractive buttons regardless of how large the button needs to be. When it comes time to use the images in a few minutes, you'll treat them like any other image; the slicing is applied automatically when they are resized.

TIP

Slicing works great for buttons, but can be used for *any* image that you may want to resize. Just set the slicing for the graphics and when you stretch them, they'll resize using your slicing preferences.

Designing the Interface

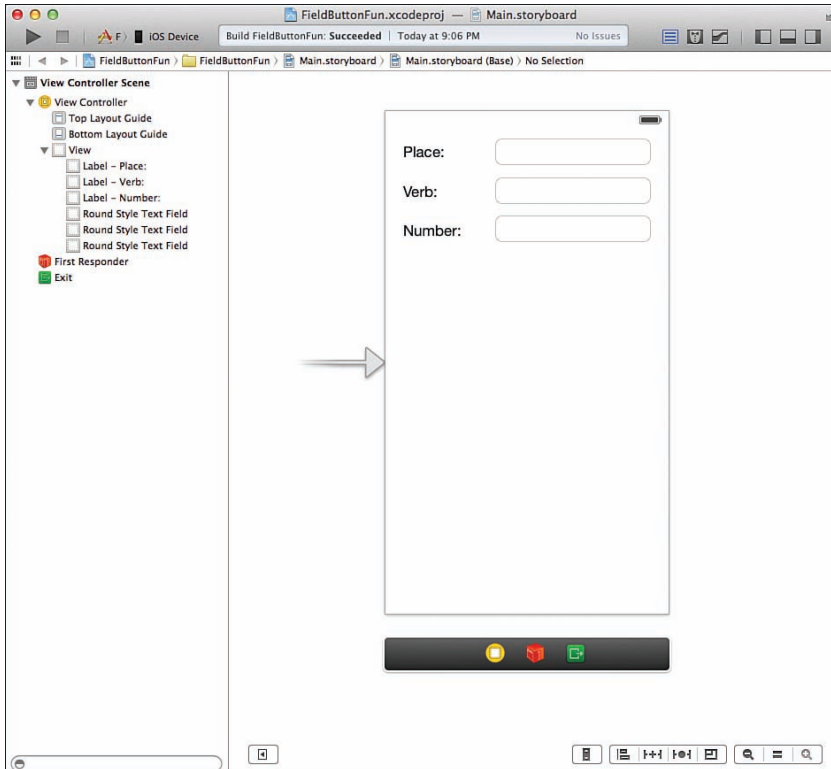
In the preceding hour, you learned that the `Main.storyboard` is loaded when the application launches and that it instantiates the default view controller, which subsequently loads its view from the storyboard file. Locate `MainStoryboard.storyboard` in the project's code folder, and click to select it and open the IB editor.

When IB has started, be sure that the document outline area is visible (Editor, Show Document Outline), hide the navigation area if you need room, and open the Object Library (View, Utilities, Show Object Library).

Adding Text Fields

Begin creating the UI by adding three text fields to the top of the view. To add a field, locate the Text Field object (`UITextField`) in the library and drag it into the view. Repeat this two more times for the other two fields.

Stack the fields on top of one another, leaving enough room so that the user can easily tap a field without hitting all of them. To help the user differentiate between the three fields, add labels to the view. Click and drag the label (`UILabel`) object from the library into the view. Align three labels directly across from the three fields. Double-click the label within the view to set its text. I labeled my fields Place, Verb, and Number, from top to bottom, as shown in Figure 7.9.

**FIGURE 7.9**

Add text fields and labels to differentiate between them.

Editing Text Field Attributes

The fields that you've created are technically fine as is, but you can adjust their appearance and behavior to create a better user experience. To view the field attributes, click a field, and then press Option-Command-4 (View, Utilities, Show Attributes Inspector) to open the Attributes Inspector (see Figure 7.10).

For example, you can use the Placeholder field to enter text that appears in the background of the field until the user begins editing. This can be a helpful tip or an additional explanation of what the user should be entering.

You may also choose to activate the Clear button. The Clear button is a small X icon added to a field that the user can touch to quickly erase the contents. To add the Clear button, just choose one of the visibility options from the Clear button pop-up menu; the functionality is added for free to your application. Note that you may also choose to automatically clear the field when the user taps it to start editing. Just check the Clear When Editing Begins check box.

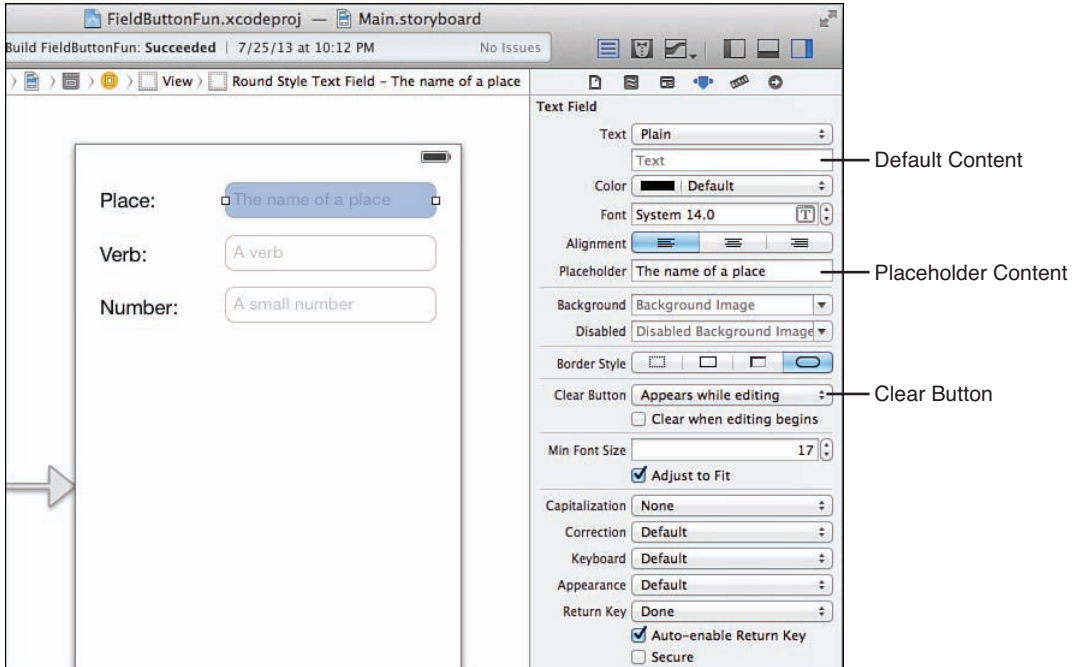


FIGURE 7.10
Editing a field’s attributes can help create a better UI.

Add these features to the three fields within the view. Figure 7.11 shows how they appear in the application.

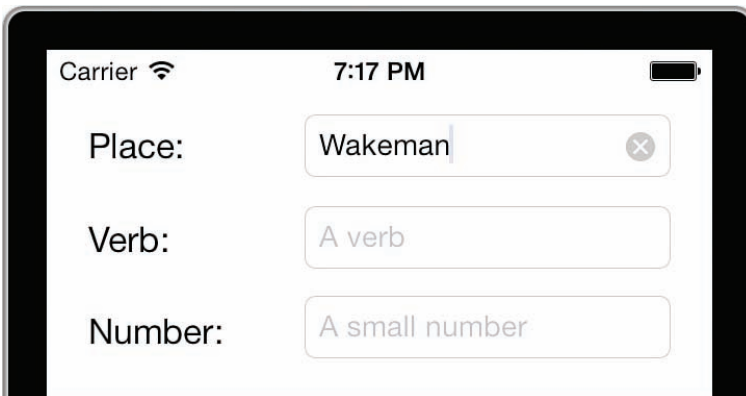


FIGURE 7.11
Placeholder text can provide helpful cues to the user, and the Clear button makes it simple to remove a value from a field.

TIP

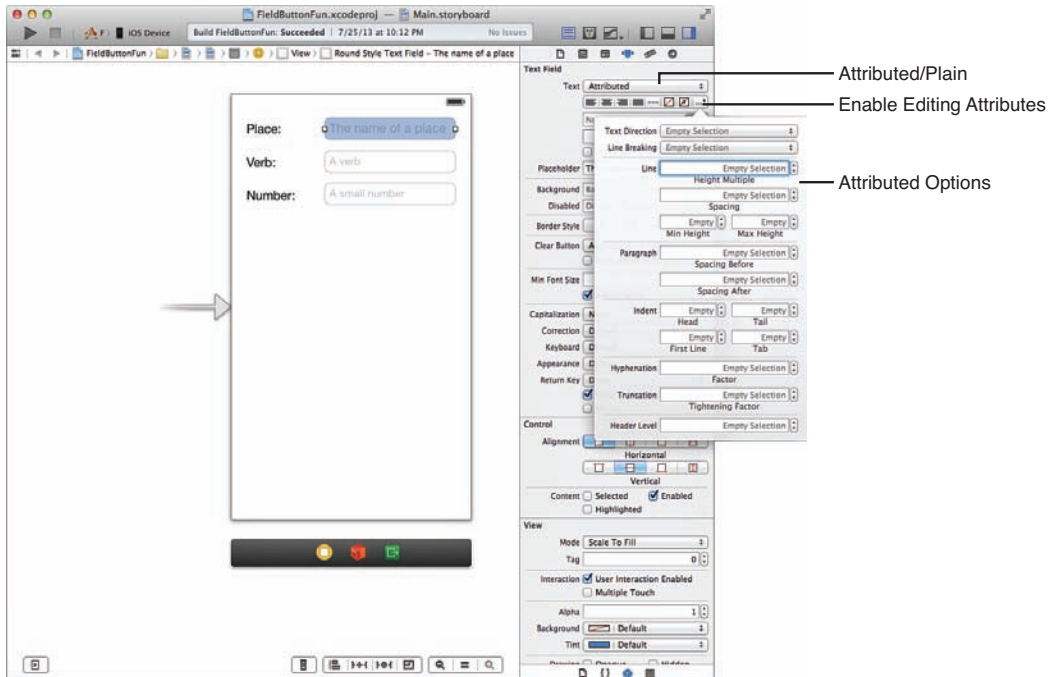
Placeholder text also helps identify which field is which within the IB editor area. It can make creating your connections much easier down the road.

In addition to these changes, attributes can adjust the text alignment, font and size, and other visual options. Part of the fun of working in the IB editor is that you can explore the tools and make tweaks (and undo them) without having to edit your code.

Attributed Versus Plain Text

In many of controls that allow the display of text, you'll find the Text drop-down menu that can toggle between Plain or Attributed options. In general, you want to leave this on Plain, but by setting it to Attributed, you can gain much finer control over the layout of the text, as shown in Figure 7.12.

Using this feature, you can provide more richly styled text output and even enable user editing of the text style by simply checking the Allows Editing Attributes check box.

**FIGURE 7.12**

Attributed text fields and other UI elements offer more detailed control over presentation.

Customizing the Keyboard Display with Text Input Traits Probably the most important attributes that you can set for an input field are the “text input traits,” or simply, how the keyboard is going to be shown onscreen. Seven different traits appear at the bottom of the text field attributes section:

- ▶ **Capitalize:** Controls whether iOS automatically capitalizes words, sentences, or all the characters entered into a field.
- ▶ **Correction:** If explicitly set to on or off, the input field corrects (on) or ignores (off) common spelling errors. If left to the defaults, it inherits the behavior of the iOS settings.
- ▶ **Keyboard:** Sets a predefined keyboard for providing input. By default, the input keyboard lets you type letters, numbers, and symbols. A total of 10 different keyboards are available, ranging from Number Pad to Web Search. If the option Number Pad is chosen, for example, only numbers can be entered. Similarly, the Email Address option constrains the input to strings that look like email addresses.
- ▶ **Appearance:** Changes the appearance of the keyboard to look more like an alert view (which you learn about in a later hour).
- ▶ **Return Key:** If the keyboard has a Return key, it is set to this label. Values include Done, Search, Next, Go, and so on.
- ▶ **Auto-Enable Return Key:** Disables the Return key on the keyboard unless the user has entered at least a single character of input into the field.
- ▶ **Secure:** Treats the field as a password, hiding each character as it is typed.

Of the three fields that we’ve added to the view, the Number field can definitely benefit from setting an input trait. With the Attributes Inspector still open, select the Number field in the view, and then choose the Number Pad option from the Keyboard pop-up menu (see Figure 7.13).

You may also want to alter the capitalization and correction options on the other two fields and set the Return key to Done. Again, all this functionality is gained “for free.” So, you can return to edit the interface and experiment all you want later on. For now, let’s call these fields “done” and move on to the text areas.

Copy and Paste

Your text entry areas automatically gain copy and paste without your having to add anything to your code. For advanced applications, you can override the protocol methods defined in `UIResponderStandardEditActions` to customize the copy, paste, and selection process.

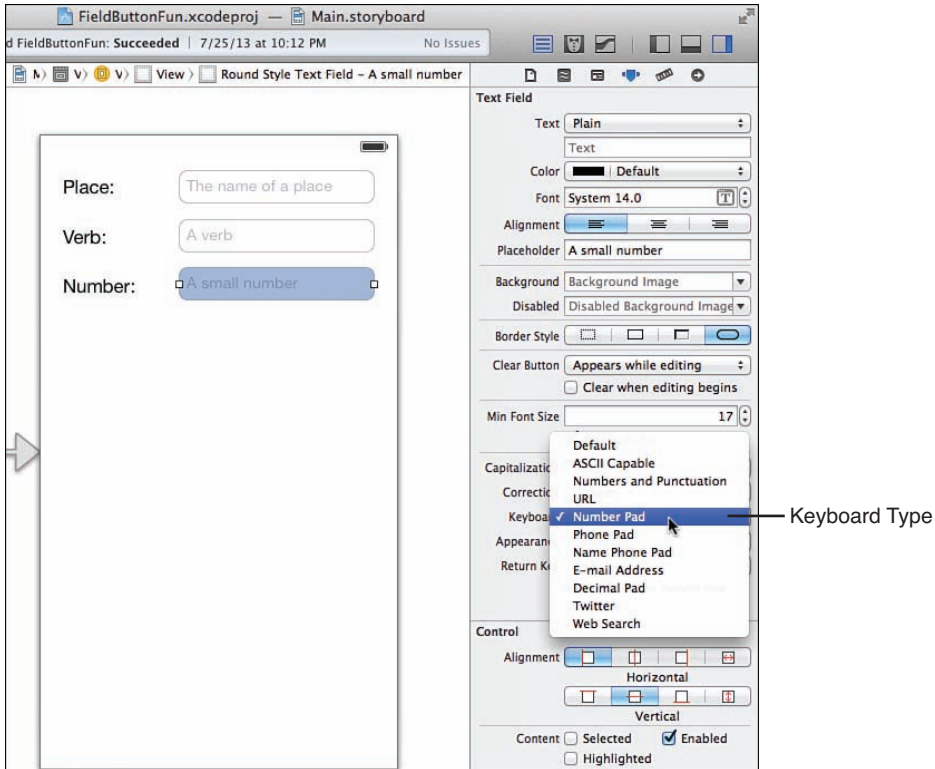


FIGURE 7.13

Choosing a keyboard type will help constrain a user's input.

Adding Text Views

Now that you know the ins and outs of text fields, let's move on to the two text views (`UITextView`) present in this project. Text views, for the most part, can be used just like text fields. You can access their contents the same way, and they support many of the same attributes as text fields, including text input traits.

To add a text view, find the Text View object (`UITextView`) and drag it into the view. Doing so adds a block to the view, complete with Greeked text (*Lorem ipsum...*) that represents the input area. Using the resizing handles on the sizes of the block, you can shrink or expand the object to best fit the view. Because this project calls for two text views, drag two into the view and size them to fit underneath the existing three text fields.

As with the text fields, the views themselves don't convey much information about their purpose to the user. To clarify their use, add two text labels above each of the views: **Template** for the first, and **Story** for the second. Your view should now resemble Figure 7.14.

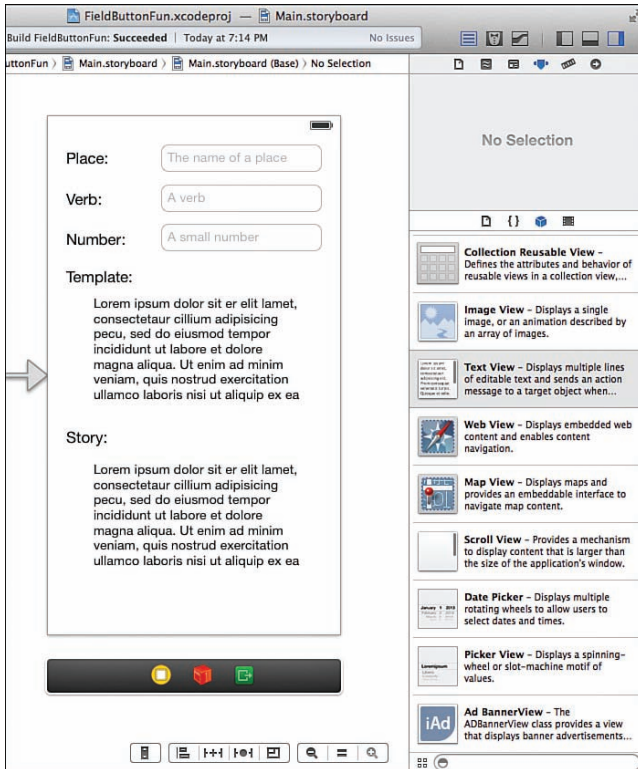


FIGURE 7.14
Add labels to clarify your text views.

CAUTION

Size Doesn't Matter (For Now)

The size of your iPhone design doesn't matter at this stage. In Hour 16, "Building Responsive and Backward-Compatible User Interfaces," you learn how a single layout can work on the multiple screen sizes as well as multiple versions of iOS. Right now, be aware that when you position controls, they probably won't adjust correctly to work on other devices or earlier versions of iOS. In addition, your layout might not exactly match my screenshots. That's okay. I don't want you to get caught up in the details of UI design while you're still learning to program!

Editing Text View Attributes Text view attributes provide many of the same visual controls as text fields, including plain and attributed modes. Select a view, and then open the Attributes Inspector (Option-Command-4) to see the available options, as shown in Figure 7.15.

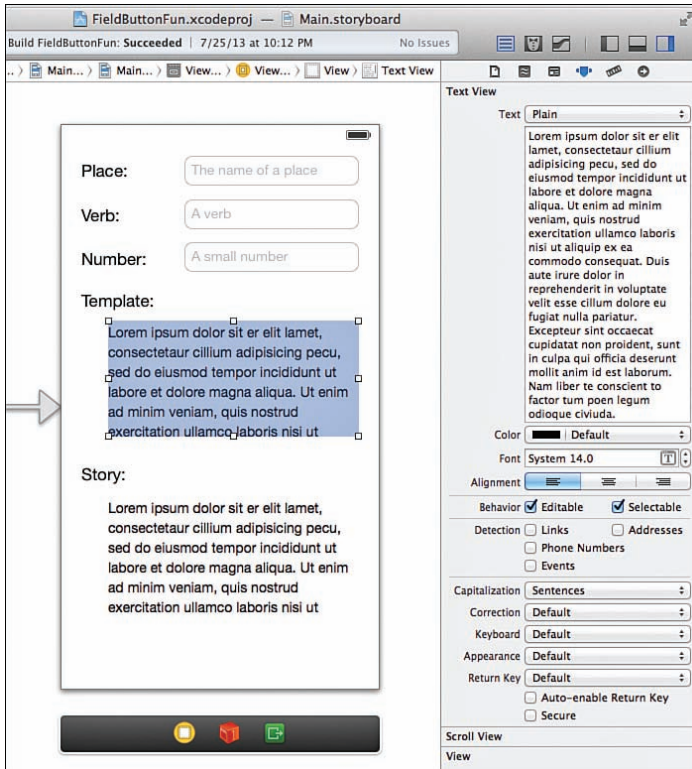


FIGURE 7.15

Edit the attributes of each text view to prepare them for input and output.

To start, we need to update the default content to remove the initial Greeked text and provide our own input. For the top field, which will act as the template, select the content within the Text attribute of the Attributes Inspector (this is directly below the Plain/Attributed drop-down menu), and then clear it. Enter the following text, which will be available within the application as the default:

The iOS developers descended upon <place>. They vowed to <verb> night and day, until all <number> Android users came to their senses. <place> would never be the same again.

When we implement the logic behind this interface, the placeholders (<place>, <verb>, <number>) are replaced with the user's input.

Next, select the "story" text view, and then again use the Attributes Inspector to clear the contents entirely. Because the contents of this text view are generated automatically, we can leave it empty. This view is a read-only view, as well, so uncheck the Editable attribute.

In this example, to help provide some additional contrast between these two areas, I set the background color of the template to a light red and the story to a light green. To do this in your copy, simply select the text view to stylize, and then click the Attributes Inspector's View Background attribute to open a color chooser. Figure 7.16 shows our final text views.

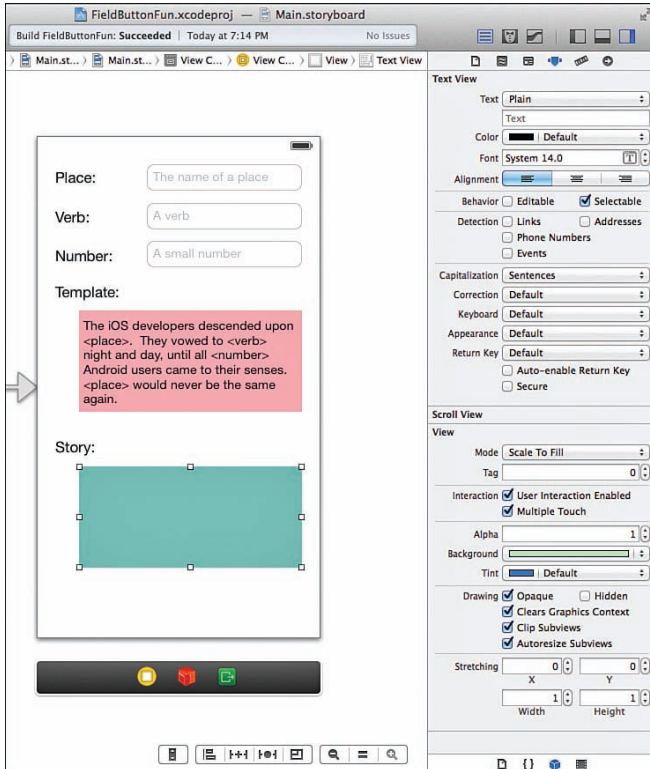


FIGURE 7.16

When completed, the text views should differ in color, editability, and content.

Using Data Detectors

Data detectors automatically analyze the content within onscreen controls and provide helpful links based on what they find. Phone numbers, for example, can be touched to dial the phone; detected web addresses can be set to launch Safari when tapped by the user. All of this occurs without your application having to do a thing. No need to parse out strings that look like URLs or phone numbers. In fact, all you need to do is click a button.

To enable data detectors on a text view, select the view and return to the Attributes Inspector (Command-1). Within the Text View Attributes area, click the check boxes under Detection: Phone Numbers to identify any sequence of numbers that looks like a phone number, Addresses for mailing addresses, Events for text that references a day/time, and Links to provide a clickable link for web and email addresses.

CAUTION

Practice Moderation!

Data detectors are a great convenience for users, but *can* be overused. If you enable data detectors in your projects, be sure they make sense. For example, if you are calculating numbers and outputting them to the user, chances are you don't want the digits to be recognized as telephone numbers.

Setting Scrolling Options When editing the text view attributes, you'll notice that a range of options exist that are specifically related to its ability to scroll, as shown in Figure 7.17.

Using these features, you can set the color of the scroll indicator (black or white), choose whether both horizontal and vertical scrolling are enabled, and even choose whether the scrolling area should have the rubber band “bounce” effect when it reaches the ends of the scrollable content.

Adding Styled Buttons

We need a single button in this project, so drag an instance of a button (`UIButton`) from the Objects Library to the bottom of the view. Title the button **Generate Story**. Figure 7.18 shows the final view and document outline, with a default button.

Although you're certainly welcome to use the standard buttons, our goal is do to something a bit more “flashy.” Before we get to the details, let's see what we can configure using the Xcode button attributes.

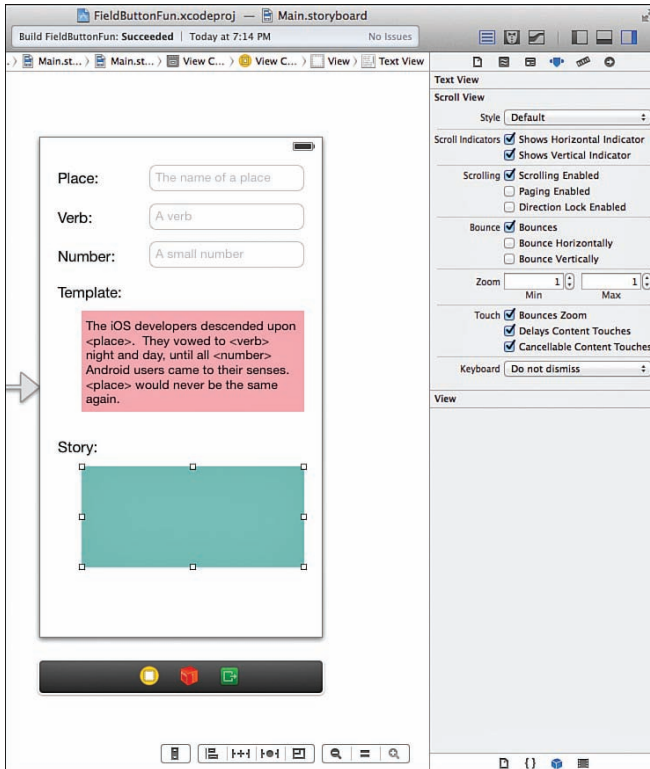


FIGURE 7.17 Scrolling regions have a number of attributes that can change their behavior.

Editing Button Attributes To edit a button's appearance, your first stop is, once again, the Attributes Inspector (Option-Command-4). Using the Attributes Inspector, you can dramatically change the appearance of the button. Use the Type drop-down menu, shown in Figure 7.19, to choose common button types:

- ▶ **System:** The default iOS 7 button style.
- ▶ **Detail Disclosure:** An arrow button used to indicate additional information is available.
- ▶ **Info Light:** An *i* icon, typically used to display additional information about an application or item. The light version is intended for dark backgrounds.
- ▶ **Info Dark:** The dark (light background) version of the Info Light button.
- ▶ **Add Contact:** A + button, often used to indicate the addition of a contact to the address book.
- ▶ **Custom:** A button that has no default appearance. Usually used with button images.

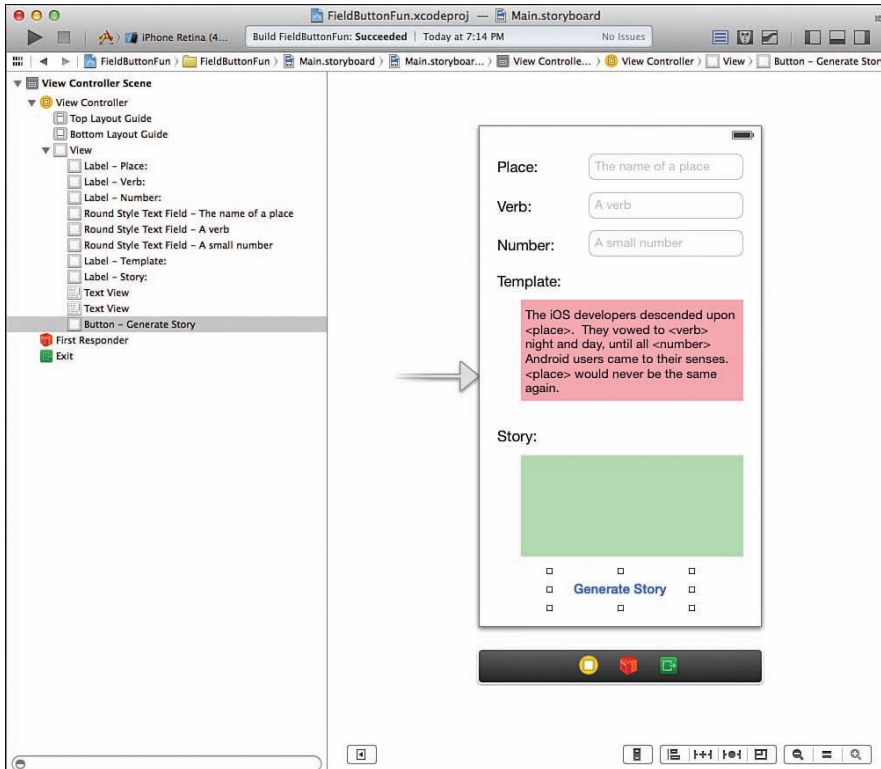


FIGURE 7.18

The default button style is little more than a label.

In addition to choosing a button type, you can make the button change depending on the current state of user interaction, a concept known as *changing state*. For instance, when a button is displayed before being touched, it is considered to be in its *default* state. When a user touches a button, it changes to show that it has been touched; this is the *highlighted* state. Use the State Config menu to select the button state you want to set up, and then use the various button attributes (images, colors, fonts, and so on) that should be applied in that state.

Other options include the ability to create shadowed text (Shadow Offset), show a tinted color when the button is highlighted, or display a “glow” around a user’s finger when he or she touches a button (Shows Touch on Highlight).

Setting Custom Button Images Even with all the settings available to configure your buttons, to create truly custom controls, you need to make custom images, including versions for the highlighted on state and the default off state. These can be any shape or size, but the PNG format is recommended because of its compression and transparency features.

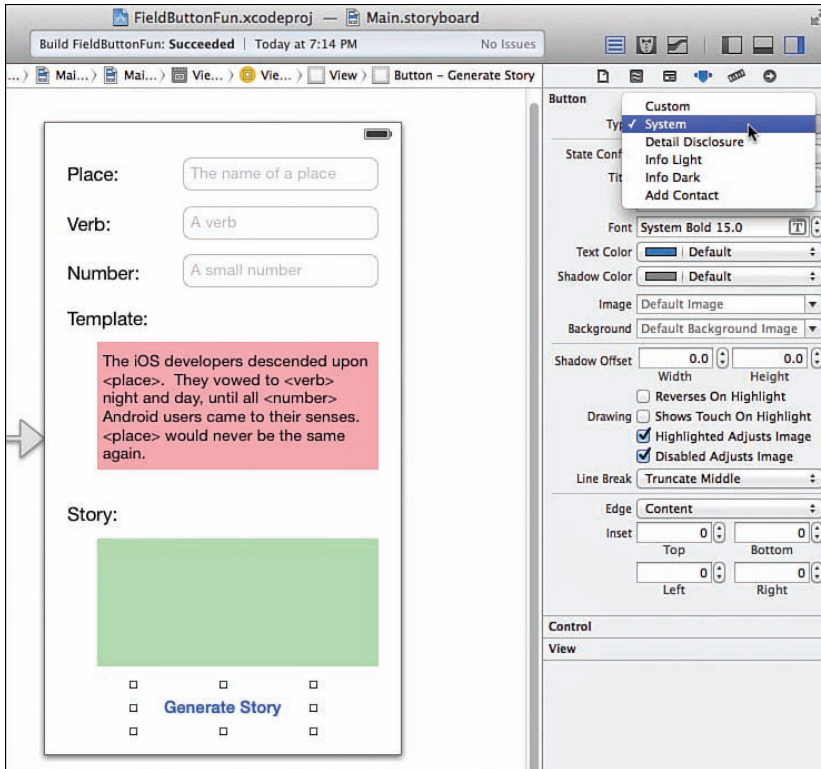


FIGURE 7.19
The Attributes Inspector gives several options for common button types and even a custom option.

After you’ve added these to the project through Xcode, you can select the image from the Image or Background drop-down menus in the Attributes Inspector. Using the Image menu sets an image that appears inside the button alongside the button title. This option enables you to decorate a button with an icon.

Using the Background menu sets an image that is stretched to fill the entire background of the button. The option lets you create a custom image as the entire button, but you must size your button exactly to match the image (*or* define slices for the image using the Xcode asset catalog). Do you get where we’re heading with this?

Assuming you followed the steps for defining button templates earlier, you should already have images that can resize to create “pretty” buttons. To create the fancy button for our project, select the `UIButton` you’ve added to the layout, make sure that the Attributes Inspector is open (Option+Command+4), and then complete these steps:

1. Set the button type to Custom. You can try using the default system type, but it applies its own highlighting effect that (in my opinion) looks “off” when applied to images.
2. Set the State Config drop-down to Default.
3. Use the Text Color drop-down to choose Black Color.
4. Use the Background drop-down to pick the `whiteButton` image.
5. Your layout should immediately update to show the fancy button. Figure 7.20 shows the button appearance and settings. Be sure to try resizing the button so you can see how the slicing comes into play.

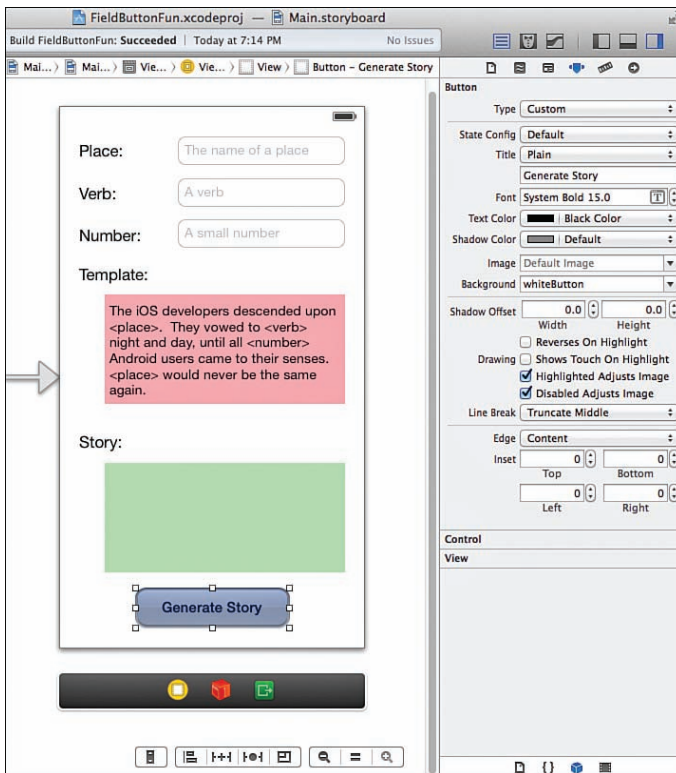


FIGURE 7.20

The “pretty” button appears, thanks to your earlier slicing.

6. Repeat steps 3–5, this time with the State Config set to Highlighted, and choosing white as the text color and `blueButton` for the image.

Because of the Xcode 5 slicing tools, everything “just works.” Your button looks great, regardless of the size, and you don’t have to write a single line of code to completely customize what it looks like!

NOTE

Prior to Xcode 5, we had to define resizable images using code, and *then* add those images to a button through code. For a peek inside this process, I provide you with these two lines from the previous edition of this book:

```
UIImage *normalImage = [[UIImage imageNamed:@"whiteButton.png"]
    resizableImageWithCapInsets:UIEdgeInsetsMake(0, 12, 0, 12)
[self.theButton setBackgroundImage:normalImage
    forState:UIControlStateNormal];
```

If you find this interesting, be sure to read more about the UIImage method `resizableImageWithCapInsets` and the UIButton method `setBackgroundImage:forState` in the Xcode documentation system.

Creating and Connecting the Outlets and Actions

With the interface finished, we now have a total of six text input/output areas that we need to access through our view controller code. In addition, we must create an action for the button that will trigger the generation of our story using the template and field contents.

In summary, a total of six outlets and one action require creation and connection:

- ▶ **Place field (UITextField):** `thePlace`
- ▶ **Verb field (UITextField):** `theVerb`
- ▶ **Number field (UITextField):** `theNumber`
- ▶ **Template Text view (UITextView):** `theTemplate`
- ▶ **Story Text view (UITextView):** `theStory`
- ▶ **Action triggered from Generate Story button:** `createStory`

Making sure that the `MainStoryboard.storyboard` file is open in the IB editor, use the editor toolbar buttons to switch to the assistant mode. You should now see your UI design and the `ViewController.m` file where you will be making your connections side by side.

Adding the Outlets

Start by Control-dragging from the Place text field to the line following the `@interface` directive in the `ViewController.m` file. When prompted, be sure to configure the connection as an outlet and the name as `thePlace`, leaving the other values set to their defaults (type `UITextField`, storage `weak`), as shown in Figure 7.21.

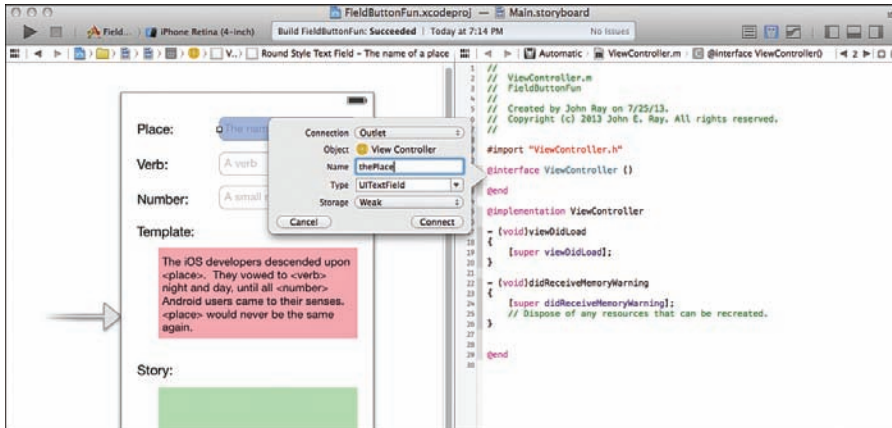


FIGURE 7.21
Create and connect outlets for each input/output element.

Repeat the process for the Verb and Number fields, connecting them to `theVerb` and `theNumber` outlets, this time dragging to just below the `@property` directive created when you added the first output. Connect the text views to `theTemplate` and `theStory` outlets. The process is identical (but the type is `UITextView`.)

That does it for the outlets. Now let's create our action.

Adding the Action

In this project, we add an action for a method we will call `createStory`. This action is triggered when the user clicks the Generate Story button. To create the action and generate an empty method that we can implement later, Control-drag from the Generate Story button to below the last `@property` directive in the `ViewController.m` file.

Name the action `createStory`, when prompted, as shown in Figure 7.22.

The connections to our interface are complete. The resulting `ViewController.m` file should have an `@interface` block at the top resembling the following:

```
@interface ViewController ()
@property (weak, nonatomic) IBOutlet UITextField *thePlace;
@property (weak, nonatomic) IBOutlet UITextField *theVerb;
@property (weak, nonatomic) IBOutlet UITextField *theNumber;
@property (weak, nonatomic) IBOutlet UITextView *theTemplate;
@property (weak, nonatomic) IBOutlet UITextView *theStory;

- (IBAction)createStory:(id)sender;
@end
```

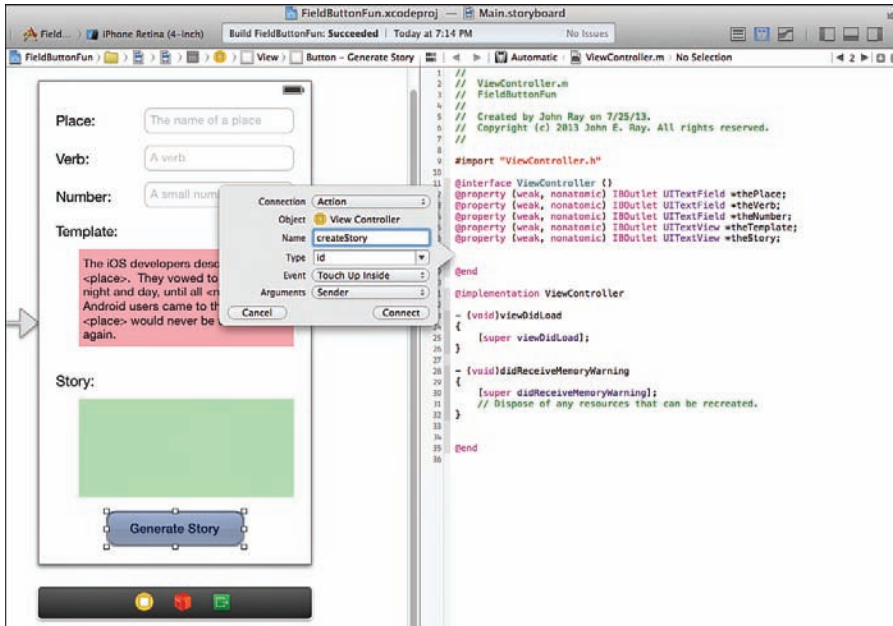


FIGURE 7.22

Create the action that will ultimately be used to generate our story.

Implementing Keyboard Hiding

Before completing the application by implementing the view controller logic to construct the story, we need to look at a “problem” that is inherent to applications with character entry: keyboards that won’t go away! To see what we mean, start the application again either on your device or in the iOS Simulator.

With your app up and running, click in a field. The keyboard appears. Now what? Click in another field; the keyboard changes to match the text input traits you set up, but it remains onscreen. Touch the word *Done*. Nothing happens. And even if it did, what about the number pad that doesn’t include a Done button? If you try to use this app, you’ll also find a keyboard that sticks around and that covers up the Generate Story button, making it impossible to fully utilize the UI. So, what’s the problem?

Hour 4, “Inside Cocoa Touch,” described *responders* as objects that process input. The first responder is the first object that has a shot at handling user input. In the case of a text field or text view, when it gains first-responder status, the keyboard is shown and remains onscreen until the field gives up or resigns first-responder status. What does this look like in code? For the field

thePlace, we could resign first-responder status and get rid of the keyboard with this line of code:

```
[self.thePlace resignFirstResponder];
```

Calling the `resignFirstResponder` method tells the input object to give up its claim to the input; as a result, the keyboard disappears.

Hiding with the Done Button

The most common trigger for hiding the keyboard in iOS applications is through the `Did End on Exit` event of the field. This event occurs when the Done (or similar) keyboard button is pressed.

We'll implement a new action method called `hideKeyboard` that is activated by the `Did End on Exit` events from our fields.

Turn your attention back to the `Main.storyboard` file and open the assistant editor. Control-drag from the Place field to the line just below the `createStory` action in the `ViewController.h` file. When prompted, configure a new action, `hideKeyboard`, for the `Did End on Exit` event. Leave all the other defaults the same, as shown in Figure 7.23.

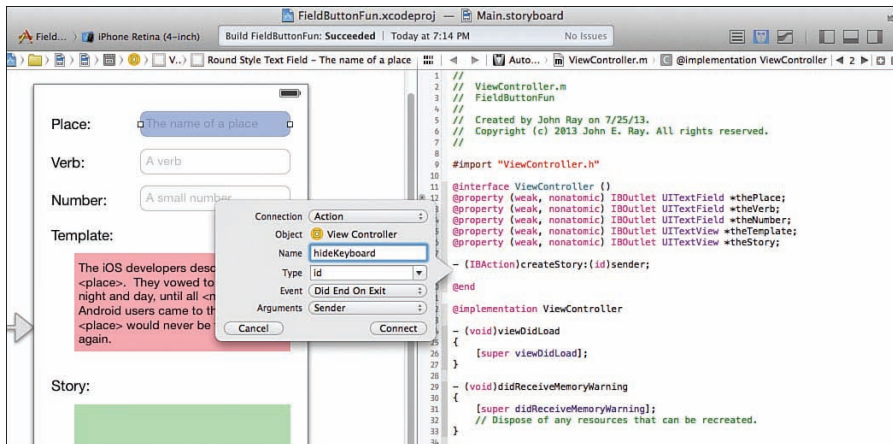


FIGURE 7.23
Add a new action method for hiding the keyboard.

Now you must connect the Verb field to the newly defined `hideKeyboard` action. There are many different ways to make connections to existing actions, but only a few enable us to target specific events. We'll use the technique you learned about in the tutorials in Hour 5, "Exploring Interface Builder": the Connections Inspector.

First switch back to the standard editor and make sure that the document outline is visible (Editor, Show Document Outline). Select the Verb field, and open the Connections Inspector by pressing Option-Command-6 (or choosing View, Utilities, Connections Inspector). Drag from the circle beside `Did End on Exit` to the View Controller icon in the document outline area. Release your mouse button and choose `hideKeyboard` when prompted, as shown in Figure 7.24.

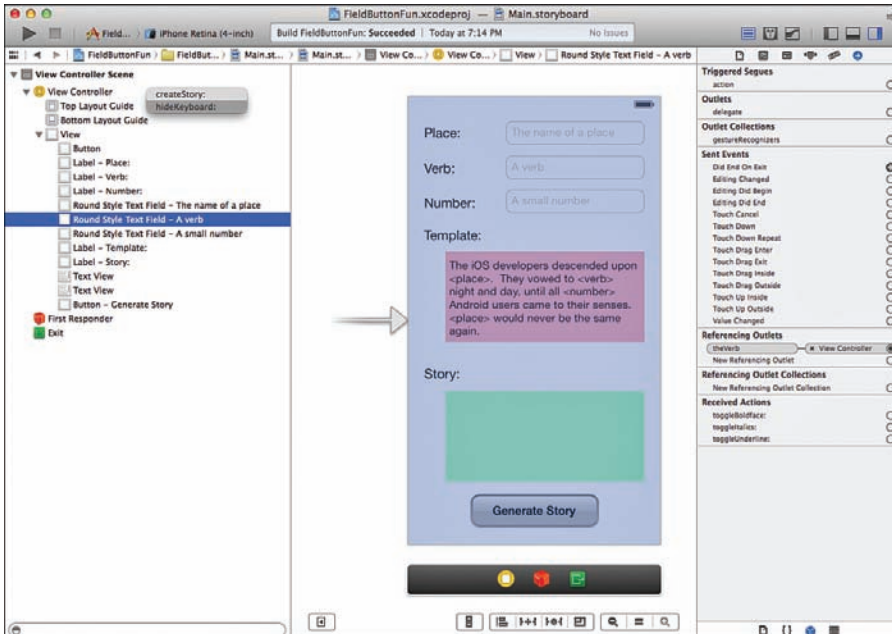


FIGURE 7.24
Connect the Verb field to the `hideKeyboard` action.

Unfortunately, now we run into a problem. The number input doesn't enter a Done button, and the text view doesn't support the `Did End on Exit` event, so how do we hide the keyboard for these variations?

Hiding with a Background Touch

A popular iOS interface convention is that if a keyboard is open and you touch the background (outside of a field) the keyboard disappears. This is the approach we need to take for the number-input text field and the text view—and functionality that we need to add to all the other fields to keep things consistent.

Wondering how we detect an event outside of a field? Nothing special: All we do is create a big invisible button that sits behind all the other controls, and then attach it to the `hideKeyboard` action method.

Within the IB editor, access the Object Library (View, Utilities, Object Library) and drag a new button (`UIButton`) from the library into the view.

Because this button needs to be invisible, make sure that it is selected, and then open the Attributes Inspector (Option-Command-4) and set the type to Custom and delete the button title. This makes the button entirely transparent. Use the selection handles to size the button to fill the entire view. With the button selected, choose Editor, Arrange, Send to Back to position the button in the back of the interface.

TIP

You can also drag an object to the top of the view hierarchy in the document outline to position it in the back. The objects are layered from the top (back) to the bottom (front).

To connect the button to the `hideKeyboard` method, it's easiest to use the document outline. Select the custom button you created (it should be at the top of the view hierarchy list), and then Control-drag from the button to the View Controller line. When prompted, choose the `hideKeyboard` method.

Nicely done. You're now ready to implement the `hideKeyboard` so that the Place and Verb fields can hide the keyboard when Done is touched, or the background can be touched to hide the keyboard in any situation.

Adding the Keyboard-Hiding Code

Because the user could be making changes in four potential places (`thePlace`, `theVerb`, `theNumber`, `theTemplate`), we must either identify the field the user is editing or simply resign first-responder status on all of them. As it turns out, if you resign first-responder status on a field that isn't the first responder, it doesn't matter. That makes implementing the `hideKeyboard` method as simple as sending the `resignFirstResponder` message to each of the properties representing our editable UI elements.

Scroll to the bottom of the `ViewController.m` file to find the `hideKeyboard` method stub that Xcode inserted for us when we created the `IBAction`. Edit the method so that it reads as shown in Listing 7.1.

LISTING 7.1 Hiding the Keyboard

```
- (IBAction)hideKeyboard:(id)sender {
    [self.thePlace resignFirstResponder];
    [self.theVerb resignFirstResponder];
    [self.theNumber resignFirstResponder];
    [self.theTemplate resignFirstResponder];
}
```

```
5:     self.theStory.text=[self.theStory.text
6:                         stringByReplacingOccurrencesOfString:@"<verb>"
7:                         withString:self.theVerb.text];
8:     self.theStory.text=[self.theStory.text
9:                         stringByReplacingOccurrencesOfString:@"<number>"
10:                        withString:self.theNumber.text];
11: }
```

Lines 2–4 replace the `<place>` placeholder in the template with the contents of the `thePlace` field, storing the results in the story text view. Lines 5–7 then update the story text view by replacing the `<verb>` placeholder with the appropriate user input. This is repeated again in lines 8–10 for the `<number>` placeholder. The end result is a completed story, output in the `theStory` text view.

Our application is finally complete.

Building the Application

To view and test the `FieldButtonFun`, click the Run icon in the Xcode toolbar. Your finished app should look similar to Figure 7.25, fancy button and all!

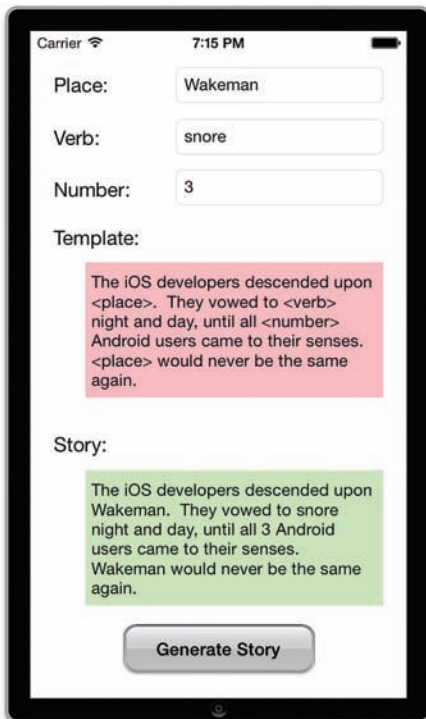


FIGURE 7.25

The finished application includes scrolling views, text editing, and a pretty button. What more could we want?

This project provided a starting point for looking through the different properties and attributes that can alter how objects look and behave within an iOS interface. The takeaway message: Don't assume anything about an object until you've reviewed how it can be configured.

CAUTION

Plain Is Pretty, Pretty Is Pretty

In this tutorial, we built a button that looks nothing like the default iOS 7 (or even iOS 6) buttons. Be careful, though, when creating “graphically rich” UIs for iOS 7 apps. Apple is embracing a simple and subtle approach to its UIs, so you might want to follow its lead. Create your UI to best fit with the application experience you want for your users.

Further Exploration

Throughout the next few hours, you'll explore a large number of UI objects, so your next steps should be to concentrate on the features you've learned in this hour—specifically, the object properties, methods, and events that they respond to.

For text fields and text views, the base object mostly provides for customization of appearance. However, you may also implement a delegate (`UITextFieldDelegate`, `UITextViewDelegate`) that responds to changes in editing status, such as starting or ending editing. You'll learn more about implementing delegates in Hour 10, “Getting the User's Attention,” but you can start looking ahead to the additional functionality that can be provided in your applications through the use of a delegate.

In addition, keep in mind that although there are plenty of properties to explore for these objects, there are additional properties and methods that are inherited from their superclasses. All UI elements, for example, inherit from `UIControl`, `UIView`, and `UIResponder`, which bring additional features to the table, such as properties for manipulating size and location of the object's onscreen display, as well as for customizing the copy and paste process (through the `UIResponderStandardEditActions` protocol). By accessing these lower-level methods, you can customize the object beyond what might be immediately obvious.

Apple Tutorials

Apple has provided a sample project that includes examples of almost all the available iOS UI controls:

UICatalog (accessible via the Xcode documentation): This project also includes a wide variety of graphic samples, such as the button images used in this hour's tutorial. It's an excellent playground for experimenting with the UI.

Summary

This hour described the use of common input features and a few important output options. You learned that text fields and text views both enable the user to enter arbitrary input constrained by a variety of different virtual keyboards. Unlike text fields, however, text views can handle multiline input as well as scrolling, making them the choice for working with large amounts of text. We also covered the use of buttons and button states, including how buttons can be manipulated through code.

We continue to use the same techniques you used in this hour throughout the rest of the book, so don't be surprised when you see these elements again.

Q&A

Q. Why can't I use a `UILabel` in place of a `UITextView` for multiline output?

A. You certainly can. The text view, however, provides scrolling functionality “for free,” whereas the label displays only the amount of text that fits within its bounds.

Q. Why doesn't Apple just handle hiding text input keyboards for us?

A. Although I can imagine some circumstances where it would be nice if this were an automatic action, it isn't difficult to implement a method to hide the keyboard. This gives you total control over the application interface—something you'll grow to appreciate.

Q. Are text views (`UITextView`) the only way to implement scrolling content?

A. No. You'll learn about implementing general scrolling behavior in Hour 9, “Using Advanced Interface Objects and Views.”

Workshop

Quiz

1. What tool will you use to create resizable images?
2. How do you get rid of an onscreen keyboard?
3. Are text views used for text input or output?

Answers

1. To define images that can be resized, you'll use the Xcode slicing tool, found within your asset catalog.
2. To clear the onscreen keyboard, you must send the `resignFirstResponder` message to the object that currently controls the keyboard (such as a text field).
3. Text views (`UITextView`) can be implemented as scrollable output areas or multiline input fields. It's entirely up to you.

Activities

1. Expand the story creator with additional placeholders, word types, and styled text editing. Use the same string manipulation functions described in this lesson to add the new functionality.
2. Modify the story creator to use a graphical button of your design. Use either an entirely graphical button or the stretchable image approach described in this hour's tutorial.

This page intentionally left blank

Index

Symbols

#import directive, 73-74, 81
sharing properties and methods, 329-330

@class directive, sharing properties and methods, 329-330

@implementation directive, 81

@import directive, 73-74

@interface directive, 74-75

@property directive, 76-78, 171

A

Accelerate framework (Core OS layer), 108

acceleration

- detecting, 631-643
- handling data, 640-641
- reading, Core Motion, 624-626

accelerometers, 620-622

Accessibility options, IB (Interface Builder), 150-151

Accounts framework (Core Services layer), 106

action sheets, 295-298

- button press response, 314-316
- implementing, 313-316

actions

- BackgroundColor project, creating and connecting, 516-518

BestFriend project, creating and connecting, 711-712

ColorTilt project, creating and connecting, 633-635

CustomPicker project, creating and connecting, 406-407

DateCalc project, creating and connecting, 391-392

FieldButtonFun project, creating and connecting, 219

FlowerWeb project, creating and connecting, 271-274

Gestures project, creating and connecting, 605-607

GettingAttention project, creating and connecting, 303-305

HelloNoun project, creating and connecting, 186-188

IB (Interface Builder), connecting to, 159-161

ImageHop project, creating and connecting, 242-246

LetsNavigate project, creating and connecting, 442-443

LetsTab project, creating and connecting, 450-452

ModalEditor project, creating and connecting, 360-361

Scroller project, creating and connecting, 285-286

Survey project, creating and connecting, 536-539

Address Book, 693-697, 708-725

- framework, 104, 106, 697
- implementing logic, 712-717

people picker delegate, 695-697

UI Framework, 694-695

alerts, 291-319

action sheets, implementing, 313-316

multibutton, creating, 306-309

System Sound Services, 298-300, 316-318

views, 292-295

- action sheets, 295-298
- button press response, 309-310
- fields, 310-312
- implementing, 305-312

AllInCode project, 577-583

building, 582-583

interface

- enabling orientation changes, 577
- programming, 577-582
- planning properties and connections, 577

allocation, objects, 86-88

animations

- loading, 247-248
- resources, adding, 233
- speed
 - incrementing, 251
 - setting, 249-251
- starting and stopping, 248-249

annotations, maps, 703-705

Apple, codec support, 648-650

Apple Developer Program, 6-9

- development provisioning profile, 13
- creating, 12-16

Apple developer tools, 20**application data sources, implementing, 488-492****application logic, 229**

- Address Book, implementing, 712-717
- BackgroundColor project, implementing, 518-521
- BackgroundDownload project, implementing, 782-784
- ColorTilt project, implementing, 635-642
- Cupertino project
 - implementing, 738-742
 - updating, 747-753
- FieldButtonFun project, implementing, 224-225
- FlowerColorTable project, implementing, 476-481
- FlowerWeb project, implementing, 274-278
- Gestures project, implementing, 607-616
- ImageHop project, implementing, 246-252
- LetsTab project, implementing, 452-455
- Maps, implementing, 717-719
- ModalEditor project, implementing, 362-363
- Orientation project, implementing, 628-630
- PopoverEditor project, implementing, 368-370
- ReturnMe project, implementing, 532-534
- Scroller project, implementing, 286-287
- SlowCount project, implementing, 776-778

- Survey project, implementing application logic, 539-543

- Swapper project, implementing, 587-589

- Universal project, implementing, 797-798

- UniversalToo project, implementing, 802

application resource constraints, 4**applications. See also projects**

- backgrounding, 757-762
- disabling, 763
- life-cycle methods, 761-762
- types, 758-760
- data storage, 503-505
 - direct file system access, 510-514
 - settings bundles, 508-510
 - user defaults, 507-508
- debugging, 809, 828-830
 - Xcode debugger, 813-828
- HelloXcode, 26
 - adding new code files to, 28
 - adding resources to, 28-32
 - building, 46-51
 - code completion, 38-40
 - managing properties, 51
 - navigating, 28-30
 - navigating code, 35-37
 - pragma marks, 41-42
 - removing files and resources, 32
 - searching code, 40-41
 - launching, Simulator, 59-60
- life cycle, tracing, 109-110
- logic, implementing, 189-190
- MVC (Model-View-Controller), 167-168
 - Single View Application template, 173-190

- structured design, 168-169

- Xcode implementation, 169-172

- running first, 17-18

- sliders, adding, 236-238

- steppers, adding, 239-240

- suspension, 758

- tracing, 809

- NSLog function, 810

- universal, 789, 805

- creating, 794-803

- development, 789-794

- multiple targets, 804-805

- setup changes, 791-794

- templates, 791-792

applicationWillEnterForeground method, 764**apps. See applications****ARC (Automatic Reference Counting), 662**

- Objective-C, 96-97

Archives and Serializations Programming Guide for Cocoa, 543**asset catalogs (Xcode), 32-35****assistant editor (Xcode), 42-44****attitude**

- displaying data, 639-640

- reading, Core Motion, 624-626

attributed text fields, versus plain, 207-209**attributes**

- bar button items, 376-377

- buttons, editing, 214-215

- date pickers, 378-379

- tables, setting, 462

- text fields

- editing, 205-207

- traits, 207

- text views, editing, 210-212

Attributes Inspector (IB), 149-150

audio, Cupertino project,
preparing for, 768-773

Auto Layout, 548, 550, 597-598
building responsive interfaces,
552-575
constraints, 552-562
centering, 562-565
Constraints object, navigating,
554-562
expanding controls, 567-572

Auto Layout system (IB), 145-148

**Automatic Reference Counting
(ARC). See ARC (Automatic
Reference Counting)**

AV Audio Player, 655
controlling audio playback,
675-677
handling playback completion,
656
implementing, 674-675

AV Audio Recorder, 656-657
implementing, 670-674

**AV Foundation framework, 105,
655-657**

AV Audio Player, 655
controlling playback,
675-677
handling playback
completion, 656
implementing, 674-675
AV Audio Recorder, 656-657
implementing, 670-674

B

background fetches, 760
performing, 780-784

background processing
local notifications, 758-759
implementing, 765-767
task-specific, 759, 768-773

background suspension, 758
handling, 764-765

background tasks, 785-786
completing long-running,
773-780
processing, enabling, 778-780

BackgroundColor project, 544
application logic,
implementing, 518-521
building, 521-522
designing interface, 515
outlets and actions, creating
and connecting, 516-518
setting up, 515

BackgroundDownload project
building, 784
creating and connecting
outlet, 782
designing interface, 781
implementing application
logic, 782-784
setting up, 780-781

backgrounding, 757-762
disabling, 763
life-cycle methods, 761-762
types, 758-760

bar button items, 375-376
attributes, 376-377
navigation controllers,
424-425

BestFriend project
Address Book, implementing
logic, 712-717
building, 723-724
designing interface, 745
email logic, implementing,
720-721
Maps, implementing logic,
717-719
outlets and actions, creating
and connecting, 711-712
setting up, 708-709

social networking logic,
implementing, 721-723
status bar, setting to white,
723

blocks, 90-91

blur effect (iOS 7), adding, 743

blurring effect, 278, 525

bookmarks, setting, 124-125

**breakpoint navigator, Xcode
debugger, 826-828**

breakpoints, setting, 816-818

**browsing, documentation,
123-124**

**building applications, Xcode,
46-51**

**button templates, preparing with
slicing, 199-204**

buttons, 195-196, 226-227

action sheets, button press
response, 314-316

alert views, button press
response, 309-310

attributes, editing, 214-215

bar button items, 375-376
attributes, 376-377

images
adding to, 200
setting custom, 215-218

styled, adding, 212

templates, preparing with
slicing, 199-204

C

**calculateDateDifference method,
398-400**

case sensitivity, Objective-C, 72

Castillo, Cesar Pinto, 275

categories

classes, 70

defining, 82-83

cells, tables, 460

- creating, 493-494
- disabling editing, 495

centering constraints, Auto Layout, 562-565**CFNetwork framework (Core Services layer), 106****chooseImage method, 677-678****class methods, 70****classes, 69**

- categories, 70
- Cocoa Touch, 110-111
 - core application, 111-113
 - data type, 113-117
 - interface, 117-120
- GenericViewController, 440
- MPMediaItem, 648
- MPMediaItemCollection, 648
- MPMediaPickerController, 648
- MPMoviePlayerController, 648
- MPMusicPlayerController, 648-650
- NSArray, 113-114
- NSDate, 116
- NSDecimalNumber, 115-116
- NSDictionary, 114-115
- NSMutableArray, 113-114
- NSMutableDictionary, 114-115
- NSMutableString, 113
- NSNumber, 115-116
- NSObject, 111
- NSString, 113
- NSURL, 116-117
- singletons, 69
- subclasses, 69
- superclasses, 69
- UIActionSheet, 295-298
- UIAlertView, 292-295
- UIApplication, 111
- UIBarButtonItem, 375
- UIButton, 117, 195, 199-204, 231

UIControl, 112

- UIDatePicker, 119, 377-378, 417
- UIDevice, 623-624
- UIGestureRecognizer, 617
- UIImagePickerController, 648
- UIImageView, 231, 681
- UILabel, 117, 196, 231
- UILongPressGesture Recognizer, 594, 617
- UIPanGestureRecognizer, 594, 617
- UIPicker, 119
- UIPickerView, 377, 417
- UIPinchGestureRecognizer, 594
- UIPopoverController, 120, 347-348, 373
- UIResponder, 112
- UIRotationGestureRecognizer, 594
- UIScrollView, 261, 279, 288
- UIScrollViewDelegate, 288
- UISegmentedControl, 118, 259, 288
- UISlider, 118, 230-231, 252
- UIStepper, 118, 230-231, 252
- UISwipeGestureRecognizer, 594
- UISwitch, 118, 258, 288
- UITapGestureRecognizer, 594
- UITextField, 119, 196
- UITextFieldDelegate, 226
- UITextView, 119, 196
- UITextViewDelegate, 226
- UIView, 112, 259
- UIViewController, 113
- UIWebView, 288
- UIWindow, 112
- ViewController, 436, 446

Cocoa, versus Cocoa Touch, 103**Cocoa Touch, 20, 101-103, 129**

- application life cycle, tracing, 109-110
- classes, 110-111
 - core application, 111-113
 - data type, 113-117
 - interface, 117-120
- versus Cocoa, 103
- functionality, 102-103
- iOS technology layers, 104-105

code

- HelloXcode app
 - adding pragma marks, 41-42
 - searching, 40-41
- loops, 93-95
- setting breakpoints, 816-818
- stepping through, 820-822
- writing
 - IB (Interface Builder), 162-163
 - Xcode, 188

code files, Xcode projects, adding to, 28-32**code listings**

- Accessing Properties from the Popover Upon Dismissal, 348
- Accessing the Popover's contentViewController, 368
- Activating Interface Rotation, 549
- Adding a Method in GenericViewController.m to Update Each Scene's Counter, 455
- Adding Audio Feedback When the Heading Updates, 771-772
- Adding the foundRotation Method, 613
- Adding the getFlower Implementation, 276

- Applying a Filter to the Image in the UIImageView, 681
- Asking to Become a First Responder, 615-616
- Calculating a Heading to a Destination, 750-751
- Calculating the Date Difference, 400
- Calculating the Difference Between Two Dates, 398-399
- Calculating the Distance When the Location Updates, 741
- Calling the NSLog Function, 810
- Centering the Map and Adding an Annotation, 718
- Changing the Label as the Orientation Changes, 629-630
- Cleaning Up After the Movie Player, 670
- Completed setOutput Method, 190
- Completed setSpeed Method, 249-250
- Completed ViewController.h Interface File, 246
- Completing the recordAudio Method, 676
- Configuring a Cell to Display in the Table View, 478
- Configuring and Displaying the Mail Compose View, 720
- Configuring the Detail View Using the detailItem, 497
- Configuring the Sections and Row Count for the Table View, 465
- Creating a Method to Display the User's Selection, 414-415
- Creating and Initializing the Audio Recorder, 672-673
- Creating the Location Manager Instance, 739
- Customizing the Annotation View, 705, 719
- Defining the Minimum Background Fetch Interval, 783
- Detecting and Displaying the Active Device, 798
- Disabling Editing of Table Cells, 495
- Disabling Editing of the UI, 495
- Disabling Interface Rotation, 642
- Disabling the Upside-Down Orientation, 584
- Disallowing Drilldown Past an Individual Contact, 714
- Dismissing the Mail Compose View, 721
- Dismissing the Modal Scene, 396, 409
- Dismissing the People Picker, 695
- Displaying the Media Picker, 685-686
- doAlertInput Implementation, 311
- Editing the viewDidLoad Method, 815
- Enabling iPhone Access to the Detail View Controller, 499
- Enabling Scrolling in Your Scroll View, 287
- Enabling the Ability to Be a First Responder, 615
- Example of the Tap Gesture Recognizer, 594-595
- Final viewDidLoad Implementation, 684-685
- Finishing the Background Fetch by Implementing application:performFetchWithCompletionHandler, 783
- Forward Geocoding, 706
- Grabbing and Configuring the UIPopoverController, 347
- Handling a Row Selection Event, 480
- Handling a User's Music Selection, 686
- Handling Button Touches, 582
- Handling Drilling Down to Individual Properties, 696
- Handling Empty Selections in the Media Picker, 687
- Handling Heading Updates, 734
- Handling Location Manager Errors, 740
- Handling Playback Completion, 656
- Handling Rotation in didRotateFromInterfaceOrientation, 582
- Handling the Alert View Input Fields, 312
- Handling the Canceling of the People Picker Display, 713
- Handling the Cancellation of a Media Selection, 652
- Handling the Cancellation of an Image Selection, 660, 680
- Handling the Composition Completion, 700
- Handling the Dismissal of the Popover, 370
- Handling the Heading Updates, 752
- Handling the Notification of Playback Completion, 651
- Handling the Selection of a Contact, 715
- Handling the Selection of an Address Book Record, 696
- Handling the Selection of an Image, 659
- Handling the Selection of Media Items, 653
- Handling the User's Selection of an Image, 679
- Hiding the Keyboard, 223

- Hiding the Keyboard When It Isn't Needed, 539
- Hiding the Keyboard When Its Done Key Is Pressed, 363
- Implementation File, 79
- Implementing a Custom Picker Data Source Protocol, 380-381
- Implementing a Custom Picker Delegate Protocol, 382
- Implementing a Simple Tweet Compose View, 722-723
- Implementing a UIActionSheet Class, 295
- Implementing a UIAlertView Object, 293
- Implementing Location Updates, 729
- Implementing playAudio Method, 675
- Implementing the applicationWillEnter Foreground Method, 764
- Implementing the chooseImage Method, 677-678
- Implementing the control Hardware Method, 637
- Implementing the createStory Method, 224-225
- Implementing the describe Integer Method, 815
- Implementing the doAcceleration Method, 640-641
- Implementing the doActionSheet Method, 313
- Implementing the doAlert Method, 306
- Implementing the doAttitude Method, 640
- Implementing the doRotation Method, 641
- Implementing the doSound Method, 317
- Implementing the doVibration Method, 318
- Implementing the Final setBackgroundHueValue Method, 519
- Implementing the foundPinch Method, 610-611
- Implementing the foundSwipe Method, 610
- Implementing the foundTap Method, 609
- Implementing the incrementCount Method, 444
- Implementing the Initial doMultipleButtonAlert Method, 308
- Implementing the Initial setBackgroundHueValue Method, 518
- Implementing the iPad's Popover-Enabled chooseImage Method, 678-679
- Implementing the newBFF Method, 713
- Implementing the playMusic Method, 688
- Implementing the setIncrement Method, 251
- Implementing the setValuesFromPreferences Method, 533
- Implementing the showResults Method, 541
- Implementing the storeSurvey Method, 540
- Implementing the toggleFlowerDetail Method, 275
- Implementing the viewDidAppear Method, 608-609
- Implementing updateInterface, 580
- Initializing the Interface When the Application Loads, 581
- Initializing the Motion Manager, 637
- Initializing the Movie Player, 668
- Initializing the Sound File References in viewDidLoad, 770
- Initiating Movie Playback, 669
- Loading and Playing a Sound, 299
- Loading the Animation, 247
- Loading the Data Required for the Picker View, 410
- Loading the Settings When the Initial View Loads, 534
- Performing a Default Calculation When the Date Chooser Is First Displayed, 400
- Placing an Annotation, 704
- Populating the Field with the Current Email Address, 362
- Populating the Flower Arrays, 476
- Populating the Flower Data Structures, 489-490
- Preparing and Showing the Compose Dialog, 699
- Preparing the Audio Player with a Default Sound, 674-675
- Preparing the Interface (But Don't Display It Yet), 578-579
- Preparing to Post to Facebook, 701
- Presenting the Picker with Custom Views, 383-384
- Processing a UIImage with a CIFilter, 661
- Providing a Custom View for Each Possible Picker Element, 412

- Reacting to a User's Selection, 415
- Reacting to a User's Touch, 467-468
- Reacting to Core Location Errors, 731
- Requesting Heading Updates, 747-748
- Responding to a Shake Gesture, 616
- Responding to an Action Sheet, 298
- Responding to an Action Sheet Button Press, 314
- Responding to an Alert View, 295
- Responding to an Alert View Button Press, 310
- Returning a Count of the Rows (Array Elements) in Each Section, 477
- Returning a Heading for Each Section, 478
- Returning the Number of Components, 411
- Returning the Number of Elements per Component, 411
- Returning the Number of Sections in the Table, 477
- Reverse Geocoding, 707
- Rotating the View into the Proper Orientation, 589
- Sample Interface File, 73
- Scheduling a Timer When the Application Starts, 777
- Setting a Custom Height and Width for the Picker Components and Rows, 413-414
- Setting a Default Selection, 416
- Setting the Delegate During the Segue, 394, 408
- Setting the Detail View Controller's detailItem, 496
- Setting the End of Background Processing, 779
- Setting the Initial Scene's Label to the Editor Scene's Field, 363
- Setting the Start of Background Processing, 779
- Setting the Status Bar Appearance in preferredStatusBarStyle, 252, 742
- Setting Up and Displaying the Image Picker, 658
- Showing the Animal Chooser Scene, If Needed, 408
- Showing the Date Chooser Scene, If Needed, 395
- Silly Implementation of table View:cellForRowAtIndexPath, 466-467
- Starting and Stopping the Animation in toggleAnimation, 248
- Storing the Recently Received Location for Later Use, 748-749
- Supporting All Interface Orientations, 577
- Toggling the Animal Chooser's Visibility Flag to NO, 408
- Toggling the Date Chooser's Visibility Flag to NO, 395
- Typical Setup and Display of a Media Picker, 652
- Updating doAlert to Register a Local Notification, 766-767
- Updating the Counter, 778
- Updating the Display in viewWillAppear:animated, 445
- Updating the Display Using the Counter Values, 453
- Updating the Initial recordAudio Method, 673
- Updating the Settings in viewDidLoad, 520
- Updating the Tab Bar Item's Badge, 454
- Updating the viewDidLoad Method to Set the Initial Display, 277
- Updating viewDidLoad to Loop 2,000 Times, 823-824
- Using prepareForSegue:sender to Grab the View Controllers, 340
- Using the Motion Manager, 625
- Watching for Orientation Changes, 629
- Your First Code Exercise, 38-39
- codec support, Apple Developer Program, 648-650**
- ColorTilt project, 631-632**
 - building, 642-643
 - creating and connecting outlets and actions, 633-635
 - designing interface, 633-634
 - implementing application logic, 635-642
 - setting up, 632-633
- commands, File menu, New Project, 24**
- compass, 744-753**
- configuring devices for development, 13-16**
- connections**
 - AllInCode project, planning, 577
 - editing, Quick Inspector, 161-162
 - Single View Application template (MVC), planning, 179-182
- connectivity, iOS devices, 5**
- constraints**
 - Auto Layout, 552-562
 - centering, 562-565
 - IB Auto Layout system, 145-148

controlHardware method, 637-638

controller delegate

Image Picker, 659-660

Media Picker, 652-653

controllers

detail view

fixing broken reference, 498-499

implementing, 496-498

master view, implementing, 492-496

navigation, 423-428, 434-445, 456-457

bar button items, 424-425

storyboards, 425-428

segmented, 288-289

split view, 499

tab bar, 429-434, 445-457

creating relationships, 448-449

storyboards, 430-434

view, 323, 421-422

multiscene development, 422

MVC (Model-View-Controller), 170-172

relationships, 323

storyboards, adding supporting subclasses, 326-329

controls

expanding, Auto Layout, 567-572

segmented, 258-259

convenience methods, 87

core application classes (Cocoa Touch), 111-113

Core Audio framework (Media layer), 105

Core Bluetooth framework (Core OS layer), 108

Core Data, 106, 543

Core Data Core Competencies tutorial, 543

Core Foundation framework (Core Services layer), 107

Core Graphics framework (Media layer), 105

Core Image, 105, 660-662

filters, 660-661, 681-682

Core Location, 107, 727, 734-735, 754-755

headings, obtaining, 732-734

location manager, 728-730

creating instance, 739

implementing delegate, 739

preparing, 738-739

locations

accuracy and update filter, 732

handling errors, 730-731

obtaining, 727-732

Core Motion, 107, 643-644

motion manager, initializing, 636-637

Core OS technology layer (iOS), 108-109

Core Services technology layer (iOS), 106-108

Core Text framework (Media layer), 105

CPUs (central processing units), monitoring, 828

createFlowerData method, 489, 491

createStory method, implementing, 224-225

Cupertino project, 735

adding background modes, 773

application logic, updating, 747-753

building, 742-743, 753

creating and connecting outlets, 737, 747

designing view, 736-738

implementing application logic, 738-742

preparing for audio, 768-773

setting up, 735-736, 744-745

updating user interface, 745-746

custom images, buttons, setting, 215-218

custom pickers, 401-416

CustomPicker project, 401-402

building, 416

creating segues, 406

custom picker view, implementing, 409-416

designing interface, 404-406

outlets and actions, creating and connecting, 406-407

scene segue logic, implementing, 407-409

setting up, 402-403

D

data detectors, 213

data models, MVC (Model-View-Controller), 172

data source outlets, FlowerColorTable project, connecting, 474-475

data storage, 503-505

direct file system access, 510-514

file system storage, implementing, 535-543

locations, 511-512

settings bundles, 508-510

user defaults, 507-508

data type classes (Cocoa Touch), 113-117

data types, 85-86

primitive, 85

- date format strings, 397
 - date pickers, 378-379, 385-401
 - DateCalc project, 386**
 - building, 400-401
 - creating segue, 389-391
 - date calculation logic, implementing, 396-400
 - designing interface, 388-389
 - outlets and actions, creating and connecting, 391-392
 - scene segue logic, implementing, 393-396
 - setting up, 386-388
 - debug navigator, Xcode debugger, 826-828**
 - debugging applications, 809, 828-830**
 - Xcode debugger, 813-828
 - declaring variables, 84-86**
 - defining methods, 78-79**
 - delegate protocols, picker view, 381-383**
 - delegates**
 - FlowerColorTable project, connecting, 474-475
 - segues, setting, 408
 - describeInteger method, 815**
 - designing applications, MVC (Model-View-Controller), 167-168**
 - Single View Application template, 173-190
 - structured design, 168-169
 - Xcode implementation, 169-172
 - detail scene, updating, 484-488**
 - detail view controllers**
 - fixing broken reference, 498-499
 - implementing, 496-498
 - detailItem public property, 496-497**
 - developers (iOS), becoming, 6-12**
 - development provisioning profile, 13**
 - creating, 12-16
 - Device command (Simulator), 62**
 - Device iOS Version command (Simulator), 63**
 - devices (iOS), 1-2**
 - application resource constraints, 4
 - configuring for development, 13-16
 - connectivity, 5
 - devices, display and graphics, 2-3
 - input and feedback mechanisms, 5
 - orientations, setting, 53
 - direct file system access, data storage, 510-514**
 - directives, 74**
 - #import, 73-74, 81
 - @implementation, 81
 - @import, 73-74
 - @interface, 74-75
 - @property, 76-78, 171
 - disabling**
 - backgrounding, 763
 - editing of table cells, 495
 - editing of the UI, 495
 - interface rotation, 642
 - dismissals, popovers, responding to, 346-348**
 - displaying**
 - media picker, 685-686
 - popovers manually, 346
 - displays**
 - iOS devices, 2-3
 - popovers, programming, 349-352
 - doAcceleration method, 640-641**
 - doActionSheet method, 313**
 - doAlert method, 305-306, 311**
 - doAttitude method, 639-640**
 - document outlines, IB (Interface Builder) storyboards, 133-137**
 - documentation**
 - browsing, 123-124
 - downloading, 121
 - searching, 121-122
 - documentation system, Xcode, 121-125**
 - documents**
 - navigating, 124-125
 - sharing, 125
 - doRotation method, 641-642**
 - downloading documentation, 121**
- ## E
- editing**
 - button attributes, 214-215
 - text field attributes, 205-207
 - text view attributes, 210-212
 - tools, IB (Interface Builder), 141-145
 - email messages**
 - composing, 698-700
 - logic, implementing, 720-721
 - Event Kit framework (Core Services layer), 107**
 - Event Kit UI framework (Cocoa Touch), 104**
 - events, multi-touch, generating, 61**
 - exits, scenes, 323, 334-336**
 - expressions, 91-95**
 - External Accessory framework (Core OS layer), 108**

F**Facebook, posting to, 701-702****feedback mechanisms, iOS devices, 5****fetches, background, 760**
performing, 780-784**FieldButtonFun project, 197**actions, creating and
connecting, 219application logic,
implementing, 224-225

building, 225-226

designing interface, 204-218

keyboard hiding,
implementing, 220-224outlets, creating and
connecting, 218-219preparing button templates
with slicing, 199-204

setting up, 198-199

**fields, alert views, adding to,
310-312****File menu commands, New
Project, 24****file patches, obtaining, 512-513****file structure, Objective-C, 72-84****file system access, data storage,
510-514****file system storage, implementing,
535-543****files**

header, 73-80

ending, 79-80

implementation, 79-82

Xcode projects, removing, 32

filters, Core Image, 660-661**FlowerColorTable project, 471-481**application logic,
implementing, 476-481

building, 481

connecting delegates and
data source outlets,
474-475

designing interface, 473-475

FlowerWeb projectactions, creating and
connecting, 271-274application logic,
implementing, 274-278

building, 278-279

designing interface, 263-270

outlets, creating and
connecting, 270-272

setting up, 263

**Foundation framework (Core
Services layer), 107****frameworks, 101**Cocoa Touch technology layer,
104-105

Media Player, 648-654

G**Game Kit framework (Cocoa
Touch), 104****GenericViewController class, 440****geocoding, 705-708****gesture recognizers**

adding, 594-595

adding to views, 601-605

tap gesture recognizer,
594-595

using, 596-617

gestures, 593, 617

gesture recognizers

adding, 594-595

adding to views, 601-605

tap gesture recognizer,
594-595**using, 596-617**

long pressing, 594

multi-touch gesture
recognition, 593-594

panning, 594

pinching, 594

rotating, 594

shaking, 594

swiping, 594

tapping, 594

Gestures project, 596-598application logic,
implementing, 607-616

building, 607-616

designing interface, 599-601

gesture recognizers, adding to
views, 601-605outlets and actions, creating
and connecting, 605-607

setting up, 598-599

**Getting Started with Audio &
Video, 689-691****GettingAttention project**action sheets, implementing,
313-316actions and outlets, creating
and connecting, 303-305alert views, implementing,
305-312

designing interface, 302-303

setting up, 301-302

graphics, iOS devices, 2-3**GUI (graphical user interface), 131****guides (IB), 141-142****gyroscopes, 622-623****H****hardware, motion, 619-623**

accelerometers, 620-622

gyroscopes, 622-623

header files, 73-80

ending, 79-80

**headings, Core Location,
obtaining, 732-734****HelloNoun project**actions, creating and
connecting, 186-188

building, 190

designing interface, 182-185
 implementing application logic, 189-190
 outlets, creating and connecting, 184-188
 planning properties and connections, 179-182
 setting up, 174-179
HelloXcode app, 26
 adding new code files to, 28
 adding resources to, 28-32
 assistant editor, 42-44
 building, 46-51
 code
 completion, 38-40
 navigating, 35-37
 pragma marks, 41-42
 searching, 40-41
 managing properties, 51
 navigating, 28-30
 removing files and resources, 32
hiding keyboard, 223, 363, 539
Home command (Simulator), 63



iAD framework (Cocoa Touch), 105
IB (Interface Builder), 131-132, 165
 Accessibility options, 150-151
 actions, connecting to, 159-161
 Attributes Inspector, 149-150
 Auto Layout system, 145-148
 editing connections, Quick Inspector, 161-162
 editing tools, 141-145
 Identity Inspector, 150-151
 object identity, 163

interfaces
 creating, 137-148
 customizing appearance, 148-153
 previewing, 151-153
 Object Library, 137-139
 outlets, 156-158
 projects, opening, 154-155
 resources, 164-165
 storyboard, 133-137
 views, adding objects to, 140-141
 writing code with, 162-163
Identity Inspector (IB), 150-151
 object identity, 163
if-then-else statements, 91-93
Image I/O framework (Media layer), 105
Image Picker, 657-660
 controller delegate, 659-660
 implementing, 677-681
image views, 231
ImageHop project, 231-253
 application logic, implementing, 246-252
 building, 252-253
 designing interface, 234-243
 implementing animated image views, 246-248
 outlets and actions, creating and connecting, 242-246
 setting up, 233-234
images
 adding to asset catalogs, 33-35
 adding to buttons, 200
 buttons, setting custom, 215-218
 retina image assets, 35
imperative development, OOP (object-oriented programming), 68
implementation files, 79-82

implicit preferences, creating, 514-522
incrementCount method, 444
Information Property List Key Reference, 543
initialization, objects, 86-88
input mechanisms, iOS devices, 5
Inspector (Quick Help), 127-128
installation, Xcode, 9-12
instance method, 70
instance variables, 70
 versus properties, 439
instantiation, 70
Interface Builder (IB). See IB (Interface Builder)
interface classes (Cocoa Touch), 117-120
interface files, 73-80
 ending, 79-80
interfaces, 547-548. See also motion; orientation
 BackgroundColor project, designing, 515
 BackgroundDownload project, designing, 781
 building responsive, Auto Layout, 552-575
 buttons
 adding, 212
 editing attributes, 214-215
 setting custom images, 215-218
 ColorTilt project, designing, 633-634
 controls, segmented, 258-259
 creating, 137-148
 Cupertino project, updating, 745-746
 customizing appearance, 148-153
 CustomPicker project, designing, 404-406
 DateCalc project, designing, 388-389

- disabling rotation, 642
 - FieldButtonFun project, designing, 204-218
 - FlowerColorTable project, designing, 473-475
 - FlowerWeb project, designing, 263-270
 - Gestures project, designing, 599-601
 - GettingAttention project, designing, 302-303
 - HelloNoun project, designing, 182-185
 - ImageHop project, designing, 234-243
 - iPad, tweaking, 484-486
 - iPhone, tweaking, 486-488
 - keyboard hiding, implementing, 220-224
 - LetsNavigate project, designing, 440-442
 - LetsTab project, designing, 449-451
 - MediaPlayground project, designing, 664-665
 - modal, 292
 - ModalEditor project, designing, 356-358
 - orientation changes, enabling, 577
 - Orientation project, designing, 627-628
 - pickers, 373, 377-378
 - custom, 401-416
 - date, 378-379, 385-401
 - views, 379-385
 - PopoverEditor project, designing, 366
 - previewing, 151-153
 - programmatically defined, 574-583
 - programming, 577-582
 - resizable, designing, 550-552
 - ReturnMe project, designing, 523-525
 - rotatable, designing, 550-552
 - rotation, 590
 - enabling, 548-549
 - swapping views on, 582-590
 - rotation events, 624
 - Scroller project, designing, 280-285
 - scrolling options, setting, 212
 - scrolling views, 261, 279-288
 - sliders, 230
 - SlowCount project, designing, 775
 - steppers, 230-231
 - Survey project, designing, 535-537
 - Swapper project, designing, 584-585
 - switches, 258
 - adding, 266-267
 - text fields
 - adding, 204-205
 - attribute traits, 207
 - editing attributes, 205-207
 - text views
 - adding, 209-210
 - editing attributes, 210-212
 - toolbars, 373-377
 - bar button items, 375-377
 - Universal project, designing, 796-797
 - UniversalToo project, designing, 801-802
 - web views, 259-261
 - adding, 267
 - Xcode, 27-28
- iOS**
- application life cycle, tracing, 109-110
 - devices, 1-2
 - application resource constraints, 4
 - configuring for development, 13-16
 - connectivity, 5
 - display and graphics, 2-3
 - input and feedback mechanisms, 5
 - screen, accommodating different sizes, 3
 - technology layers, 103
 - Cocoa Touch, 104-105
 - Core OS, 108-109
 - Core Services, 106-108
 - Media, 105-106
 - version detection, 583
 - iOS Simulator, 23, 58-63**
 - generating multi-touch events, 61
 - Hardware menu commands, 62-64
 - launching applications, 59-60
 - rotating device, 62
 - iPad**
 - interface, tweaking, 484-486
 - popovers, 315, 341-352, 364-370
 - creating segues, 343-346
 - displaying manually, 346
 - preparing, 342-343
 - programming displays, 349-352
 - responding to dismissals, 346-348
 - screen, 2
 - split view controllers, 468-469
 - targets, 804-805
 - iPhone**
 - interface, tweaking, 486-488
 - screen, 2
 - targets, 804-805

K-L**keyboard hiding, 363**

interfaces, implementing,
220-224

labels, 196**launch images, setting, 57-58****launching applications, Simulator, 59-60****LetsNavigate project**

building, 445
designing interface, 440-442
outlets and actions, creating
and connecting, 442-443
push segues, creating,
440-441
setting up, 436-440

LetsTab project

application logic,
implementing, 452-455
building, 455-456
designing interface, 449-451
outlets and actions, creating
and connecting, 450-452
setting up, 446-448

libraries, IB (Interface Builder), Object Library, 137-139**life-cycle methods,**

background-aware, 761-762

listings. See code listings**local notifications**

background processing,
758-759
implementing, 765-767

LocateMe, 754**location manager (Core Location), 728-730**

delegate, implementing,
740-742
instance, creating, 739
preparing, 738-739

locations

accuracy and update filter,
732

adding constants, 736

compass, 744-753

data storage, 511-512

handling errors, 730-731

obtaining, 727-732

Lock command (Simulator), 63 logic, 229

Address Book, implementing,
712-717

BackgroundColor project,
implementing, 518-521

BackgroundDownload project,
implementing, 782-784

ColorTilt project,
implementing, 635-642

Cupertino project
implementing, 738-742
updating, 747-753

FieldButtonFun project,
implementing, 224-225

FlowerColorTable project,
implementing, 476-481

FlowerWeb project,
implementing, 274-278

Gestures project,
implementing, 607-616

ImageHop project,
implementing, 246-252

LetsTab project, implementing,
452-455

Maps, implementing, 717-719

ModalEditor project,
implementing, 362-363

Orientation project,
implementing, 628-630

PopoverEditor project,
implementing, 368-370

ReturnMe project,
implementing, 532-534

Scroller project, implementing,
286-287

SlowCount project,
implementing, 776-778

Survey project, implementing
application logic, 539-543

Swapper project,
implementing, 587-589

Universal project,
implementing, 797-798

UniversalToo project,
implementing, 802

long pressing, 594**long-running background tasks, completing, 773-780****long-running tasks, completing, 759-760****loops, 93-95****M****magnetic compass, 744-753****Mail, 698-700****Manage UI framework (Cocoa Touch), 104****manually controlling modal segues, 333-334****manually displaying popover segues, 346****Map Kit, 104, 724-725, 727****mapping, 702-708**

annotations, 703-705

geocoding, 705-708

Maps, logic, implementing, 717-719**master scene, updating, 484-487****master view controllers, implementing, 492-496****Master-Detail Application template, 500**

creating, 481-499

split view controllers, 471

media items, accessing, Media Player, 654**Media layer (iOS), 105-106**

Media Picker, 651-652

- controller delegate, 652-653
- displaying, 685-686
- preparing, 683-684

Media Player framework, 106, 648-654

- accessing media items, 654
- Media Picker, 651-652
 - controller delegate, 652-653
- Movie Player, 648-650
 - handling playback completion, 650-651
 - implementing, 667-671
- Music Player, 653-654

MediaPlayground project, 662-663

- audio playback
 - controlling, 675-677
 - implementing, 674-675
- audio recording, implementing, 670-674
- Core Image filter, implementing, 681-682
- designing interface, 664-665
- Movie Player, implementing, 667-671
- music libraries, accessing and playing, 683-688
- outlets and actions, creating and connecting, 665-667
- photo library, implementing, 677-681
- setting up, 663-664

memory management, Objective-C, 95-97**memory usage, monitoring, 828****messages, 70, 88-91**

- nested messaging, 89-90

methods, 88-91

- applicationDidEnterForeground, 764
- background-aware application life cycle, 761-762

- calculateDateDifference, 398-400
- chooseImage, 677-678
- class, 70
- controlHardware, 637-638
- convenience, 87
- createFlowerData, 489, 491
- createStory, 224-225
- defining, 78-79
- describeInteger, 815
- doAcceleration, 640-641
- doActionSheet, 313
- doAlert, 305-306
- doAttitude, 639-640
- doRotation, 641-642
- implementing, 81-82
- incrementCount, 444
- instance, 70
- newBFF, 713
- playAudio, 675
- playMusic, 688
- prepareForSegue:sender, 339-340
- recordAudio, 673, 676
- setBackgroundHueValue, 519-520
- setIncrement, 251
- setSpeed, 249-251
- setValuesFromPreferences, 533-534
- showFromRect, 315
- showResults, 541
- storeSurvey, 539-540
- tableView:titleForHeaderInSection Section, 466-467
- toggleAnimation, 248-249
- toggleFlowerDetail, 275
- viewDidLoad, 275, 520-521, 815, 823-824
- viewWillAppear:animated, 445

modal scene switches, programming, 336-339**modal segues, 352-364**

- controlling manually, 333-334
- creating, 358

modal user interfaces, 292**modal views, 323****ModalEditor project**

- actions and outlets, creating and connecting, 360-361
- application logic, implementing, 362-363
- building, 364
- designing interface, 356-358
- modal segues, creating, 358
- setting up, 352-356

Model-View-Controller (MVC). See MVC (Model-View-Controller)**motion**

- acceleration, detecting, 631-643
- accessing data, 624-626
- Core Motion, reading acceleration, rotation, and attitude, 624-626
- hardware, 619-623
 - accelerometers, 620-622
 - gyroscopes, 622-623
- managing updates, 637-638
- methods, preparing, 638-639
- rotation, detecting, 631-643
- tilt, detecting, 631-643

motion manager (Core Motion), 624-626, 643-644

- initializing, 636-637

motion sensing, Nintendo Wii, 619**Movie Player, 648-650**

- handling playback completion, 650-651
- implementing, 667-671

MPMediaItem class, 648**MPMediaItemCollection class, 648****MPMediaPickerController class, 648**

MPMoviePlayerController class, 648

MPMusicPlayerController class, 648-650

multibutton alerts, creating, 306-309

multiple targets, universal applications, 803-805

multiscene storyboards, 322-341
 preparing, 324-330
 scenes
 adding, 324
 configuring segues, 332-333
 creating segues, 330-333
 manually controlling modal segues, 333-334
 naming, 326-327
 passing data between, 339-341
 programming modal scene switches, 336-339
 unwind segues, 334-336
 view controllers, adding supporting subclasses, 326-329

multi-touch events, generating, Simulator, 61

multi-touch gesture recognition, 593-594

Music Player, 653-654

MVC (Model-View-Controller), 21, 167-168, 191-192
 data models, 172
 Single View Application template, 173-190
 creating and connecting outlets and connections, 184-188
 designing interface, 182-185
 implementing application logic, 189-190
 planning properties and connections, 179-182
 setting up project, 174-179
 structured application design, 168-169
 view controllers, 170-172
 views, 169-170
 Xcode implementation, 169-172

N

naming scenes, 326-327

navigating
 documents, 124-125
 Xcode projects, 28-30

navigation bars, 424-425

navigation controllers, 423-428, 434-445, 456-457
 bar button items, 424-425
 storyboards, 425-428

navigation scenes, sharing data between, 428

navigators, Xcode debugger, 826-828

nested messaging, 89-90

New Project command (File menu), 24

newBFF method, 713

Newsstand framework (Core Services layer), 107

NeXTSTEP, 103

Nintendo Wii, motion sensing, 619

NSArray class, 113-114

NSDate class, 116

NSDecimalNumber class, 115-116

NSDictionary class, 114-115

NSLog function, 810-812

NSMutableArray class, 113-114

NSMutableDictionary class, 114-115

NSMutableString class, 113

NSNumber class, 115-116

NSObject class, 111

NSString class, 113

NSURL class, 116-117

O

Object Library (IB), 137-139

Objective-C, 20, 67, 71-72, 97-98
 ARC (Automatic Reference Counting), 96-97
 blocks, 90-91
 case sensitivity, 72
 categories, defining, 82-83
 decision making, 91-95
 expressions, 91-95
 file structure, 72-84
 header files, 73-80
 implementation files, 79-82
 loops, 93-95
 memory management, 95-97
 messages, 88-91
 methods, 88-91
 objects
 allocating, 86-88
 initializing, 86-88
 OOP (object-oriented programming), 67-72
 protocols, creating, 84
 syntax, 72
 typecasting, 88
 variables, declaring, 84-86

Objective-C Phrasebook, 97

object-oriented programming (OOP). See **OOP (object-oriented programming)**

objects, 70, 489

- allocating, 86-88
- identifying, IB (Interface Builder), 163
- initializing, 86-88
- self, 71

OOP (object-oriented programming)

- classes, 69
 - categories, 70
- imperative development, 68
- instance method, 70
- instance variables, 70
- instantiation, 70
- messages, 70
- Objective-C, 67-72
- objects, 70
 - self, 71
- parameters, 70
- properties, 70
- singletons, 69
- subclasses, 69
- superclasses, 69
- variables, 70

OpenGL ES framework (Media layer), 106

OpenStep, 103

orientation

- accessing data, 622-624
- devices, setting, 53
- interfaces, enabling changes, 577
- preventing changes, 642
- registering for updates, 628-629
- rotation, swapping views on, 582-590
- sensing, 626-631
- upside-down, disabling, 584

Orientation project, 626-631

- building, 631
- creating and connecting
 - outlet, 627
- designing interface, 627-628
- implementing application logic, 628-630
- setting up, 627

outlets

- BackgroundColor project, creating and connecting, 516-518
- BestFriend project, creating and connecting, 711-712
- ColorTilt project, creating and connecting, 633-635
- Cupertino project, creating and connecting, 737, 747
- CustomPicker project, creating and connecting, 406-407
- DateCalc project, creating and connecting, 391-392
- FieldButtonFun project, creating and connecting, 218-219
- FlowerWeb project, creating and connecting, 270-272
- Gestures project, creating and connecting, 605-607
- GettingAttention project, creating and connecting, 303-305
- HelloNoun project, creating and connecting, 184-188
- IB (Interface Builder), 156-158
- ImageHop project, creating and connecting, 242-246
- LetsNavigate project, creating and connecting, 442-443
- LetsTab project, creating and connecting, 450-452
- ModalEditor project, creating and connecting, 360-361
- Orientation project, creating and connecting, 627

PopoverEditor project, creating and connecting, 367-368

ReturnMe project, creating and connecting, 524

Scroller project, creating and connecting, 285-286

SlowCount project, creating and connecting, 776

Survey project, creating and connecting, 536-539

Swapper project, creating and connecting, 586

Universal project, creating and connecting, 796-797

UniversalToo project, creating and connecting, 802

P

panning, 594

parameters, 70

parent classes, 69

Pass Kit framework (Core Services layer), 107

passing data between scenes, 339-341

people picker, Address Book, 695-697

pickers, 373, 377-378, 417-418

- custom, 401-416

- date, 378-379, 385-401

- Image Picker, 657-660

- controller delegate, 659-660

- implementing, 677-681

- Media Picker, 651-652

- controller delegate, 652-653

- displaying, 685-686

- preparing, 683-684

- views, 379-385

pinching, 594

- plain text fields, versus attributed, 207
- playAudio method, 675**
- playback**
 - AV Audio Player, handling completion, 656
 - Movie Player, handling completion, 650-651
- playMusic method, 688**
- pointers, 85-86**
- PopoverEditor project**
 - application logic, implementing, 368-370
 - building, 370
 - creating popover segue, 366-367
 - designing interface, 366
 - outlets, creating and connecting, 367-368
 - setting up, 365-366
- popovers, 341-352, 364-370**
 - creating segues, 343-346
 - dismissals, responding to, 346-348
 - displaying manually, 346
 - iPad, 315
 - preparing, 342-343
 - programming displays, 349-352
 - segues, creating, 366-367
- pragma marks, code, Xcode projects, 41-42**
- preferences, implicit, creating, 514-522**
- Preferences and Settings Programming Guide, 543**
- prepareForSegue:sender method, 339-340**
- primitive data types, 85**
- programmatically defined interfaces, 574-583**
- Programming in Objective-C 2.0, Third Edition, 97**
- projects**
 - AllInCode, 577-583
 - building, 582-583
 - enabling orientation changes, 577
 - planning properties and connections, 577
 - programming interface, 577-582
 - BackgroundColor, 544
 - building, 521-522
 - creating and connecting outlets and actions, 516-518
 - designing interface, 515
 - implementing application logic, 518-521
 - setting up, 515
 - BackgroundDownload
 - building, 784
 - designing interface, 781
 - implementing application logic, 782-784
 - setting up, 780-781
 - BestFriend
 - Address Book logic, 712-717
 - building, 723-724
 - creating and connecting outlets and actions, 711-712
 - implementing email logic, 720-721
 - implementing social networking logic, 721-723
 - Maps logic, 712-717
 - setting to white, 723
 - setting up, 708-709, 745
 - ColorTilt, 631-632
 - building, 642-643
 - creating and connecting outlets and actions, 633-635
 - designing interface, 633-634
 - implementing application logic, 635-642
 - setting up, 632-633
 - Cupertino, 735
 - adding background modes, 773
 - building, 742-743, 753
 - creating and connecting outlets, 737, 747
 - designing view, 736-738
 - implementing application logic, 738-742
 - preparing for audio, 768-773
 - setting up, 735-736, 744-745
 - updating application logic, 747-753
 - updating user interface, 745-746
 - CustomPicker, 401-402
 - building, 416
 - creating and connecting outlets and actions, 406-407
 - creating segues, 406
 - designing interface, 404-406
 - implementing custom picker view, 409-416
 - implementing scene segue logic, 407-409
 - setting up, 402-403
 - DateCalc, 386
 - building, 400-401
 - creating and connecting outlets and actions, 391-392
 - creating segue, 389-391
 - designing interface, 388-389

- implementing date calculation logic, 396-400
 - implementing scene segue logic, 393-396
 - setting up, 386-388
- FieldButtonFun, 197
 - building, 225-226
 - creating and connecting outlets and actions, 218-219
 - designing interface, 204-218
 - implementing application logic, 224-225
 - implementing keyboard hiding, 220-224
 - preparing button templates with slicing, 199-204
 - setting up, 198-199
- FlowerColorTable, 471-481
 - application logic, 476-481
 - building, 481
 - connecting delegates and data source outlets, 474-475
 - designing interface, 473-475
- FlowerWeb
 - building, 278-279
 - creating and connecting actions, 271-274
 - creating and connecting outlets, 270-272
 - designing interface, 263-270
 - implementing application logic, 274-278
 - setting up, 263
- Gestures, 596-598
 - adding gesture recognizers to views, 601-605
 - creating and connecting outlets and actions, 605-607
 - designing interface, 599-601
 - implementing, 607-616
 - implementing application logic, 607-616
 - setting up, 598-599
- GettingAttention
 - creating and connecting actions and outlets, 303-305
 - designing interface, 302-303
 - implementing action sheets, 313-316
 - implementing alert views, 305-312
 - setting up, 301-302
- IB (Interface Builder), opening, 154-155
- ImageHop, 233-234
 - building, 252-253
 - creating and connecting outlets and actions, 242-246
 - designing interface, 234-243
 - implementing application logic, 246-252
- LetsNavigate, 445
 - creating and connecting outlets and actions, 442-443
 - creating push segues, 440-441
 - designing interface, 440-442
 - setting up, 436-440
- LetsTab
 - building, 455-456
 - creating and connecting outlets and actions, 450-452
 - designing interface, 449-451
 - implementing application logic, 452-455
 - setting up, 446-448
- Master-Detail Application template, creating, 481-499
- MediaPlayground, 662-663
 - accessing and playing music library, 683-688
 - controlling audio playback, 675-677
 - creating and connecting outlets and actions, 665-667
 - designing interface, 664-665
 - implementing audio playback, 674-675
 - implementing audio recording, 670-674
 - implementing camera, 677-681
 - implementing Core Image filter, 681-682
 - implementing Movie Player, 667-671
 - implementing photo library, 677-681
 - setting up, 663-664
- ModalEditor
 - building, 364
 - creating and connecting actions and outlets, 360-361
 - creating modal segues, 358
 - designing interface, 356-358
 - implementing application logic, 362-363
 - setting up, 352-356
- Orientation
 - building, 631
 - creating and connecting outlet, 627

- designing interface, 627-628
 - implementing application logic, 628-630
 - setting up, 627
 - PopoverEditor
 - building, 370
 - creating and connecting outlets, 367-368
 - creating popover segue, 366-367
 - designing interface, 366
 - implementing application logic, 368-370
 - setting up, 365-366
 - ReturnMe, 544
 - creating and connecting, 524
 - creating settings bundles, 524-531
 - designing interface, 523-525
 - implementing application logic, 532-534
 - setting up, 522-523
 - Scroller
 - building, 287-288
 - creating and connecting outlets and actions, 285-286
 - designing interface, 280-285
 - implementing application logic, 286-287
 - setting up, 279
 - SlowCount
 - building, 780
 - creating and connecting outlets, 776
 - designing interface, 775
 - enabling background task processing, 778-780
 - implementing application logic, 776-778
 - setting up, 774-775
 - Survey, 544
 - creating and connecting outlets and actions, 536-539
 - designing interface, 535-537
 - implementing application logic, 539-543
 - setting up, 535
 - Swapper
 - building, 590
 - creating and connecting outlets, 586
 - designing interface, 584-585
 - implementing application logic, 587-589
 - setting up, 582-584
 - Universal
 - building, 799
 - creating and connecting outlets, 796-797
 - designing interface, 796-797
 - implementing application logic, 797-798
 - setting up, 796
 - UniversalToo, 803
 - creating and connecting outlets, 802
 - designing interface, 801-802
 - implementing application logic, 802
 - setting up, 799-801
 - Xcode
 - adding new code files to, 28
 - adding resources to, 28-32
 - assistant editor, 42-44
 - choosing type, 24-27
 - code completion, 38-40
 - creating, 24-28
 - managing properties, 51
 - relationships
 - navigating, 28-30
 - navigating code, 35-37
 - pragma marks, 41-42
 - removing files and resources, 32
 - searching code, 40-41
 - properties, 70**
 - AllInCode project, planning, 577
 - detailItem, 496-497
 - versus instance variables, 439
 - respecting point of, 771
 - Single View Application template (MVC), planning, 179-182
 - protocols, 70, 75**
 - creating, 84
 - pseudo preferences, 506**
 - push segues, creating, 440-441**
- Q**
- Quartz Core framework (Media layer), 106**
 - Quick Help (Xcode), 125-129**
 - Quick Inspector (IB), editing connections, 161-162**
 - Quick Look framework (Core Services layer), 107**
- R**
- range attributes**
 - sliders, setting, 237-239
 - steppers, setting, 240-241
 - recordAudio method, 673, 676**
 - relationships**
 - tab bar, creating, 448-449
 - view controllers, 323

resizable interfaces, designing, 550-552

resources, Xcode projects

adding to, 28-32

removing, 32

responsive interfaces, 547-548

building, Auto Layout, 552-575

designing rotatable and resizeable, 550-552

rotation, enabling, 548-549

Retina display, 2

retina image assets, 35

ReturnMe project, 544

application logic, implementing, 532-534

designing interface, 523-525

outlets, creating and connecting, 524

setting up, 522-523

settings bundles, creating, 524-531

rich media, 647-648, 689-691

AV Foundation framework, 655-657

Core Image framework, 660-662

Image Picker, 657-660

Media Player framework, 648-654

rotatable interfaces, designing, 550-552

rotation, 594

detecting, 631-643

interfaces, 590

enabling, 548-549

swapping views on, 582-590

reacting to, 641-642

reading, Core Motion, 624-626

swapping views on, 582-590

running first app, 17-18

S

scenes, 323

exits, 323

modal, programming switches, 336-339

multiscene storyboards, adding to, 324

naming, 326-327

navigation, sharing data between, 428

passing data between, 339-341

segues, 323

configuring, 332-333

controlling modal manually, 333-334

creating, 330-333, 406

creating push, 440-441

implementing logic, 393-396, 407-409

modal, 352-364

unwind, 323, 334-336

tab bar, sharing data between, 430-434

scroll views, 288-289

Scroller project

actions and outlets, creating and connecting, 285-286

application logic, implementing, 286-287

building, 287-288

designing interface, 280-285

setting up, 279

scrolling options, interfaces, setting, 212

scrolling views, 261, 279-288

SDK (software development kit), 6, 321

search navigator (Xcode), 40-41

searching documentation, 121-122

Security framework (Core OS layer), 108

segmented controls, 258-259, 288-289

segues, 370-371

creating, 389-391, 406

modal, 352-364

creating, 358

popovers, creating, 343-346, 366-367

push, creating, 440-441

scenes

configuring, 332-333

controlling modal manually, 333-334

creating, 330-333

implementing logic, 393-396

unwind, 323, 334-336

setting delegates, 408

selection handles (IB), 142-144

services

Address Book, 693-697, 724-725

email, 698-700

mapping, 702-708

posting to social networking sites, 700-702

setBackgroundHueValue method, 519-520

setIncrement method, 251

setSpeed method, 249-251

settings bundles

creating, 524-531

data storage, 508-510

setValuesFromPreferences method, 533-534

Shake Gesture command (Simulator), 63

shaking devices, 594

showFromRect method, 315

showResults method, 541

- Simulate Hardware Keyboard command (Simulator), 63**
 - Simulate Memory Warning command (Simulator), 63**
 - Simulator (iOS), 58-63**
 - generating multi-touch events, 61
 - Hardware menu commands, 62-64
 - launching applications, 59-60
 - rotating device, 62
 - Single View Application template, table views, 471-481**
 - Single View Application template (MVC), 173-190**
 - creating and connecting outlets and connections, 184-188
 - designing interface, 182-185
 - implementing application logic, 189-190
 - planning properties and connections, 179-182
 - setting up project, 174-179
 - singletons, 69**
 - Size Inspector (IB), 143-145**
 - slicing, preparing button templates, 199-204**
 - sliders, 230, 252-253**
 - applications, adding, 236-238
 - setting range attributes, 237-239
 - SlowCount project**
 - background task processing, enabling, 778-780
 - building, 780
 - creating and connecting outlets, 776
 - designing interface, 775
 - implementing application logic, 776-778
 - setting up, 774-775
 - Social framework (Core Services layer), 108**
 - social networking platforms, posting to, 700-702**
 - software development kit (SDK), 6, 321**
 - split view controllers, 468-469, 499**
 - implementing, 469-471
 - Master-Detail Application template, 471
 - state preservation, 506**
 - statements**
 - if-then-else, 91-93
 - switch, 91-93
 - status bar, setting display, 54-58**
 - steppers, 230-231, 252-253**
 - applications, adding to, 239-240
 - range attributes, setting, 240-241
 - stepping through code, 820-822**
 - Store Kit framework (Core Services layer), 108**
 - storeSurvey method, 539-540**
 - storyboards, 323, 370-371, 421**
 - IB (Interface Builder), 133-137
 - multiscene, 322-341
 - adding scenes, 324
 - configuring segues, 332-333
 - creating segues, 330-333
 - manually controlling modal segues, 333-334
 - naming scenes, 326-327
 - passing data between scenes, 339-341
 - preparing, 324-330
 - programming modal scene switches, 336-339
 - unwind segues, 334-336
 - navigation controllers, 425-428
 - tab bar controllers, 430-434
 - view controllers, adding supporting subclasses, 326-329
- strings, date format, 397**
- structured application design, MVC (Model-View-Controller), 168-169**
- styled buttons, adding, 212**
- subclasses, 69**
- superclasses, 69**
- Survey project, 544**
 - application logic, implementing, 539-543
 - designing interface, 535-537
 - outlets and actions, creating and connecting, 536-539
 - setting up, 535
- suspension, background, 758**
 - handling, 764-765
- Swapper project**
 - application logic, implementing, 587-589
 - building, 590
 - creating and connecting outlets, 586
 - interface, designing, 584-585
 - setting up, 582-584
- swiping, 594**
- switch statements, 91-93**
- switches, 258, 288-289**
 - adding, 266-267
 - modal scene, programming, 336-339
 - setting default state, 267
- symbol navigator (Xcode), 37**
- syntax, Objective-C expressions, 91-95**
- System Configuration framework (Core Services layer), 108**
- System framework (Core OS layer), 109**
- system settings, implementing, 522-534**
- System Sound Services, 298-300, 647**

T

tab bar controllers, 429-434, 445-457

- creating relationships, 448-449
- storyboards, 430-434

tab bar scenes, sharing data between, 430-434

tabbed editing, Xcode, 44

table views, 459, 471-481, 499

- adding, 461-468

tables, 459-460

- appearance, 460-461
- attributes, setting, 462
- cells, 460
 - creating, 493-494
 - disabling editing, 495
 - setting prototype attributes, 463-464

tableView:titleForHeaderInSection method, 466-467

tap gesture recognizer, 594-595

tapping, 594

targets, multiple, universal applications, 803-805

tasks

- background, 785-786
 - completing long-running, 773-780
 - enabling processing, 778-780
- long-running, completion, 759-760

task-specific background processing, 759, 768-773

technology layers (iOS), 103

- Cocoa Touch, 104-105
- Core OS, 108-109
- Core Services, 106-108
- Media, 105-106

templates

- button, preparing with slicing, 199-204
 - Master-Detail Application, 500
 - creating, 481-499
- universal applications, 791-792

text fields, 196-197, 226-227

- adding, 204-205
- attributed versus plain, 207-209
- attributes
 - editing, 205-207
 - traits, 207

text views, 196, 226-227

- adding, 209-210
- attributes, editing, 210-212

tilt, detecting, 631-643

Toggle In-Call Status Bar command (Simulator), 63

toggleAnimation method, 248-249

toggleFlowerDetail method, 275

toolbars, 373-377, 417-418

- bar button items, 375-376
- attributes, 376-377

touch, multi-touch gesture recognition, 593-594

tracing

- application life cycle, 109-110
- applications, 809, 828-830
 - NSLog function, 810

TV Out command (Simulator), 63

typecasting, 88

U

UI Framework, Address Book, 694-695

UIActionSheet class, 295-298

UIAlertView class, 292-295

UIApplication class, 111

UIBarButtonItem class, 375

UIButton class, 117, 195, 199-204, 231

UIControl class, 112

UIDatePicker class, 119, 377-378, 417

UIDevice class, requesting orientation notifications through, 623-624

UIGestureRecognizer class, 617

UIImagePickerController class, 648

UIImagePickerControllerDelegate protocol, 659-660

UIImageView class, 231, 681

UIKit framework (Cocoa Touch), 104

UILabel class, 117, 196, 231

UILongPressGestureRecognizer class, 594, 617

UIPanGestureRecognizer class, 594, 617

UIPicker class, 119

UIPickerView class, 377, 417

UIPinchGestureRecognizer class, 594

UIPopoverController class, 120, 347-348, 373

UIResponder class, 112

UIRotationGestureRecognizer class, 594

UIScrollView class, 261, 279, 288

UIScrollViewDelegate class, 288

UISegmentedControl class, 118, 259, 288

UISlider class, 118, 230-231, 252

UIStepper, 252

UIStepper class, 118, 230-231

UISwipeGestureRecognizer class, 594

UISwitch class, 118, 258, 288

UITapGestureRecognizer class, 594
UITextField class, 119, 196
UITextFieldDelegate class, 226
UITextView class, 119, 196
UITextViewDelegate class, 226
UIView class, 112, 259
UIViewController class, 113
UIWebView class, 288
UIWindow class, 112
universal applications, 789, 805
 creating, 794-803
 development, 789-794
 multiple targets, 804-805
 setup changes, 791-794
 templates, 791-792
Universal project
 application logic, implementing, 797-798
 building, 799
 designing interface, 796-797
 outlets, creating and connecting, 796-797
 setting up, 796
UniversalToo project
 application logic, implementing, 802
 building, 803
 designing interface, 801-802
 outlets, creating and connecting, 802
 setting up, 799-801
unwind segues, scenes, 323, 334-336
updates
 motion, managing, 637-638
 orientation, registering for, 628-629
upside-down orientation, disabling, 584
user alerts, 291-319
 multibutton, creating, 306-309
 System Sound Services, 298-300, 316-318

 views, 292-295
 action sheets, 295-298
 implementing, 305-312
user defaults, data storage, 507-508
user input, 195, 257-258
 buttons, 195-196
 implementation, 197
 labels, 196
 text fields, 196-197
 text views, 196
user interfaces. See interfaces
user locations
 accuracy and update filter, 732
 adding constants, 736
 compass, 744-753
 handling errors, 730-731
 obtaining, 727-732

V

variable list, Xcode debugger, accessing, 823-824
variables, 70
 declaring, 84-86
 instance, 70
version detection, iOS, 583
vibrations, alerts, 317
view controllers, 323, 421-422
 multiscene development, 422
 MVC (Model-View-Controller), 170-172
 relationships, 323
 storyboards, adding supporting subclasses, 326-329
ViewController class, 436, 446
viewDidLoad method, 275, 520-521, 815, 823-824

views, 323
 alert, 292-295
 button press response, 309-310
 fields, 310-312
 implementing, 305-312
 Cupertino project, designing, 736-738
 gesture recognizers, adding to, 601-605
 image, 231
 modal, 323
 MVC (Model-View-Controller), 169-170
 objects, adding to, 140-141
 pickers, 379-385
 rotation, swapping on, 582-590
 scrolling, 261, 279-288
 split view controllers, 468-469
 implementing, 469-471
 Master-Detail Application template, 471
 table views, 459
 adding, 461-468
 web, 259-261
 adding, 267
viewWillAppear:animated method, 445

W- X-Y-Z

web views, 259-261, 288-289
 adding, 267
Xcode, 21, 23-58, 63-64
 animation resources, adding, 233
 asset catalogs, 32-35
 assistant editor, 42-44

- debugger, 813-828
 - accessing variable list, 823-824
 - navigators, 826-828
 - setting breakpoints, 816-818
 - stepping through code, 820-822
- devices, setting orientations, 53
- documentation system, 121-125
- installing, 9-12
- interface, 27-28
- iOS frameworks, 120-129
- iOS Simulator, 58-63
 - generating multi-touch events, 61
 - launching applications, 59-60
- managing snapshots, 45
- MVC (Model-View-Controller), implementation, 169-172
- projects
 - adding new code files to, 28
 - adding resources to, 28-32
 - building applications, 46-51
 - choosing type, 24-27
 - code completion, 38-40
 - creating, 24-28
 - managing properties, 51
 - navigating, 28-30
 - navigating code, 35-37
 - pragma marks, 41-42
 - removing files and resources, 32
 - searching code, 40-41
 - setting status bar display, 54-58
 - Quick Help, 125-129
 - Single View Application option, 174
 - tabbed editing, 44
 - writing code with, 188

Xcode 4 Unleashed, 97