John Ray

In Full Color

Figures and code appear as they do in Xcode 4.5+

Covers iOS 6, Xcode 4.5+, iPhone, iPad, and More!

Additional files and updates available online

Sams Teach Yourself iOS® 6 Application Development in 24 Hours

SAMS

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

John Ray

Sams Teach Yourself

iOS® 6 Application Development in 24 Hours

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

Sams Teach Yourself iOS® 6 Application Development in 24 Hours

Copyright © 2013 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: 978-0-672-33443-6

ISBN-10: 0-672-33443-7

Library of Congress Cataloging-in-Publication Data is on file and available upon request.

Printed in the United States of America

First Printing January 2013

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.

Bulk Sales

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

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

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

International Sales
international@pearsoned.com

Editor-in-Chief
Greg Wiegand

Acquisitions Editor
Laura Norman

Development Editor
Keith Cline

Managing Editor
Kristy Hart

Senior Project Editor
Lori Lyons

Indexer
Cheryl Landes

Proofreader
Sarah Kearns

Technical Editor
Anne Groves

Editorial Assistant
Cindy Teeters

Interior Designer
Gary Adair

Cover Designer
Anne Jones

Compositor
Gloria Schurick

Contents at a Glance

Introduction 1

HOUR 1 Preparing Your System and iDevice for Development 3

2 Introduction to Xcode and the iOS Simulator 27

3 Discovering Objective-C: The Language of Apple Platforms 65

4 Inside Cocoa Touch 97

5 Exploring Interface Builder 125

6 Model-View-Controller Application Design 161

7 Working with Text, Keyboards, and Buttons 189

8 Handling Images, Animation, Sliders, and Steppers 221

9 Using Advanced Interface Objects and Views 247

10 Getting the User's Attention 281

11 Implementing Multiple Scenes and Popovers 311

12 Making Choices with Toolbars and Pickers 363

13 Advanced Storyboards Using Navigation and Tab Bar Controllers 409

14 Navigating Information Using Table Views and Split View Controllers 447

15 Reading and Writing Application Data 489

16 Building Responsive User Interfaces 533

17 Using Advanced Touches and Gestures 571

18 Sensing Orientation and Motion 599

19 Working with Rich Media 625

20 Interacting with Other Applications 673

21 Implementing Location Services 707

22 Building Background-Aware Applications 737

23 Building Universal Applications 763

24 Application Tracing and Debugging 781

Index 803

Table of Contents

Introduction 1

Who Can Become an iOS Developer? 1

Who Should Use This Book? 2

What Is (and Isn't) in This Book? 2

HOUR 1: Preparing Your System and iDevice for Development 3

Welcome to the iOS Platform 3

Becoming an iOS Developer 8

Creating and Installing a Development Provisioning Profile 15

Running Your First iOS App 19

Developer Technology Overview 22

Further Exploration 23

Summary 24

Q&A 24

Workshop 25

Activities 25

HOUR 2: Introduction to Xcode and the iOS Simulator 27

Using Xcode 27

Using the iOS Simulator 56

Further Exploration 62

Summary 62

Q&A 63

Workshop 63

Activities 64

HOUR 3: Discovering Objective-C: The Language of Apple Platforms 65

Object-Oriented Programming and Objective-C 65

Exploring the Objective-C File Structure 70

Objective-C Programming Basics 81

Memory Management and Automatic Reference Counting 91

Further Exploration 93

Summary 94
Q&A 94
Workshop 95
Activities 95

HOUR 4: Inside Cocoa Touch 97

What Is Cocoa Touch? 97
Exploring the iOS Technology Layers 99
Tracing the iOS Application Life Cycle 105
Cocoa Fundamentals 106
Exploring the iOS Frameworks with Xcode 116
Further Exploration 122
Summary 122
Q&A 122
Workshop 123
Activities 123

HOUR 5: Exploring Interface Builder 125

Understanding Interface Builder 125
Creating User Interfaces 132
Customizing the Interface Appearance 142
Connecting to Code 146
Further Exploration 156
Summary 157
Q&A 157
Workshop 158
Activities 159

HOUR 6: Model-View-Controller Application Design 161

Understanding the Model-View-Controller Design Pattern 161
How Xcode Implements MVC 163
Using the Single View Application Template 167
Further Exploration 184
Summary 185
Q&A 186
Workshop 186
Activities 187

HOUR 7: Working with Text, Keyboards, and Buttons 189

Basic User Input and Output 189
Using Text Fields, Text Views, and Buttons 191
Further Exploration 218
Summary 218
Q&A 219
Workshop 219
Activities 219

HOUR 8: HANDLING IMAGES, ANIMATION, SLIDERS, AND STEPPERS 221

User Input and Output 221
Creating and Managing Image Animations, Sliders, and Steppers 223
Further Exploration 244
Summary 245
Q&A 245
Workshop 246
Activities 246

HOUR 9: Using Advanced Interface Objects and Views 247

User Input and Output (Continued) 247
Using Switches, Segmented Controls, and Web Views 252
Using Scrolling Views 268
Further Exploration 277
Summary 277
Q&A 278
Workshop 278
Activities 279

HOUR 10: Getting the User's Attention 281

Alerting the User 281
Exploring User Alert Methods 290
Further Exploration 308
Summary 309
Q&A 309
Workshop 310

HOUR 11: Implementing Multiple Scenes and Popovers 311
Introducing Multiscene Storyboards ... 311
Understanding the iPad Popover ... 332
Using a Modal Segue ... 342
Using a Popover ... 355
Further Exploration ... 360
Summary ... 361
Q&A ... 361
Workshop ... 362
Activities ... 362

HOUR 12: Making Choices with Toolbars and Pickers 363
Understanding the Role of Toolbars ... 363
Exploring Pickers ... 367
Using the Date Picker ... 375
Using a Custom Picker ... 391
Further Exploration ... 406
Summary ... 407
Q&A ... 407
Workshop ... 408
Activities ... 408

HOUR 13: Advanced Storyboards Using Navigation and Tab Bar Controllers 409
Advanced View Controllers ... 409
Exploring Navigation Controllers ... 411
Understanding Tab Bar Controllers ... 417
Using a Navigation Controller ... 422
Using a Tab Bar Controller ... 433
Further Exploration ... 443
Summary ... 444
Q&A ... 445
Activities ... 446

HOUR 14: Navigating Information Using Table Views and Split View Controllers 447
Understanding Tables ... 447
Exploring the Split View Controller (iPad Only) ... 456

A Simple Table View Application 459
Creating a Master-Detail Application 469
Further Exploration 487
Summary 487
Q&A 488
Workshop 488
Activities 488

HOUR 15: Reading and Writing Application Data **489**
iOS Applications and Data Storage 489
Data Storage Approaches 492
Creating Implicit Preferences 500
Implementing System Settings 508
Implementing File System Storage 520
Further Exploration 528
Summary 529
Q&A 530
Activities 530

HOUR 16: Building Responsive User Interfaces **533**
Responsive Interfaces 533
Using Auto Layout 538
Programmatically Defined Interfaces 555
Swapping Views on Rotation 562
Further Exploration 569
Summary 569
Q&A 569
Workshop 570
Activities 570

HOUR 17: Using Advanced Touches and Gestures **571**
Multitouch Gesture Recognition 571
Adding Gesture Recognizers 572
Using Gesture Recognizers 574
Further Exploration 595
Summary 596
Q&A 596

Workshop ... 596
Activities ... 597

HOUR 18: Sensing Orientation and Motion 599
Understanding Motion Hardware ... 599
Accessing Orientation and Motion Data ... 602
Sensing Orientation ... 606
Detecting Tilt and Rotation ... 611
Further Exploration ... 622
Summary ... 623
Q&A ... 623
Workshop ... 624
Activities ... 624

HOUR 19: Working with Rich Media 625
Exploring Rich Media ... 625
The Media Playground Application ... 640
Further Exploration ... 668
Summary ... 669
Q&A ... 670
Workshop ... 670

HOUR 20: Interacting with Other Applications 673
Extending Application Integration ... 673
Using the Address Book, Email, Social Networking, and Maps ... 686
Further Exploration ... 703
Summary ... 704
Q&A ... 704
Workshop ... 704
Activities ... 705

HOUR 21: Implementing Location Services 707
Understanding Core Location ... 707
Creating a Location-Aware Application ... 714
Using the Magnetic Compass ... 724
Further Exploration ... 733
Summary ... 734
Q&A ... 734

Workshop....735
Activities....735

HOUR 22: Building Background-Aware Applications **737**

Understanding iOS Backgrounding....737
Disabling Backgrounding....742
Handling Background Suspension....743
Implementing Local Notifications....745
Using Task-Specific Background Processing....748
Completing a Long-Running Background Task....755
Further Exploration....760
Summary....760
Q&A....761
Workshop....761
Activities....762

HOUR 23: Building Universal Applications **763**

Universal Application Development....763
Creating a Universal Application (Take 1)....767
Creating a Universal Application (Take 2)....772
Using Multiple Targets....776
Further Exploration....778
Summary....779
Q&A....779
Workshop....780
Activities....780

HOUR 24: Application Tracing and Debugging **781**

Instant Feedback with `NSLog`....781
Using the Xcode Debugger....784
Further Exploration....799
Summary....800
Q&A....800
Workshop....800
Activities....801

Index **803**

About the Author

John Ray is currently serving as the Interim Director of the Office of Research Information Systems for 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 Mountain Lion MacBook*, and *Teach Yourself iOS 5 Development in 24 Hours*. As a Macintosh user since 1984, he strives to ensure that each project presents the Macintosh with the equality and depth it deserves. Even technical titles such as *Using TCP/IP* contain extensive information about the Macintosh and its applications and have garnered numerous positive reviews for their straightforward approach and accessibility to beginner and intermediate users.

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

Dedication

This edition of iOS 6 is dedicated to the memory of iOS 5 and the upcoming birth of iOS 7.

Acknowledgments

Thank you to the group at Sams Publishing—Laura Norman, Keith Cline, Anne Groves—for slowly nudging me toward the finish line. As I read and re-read these chapters, I'm amazed at how you can see the intention behind the seemingly random words that I type. The team has performed pure magic in transforming the files I've submitted into clean, well-styled, content.

As always, thanks to everyone who puts up with me over the months of writing. I'm grateful for your support.

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.

This page intentionally left blank

Introduction

When you pick up an iOS device and use it, you feel connected. Whether it be an iPad, 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.

When creating iOS and the Xcode tools used to develop for it, 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 6 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 6 and later on Xcode 4.5 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, 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.

HOUR 16
Building Responsive User Interfaces

What You'll Learn in This Hour:

- ▶ How to make an application "responsive"
- ▶ Using Auto Layout to enable automatic resizing and rotation
- ▶ Creating UI elements programmatically for the ultimate control
- ▶ How to swap views for landscape and portrait viewing

You can use almost every iOS interface widget available, create multiple views and view controllers, add sounds and alerts, write files, and even manage application preferences, but until now, your applications have been missing a very important feature: responsive interfaces. The ability to create interfaces that display correctly regardless of your iDevice's screen size or orientation is one of the key features that users expect in an application—and the key goal of responsive interface design.

This hour's lesson explores the Xcode Auto Layout system and three different ways of adding rotatable and resizable interfaces to your apps. As you read this hour's lesson, an important fact to keep in mind is that there are *many* different ways to solve the problem of responsive interface design. What works best for you may be entirely different than what I describe here.

Responsive Interfaces

Years ago, when I had my first Windows Mobile smartphone, I longed for an easy way to look at web content in landscape mode. There was a method for triggering a landscape view, but it was glitchy and cumbersome to use. The iPhone introduced the first consumer phone with on-the-fly interface rotation that feels natural and doesn't get in the way of what you're trying to do.

The iPhone 5 shakes things up by introducing the first 4" iPhone screen with a different aspect ration and resolution than the traditional 3.5" models. Does this mean that you need to worry about iPhone 3.5" displays, 4" displays, iPad displays, and portrait and landscape orientations for each? Nope! The iOS Auto Layout system makes it easy to adapt to different resolution displays and change your interface layout as your device resolution changes.

TIP

Using Auto Layout to adapt between 3.5" and 4" screens is no different from adapting to a device that rotates between portrait and landscape. The resolution changes on-the-fly in the case of rotation, but you'll use the same tools to compensate. The only difference is that you can control whether or not your application handles different orientations—you can't control whether or not your users buy a 4" iPhone screen.

Although you should definitely plan to accommodate all sizes of the iPhone screen in your iPhone apps, the decision to handle rotation is entirely up to you. Consider how the user will be interfacing with the app. Does it make sense to force a portrait-only view? Should the view rotate to accommodate any of the possible orientations the phone may assume? The more flexibility you give users to adapt to their own preferred working style, the happier they'll be. Best of all, enabling rotation is a simple process.

TIP

Apple's user interface guidelines for the iPad strongly encourage the support of any orientation/rotation: portrait, left landscape, right landscape, and upside-down.

Enabling Interface Rotation

The projects that you've built to-date have had supported limited interface rotation because of a single line of code in a single method in your view controller. This is added by default when you use one of Apple's iOS templates.

When an iOS device wants to check to see whether it should rotate your interface, it sends the `supportedInterfaceOrientations` message to your view controller. The implementation of `supportedInterfaceOrientations` just returns a value that describes the supported orientation. To cover more than one orientation, you just return the list of constants you want to use, separated by the | character.

There are seven screen orientation support constants, as listed here.

Orientation	Orientation Support Constant
Portrait	`UIInterfaceOrientationMaskPortrait`
Portrait upside-down	`UIInterfaceOrientationMaskPortraitUpsideDown`
Landscape left	`UIInterfaceOrientationMaskLandscapeLeft`

Orientation	Orientation Support Constant
Landscape right	`UIInterfaceOrientationMaskLandscapeRight`
Any Landscape	`UIInterfaceOrientationMaskLandscape`
Anything but upside-down	`UIInterfaceOrientationMaskAllButUpsideDown`
Anything	`UIInterfaceOrientationMaskAll`

For example, to allow your interface to rotate to either the portrait or landscape left orientations, you implement `supportedInterfaceOrientations` in your view controller with the code in Listing 16.1.

LISTING 16.1 This Method Activates Interface Rotation

```
- (NSUInteger)supportedInterfaceOrientations {
    return UIInterfaceOrientationMaskPortrait |
                      UIInterfaceOrientationMaskLandscapeLeft;
}
```

The `return` statement handles everything. It returns the combination of the two constants that describe landscape left and portrait orientations.

NOTE

In addition to adding this method to your view controllers, you must also choose the orientations supported by your app by clicking the Supported Interface Orientation icons in the application summary. This is described in the section “Setting Supported Device Orientations” in Hour 2, “Introduction to Xcode and the iOS Simulator.”

At this point, take a few minutes and go back to some of the earlier hours and modify this method in your view controller code to allow for different orientations. Test the applications in the iOS simulator or on your device. For iPhone apps, try switching between 3.5" and 4" displays.

Some of the applications will probably look just fine, but you’ll notice that others, well, don’t quite “work” in the different screen orientations and sizes, as shown in Figure 16.1 (ImageHop, from Hour 8, “Handling Images, Animation, Sliders, and Steppers”).

Everything we’ve been building has defaulted to a portrait design and, for the iPhone, a 4" screen, so how can we create interfaces that look good regardless of the orientation and size of the screen? We obviously need to make some tweaks.

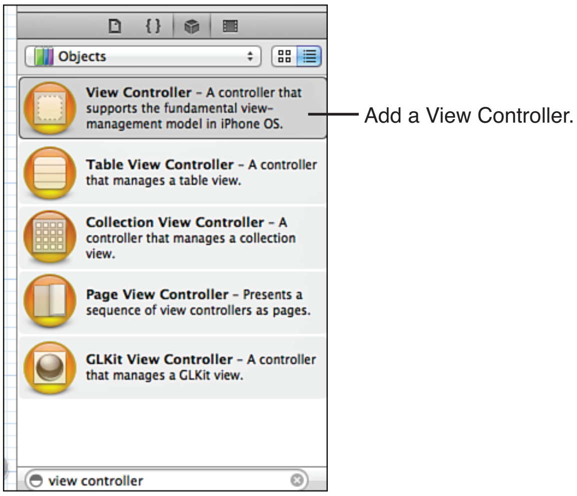

Objects
View Controller - A controller that supports the fundamental view-management model in iPhone OS.
Table View Controller - A controller that manages a table view.
Collection View Controller - A controller that manages a collection view.
Page View Controller - Presents a sequence of view controllers as pages.
GLKit View Controller - A controller that manages a GLKit view.
view controller
Add a View Controller.

FIGURE 16.1
Enabling an orientation doesn't mean it will look good.

Designing Rotatable and Resizable Interfaces

In the remainder of this hour, we explore several different techniques for building interfaces that rotate and resize themselves appropriately depending on the device, or when the user changes the device's screen orientation. Before we get started, let's quickly review the different approaches and when you'd want to use them.

Auto Layout

The Xcode Interface Builder (IB) editor provides tools for describing how your interface should react when it is rotated or when the size of the screen can be variable. It is possible to define a single view in IB that positions and sizes itself appropriately—no matter what the situation—without writing a single line of code. This bit of Apple magic is called *Auto Layout*. It will be your best friend and worst enemy in creating responsive interfaces.

Using Auto Layout should be the starting point for all interfaces. If you can successfully define portrait and landscape modes in a single view in the IB editor, your work is done.

Unfortunately, Auto Layout doesn't work well when there are many irregularly positioned interface elements. A single row of buttons? No problem. Half a dozen fields, switches, and images all mixed together? Probably not going to work.

Programming Interfaces

As you've learned, each UI element is defined by a rectangular area on the screen: its `frame` property. If we programmatically define an application interface, we can make it into anything we want. Although this might sound challenging, it's actually very simple and very effective for creating interfaces that vary dramatically between different sizes and orientations.

For example, to detect when the interface rotates, we can implement the method `didRotate-FromInterfaceOrientation` in our view controller. The view controller property `interface Orientation` will tell us the current orientation of the interface, by containing one of these four constants:

Orientation	**Current Orientation Constant**
Portrait	`UIInterfaceOrientationPortrait`
Portrait upside-down	`UIInterfaceOrientationPortraitUpsideDown`
Landscape left	`UIInterfaceOrientationLandscapeLeft`
Landscape right	`UIInterfaceOrientationLandscapeRight`

The drawback to doing everything in code is that you are doing *everything* in code. It becomes very difficult to prototype interface changes and visualize what your final interface will look like. In addition, it is harder to adapt to changes in the future. It's best, in my opinion, to use a combination of Auto Layout with hand-coded interfaces to provide the best flexibility and speed of development.

Swapping Views

A more dramatic approach to changing your view to accommodate different screen orientations is to use entirely different views for landscape and portrait layouts. When the user rotates the device, the current view is replaced by another view that is laid out properly for the orientation.

This means that you can define two views in a single scene that look exactly the way you want, but it also means that you must keep track of separate `IBOutlets` for each view. Although it is certainly possible for elements in the views to invoke the same `IBActions`, they cannot share the same outlets, so you'll potentially need to keep track of twice as many UI widgets within a single view controller.

NOTE

To know when to change frames or swap views, you will be implementing the method `didRotate-FromInterfaceOrientation` in your view controller. This message is sent to a view controller when the interface has finished changing orientation. If you want to react *prior* to the orientation change taking place, you could implement the `willRotateToInterfaceOrientation:duration` method.

TIP

Apple has implemented a screen-locking function so that users can lock the screen orientation without it changing if the device rotates. (This can be useful for reading while lying on your side.) When the lock is enabled, your application will not receive notifications about a change in orientation. In other words, to support the orientation lock, you don't need to do a thing.

Using Auto Layout

As already mentioned, Auto Layout enables us to build responsive interfaces without writing a line of code. Because of that, our first tutorial is a series of examples that you can follow along with, rather than a project that we build. I've included several sample storyboard files in this hour's Projects folder, but you should definitely get yourself into IB and attempt the examples on your own.

NOTE

You'll be making changes to the Empty.storyboard file in this example. I recommend copying this file and making changes to your copies. This will save you the time of having to create a new storyboard file (or deleing the contents of the current file) each time you want to try something new.

In addition, the examples in this section all target the iPhone. Why? Because the iPhone screen is the perfect place for practice. It can change between both portrait and landscape orientations *and* resize from 3.5" to 4" displays.

The Language and Tools of Auto Layout

When I see something with *Auto* in the name, I expect that it is going to do things for me: automatically. Auto Layout lives up to this, but not without a reasonably long "getting to know you" phase. To use Auto Layout effectively, you must understand the terminology and know where to look for the tools. We spend the next few minutes getting into details, and then walk through a few examples that should get you started on the way to mastery. (Perhaps of archery or speed skating. Auto Layout if you're lucky.)

Constraint Basics

Auto Layout works by building a series of *constraints* for your onscreen objects. Constraints define placement of objects, distances between objects, and how flexible these relationships are. A vertical constraint for a button, for example, might be along the lines of "keep X many points of space between this object and the top of the screen" or "keep object Y vertically centered in the view." Unfortunately, the language used for positioning can be a bit confusing; you may not be familiar with these terms:

- **Leading:** The “front” or left side of an object.
- **Trailing:** The “back” or right side of an object.
- **Superview:** The view that contains an object. In our examples, this is the “view” that makes up the scene itself.

NOTE

Xcode *must* always have enough constraints defined for an object that it can determine its horizontal and vertical position at any point in time (regardless of the screen orientation/size). This means that if you don’t add constraints of your own (called user constraints and represented by a blue icon), Xcode will add system constraints automatically (represented by a purple icon).

For example, try adding a button to the window in the Empty.storyboard file; make sure is located toward the top-left side of the view and that you’ve resized the button slightly. Notice that parallel to the button in the document outline hierarchy a new Constraints entry shows up, as well as a Constraints item within the button object itself, as shown in Figure 16.2.

Navigating the Constraints Objects

Within the view’s Constraints object are two constraints: horizontal space and vertical space constraint. The horizontal constraint states that the left side of the button will be a certain number of points from the left edge of the view. This is known as the *leading space* (versus the space between the right side and the edge of the view, which is called the *trailing space*). The vertical constraint is the distance from the top of the view to the top of the button.

What constraints are added depend on where the object is in relation to its containing view. Xcode does its best to choose what constraints make the most sense. If you position a button near the bottom of the view, for example, it will add a constraint for the distance between the bottom of the button and the bottom of the view.

Constraints, however, are more than just entries that tie an object to the view it is within. They can be flexible, ensuring that an object maintains *at least* or *at most* a certain distance from another object, or even that two objects, when resized, maintain the same distance between one another.

TIP

Alignment of objects is also a constraint, and is managed using the alignment tools you’ve already learned about. In other words, when you set vertical or horizontal alignments, you’re actually configuring constraints that will be maintained regardless of the device’s screen size or orientation!

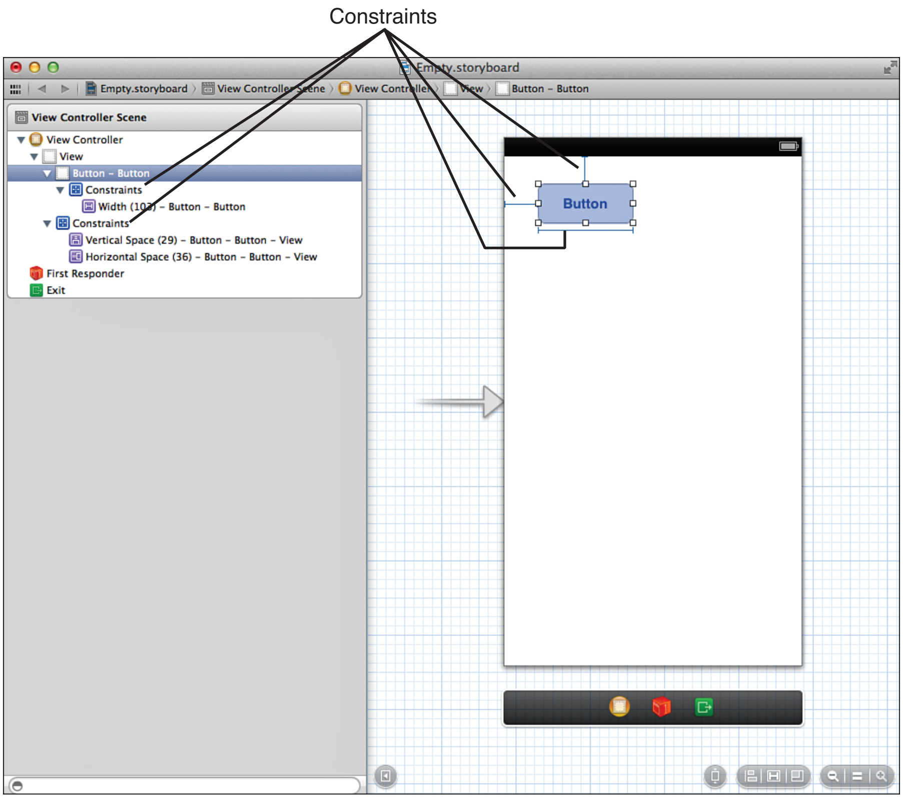


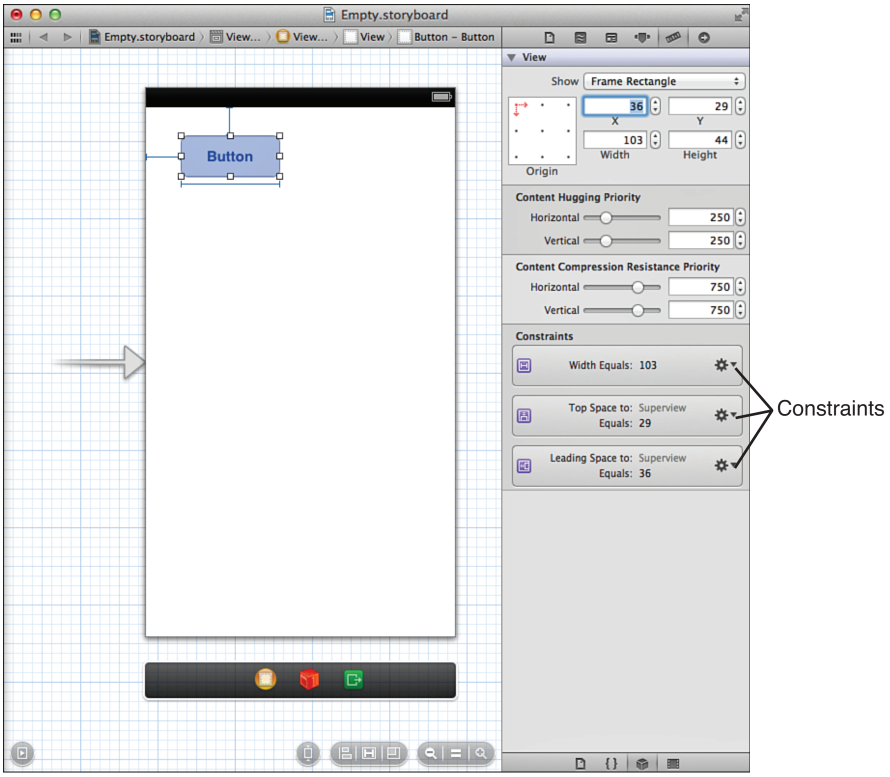
FIGURE 16.2
The Constraints object represents the positioning relationships within a view, or Constraints on the UI elements themselves.

Constraints that set a specific size or distance between objects are called *pinning*. The flexibility (or inflexibility of a constraint) is managed by configuring a *relationship*.

This brings us to the second constraint object in the document outline: constraints on the object itself, not on its positioning. When you resize a user interface (UI) element or set constraints on how the content *within* it is positioned, Xcode adds (or modifies) a constraint object *within* the UI element itself. These constraints determine the width or height of the element. Not surprisingly, when you've got dozens of components in your UI, you'll have *lots* of constraints in your document outline.

Viewing and Editing Constraints via the Size Inspector

While sorting through constraints in the document outline can be a pain, you can quickly view *all* the constraints applied to an element (whether located in the Constraints object on the view the element is in, or within the UI element itself) within the Size Inspector (Option-Command-5). For example, if you've added a resized button to the storyboard, opening the Size Inspector should look something like Figure 16.3. Here horizontal and vertical positioning for the button are represented as constraints, as well as the width of the button.


Empty.storyboard

Empty.storyboard 〉 View… 〉 View… 〉 View 〉 Button – Button

View

Show: Frame Rectangle

X: 36 Y: 29

Width: 103 Height: 44

Origin

Content Hugging Priority

Horizontal: 250

Vertical: 250

Content Compression Resistance Priority

Horizontal: 750

Vertical: 750

Constraints

Width Equals: 103

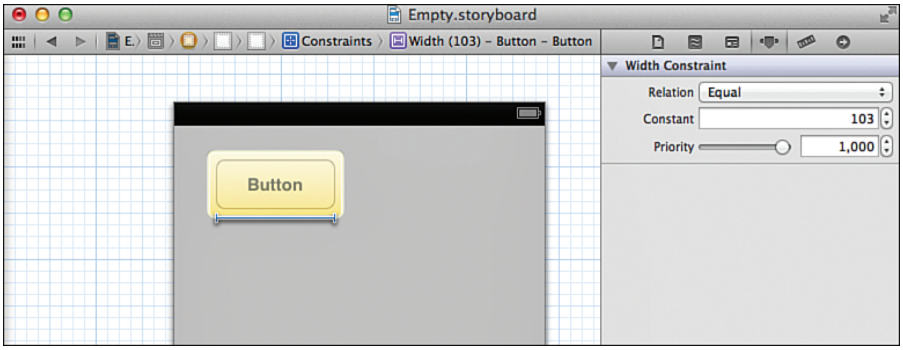
Top Space to: Superview Equals: 29

Leading Space to: Superview Equals: 36

FIGURE 16.3
Use the Size Inspector to view your constraints.

Using the gear menu to the right side of each constraint to do one of three things. First, you can select and edit a constraint. For example, if you pick a width constraint and choose Select and Edit, Xcode jumps you into the Attributes Inspector, where you can see and adjust the details of the constraint, as shown in Figure 16.4.

The Relation drop-down determines what the constraint is trying to do. Is it trying to keep a distance or size equal to a value? Greater than or equal to it? Less than or equal? You choose the relation that gives your interface the flexibility you need. With each relation, you can also set a constant. This is a measurement, in points, that the relationship will be maintaining.



Width Constraint
Relation Equal
Constant 103
Priority 1,000

FIGURE 16.4
Use the Attributes Inspector to edit constraints.

You'll also notice a Priority slider and a Standard check box (sometimes) appear in the editor. The Priority slider determines how "strong" the constraint relationship is. There may, for example, be instances when multiple constraints must be evaluated against one another. The priority setting determines the importance of any given constraint. A value of 1000 is a constraint that is required. If you move the Constraint slider, Xcode shows a description of what to expect at a given constraint priority.

The Standard check box lets Xcode use its internal database of spacing to set the recommended space between two objects. This, in many cases, is preferred because it ensures a consistent interface.

In a few minutes, we'll make use of the editor to configure a few relationships. Trust me, it makes more sense once you get started.

TIP

Constraints can also be edited by selecting them in the design view and opening the Attributes Inspector. If you're good at clicking on thin lines, you can do this without first going into the Size Inspector. I, personally, prefer starting by selecting the UI element, viewing the constraints in the Size Inspector, then jumping into editing from there.

The other options available in the gear menu in the Size Inspector are Promote to User Constraint and Delete. With Promote to User Constraint, you can select a constraint that you didn't add manually, turn it into a user constraint. The other option, Delete, removes the constraint from the Storyboard. Remember that there *must* be constraints to describe the object's position complete, so if you delete a constraint, don't be surprised to see Xcode add a new constraint back into the storyboard to fill in missing positioning details.

NOTE

You can edit system constraints *and* user constraints, so if the system constraints are serving your needs, there's no reason to turn it into a user constraint.

Content Hugging and Content Compression Resistance

When viewing an interface object with the Size Inspector, you probably noticed the settings Content Hugging Priority and Content Compression Resistance Priority. These features are related to Auto Layout, but what do they do?

When inspecting an object that is within an Auto Layout XIB, these settings control how closely the sides of an object "hug" the content in the object and how much the content can be compressed or clipped. Imagine a button that you want to expand horizontally, but not vertically. To allow horizontal expansion, you'd set horizontal hugging is a low priority. To keep it from growing vertically, you set vertical hugging to a high priority.

Similarly, the content (the button label) should not be compressed or clipped at all, so the content compression resistance settings for both horizontal and vertical compression should be very high priority.

You will not often need to adjust these settings beyond their defaults, which IB adds for you.

TIP

UI objects like buttons have *intrinsic* constraints that do not appear in the interface. These are the sizes that Xcode determines the object should have in order to display your content (such as a button label or button image). Xcode will automatically try to maintain these constraints unless you override them with your own user constraints and content compression/hugging settings. Other objects, such as switches, cannot resize at all, so there's no tweaking that will cause them to violate their intrinsic constraints. Text views, image views, and other "generic" content holders, however, will happily resize with no difficulty. You'll see this behavior in a few minutes.

I'm Beginning to See How This Works, But How Can You Create Complex Responsive Layouts with Just These Simple Tools?

There is no "one way" to build interfaces with Auto Layout, but there is a mindset that will be helpful as you build your own designs. Specifically, you should start by identifying the interface elements that you want to anchor somewhere on the screen, regardless of orientation or screen size. From there, begin looking for the relationships between those elements and the other objects that make up your interface. Building relationships between objects, rather than trying to tie each element to specific coordinates within a view, is a good tactic for successful UI design.

Example One: Centering Constraints

Let's look at a very simple example: a button that stays centered regardless of whether you use a 3.5" or 4" iPhone (or any iPad), or rotates between portrait and landscape orientations. To do this, begin by adding and positioning the button:

1. If you already have a push button added to a window in the Empty.storyboard file, you're in good shape. If not, open the file and add one now.
2. With the button highlighted, select Editor, Align, Vertical Center in Container.
3. Reselect the button, then choose Editor, Align, Horizontal Center in Container. The layout should now resemble Figure 16.5.

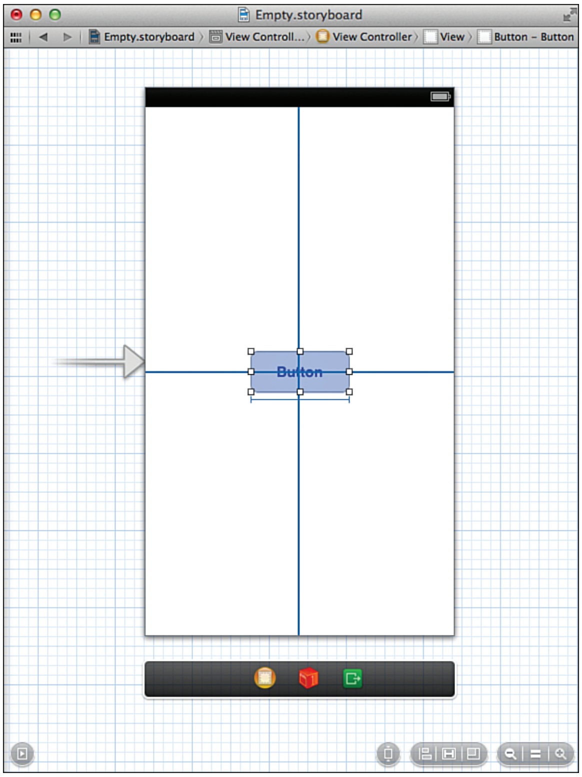


FIGURE 16.5
Align the button to the vertical and hortizontal centers.

The constraints that you've added show up as dark blue lines in the layout. Dark blue lines indicate user constraints; that is, *you* added the alignment constraints to the system.

TIP

If you don't add your own constraints Xcode will add some for you. In this example, had you dragged the button into the center of your view until both horizontal and vertical alignment guides appeared, Xcode would actually have created these same constraints for you! They would be system constraints, not user constraints, but they'd do the exact same thing.

Take a moment to view the constraints on the button via the Size Inspector. Select the button, then press Command-Option-5; the Size Inspector should show at least two constraints (more if you changed the size of the button), as shown in Figure 16.6.

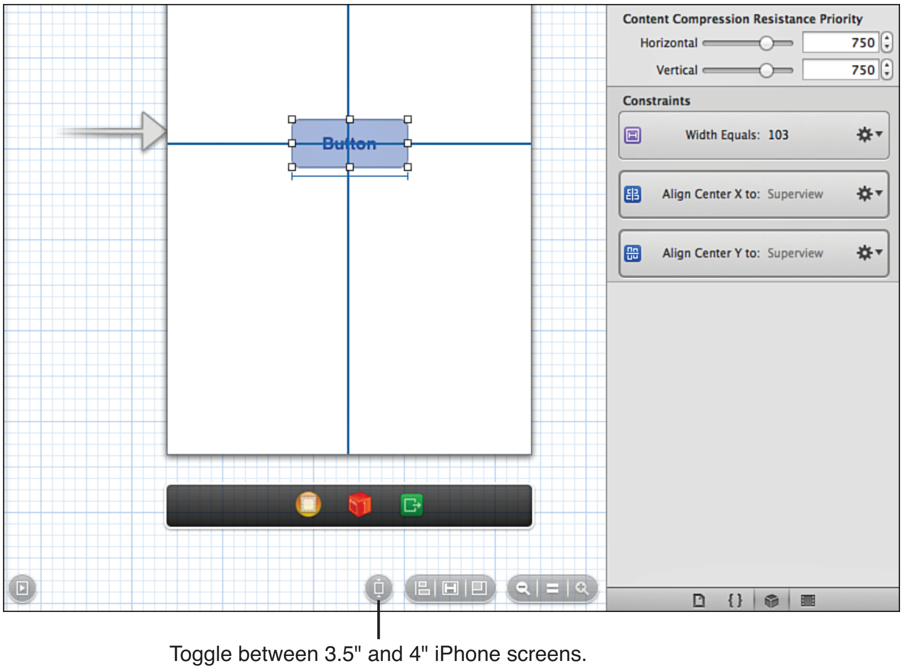

Content Compression Resistance Priority
Horizontal 750
Vertical 750
Constraints
Width Equals: 103
Align Center X to: Superview
Align Center Y to: Superview

FIGURE 16.6
Toggle the size of the screen.

Now try toggling the size of the screen between 3.5" and 4", using the toggle icon shown in Figure 16.6. The button should stay centered, regardless of the screen size. The final test is to try the interface in another orientation.

Make sure that the document outline is visible, select the view controller for the scene. Open the Attributes Inspector (Option-Command-4) and choose Landscape from the Orientation drop-down menu. IB updates to show the interface in landscape mode. Assuming your constraints are correct, the button should also properly center itself in the view, as demonstrated in Figure 16.7.

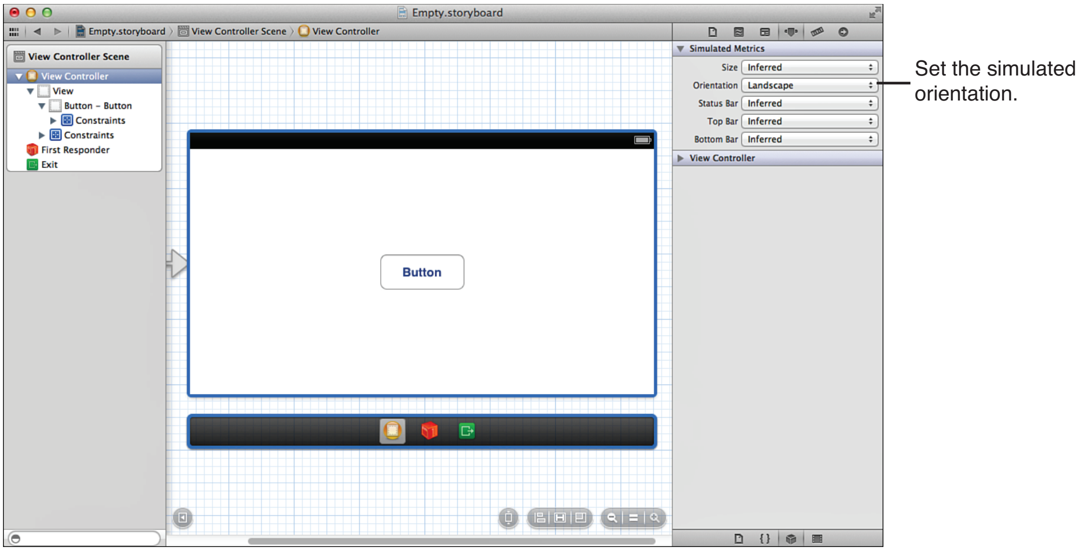


FIGURE 16.7
The button centers itself, regardless of orientation and screen size.

That was pretty simple, wasn't it? Of course, not all your applications will consist of just a single button (only the best ones). For that reason, we cover a few more examples, starting with resizing controls.

Example Two: Expanding Controls

A common design problem is where you have a series of controls on the screen and they need to expand to fill the size of the screen (or, in the case of a 3.5" iPhone display, properly shrink to fit on the screen). For example, let's create an interface with two text views at the top and a button at the bottom. The button should always remain anchored at the bottom, but should grow horizontally to fill the width of the display. The text views should grow or shrink vertically, and expand horizontally to fill the screen.

Creating the Interface

Starting from a fresh copy of Empty.storyboard, follow these steps:

1. Drag a button to the storyboard, positioning it using the guide at the bottom of the view.
2. Resize each side of the button by dragging toward the side of the view, and then release when you see the blue guide appear.

By using the guides in steps 1 and 2, you've created three constraints that anchor to the side and bottom margins of the view, as shown in Figure 16.8.

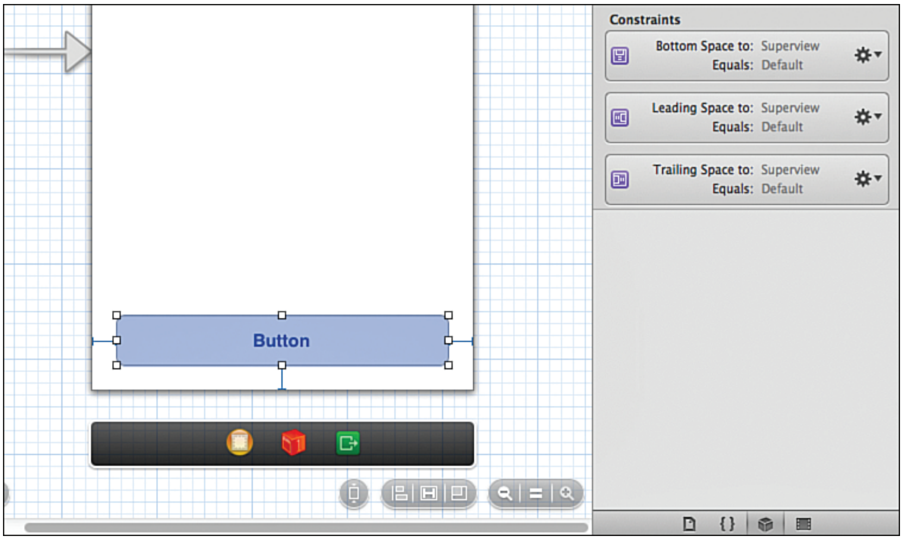

Constraints
Bottom Space to: Superview
Equals: Default
Leading Space to: Superview
Equals: Default
Trailing Space to: Superview
Equals: Default
Button

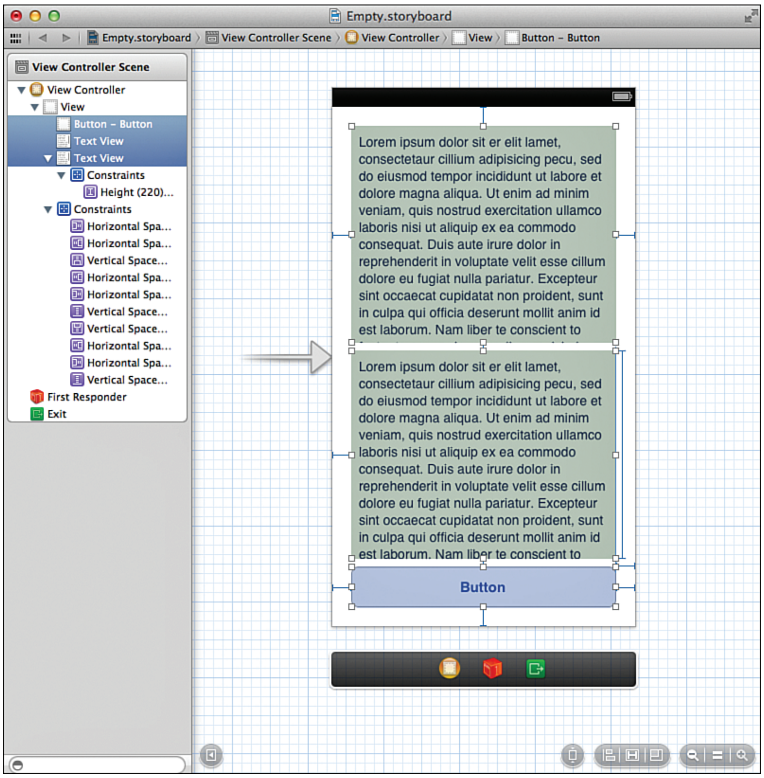
FIGURE 16.8
These three constraints make the button resize and stay anchored at the bottom, regardless of the screen size or orientation.

The constraints that have been automatically created are a good start for the button. These ensure that the sides of the button are always an equal distance from the sides of the view, regardless of the orientation. The same goes for the distance from the bottom of the view to the bottom of the button.

Next, add two text views to the scene:

1. Drag two text views to the scene, stacking one on top of the other.
2. Resize the text views to fill the scene horizontally, until the blue guides appear.
3. Size the text views vertically until the guides appear to keep the views from overlapping the button, one another, or the top margin of the scene.
4. Set the color of the text views to something other than white so that they are easier to see. Your finished view should look like Figure 16.9.

The top text view has constraints generated to attach it to the top of the view and the sides of the view. The bottom text view has constraints that keep it a set distance from the sides of the view and the button at the bottom. Between the two text views is a constraint that keeps them a set distance apart. Finally, one of the text views will have a height constraint applied.



Lorem ipsum dolor sit er elit lamet, consectetaur cillium adipisicing pecu, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Nam liber te conscient to

Lorem ipsum dolor sit er elit lamet, consectetaur cillium adipisicing pecu, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Nam liber te conscient to

FIGURE 16.9
You've added three UI elements and quite a few system constraints.

Setting the Constraints

Try resizing and rotating the display. What you'll notice is that resizing it vertically (switching between 3.5" and 4" screens) likely works okay. What definitely doesn't is rotating to landscape. In landscape mode, one of the text views will shrink so much that it isn't even visible. To make the scene work right, you need to set some additional constraints.

Specifically, you need to accomplish these things:

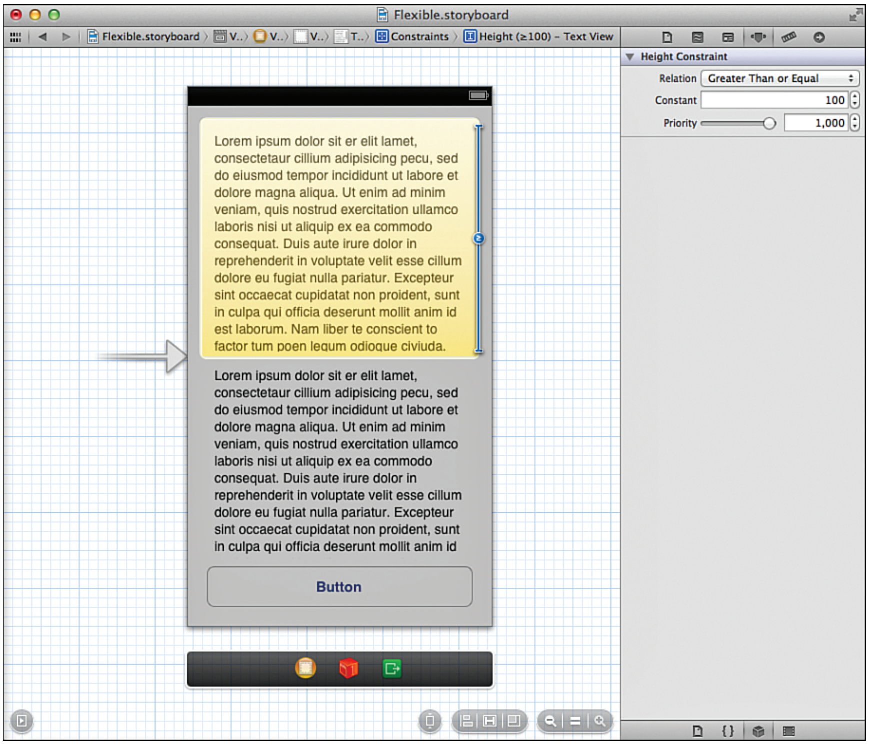
1. Pin the height of the button so that at no point in time is it resized.
2. Pick one of the text views to be the first to shrink when the size changes. We set this view so that its height is greater than or equal to the smallest size we want it to reach. I will use the top view for this purpose and set its minimum height to 100 points or greater.
3. Set a fixed width on the bottom text view (I'll use 210 points) so that it doesn't change size. *But*, we need to set its priority to less than 1000 (the default). This lets the text view change

sizes in an extreme situation. That “extreme situation” would otherwise occur in landscape mode where the top view would try to maintain a height of 100 points but the bottom view’s fixed height of 210 would force a condition where both can’t hold.

Begin by selecting the button and choosing, Editor, Pin, Height from the menu bar. Simple enough!

Next, choose the top text view and choose Editor, Pin, Height (again). You’ve just created a height constraint, but it needs to be adjusted so that it is flexible. Select the height constraint within the design view (or within the size inspector), and then open the Attributes inspector.

Using the Relation drop-down, choose Greater Than or Equal. Set the constant to 100 (the text view must be 100 points in height or greater) and leave the priority alone. Your configuration should look like Figure 16.10.


Flexible.storyboard
Flexible.storyboard › V.. › V.. › V.. › T.. › Constraints › Height (≥100) – Text View
Button

Height Constraint

Relation: Greater Than or Equal

Constant: 100

Priority: 1,000

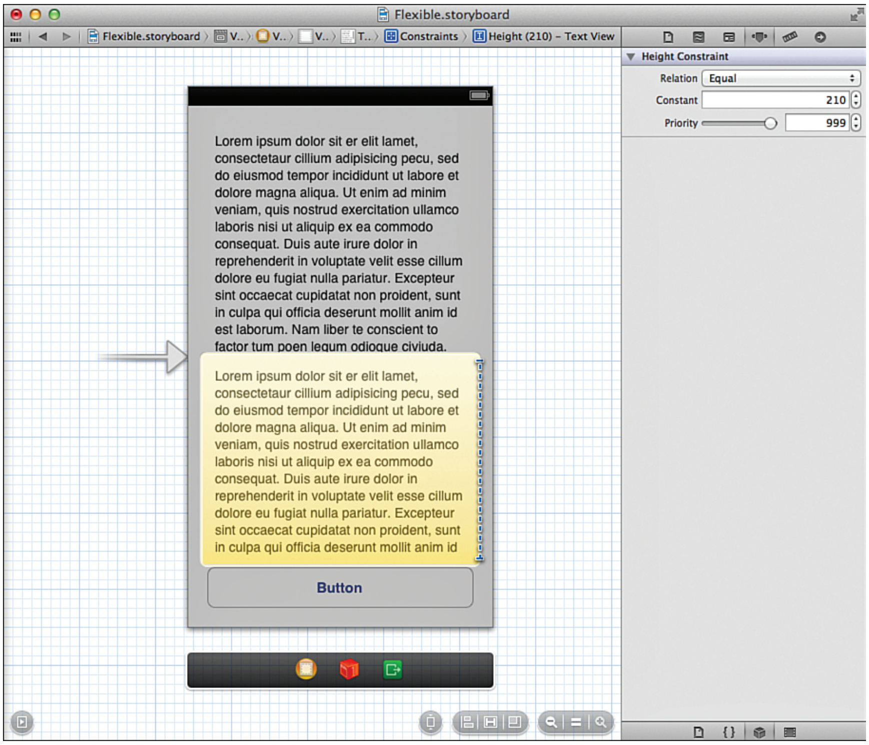
Lorem ipsum dolor sit er elit lamet, consectetaur cillium adipisicing pecu, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Nam liber te conscient to factor tum poen legum odioque civiuda.

Lorem ipsum dolor sit er elit lamet, consectetaur cillium adipisicing pecu, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id

FIGURE 16.10
Set the relation for the height constraint.

Now the top text view will shrink when it needs to, but it can also grow; in fact, until a constraint is set on the bottom text view, it can grow *too large*, pushing the bottom text view until it is too small in any orientation.

Select the bottom text view and again add a height constraint (Editor, Pin, Height, yet again). Select the constraint and make sure that the relation is set to Equal with a constant around 210, as shown in Figure 16.11. This time, however, set the priority down to 999. This makes the constraint turn into a dotted line (visible in Figure 16.11) showing that the constaint will *try* to be maintained, but, if necessary, it can resize to avoid violating other constraints.


Flexible.storyboard
Flexible.storyboard 〉 V.. 〉 V.. 〉 V.. 〉 T.. 〉 Constraints 〉 Height (210) – Text View
Button

Height Constraint

Relation: Equal
Constant: 210
Priority: 999

Lorem ipsum dolor sit er elit lamet, consectetaur cillium adipisicing pecu, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Nam liber te conscient to factor tum poen legum odioque civiuda.

Lorem ipsum dolor sit er elit lamet, consectetaur cillium adipisicing pecu, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id

FIGURE 16.11
Set a height constraint with a priority lower than 1000 to enable the text view to change sizes if absolutely necessary.

Go ahead and try resizing the view between 3.5" and 4" iPhone screen sizes and rotating to landscape mode. Everything should work like a charm, with the top text view resizing when needed and the bottom text view resizing when it absolutely has to.

CAUTION

What If It Doesn't Work?

It's *really* easy to end up adding constraints you don't want by inadvertantly dragging things around in your view. If something doesn't appear to be working, it's likely because there are extra height constraints. Look for multiple height constraints (or any others that look out of place) and delete what you didn't intentionally add.

TIP

Combining multiple constraints with variable relationships, like this, is what you need to do to accommodate complex interfaces. Don't be shy about trying different relationships and priorities to see how they interact. There isn't a "recipe" for making effective Auto Layout designs. Spend a few hours playing with the tools—laying out interfaces in different ways. Hands-on is the only way you'll get the experience you need to be comfortable with Auto Layout.

Example Three: Variable and Matching Sizes (and Auto Layout Craziness!)

Okay, one more quick exercise to round things out. I promise this one is easier than the last. This time, we add three buttons in a horizontal line, and we add enough constraints to make the buttons all resize equally when we shift to a landscape mode.

CAUTION

They're *My* Constraints, Not Yours, Xcode!

When I say "Auto Layout craziness," I mean it. Even if you aren't building the example, be sure to read through this section to understand how Xcode can sometimes, without warning, start rewriting your Auto Layout constraints and wreaking havock on your work.

Let's make this happen! Create a new copy of the Empty.storyboard document, and then drag three buttons into the horizontal line on the screen. Label the buttons **Button A**, **Button B**, and **Button C**, as shown in Figure 16.12.

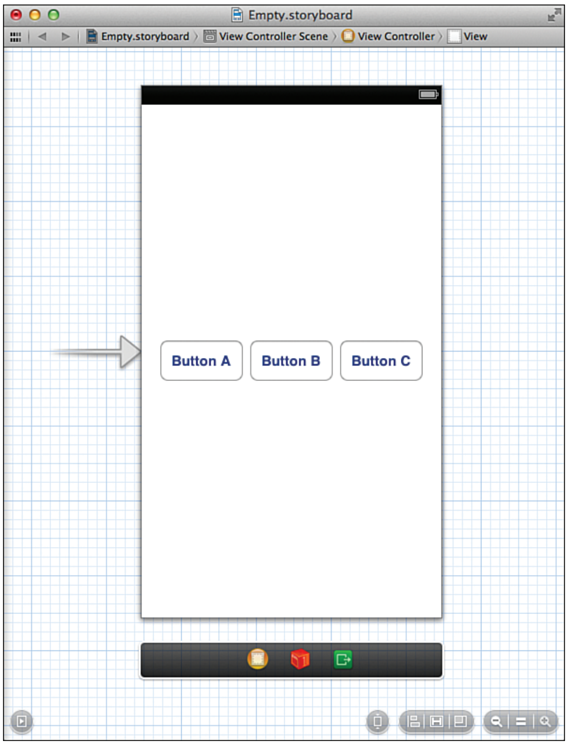


FIGURE 16.12
Add three buttons, nothing more.

You now know enough to make one of the buttons resize, but how could we constrain them so that *all* the buttons resize to fill in the screen in landscape mode? These are the constraints that I came up with:

1. Align all the buttons to the vertical center in the container. This keeps them centered vertically on the screen.
2. Pin the Leading Space of Button A to the superview (the edge of the view). This is unnecessary if you used a guide to align the left edge of the button and Xcode already added a constraint tying the edge of the button to the edge of the view.
3. Pin the Trailing Space of Button C to the superview. Like the first constraint, you don't technically need this one if you aligned the right side of the button to a guide and Xcode added a constraint for you.
4. Pin the Horizontal Spacing between Button A and Button B.

5. Pin the Horizontal Spacing between Button B and C. (If we stopped here, the center button (Button B) would stretch while the other buttons would stay the same size.)
6. Pin the buttons so that they all share the same width. (Select all the buttons, and then choose Editor, Pin, Widths Equally.)

Use the Editor menu to add these constraints to your project; you should know how to do it by now. There is no need to adjust the priority or relationships in this case. Your finished layout will hopefully be similar to Figure 16.13 (one of the = constraints may be on a different button. This is fine). *Save your work before continuing.*

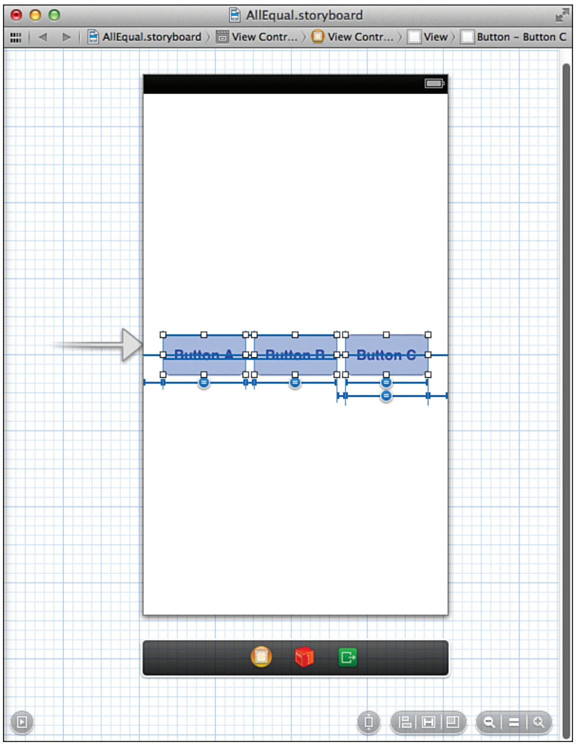
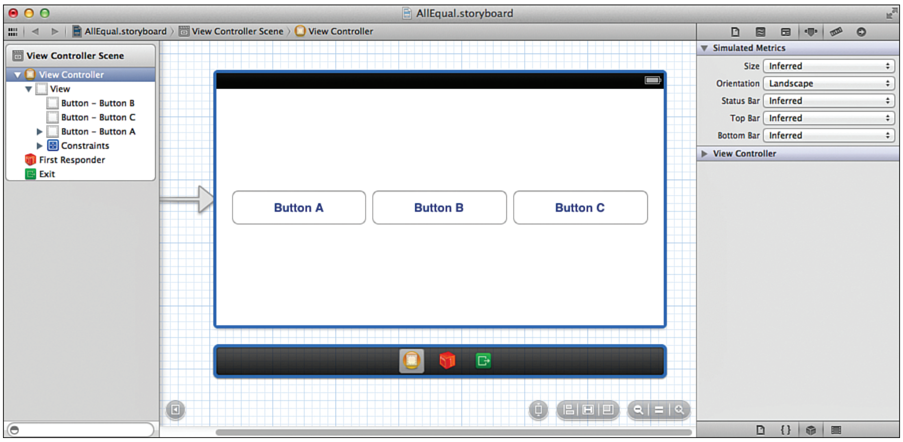


FIGURE 16.13
Add these constraints to your scene.

Now, here's where it gets interesting (wake up!). When you rotate the view to landscape, all the buttons should resize equally and fill in the space across the screen, as shown in Figure 16.14.



Size: Inferred
Orientation: Landscape
Status Bar: Inferred
Top Bar: Inferred
Bottom Bar: Inferred

FIGURE 16.14
The buttons' constraints force them to spread out across the screen and match sizes.

This is exactly what we want. *But*, what happens when you rotate back to a vertical position? Chances are that the buttons get all weird looking and overlap. Switch back and forth between horizontal and landscape orientations, and they may get even worse.

So, what's going on here? Xcode is being a pain, that's what. When you switch to landscape, it is rewriting the constraints as if you designed the view in landscape mode. You actually *lose* work and constraint logic depending on how the complex your interface is.

To deal with this bit of insanity, just use Command-Z to undo the switch to landscape orientation after validating that the interface looks okay. As long as it looks good in your original view and when it first switches to landscape, you're in good shape. The compiled view used in your actual application will work just fine (even if it switches back and forth between portrait and landscape orientations).

Wait, There's More

There is much more to the Auto Layout system than can be described in an hour. Be sure to explore the Pin menu to see the different types of constraints that you can put in place. Width/height constraints enforce a given width or height on an object. Equal width/height constraints ensure multiple objects maintain an equal width or height. The leading/trailing space pinnings tie the left side of an object to the left side of its parent view (leading), or the right side of an object to the right side of its parent view (trailing).

Review Apple's documentation, starting with *Cocoa Auto Layout Guide*, for more information.

Programmatically Defined Interfaces

In the previous example, you learned how the IB editor can help quickly create interface layouts that look as good horizontally as they do vertically and that can resize between the 3.5" and 4" iPhone displays. Unfortunately, in plenty of situations, IB can't quite accommodate. Irregularly spaced controls and tightly packed layouts rarely work out the way you expect. You may also find yourself wanting to tweak the interface to look completely different—positioning objects that were at the top of the view down by the bottom and so on.

In cases like this, consider implementing the UI (or portions of it) completely in code. But, what about the nice and neat drag-and-drop approach? Well, using code isn't as convenient as drawing your interface in IB, but it *isn't* difficult. We're going to move to a project that is the exact opposite of our last few examples—rather than building a responsive interface without code, we're going to *only* use code to create an interface.

Implementation Overview

In this tutorial, we create a very simple application with three UI elements: two buttons (Button A and Button B) and a label. The buttons trigger a method to set the label to the title of the button. Most important, however, is that the interface reacts properly to orientation and size changes.

In the portrait orientation, the buttons are drawn with the label sandwiched between them. In landscape, the buttons move closer to the bottom, and the label repositions above them. The final output will resemble Figure 16.15.

Take note that the positioning of the buttons and the label cannot be handled in Auto Layout (at least not through any straightforward approach). When you encounter issues that can't be solved in Auto Layout, there's no harm in coding your way out of the forest.

To handle the rotation and resizing of the objects, we use the `bounds` property of the scene's view. This gives us the width and height of the device corresponding to whatever orientation it is in. We then use these values to position the UI elements on the screen. By basing the positioning on the `bounds`, the size of the device screen and orientation are largely irrelevant, as you'll soon see.

Setting Up the Project

Unlike the previous example, we can't rely pointing and clicking for the interface, so there is a bit of code in the project. Once again, create a new single-view iOS application project and name it **AllInCode**.

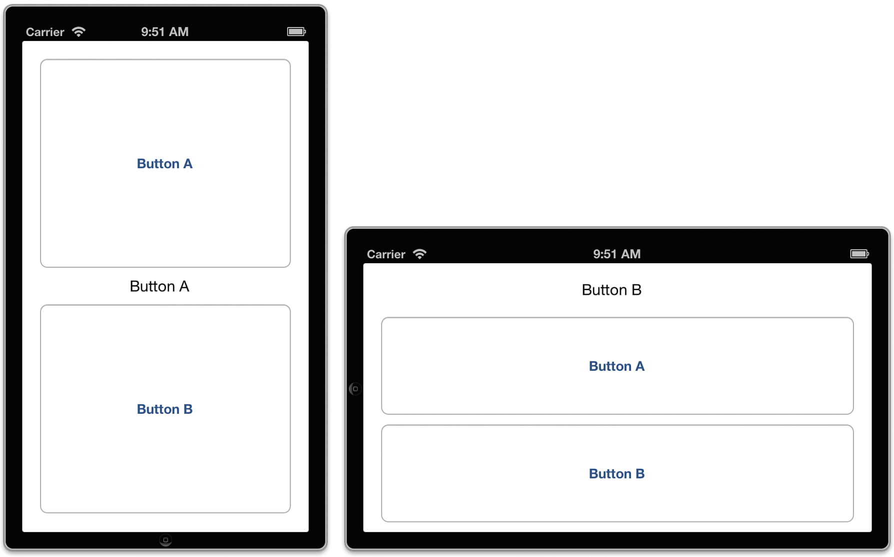


FIGURE 16.15
Buttons resize and reposition appropriately (all handled in code).

Planning the Properties and Connections

In this exercise, you manually resize and reposition three UI elements: two buttons (`UIButton`) and one label (`UILabel`). Although we aren't creating these with outlets, we define properties for them: `buttonA`, `buttonB`, and `theLabel` should suffice.

Why Aren't You Using Private Instance Variables?

Good question. You're welcome to do this, exactly as we have in some of our other projects. In this case, however, it might be useful for our view controller to expose these other elements so that they could be used by other classes. This also matches how IB itself handles UI elements. If you disagree with this approach, no worries; instance variables will work just fine.

We also implement a method: `handleButton`, which updates the onscreen label to show the title of a button that was tapped. Like the properties, this won't be declared using IB, but we'll be using it just like an `IBAction`. We also add two additional methods, `initInterface` and `updateInterface`, to handle setting up and updating the interface, respectively. These will be triggered by a change in orientation, so our next step is to set up the project to properly handle orientation changes.

Enabling Orientation Changes

For this project, enable support of all orientations. To do this, start by updating the project summary (click the blue project icon at the top of the project navigator) and select all the device orientations within the Supported Interface Orientations section. Next, add the `supportedInterfaceOrientations` method to `ViewController.m` and have it return the constant `UIInterfaceOrientationMaskAll`, as shown in Listing 16.2.

LISTING 16.2 Support All Interface Orientations

```
- (NSUInteger)supportedInterfaceOrientations {
    return UIInterfaceOrientationMaskAll;
}
```

Programming the Interface

We've now reached the point in the project where normally I'd say, "Let's design the interface." This time, however, there isn't going to be a visual design, just code. In fact, you've already seen the two screenshots that accompany this project, so if you're just skimming pictures, you'd better flip through a few more pages.

Defining Properties and Methods

We start by defining the properties and methods that the view controller will be using. Recall that we're adding three properties: `buttonA`, `buttonB`, and `theLabel`. We' also have three methods that we should prototype in ViewController.h: `initInterface`, `updateInterface`, and `handleButton`.

Edit ViewController.h, adding the properties and method prototypes, as shown in Listing 16.3.

LISTING 16.3 Add the Properties and Method Prototypes to ViewController.h

```
#import <UIKit/UIKit.h>

@interface ViewController : UIViewController

@property (weak,nonatomic) UIButton *buttonA;
@property (weak,nonatomic) UIButton *buttonB;
@property (strong,nonatomic) UILabel *theLabel;

- (void)initInterface;
- (void)updateInterface;
- (void)handleButton:(id)sender;

@end
```

Look closely and you'll notice that theLabel is set to use a strong property reference. Unfortunately, this is a result of overzealous memory cleanup by ARC. If we use weak instead, the label will actually be removed from memory before we can even get it onto the screen. Using strong ensures that the label sticks around as long as the ViewController object is active.

Everything else should look pretty straightforward. The initInterface and updateInterface methods don't take any arguments or return any values. The handleButton method is styled after a typical IBAction because it will behave exactly like a typical action; it's just being defined by hand rather than being built for us.

Initializing the Interface Properties

The next step is to add the initInterface method to ViewController.m. The purpose of this method is to configure all the interface elements (the two buttons and the label) so that they're ready to be added to the interface, but *not* display them just yet.

By keeping the display logic separated from the initialization logic, we can build a method that can be called at any time to update the interface. This method, aptly named updateInterface, is called at the end of the initInterface and anytime interface rotation is sensed.

Add the initInterface method from Listing 16.4 to ViewController.m.

LISTING 16.4 Prepare the Interface But Don't Display It Yet

```
- (void)initInterface {
    self.buttonA =[UIButton buttonWithType:UIButtonTypeRoundedRect];
    [self.buttonA addTarget:self action:@selector(handleButton:)
            forControlEvents:UIControlEventTouchUpInside];
    [self.buttonA setTitle:@"Button A" forState:UIControlStateNormal];

    self.buttonB =[UIButton buttonWithType:UIButtonTypeRoundedRect];
    [self.buttonB addTarget:self action:@selector(handleButton:)
            forControlEvents:UIControlEventTouchUpInside];
    [self.buttonB setTitle:@"Button B" forState:UIControlStateNormal];

    self.theLabel=[[UILabel alloc] init];
    self.theLabel.text=@"Welcome";
    [self updateInterface];
}
```

This might be the first time we've manually created a number of UI elements, but because you've been working with these objects and adjusting their properties for hours, this code shouldn't seem completely foreign.

Line 2 initializes the `buttonA` property as a button of type `UIButtonTypeRoundedRect`—the standard button we use in our views. Lines 3–4 use the button's `addTarget:action:forControlEvents` method to choose what will happen when the `Touch Up Inside` event occurs for the button. The `@selector` directive specifies which method will be called during the event—such as `handleButton`. This is exactly the same as connecting a button to an `IBAction` in IB.

Line 5 sets the title for the button to Button A.

Lines 7–10 repeat the same process for Button B (`buttonB`).

Lines 12–13 allocate and initialize a label (`theLabel`) with the default text Welcome.

Lastly, line 14 invokes the `updateInterface` method so that the newly defined user elements can be placed on the screen. So, what do we do now? Implement `updateInterface`.

Implementing the Interface Update Method

The `updateInterface` method does the heavy lifting for the application. It checks to see what the current orientation is, and then it draws content based on the view's `bounds` property. By basing the drawing on the height and width contained within `bounds`, you can scale to any screen size at all.

For example, consider this code snippet:

```
float screenWidth;
float screenHeight;
screenWidth=self.view.bounds.size.width;
screenHeight=self.view.bounds.size.height;
```

This grabs and stores the current screen width and height in the variables `screenWidth` and `screenHeight`. The dimensions and position of UI objects are determined by their `frame`, which is a property of type `CGRect`. To set the `frame` of a button property named `theButton` so that it filled the top half of the screen, I'd write the following:

```
self.theButton.frame=CGRectMake(0.0,0.0,screenWidth,screenHeight/2);
```

The first two values of `CGRectMake` (which create a `CGRect` data structure) set the origin point at 0,0. The second two parameters determine the width and height of the `CGRect`. Using `screenWidth` sets the button to the same width of the screen and `screenHeight/2` sets the height of the button to half the height of the screen. In an actual implementation, you want to include some margin around the edges. This is why you'll see +20 and other values tacked onto my coordinates. Speaking of which, go ahead and implement `updateInterface`, as shown in Listing 16.5. When you're done, we step through the code.

LISTING 16.5 The `updateInterface` Implementation

```
 1: - (void)updateInterface {
 2:     float screenWidth;
 3:     float screenHeight;
 4:     screenWidth=self.view.bounds.size.width;
 5:     screenHeight=self.view.bounds.size.height;
 6:
 7:     if (self.interfaceOrientation==UIInterfaceOrientationPortrait ||
 8:         self.interfaceOrientation==UIInterfaceOrientationPortraitUpsideDown) {
 9:         self.buttonA.frame=CGRectMake(20.0,20.0,screenWidth-40.0,
10:                                       screenHeight/2-40.0);
11:         self.buttonB.frame=CGRectMake(20.0,screenHeight/2+20,
12:                                       screenWidth-40.0,screenHeight/2-40.0);
13:         self.theLabel.frame=CGRectMake(screenWidth/2-40,
14:                                        screenHeight/2-10,200.0,20.0);
15:     } else {
16:         self.buttonA.frame=CGRectMake(20.0,60.0,screenWidth-40.0,
17:                                       screenHeight/2-40.0);
18:         self.buttonB.frame=CGRectMake(20.0,screenHeight/2+30,
19:                                       screenWidth-40.0,screenHeight/2-40.0);
20:         self.theLabel.frame=CGRectMake(screenWidth/2-40,20.0,200.0,20.0);
21:     }
22:
23:     [self.view addSubview:self.buttonA];
24:     [self.view addSubview:self.buttonB];
25:     [self.view addSubview:self.theLabel];
26: }
```

Lines 2–5 grab and store the current screen size in `screenWidth` and `screenHeight`.

Lines 7–8 checks the `interfaceOrientation` property of the view controller, and, if it is in one of the portrait orientations, lines 8–14 are executed. Otherwise, lines 16–20 are evaluated. These blocks both have the same purpose: defining the `frame` properties for each of the UI elements (`buttonA`, `buttonB`, and `theLabel`).

Lines 8–14 define positions for the buttons so that there are margins on the edges of the screen and a space in the middle for the label. Lines 16–20 position the buttons lower on the screen and put the label at the top. The margins and spacing I used is completely arbitrary. You can try changing these values around to see what effect they have.

Finally, lines 23–25 add the buttons and label to the view so that they are visible onscreen.

Everything is now in place for the interface, but we need to take care of three small tasks before the project is complete. First, we need to make sure that the interface is drawn when the application first loads. Second, the interface must update when an orientation change occurs. Third, we need to implement `handleButton` to update the label when the buttons are pressed.

Drawing the Interface When the Application Launches

When the application first launches, there isn't an orientation change to trigger the interface to be drawn. To make sure there is something on the screen, we need to call `initInterface` when the application loads. Add this to `viewDidLoad`, as shown in Listing 16.6.

LISTING 16.6 Initialize the Interface When the Application Loads

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    [self initInterface];
}
```

We're getting closer. The application will now initialize and display the interface, but it still can't adapt to a change in orientation.

Updating the Interface When Orientation Changes

To handle orientation changes, the application needs to call `updateInterface` within an implementation of `didRotateFromInterfaceOrientation`. We also need to remove the existing buttons, otherwise the old version of the interface will still be visible; this is surprisingly easy to do.

Add `didRotateFromInterfaceOrientation` to ViewController.m, as shown in Listing 16.7.

LISTING 16.7 Handle Rotation in `didRotateFromInterfaceOrientation`

```
1: - (void)didRotateFromInterfaceOrientation:
2:         (UIInterfaceOrientation)fromInterfaceOrientation {
3:     [[self.view subviews]
4:                 makeObjectsPerformSelector:@selector(removeFromSuperview)];
5:     [self updateInterface];
6: }
```

Lines 3–4 use the very cool `makeObjectsPerformSelector` method on all the subviews in the scene's view (all of our UI elements) to send them the message `removeFromSuperview`. This, as expected, removes the buttons and label from the view.

Line 5 calls `updateInterface` and draws the appropriate version of the interface for whatever orientation we are currently in.

Handling the Button Touches

The last piece of the puzzle is implementing `handleButton` so that it updates the onscreen label with the label of the button being touched. This is just a single line, so add Listing 16.8 to the view controller, and you're done.

LISTING 16.8 Handle Button Touches

```
- (void)handleButton:(id)sender {
    self.theLabel.text=((UIButton *)sender).currentTitle;
}
```

The one line of the implementation uses the `sender` parameter (typecast as a `UIButton`) to grab the title of the button (`currentTitle`) that was pressed.

Building the Application

Build and run the application. It should rotate and resize with no problem. What's more, because all of the interface layout was based on the height and width of the view, this same code will work, without changes, in an iPad or iPhone project.

I hope this didn't scare you too much. The purpose of this exercise was to show that responsive and flexible interfaces can be accomplished in code without it being too much of a hassle. The biggest challenge is determining how the controls will be laid out and then coming up with the `CGRectMake` functions to define their locations.

Swapping Views on Rotation

Some applications display entirely different UIs depending on the device's orientation. The iPhone Music application, for example, displays a scrolling list of songs in portrait mode and a "flickable" Cover Flow view of albums when held in landscape. You, too, can create applications that dramatically alter their appearance by simply switching between views when the phone is rotated.

Our last tutorial in this hour is short and sweet and gives you the flexibility to manage your landscape and portrait views all within the comfort of the IB editor. What's more, you can still use Auto Layout within each of these views to position your interface objects to accommodate different screen sizes (like the 3.5" and 4" iPhone screens).

Implementation Overview

The previous examples used a single view and rearranged it (either through Auto Layout or code) to fit a different orientation. When the view is too different or complex for this to be feasible, however, you can use two individual views with a single view controller. This is precisely what we do in this application. We start by adding a second view to the traditional single-view application, and then we design both views and make sure we can easily access them through properties in our code.

Once that is complete, we write the code necessary to swap the views when the device rotates. There is a catch, which you'll learn about in a bit, but nothing that poses too much of a problem to coders as experienced as we are.

Setting Up the Project

Create a new project named **Swapper** using the Single View Application template. Although this includes a single view already (which we'll use for the default portrait display), we need to supplement it with a second landscape view.

Planning the Property and Connections

This application does not implement any real UI elements, but we need to access two `UIView` instances programmatically. One view is for portrait orientation (`portraitView`) and another for landscape orientation (`landscapeView`). We will implement a method to handle orientation changes, but it will not be triggered by any actions.

Adding a Degree to Radians Constant

Later in this exercise, we have to call a special Core Graphics method to define how to rotate views. The method requires a value to be passed in radians rather than degrees. In other words, instead of saying we want to rotate the view 90 degrees, we have to tell it we want to rotate 1.57 radians. To help us handle the conversion, we define a constant for the conversion factor. Multiplying degrees by the constant gets us the resulting value in radians.

To define the constant, add the following line after the `#import` line in ViewController.m:

```
#define kDeg2Rad (3.1415926/180.0)
```

Enabling Orientation Changes

As with the previous example, we need to ensure that the implementation of `supportedInter-faceOrientations` is behaving as we expect in our view controller. Unlike the previous implementation, however, this time we allow the device to rotate only between the two landscape modes and upright portrait.

Update ViewController.m to include the implementation in Listing 16.9.

LISTING 16.9 Disable the Upside-Down Orientation

```
- (NSUInteger)supportedInterfaceOrientations {
    return UIInterfaceOrientationMaskAllButUpsideDown;
}
```

Be sure to also go into the project summary and set the allowed orientations to everything but upside-down.

Designing the Interface

When you are swapping views, the sky is the limit for the design. You build them exactly as you would in any other application. The only difference is that if you have multiple views handled by a single view controller, you must define outlets that encompass all the interface elements.

This example demonstrates just how to swap views, so our work will be a piece of cake.

Creating the Views

Open the MainStoryboard.storyboard file and drag a new instance of the `UIView` object from the Object Library to the document outline, placing it at the same level in the hierarchy as the view controller, as shown in Figure 16.16. Don't put the `UIView` inside the existing view.

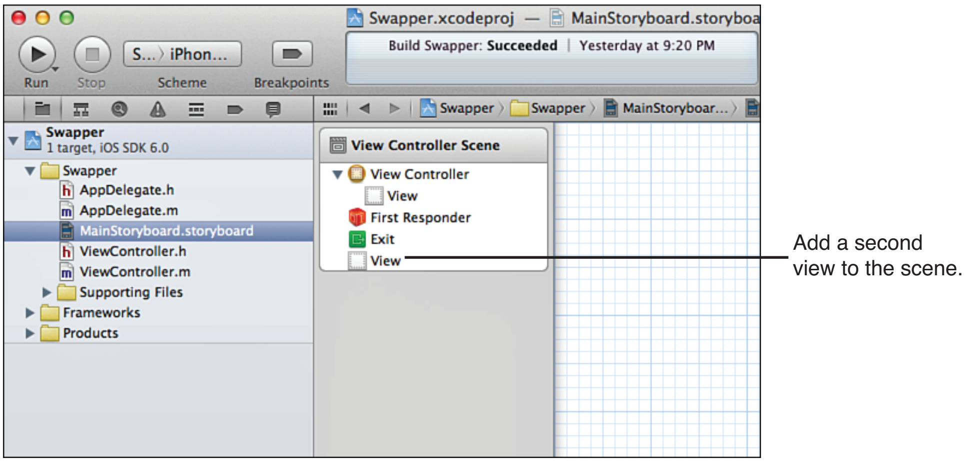


FIGURE 16.16
Add a second view to the scene.

TIP

It might seem counterintuitive, but there's no reason that a scene can't contain multiple views if they are going to be managed by the same view controller. Only the view nested within the view controller is displayed by default. Additional views have to be displayed manually.

Now, open the default view and add a label, such as **Portrait View**; make sure that it is selected. Use Editor, Align from the menu bar to set constraints so that it is aligned to the horizontal and vertical centers of the view. Now set a background color to differentiate the view. That finishes one view, but we still have another to do. Unfortunately, you can only edit a view that is assigned to a view controller in IB, so we have to be creative.

Drag the view you just created out of the view controller hierarchy in the document outline, placing it at the same level as the view controller. Drag the second view onto the view controller line in the document outline. You can now edit it by adding a unique background color and a label such as **Landscape View**. You may want to switch the view controller to simulate a landscape mode while making these edits. When the second view is done, rearrange the view hierarchy again, nesting the portrait view inside the view controller and the landscape view outside the view controller.

If you want to make this more interesting, you're welcome to add other controls and design the view as you see fit. Our finished landscape and portrait views are shown in Figure 16.17.

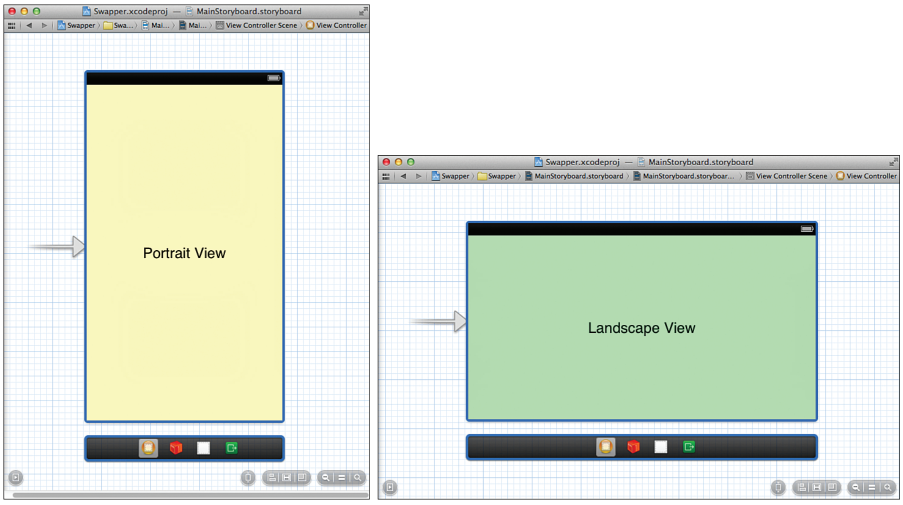

Swapper.xcodeproj — MainStoryboard.storyboard
Swapper › Swa... › Mai... › Mai... › View Controller Scene › View Controller
Portrait View
Swapper.xcodeproj — MainStoryboard.storyboard
Swapper › Swapper › MainStoryboard.storyboard › MainStoryboard.storyboar... › View Controller Scene › View Controller
Landscape View

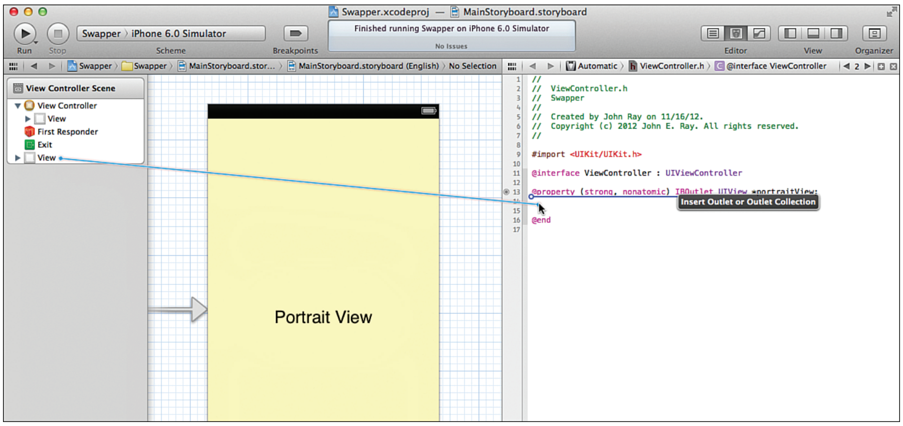
FIGURE 16.17
Edit the two views so that you can tell them apart.

Creating and Connecting the Outlets

To finish up our interface work, we need to connect the two views to two outlets. The default view (nested in the view controller) will be connected to `portraitView`. The second view will be connected to `landscapeView`. Switch to the assistant editor and make sure that the document outline is visible.

Because we're dealing with views rather than objects in our interface design, the easiest way to make these connections is to Control-drag from the respective lines in the document outline to the ViewController.h file. In addition, unlike with most of the projects in this book, we need to create outlets with the storage set to Strong; otherwise, ARC will conveniently get rid of the views when they aren't visible.

Control-drag from the default (nested) view to below the `@interface` line in ViewController.h. Create a new outlet for the view called `portraitView`, with the storage set as Strong. Repeat the process for the second view, naming the connection `landscapeView` as demonstrated in Figure 16.18.


Swapper.xcodeproj — MainStoryboard.storyboard
Run
Stop
Swapper › iPhone 6.0 Simulator
Scheme
Breakpoints
Finished running Swapper on iPhone 6.0 Simulator
No Issues
Editor
View
Organizer
Swapper › Swapper › MainStoryboard.stor... › MainStoryboard.storyboard (English) › No Selection
Automatic › ViewController.h › @interface ViewController
View Controller Scene
View Controller
View
First Responder
Exit
View
Portrait View
Insert Outlet or Outlet Collection

```
//
//  ViewController.h
//  Swapper
//
//  Created by John Ray on 11/16/12.
//  Copyright (c) 2012 John E. Ray. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface ViewController : UIViewController

@property (strong, nonatomic) IBOutlet UIView *portraitView;


@end
```

FIGURE 16.18
Connect the views to corresponding outlets using a storage type of Strong.

Implementing the Application Logic

For the most part, swapping views is actually easier than the reframing logic we implemented in the last project—with one small exception. Even though we designed one of the views to be in landscape view, it doesn't "know" that it is supposed to be displayed in a landscape orientation.

Understanding the View-Rotation Logic

For a landscape view to be successfully swapped onto the screen, we need to rotate it and define how big it is. The reason for this is that there is no inherent logic built in to a view that says "hey, I'm supposed to be sideways." As far as it knows, it is intended to be displayed in portrait mode but has UI elements that are pushed off the sides of display.

Each time we change orientation, we go through three steps: swapping the view, rotating the view to the proper orientation through the `transform` property, and setting the view's origin and size via the `bounds` property.

For example, assume we're rotating to landscape right orientation:

1. First, we can grab the current view size (after rotation) by accessing and storing `self.view.bounds`. This can be used later to make sure that the new view is set to the proper size:

   ```
   currentBounds=self.view.bounds;
   ```

2. Second, we swap out the view by assigning `self.view`, which contains the current view of the view controller, to the `landscapeView` property. If we left things at that, the view would properly switch, but it wouldn't be rotated into the landscape orientation. A landscape view displayed in a portrait orientation isn't a pretty thing. For example:

   ```
   self.view=self.landscapeView;
   ```

3. Next, to deal with the rotation, we define the `transform` property of the view. This property determines how the view will be altered before it is displayed. To meet our needs, we have to rotate the view 90 degrees to the right (for landscape right), –90 degrees to the left (for landscape left), and 0 degrees for portrait. As luck would have it, the Core Graphics C function, `CGAffineTransformMakeRotation()`, accepts a rotation value in radians and provides an appropriate structure to the `transform` property to handle the rotation. For example:

   ```
   self.view.transform=CGAffineTransformMakeRotation(deg2rad*(90));
   ```

NOTE

Note that we multiply the rotation in degrees (90, –90, and 0) by the constant `kDeg2Rad` that we defined earlier so that `CGAffineTransformMakeRotation()` has the radian value it expects.

4. The final step is to set the `bounds` property of the view to the bounds that we stored in step 1. For example:

   ```
   self.view.bounds=currentBounds;
   ```

Now that you understand the steps, let's take a look at the actual implementation.

Adding the View-Rotation Logic

All the rotation magic happens within a single method: `supportedInterfaceOrientations.`

Open the ViewController.m file and implement the method, as shown in Listing 16.7.

LISTING 16.10 Rotate the View into the Proper Orientation

```
 1: - (void)didRotateFromInterfaceOrientation:
 2:     (UIInterfaceOrientation)fromInterfaceOrientation {
 3:
 4:     CGRect currentBounds=self.view.bounds;
 5:
 6:     if (self.interfaceOrientation == UIInterfaceOrientationLandscapeRight) {
 7:         self.view=self.landscapeView;
 8:         self.view.transform=CGAffineTransformMakeRotation(kDeg2Rad*(90));
 9:     } else if (self.interfaceOrientation == UIInterfaceOrientationLandscapeLeft) {
10:         self.view=self.landscapeView;
11:         self.view.transform=CGAffineTransformMakeRotation(kDeg2Rad*(-90));
12:     } else {
13:         self.view=self.portraitView;
14:         self.view.transform=CGAffineTransformMakeRotation(0);
15:     }
16:     self.view.bounds=currentBounds;
17: }
```

Line 4 grabs the current `bounds` of the scene's view after the rotation has occurred, and stores it in `currentBounds`.

Lines 6–8 handle rotation to the right (landscape right). Lines 9–11 deal with rotation to the left (landscape left). Finally, lines 13–14 configure the view for the default orientation, portrait.

In the very last step, line 16, the bounds of the view we swapped in are set to the `currentBounds` that we stored when the method started.

NOTE

Although we used an `if-then-else` statement in this example, you could easily use a `switch` structure instead. The `toInterfaceOrientation` parameter and orientation constants are integer values, which means they can be evaluated directly in a `switch` statement.

TIP

In this hour's exercises, we used `didRotateFromInterfaceOrientation` to detect and react to a change in orientation. If you'd prefer to being reacting immediately *prior* to iOS adjusting the orientation, you can use the method `willRotateToInterfaceOrientation:duration` instead.

Building the Application

Save the implementation file, and then run and test the application. As you rotate the device or the iOS simulator, your views should be swapped in and out appropriately. This approach gives you a good combination of flexibility while keeping the benefits of Auto Layout. Unfortunately, it also means that you have to manage twice as many objects in your code.

When designing your own applications, you need to strike a balance between interface flexibility and code complexity. In some cases, it's just easier to design a different scene and use a second view and view controller to handle other orientations.

Further Exploration

Although we covered several different ways of working with rotation in the iPhone interface, you may want to explore additional features outside of this hour's lesson. Using the Xcode documentation tool, review the `UIView` instance methods. You'll see that there are additional methods that you can implement, such as `willAnimateRotationToInterfaceOrientation:duration`, which is used to set up a single-step animated rotation sequence. What's more, you can combine these methods with segues—programmatically triggering a segue to a new scene when an orientation event occurs.

In short, there are a number of ways to create interfaces that do anything you want. It's a good idea to practice the techniques discussed here to learn what will work best for your projects.

Summary

iDevices are all about the user experience—a touchable display, intuitive controls, and now, rotatable and resizable interfaces. Using the techniques described in this hour, you can adapt to almost any type of rotation scenario. To handle interface size changes without a line of code, for example, you can take advantage of the Auto Layout. For more complex changes, however, you might want to programmatically define your onscreen elements, giving you complete control over their size and placement. Finally, for a good balance in flexibility, you can create multiple different views and swap them as the device rotates.

By implementing rotation and size-aware applications, you give your users the ability to use their devices in the way that feels most comfortable to them.

Q&A

Q. Why don't many iPhone applications implement the upside-down portrait mode?

A. Although there is no problem implementing the upside-down portrait orientation using the approaches described in this hour, it isn't recommended. When the iPhone is upside-down, the Home button and sensors are not in the "normal" location. If a call comes in or the user needs to interact with the phone's controls, the user will need to rotate the phone 180 degrees—a somewhat complicated action to perform with one hand.

Q. How do I get the controls in application XYZ to behave using Auto Layout?

A. This is a difficult question with no clear answer because there may be dozens of ways of implementing constraints that have the desired effect. I have implemented Auto Layout constraints for 3.5" and 4" iPhone displays in all my sample projects. You may want to take a look at those to get an idea of how I solved resizing problems (landscape, however, is up to you!).

Workshop

Quiz

1. The iDevice interface can rotate through three different orientations. True or false?

2. How does an application communicate which rotation orientations it supports?

3. What was the purpose of the `kDeg2Rad` constant that we defined in the final exercise?

Answers

1. False. There are four primary interface orientations: landscape right, landscape left, portrait, and upside-down portrait.

2. As long as the the `supportedInterfaceOrientations` method is implemented in the view controller, the application identifies which of the four orientations it will operate in.

3. We defined the `kDeg2Rad` constant to give us an easy way of converting degrees to radians for the Core Graphics C function `CGAffineTransformMakeRotation()`.

Activities

1. Edit the Swapper example so that each view presents and processes user input. Keep in mind that because both views are handled by a single view controller, you must add all the outlets and actions for both views to the view controller interface and implementation files.

2. Return to an earlier lesson and revise the interface to support multiple different orientations and iPhone screen sizes. Use any of the techniques described in this hour's exercises for the implementation.

Index

A

A chips, 6

ABPeoplePickerNavigaion-Controller class, 675

Accelerate framework, 104

accelerometers

- described, 7, 600-601
- reading with Core Motion, 604-606
- updates to, 617-620

accessibility attributes, setting, 142-144

Accessibility Inspector, 145

accessing

- instance variables directly, 75
- orientation and motion data, 603-606
- properties inside classes, 75
- variable lists, 796

accessors, 73

Accounts framework, 102

accuracy of code, 183

action sheet buttons, 304-306

action sheets, 285-287, 303-306

actions

- buttons, toggling animations on and off with, 225-226
- creating and connecting in apps
  - BestFriend, 689-690
  - with custom pickers, 396-397
  - with the date picker, 381-383
  - detecting tilt and rotation, 614-616
  - flashlight, 502-504
  - FlowerWeb, 260-264
  - gesture apps, 583-586
  - Interface Builder, 151-153
  - looped animations, 234-237
  - MediaPlayground, 644-645
  - with modal segues, 351
  - with navigation controllers, 430-431
  - scrolling views, 274-275
  - Single View Application template, 179-182

story creators, 206-209
survey, 523-524
with tab bar controllers, 438-440
with user alerts, 293-295
defining, 164
defining in GenericViewController class, 428
described, 148-149
identifying, 175
triggering with UI elements, 193

active calls, simulating toggles to, 61

Add Contact button type, 204

addGestureRecognizer method, 573

Address Book framework, 100, 102, 677, 691

address books
BestFriend app
building, 702
creating and connecting outlets and actions, 689-690
designing the interface, 688-689
implementing logic in, 690-695
setting up the project, 686-688
integrating with apps, 674-677

addresses
email, populating with current, 353
web. *See* URLs

alert view buttons, 299-301

alert view fields, 301

alert view text fields, 302-303

alert views, 282-285

alertBody property, 745

alerts for users
action sheets, 285-287
alert views, 282-285
creating apps
designing the interface, 293-294
implementing action sheets, 303-306
implementing alert sounds and vibrations, 307-308
implementing alert views, 296-303
setting up the project, 291-292
System Sound Services, 288-290, 307
vibrations, 288, 290

alertViewStyle property, 283

aligning objects, 135-137, 539

alloc message, 83

allocating objects, 83-84

allowsEditing property, 636

allowsPickingMultipleItems property, 662

Amazon Mobile app, 7

analyzing apps, 46

anchors, setting, 380-381, 396

angle brackets, in protocols, 72

animated loops
designing the interface for
adding Hop buttons, 232
adding image views, 226
adding speed output labels, 232
building the app, 243
controlling speed with sliders, 228-231
copying image views, 227-228
creating and connecting outlets and actions, 234-237
implementing application logic, 237-243
incrementing speed of, 242-243
setting background images and colors, 232-233
setting default images, 226-227
setting the speed, 240-242
starting and stopping, 239-240
as frames, 227
setting up the project, 224-226

animation resources, adding to looped animations, 225

animationDuration property, 239

animationImages property, 239

annotations, maps, 684-686, 696-699

APIs (application programming interfaces), multitasking, 6

app developers. *See* iOS developers

app icons, setting, 53-54

App ID, 18

App Store, 2, 8

AppDelegate files, 170

Appearance trait, 198

appearances

- custom pickers, 369
- table views, 448-449
- user interfaces, customizing, 142-145

Apple Developer Program, joining, 8-12

Apple Developer Registration Center, 9

Apple TV, 4

application:didFinishLaunching WithOptions method, 741

Application category, 29

application data, storage of, 30, 497-498

application delegate class, 105, 173

application logic implementation in apps

- BestFriend app
  - address books, 690-695
  - email, 700-701
  - maps, 695-699
  - social networking, 701-702
- detecting tilt and rotation, 616-621
- flashlight apps, 504-506
- FlowerWeb app, 248-268
- gesture apps, 587-595
- location-aware apps, 719-723
- long-running background tasks, 756-760
- looped animations, 237-243
- with magnetic compass, 726-732
- with modal segues, 352-353
- with navigation controllers, 432-433
- orientation sensing apps, 608-610
- with popovers, 358-360
- scrolling views, 275-276
- Single Application View template, 183-184
- story creators, 216-217
- survey app, 525-528
- swapping views on rotation, 566-568
- with tab bar controllers, 440-443
- with table views, 463-469
- universal apps, 771, 774-775

application objects, 107

application programming interfaces (APIs), multitasking, 6

application resource constraints, iOS platform, 6

applicationDidBecomeActive method, 741

applicationDidEnterBackground method, 740, 741

applicationIconBadgeNumber property, 745

applications. *See* apps

applicationWillEnterForeground method, 741, 743-744

applicationWillResignActive method, 741

applicationWillTerminate method, 740, 741

apps

- building
  - looped animations, 243
  - with Single Application View template, 184-185
  - story creators, 217
  - in Xcode, 46-50
- closing, 184
- compiling, 211
- creating
  - best practices in, 161-163
  - in Xcode, 168-169
- data storage in
  - approaches to, 491-500
  - limiting preferences for, 489-491
- detecting devices running on, 764
- generating multitouch events, 59
- identifiers for, 29
- integrating with other apps
  - address books, 674-677
  - BestFriend app, 686-702
  - email messages, 678-680
  - maps, 682-686
  - social networking sites, 680-682
- launching, drawing interfaces while, 561
- life cycle of, 105-106
- projects for, creating and managing, 28-36
- recovering from crashes of, 61
- running, 19-22, 39, 105, 211
- simulating
  - esoteric conditions, 60-61
  - rotations, 60

technology used in developing, 22-23
testing in iOS Simulator, launching, 57-59
tracing and debugging
feedback for, 782-783
with Xcode debugger, 784-799
troubleshooting, 105
tutorial, 18
Welcome, 22
See also projects; specific app names; user interfaces

ARC. *See* automatic reference counting

areas, functional, Xcode, 31-32

ARM processors, 6

arrays, described, 109-110

arrow directions, popovers, 335-336

assignments, with variables, 75-82

assistant editor mode, 42

asterisks, in variables, 82

attributed text fields, 196

attributes
bar button, 366
date picker, 368-369
editing
buttons, 204-205
text fields, 194-196
text views, 200-202
setting
accessibility, 142-144
navigation bar item, 414-415
prototype cell, 451-452
slider range, 229-231
stepper range, 231-232
tab bar item, 420-434
tables, 450-451
web view, 259
strong, 75-76
weak, 75-76

Attributes Inspector, 142, 204, 581

audio
adding feedback, 751
adding resources to apps, 291
adding to projects, 34-35, 631
implementing directions for, 750-753
implementing in apps, 307-308
implementing recording and playback, 649-655
playing, 289, 307-308, 639
preparing apps for, 748-749
recording keys, 635
supported file formats for, 628
System Sound Services, 288-290, 307, 632
variables and constants, adding to apps, 749

AudioToolbox framework, adding to apps, 291-292, 748

Auto-Enable Return Key trait, 198

Auto Layout system
adapting between varying sizes of screens, 534
constraints
centering, 544-546
content hugging and content compression resistance, 543
described, 538-554
expanding controls, 546-551
navigating Constraints objects, 539-540
Pin menu, 554
variable and matching sizes, 551-554
viewing and editing in Size Inspector, 541-543
described, 139-141, 538
designing rotatable and resizable interfaces, 536

auto layouts, 575

autocompletion of code, 38-40

automatic reference counting (ARC), 92-93, 340, 639

autosizing storyboards, 141

AV audio player, 633-634

AV audio recorder, 634-635

AV Foundation framework, 101, 632

AVAudioPlayer class, 633

AVAudioPlayerDelegate protocol, 634

AVAudioRecorder class, 633, 635

AVEncoderAudioQualityKey, 635

AVFormatIDKey, 635

AVNumberofChannelsKey, 635

AVSampleRateKey, 635

B

back button attribute, 415

background-aware apps

adding background modes key, 753-754

backgrounding

disabling, 742

types of, 739

completing long-running background tasks, 739, 754-760

handling background suspension, 743-744

implementing directions for audio in, 750-753

life cycle methods, 739-742

task-specific background processing, 739, 747-753

background image resources, adding to location-aware apps, 715

background modes key, adding to apps, 753-754

background processing, task-specific, 739, 747-753

background suspension, 743-744

background tasks, long-running, 739, 754-760

background touch, hiding keyboards with, 214-215

backgrounds

buttons, customizing, 206

changing colors of, 201

hiding keyboards by touching, 214-215

setting for looped animations, 232-233

badges, tab bar item, 441-443

bar button attributes, 366

bar button items, 365, 412

bar button titles, setting, 485-486

barItem outlet, 438

bars

navigation, 412

tab, 418

Basic table style, 452

battery life, 712

beginGeneratingDeviceOrientationNotifications method, 603

BestFriend app

building, 702

creating and connecting outlets and actions, 689-690

designing the interface, 688-689

implementing logic in

address books, 690-695

email, 700-701

maps, 695-699

social networking, 701-702

setting up the project, 686-688

beta versions

access to, 8

Xcode, 13

blocks

functional building. *See* classes

in methods, 86-87

Bluetooth, 7

brackets, in protocols, 72

breakpoint navigator, 797

breakpoints, setting, 786-796

build schemes, choosing, 46

building apps

BestFriend, 702

with custom pickers, 406

with the date picker, 390

detecting tilt and rotation, 621-622

flashlight, 506

FlowerWeb, 268

with gestures, 595

location-aware, 723-724

long-running background tasks, 739, 754-760

looped animations, 243

with magnetic compass, 733

with Master-Detail Application templates, 486

with navigation controllers, 433

with orientation sensing, 610

with popovers, 360

programmatically defined interfaces, 562

scrolling views, 276

Settings, 520

with Single View Application templates, 184-185

story creators, 217

swapping views on rotation, 569

with tab bar controllers, 443

with table views, 469

universal, 771, 775-776

in Xcode, 46-50

bundle identifiers, 29

bundles, settings, 494-496, 511-517

button attributes, editing, 204-205

button templates, implementing, 209-212
button touches, 561-562
button types, 204-205
buttons
  action sheet, 304-306
  alert view, 299-301
  bar, 365, 412, 485-486
  Clear, 195
  constraints of, 543
  Done, hiding keyboards with, 213-214
  editing attributes of, 204-205
  editing text in, 415
  with gradients, 190
  Hop, adding to looped animations, 232
  implementing templates for, 209-212
  purpose of, 113, 189-190
  setting custom images for, 205-206
  styled, adding to user interfaces, 203-204
  tab bar item, as part of scenes, 420
  toggling animations on and off with outlets, 225-226
  X, 145
  Xcode, 31-32
buttonTitleAtIndex method, 306

C

calculations
  dates, 386-390
  mathematical operations, 82
calls, active, simulating toggles to, 61
cameras, implementing in apps, 656-659
cancelButtonTitle parameter, 283, 286
canPerformUnwindSegueAction method, 327
Capitalize trait, 197, 199
case sensitivity, Objective-C, 70
categories, 68
  Application, 29
  creating, 80
  described, 79
  empty, 78
  overriding methods with, 80
cell phones, locating devices with, 44
cells
  disabling editing of, 483
  displaying in table views, 466
  table, 448, 451-452
cellular providers, connecting iPhones and iPads with, 7
center alignment constraints, 141
centering, constraints, 544-546
central processing units (CPUs), 6
CFNetwork framework, 102
CGRectMake function, values in, 49
changing. *See* editing
characters
  hiding while typing, 198
  underscore, 72
chooseImage method, 656-657
choosing
  build schemes, 46
  contacts, address books, 675-677, 691-695
  segue styles, 321, 323
  templates for projects, 28-29
  See also pickers
chosenColor outlet, 149
circular references, 75
@class directive, 319
class files, 170-171
class methods, 68, 76
class prefixes, 30, 171
classes
  ABPeoplePickerNavigation-Controller, 675
  accessing properties inside, 75
  application delegate, 105, 173
  AVAudioPlayer, 633
  AVAudioRecorder, 633, 635
  CMMotionManager, 605
  colons in, 72
  core application, 107-108
  creating to share, 78
  data type, 109-112
  date chooser view controller, 376
  described, 23, 67
  EditorViewController, 331
  generic view controller, 424, 435

GenericViewController, 428
instance variables, 68
interface, 113-116
location of code for, Xcode, 33
master, 69
Media Player framework, 626
MPMediaItem, 626, 632
MPMediaItemCollection, 626
MPMediaPickerController, 626, 629-631, 635-636, 662-668
MPMoviePlayerController, 626-629
MPMusicPlayerController, 626, 631
MSURL, 112
names of, sharing with file names, 73
navigation controller, 424
NSArray, 109-110
NSDate, 112
NSDictionary, 110, 634-635
NSMutableArray, 109-110
NSMutableDictionary, 110
NSMutableString, 109
NSNotificationCenter, 628
NSNumber, 111
NSObject, 65, 107
NSOperationQueue, 605
NSString, 109
NSURL, 250-251
NSURLRequest, 250-251
NSUserDefault class, 493
prewritten, 69
requestWithURL, 250-251
root, 107
SLComposeViewController, 680-681
tab bar controller, 435
UIActionSheet, 285-287
UIApplication, 107, 170
UIBarButtonItem, 412
UIButton, 82, 113, 189
UIControl, 108
UIDatePicker, 115, 368-369
UIDevice, 603
UIImagePickerController, 635-638
UIImageView, 223, 239, 576, 660
UILabel, 113, 165, 174, 191, 576
UINavigationBar, 412
UINavigationController, 411-416, 424
UINavigationItem, 412
UIPicker, 115
UIPickerView, 369-374
UIPopoverController, 115-116, 340-341
UIResponder, 108, 171
UIResponderStandardEdit-Actions, 199
UIScrollView, 251
UISegmentedControl, 114
UISlider, 114, 221
UIStepper, 114, 222
UISwitch, 113
UITabBar, 418
UITabBarController, 435
UITabBarItem, 418, 420-421
UITapGestureRecognizer, 573
UITextField, 115, 190
UITextView, 115, 190
UIView, 108, 130, 171
UIViewController, 108, 316, 331, 424, 775
UIWindow, 107
view controller, 108, 171-173
ViewController, 108, 171-173, 331
See also instances; objects

Clear button, 195

clickedButtonAtIndex method, 306

CLLocationManagerDelegate protocol, 708-710

closing
- apps, 184
- Xcode, 14

CMMotionManager class, 605

Cocoa framework, 99

Cocoa Touch
- classes in
  - core application, 107-108
  - data type, 109-112
  - interface, 113-116
- described, 23, 97-99
- roles of in app life cycle, 105-106
- UIKit framework as component of, 105

Cocoa Touch layer, 100-101

code
- accuracy of, 183
- adding new files to a project, 34
- angle brackets in, 72
- asterisks in, 82
- colons in, 72, 76, 85

connecting user interfaces to, 146-156, 164
editing and navigating in Xcode
activating tabbed editing, 44
adding pragma marks, 42
assistant editor mode, 43-42
with code completion, 38-40
finding methods and properties, 37-38
finding with search navigator, 41
managing snapshots, 44-46
overview, 36-37
keyboard-hiding, adding to user interfaces, 215-216
plus and minus signs in, 76
semicolons in, 72
spaghetti, 162
stepping through, 786-796
underscore characters in, 72, 74
when and when not to, 410

code classes, location of in Xcode, 33

code completion, 38-40

code libraries, location of in Xcode, 33

Code Snippet library, 133

codecs supported by Apple, 628

colons
in classes, 72
in methods, 76, 85

colorChoice outlet, 149

colors
background
changing, 201
setting for looped animations, 232-233
setting for status bars, 53

company identifiers, 29-30

compasses
digital, 7
magnetic, *See* magnetic compass

compiling apps, 211

completing long-running tasks, 739, 754-760

components, changing sizes of, 402-403

compose view, 60-61

Compression Resistance setting, 139, 141

condition-based loops, 90-91

configureView method, 485

Connection Inspector, 151
viewing connections in, 586

connections
creating in apps
in apps detecting tilt and rotation, 614-616
in apps with custom pickers, 396-397
in apps with magnetic compass, 726
in apps with Master-Detail Application templates, 474-476
in apps with modal segues, 351
in apps with navigation controllers, 430-431
in apps with orientation sensing, 607-608
in apps with popovers, 358
in apps with tab bar controllers, 438-440
in apps with the date picker, 381-383
in apps with user alerts, 293-295
in BestFriend app, 689-690
to exit and unwind segues, 325
in flashlight app, 502-504
in flower apps, 260-264
in gesture apps, 583-586
in Interface Builder, 151-153
in location-aware apps, 719
in long-running background tasks, 755
in looped animations, 234-237
MediaPlayground app, 644-645
in scrolling views, 274-275
Settings app, 511
in Single Application View template, 179-182
in story creators, 206-209
in survey app, 523-524
universal apps, 770, 774-775
planning in apps
in apps with gestures, 576
BestFriend app, 687-688

with the date picker, 377
detecting tilt and rotation, 612-613
flashlight app, 501
location-aware apps, 715-716
long-running background tasks, 754
looped animations, 225-226
with Master-Detail Application templates, 472
MediaPlayground app, 642
modal segues, 346
with navigation controllers, 427-428
with popovers, 356
programmatically defined interfaces, 556
scrolling views, 270
Settings app, 509
Single View Application templates, 173-175
story creators, 193
swapping views on rotation, 563
with tab bar controllers, 436
with table views, 461
universal apps, 769
user alert apps, 292
from user interfaces to code, 146-156, 164
viewing in Connection Inspector, 586
views through segues, 429

connectivity, iOS devices, 7

constants

audio, adding to apps, 749
custom picker component, 393
degree conversion, 725
for displaying popovers, 342
key
adding to flashlight app, 501
adding to Settings app, 509
location, adding to location-aware apps, 716
radian conversion, 725
radians, adding degrees to, 563
table section, adding to apps with table views, 461

Constraint setting, 139

constraints

centering, 544-546
content hugging and content compression resistance, 543
described, 538-539
expanding controls, 546-551
Pin menu, 554
variable and matching sizes, 551-554
viewing and editing in Size Inspector, 541-543

Constraints objects, 140-141, 539-540

content, remote, loading, 250-251

content compression resistance, 543

content hugging, 543

Content Hugging setting, 139, 141

content types, web views, 250

contentViewController property, 358, 384

controlHardware method, 617-618

controllers

detail view, 484-485
master view, 480-484
navigation
creating apps with, 423-433
described, 411-416
split view, 456-459, 471-472
tab bar, 417-422, 433-443
view. *See* view controllers

controls

expanding, 546-551
onscreen, 108
segmented
adding to apps, 254
described, 114, 248-249
using in flower apps, 254, 255-256

convenience methods, 83-84

copy and paste process, customizing, 199

copying, image views, 227-228

core application classes, 107-108

Core Audio framework, 101

Core Bluetooth framework, 104

core data, 30, 167

Core Data layer, 102

Core Foundation framework, 103

Core Graphics framework, 101

Core Image filters, 639, 659-662

Core Image framework, 101, 638-639

Core Location framework, 103, 695-696, 707-716

Core Motion framework, 103, 612-613

Core Motion motion manager

initializing, 616-617

reading accelerometer and gyroscope with, 604-606

Core OS layer, 104

Core Services layer, 102-104

Core Text framework, 101

CoreGraphics framework, 100

correcting errors and warnings, 48-50

Correction trait, 197, 199

costs, Apple Developer Program, 8

count-based loops, 89

counters

incrementing and displaying, 432-433, 441

initializing, 757

updating, 758

viewing output of, 783-784

CountingNavigationController subclass, 424

cover vertical option, transition types, 323

CPUs (central processing units), 6

crashes in iOS Simulator, recovering from, 61

createStory method, 193, 216-217

cross dissolve option, transition types, 323

curls, page, 323

current context presentation style, 323

Custom button type, 204

custom images, setting for buttons, 205-206

custom option, segues, 321

custom pickers

appearance of, 369

creating apps with

building the app, 406

creating and connecting outlets and actions, 396-397

creating segues, 396

designing the interface, 394-395

implementing custom picker view, 399-405

setting up the project, 392-393

D

data

application, storage of, 30, 497-498

core, 30, 167

orientation and motion, accessing, 603-606

passing between scenes, 329-331

picker, loading, 399-400

reading and writing, 499-500

reusing in other applications, 162

sharing between scenes, 416, 422

data detectors, 201-202

data models, 166-167

data storage

approaches to

direct file system access, 496-500

settings bundles, 494-496

user defaults, 493-494

implementing

for file systems, 520-528

system settings, 507-520

limiting preferences for, 489-491

data type classes, 109-112

data types

float, 111

int, 111

object, 82-83

primitive, 81-82

Date & Time mode, 369

date calculation logic, implementing, 386-390

date chooser scene, 380, 384-386

date chooser view controller class, 376

date format string, 387

date method, 112

Date mode, 369

date picker

creating apps with

building the app, 390

creating and connecting outlets and actions, 381-383
creating segues, 379-381
designing the interface, 378-380
implementing date calculation logic, 386-390
implementing scene segue logic, 383-386
setting up the project, 376-377
described, 368-369

date picker attributes, 368-369

dates
determining differences between, 387-390
viewing time and, 386-387
working with, 112

debug area, 31-32

debug navigators, 798-799

debugging apps
feedback from, 782-783
with Xcode debugger, 784-799

defaults
app displays in iOS Simulator, 59
images, setting for looped animations, 226-227
iPhone screen size, 131
resetting iOS Simulator to, 58
sound, preparing audio players with, 653
status bar display, 53
user, data storage, 493-494

degree conversion constants, 725

degrees, adding to radians constant, 563

delegate object, 106

delegate parameter, 283, 286

delegate property, 398

delegates
creating and setting, 383-384
location manager, 708-711, 720-723
mail compose view controller, 680
media picker controller, 630-631
navigation controller, 638
people picker, 675-677
UI image picker controller, 637-638

deleting. *See* removing

designing
apps with custom pickers
building the app, 406
creating and connecting outlets and actions, 396-397
creating segues, 396
designing the interface, 394-395
implementing custom picker view, 399-405
setting up the project, 392-393
apps with gestures
adding gesture recognizers, 579-583
building the app, 595
creating and connecting outlets and actions, 583-586
creating the interface, 577-579
implementing application logic, 587-595
setting up the project, 576-579
apps with magnetic compass
building, 733
creating and connecting outlets, 726
setting up the project, 724-725
updating application logic, 726-732
updating the user interface, 725
apps with Master-Detail Application templates
building the app, 486
implementing application data source, 476-480
implementing master view controller, 480-484
setting up the project, 470-472
tweaking the iPad interface, 472-475
tweaking the iPhone interface, 474-476
apps with modal segues
building the app, 354
creating and connecting outlets and actions, 351
creating the interface, 347-349
creating the modal segue for, 349
implementing application logic, 352-353
setting up the project, 343-347

unwinding back to initial scene, 350-351
apps with navigation controllers, 423-433
apps with orientation sensing, 606-610
apps with popovers, 355-360
apps with tab bar controllers, 433-443
apps with table views
building the app, 469
designing the interface, 461-463
implementing application logic, 463-469
setting up the project, 460-461
apps with the date picker
building the app, 390
creating and connecting outlets and actions, 381-383
creating segues, 379-381
creating the interface, 378-380
implementing date calculation logic, 386-390
implementing scene segue logic, 383-386
setting up the project, 376-377
apps with user alerts
creating and connecting outlets and actions, 293-295
creating the interface, 293
implementing action sheets, 303-306
implementing alert sounds and vibrations, 307-308
implementing alert views, 296-303
setting up the project, 291-292
BestFriend app
building, 702
creating and connecting outlets and actions, 689
creating the interface, 688-689
implementing logic in, 690-702
setting up the project, 686-688
flashlight app
building the app, 506
creating and connecting outlets and actions, 502-504
creating the interface, 501-502
implementing application logic, 504-506
setting up the project, 501
FlowerWeb app
adding and configuring segments, 254-255
adding switches, 257-258
adding web views, 259-260
building the app, 268
creating and connecting outlets and actions, 260-264
implementing application logic, 248-268
segmented controls, 254-256
setting up the project, 253
location-aware apps
building the app, 723-724
creating and connecting outputs, 719
creating the view, 716-718
implementing application logic, 719-723
setting up the project, 715-716
looped animations
adding Hop buttons, 232
adding image views, 226
adding speed output labels, 232
building the app, 243
changing speed with steppers, 231-232
controlling speed with sliders, 228-231
copying image views, 227-228
creating and connecting outlets and actions, 234-237
implementing application logic, 237-243
incrementing speed of, 242-243
setting background images and colors, 232-233
setting default images, 226-227
setting the speed, 240-242

setting up the project, 224-226
starting and stopping, 239-240
MediaPlayground app
accessing and playing the music library, 662-668
creating and connecting outlets and actions, 644-645
designing the interface, 642
implementing audio recording and playback, 649-655
implementing Core Image filter, 659-662
implementing movie player, 646-650
implementing photo library and camera, 656-659
setting up the project, 640-642
multiscene projects
controlling model segues manually, 324
creating segues, 320-323
exit and unwind segues in, 325-327
passing data between scenes, 329-331
preparing the project, 314-319
popovers
creating popover segues, 333-336
displaying manually, 337
preparation for, 333
programming popover displays, 340-342
responding to dismissals of, 337-339
scenes in tab bar controllers, 418-422
scrolling views, 269-277
Settings app
building the app, 520
creating and connecting outlets, 511
creating the interface, 509-510
creating the settings bundles, 494-496, 511-517
implementing application logic, 518-520
setting up the project, 507-509
with Single View Application templates, 176-178
story creators
adding styled buttons, 203-204
adding text fields, 193-194
adding text views, 199
building the app, 217
creating and connecting outlets and actions, 206-209
editing button attributes, 204-205
editing text field attributes, 194-196
editing text view attributes, 199-202
hiding keyboards, 212-216
implementing application logic, 216-217
implementing button templates, 209-212
setting custom button images, 205-206
setting scrolling options, 203
setting up the project, 192-193
survey app
creating and connecting outlets and actions, 523-524
designing the interface, 521-522
implementing application logic, 525-528
setting up the project, 521
destinationViewController property, 384
destructiveButtonTitle parameter, 286
Detail Disclosure button type, 204
detail scenes, updating, 474-476
detail view, 484-485
detail view controller, 484-485
detailItem method, 484-485
detectors, data, 201-202
developers. *See* iOS developers
development devices
configuring with Xcode, 16-19
registering, 17, 18
development preparation
iOS platform tour, 3-7
overview, 3
development provisioning profile, creating and installing, 15-19

device orientations, setting, 51-53
device preparation
configuring with Xcode, 16-19
iOS platform tour, 3-7
overview, 3
device support components, adding to Xcode, 14
dictionaries, described, 110
didRotateFromInterfaceOrientation method, 561
digital compasses, 7
direction image resources, 724
directives
@class, 319
IBAction, 164, 166
IBOutlet, 164-165
@implementation, 78
#import, 71-72, 77, 319
@interface, 72-73
@optional, 80-81
@property, 73-75, 174
@required, 80-81
directories
creating groups in, 35
Xcode, 32
See also subgroups
disabling
backgrounding, 742
Welcome to Xcode screen, 28
dismissal method, 337-339, 360
dismissAnimalChooser action, 397
dismissDateChooser action, 382
displaying. *See* viewing
displays
counters, 432-433, 441
default status bar, 53
iOS devices, 4-5
keyboard, customizing with text input traits, 197-199
maps, 696-699
navigator area, filtering information in, 33
Non-Retina
app icons for, 53
launch images for, 55-56
popovers, programming, 340-342
Retina
app icons for, 53
image views, 223
launch images for, 55-56
loading high-resolution images for, 228
updating, 758
web views, hiding and showing, 264-265
doAcceleration method, 619-620
doAlert method, 745
document outline objects, 130-131
document outlines, storyboards, 127-131
documentation
Cocoa Touch, 98
using tables in apps, 456
Xcode, 117-119
domain names, using identifiers for, 29
Done button, hiding keyboards with, 213-214
doRotation method, 620-621
doSound method, 307
dot notation, 74
downloading, Xcode, 12
dragging and dropping
images in Xcode image wells, 767
objects, 579
duplicating. *See* copying

E

earlierDate method, 112
editing
attributes
text fields, 194-196
text views, 200-202
cells, disabling, 482
code in Xcode
activating tabbed editing, 44
adding pragma marks, 42
assistant editor mode, 43-42
with code completion, 38-40
finding methods and properties, 37-38
finding with search navigator, 41
managing snapshots, 44-46
overview, 36-37
connections with Quick Inspector, 153

constraints in Size Inspector, 541-543
text in buttons, 415
titles in navigation bars, 414-415
user interfaces
disabling, 482
Interface Builder, 135-139
variable states, 791
viewDidLoad method, 786-787

editor area, 31

editor scene field, setting initial scene label to, 353

EditorViewController class, 331

email
BestFriend app
building, 702
creating and connecting outlets and actions, 689
designing the interface, 688-689
implementing logic in, 700-701
setting up the project, 686-688
integrating with apps, 678-680

email addresses, populating fields with current, 353

Empty Application template, 29

empty categories, 78

empty files, adding to projects manually, 34

empty selections, media picker, 666

enterprise developer program, 8

errors
correcting, 48-50
location, 710-711
in older device versions, 741

esoteric conditions, simulating, 60-61

Event Kit framework, 103

Event Kit UI framework, 100

events
detecting outside fields, 214
interface rotation, 603
low-memory, simulating, 61
multitouch, generating in iOS Simulator, 59
Touch Up Inside, 153, 189

Exit icon, 129

exit segues, 324-325, 353

exiting
apps, 184
Xcode, 14

exits
described, 313
preparing view controllers for, 325, 347

expressions
described, 87-88
making decisions with, 88-89

extensions for files, xcodeproj, 31

External Accessory framework, 104

F

Facebook, integrating with other apps, 680-682

feedback
app tracing and debugging, 782-783
audio, 751
iOS devices, 7

fees, Apple Developer Program, 8

fields
alert views, 301
detecting events outside, 214
editor scene, setting initial scene label to, 353
populating with current email addresses, 353
text
adding to user interfaces, 193-194
alert view, 302-303
attributed, 196
editing attributes of, 194-196
plain, 196
purpose of, 115, 190
text entry, alertViewStyle property, 283
treating as passwords, 198

file extensions, xcodeproj, 31

file paths, data storage, 498-499

file system storage, implementing, 520-528

File Template library, 133

files
AppDelegate, 170
class, 170-171

empty, adding to projects manually, 34
finding strings in, 41
icon, universal apps, 766-767
implementation, 77-79
interface
described, 71
ending, 76, 79
importing, 397
logical groupings in Xcode folders, 32, 33
MainStoryboard, 172-173
media, adding to projects, 641-642
names of, sharing with class names, 73
new code, adding to projects, 34
PNG, as app icons, 54
removing from projects, 35-36
stored data, direct access to, 496-500
storyboard, 171-173
structure in Objective-C
automatic setup of, 79
categories and protocols, 79-81
defining methods, 75-76
implementation files, 77-79
#import directive, 71-72, 77
interface (header) files, 71
@interface directive and instance variables, 72-73
@property directive and instance variables, 73-75
supported formats for rich media, 628
ViewController, 170, 174-175, 237
See also resources

filtering
information displayed in project navigator, 33
search criteria in Xcode libraries, 119
search results, 41

filters
Core Image, 639, 659-662
location accuracy and update, 712
media picker, 630

finding
code with search navigator, 41
files in Xcode folders, 32
information in Xcode library, 118-119
methods and properties in symbol navigator, 37-38
sounds, 289
strings in files, 41

fireDate property, 745

first responder, 593-594

First Responder icon, 129

flashlight app
building, 506
creating and connecting outlets and actions, 502-504
designing the interface, 501-502
implementing application logic, 504-506
setting up the project, 501

flip horizontal option, transition types, 323

float data type, 111

floating-point numbers
float data type, 111
initializing strings with, 109

flowerView outlet, 149

FlowerWeb app
designing the interface for
adding and configuring segments, 254-255
adding switches, 257-258
adding web views, 259-260
building the app, 268
creating and connecting outlets and actions, 260-264
implementing application logic, 248-268
segmented controls, 254-256
setting up the project, 253

folders
creating groups in, 35
Xcode, 32
See also subgroups

for loops, 90

form sheet presentation style, 323

Foundation framework, 100

foundPinch method, 584, 589-590

foundRotation method, 584, 590-593

foundSwipe method, 583, 588

foundTap method, 573, 583, 588

Frame Rectangle setting, 139

frames

- animations as, 227
- changing, 537

frameworks

- Address Book, 100, 102, 677, 691
- Address Book UI, 674-675
- AudioToolbox, adding to apps, 291-292
- AV Foundation, 632-635
- Cocoa, 99
- Cocoa Touch layer, 100-101
- Core Foundation, 103
- Core Graphics, 101
- Core Image, 101, 638-639
- Core Location, 103, 695-696, 707-714, 715-716
- Core Motion, 103, 612-613
- Core OS layer, 104
- Core Services layer, 102-104
- exploring with Xcode, 116-122
- Map Kit. *See* Map Kit framework
- Media layer, 101-102
- Media Player, 102, 626-632, 646
- Message UI, 100, 700
- in templates, 100

Frameworks subgroup, 33

free developer accounts, testing applications with, 15

full screen presentation style, 323

functional areas, Xcode, 31-32

functional building blocks. *See* classes

functions

- CGRectMake, values in, 49
- main, 105
- NSLog, 782-783, 788
- UIApplicationMain, 105

G

Game Kit framework, 100

generic view controller classes, 424, 435

GenericViewController class, 428

gesture recognizers

- adding to projects, 572-573
- adding to views, 579-583
- responding to, 588-593

gestures

- creating apps with
  - adding gesture recognizers, 579-583
  - building the app, 595
  - creating and connecting outlets and actions, 583-586
  - designing the interface, 577-579
  - implementing application logic, 587-595
  - setting up the project, 576-579
- simulating, 61, 571

getFlower action, 149

getters (accessors), 73

global positioning systems (GPSs), 7, 712

goBack action, 153

goForward action, 153

GPSs (global positioning systems), 7, 712

GPUs (graphics processing units), 6

gradients, buttons with, 190

graphics. *See* images

graphics processing units (GPUs), 6

groups

- creating, 33, 35
- logical, files, 32-33
- Resources, 33
- Supporting Files, 33

guides, 135-136

gyroAvailable property, 605

gyroscopes

- described, 7, 599-602
- reading with Core Motion, 604-606
- updates to, 617-618, 620-621

H

handler blocks. *See* blocks

handles, selection, 136-137

hardware

- motion and orientation detection
  - accelerometers, 7, 600-601
  - gyroscopes, 7, 599, 601-602
- simulating keyboards for, 61

header files. *See* interface files
headings
- magnetic compass, 712-714, 727-732
- table view sections, 466

height value, 276
help, Quick Help, 119-122
hideKeyboard method, 214
hiding
- characters while typing, 198
- functional areas, Xcode, 32
- keyboards, 212-216
- web views, 264-265
- Welcome to Xcode screen, 28

Hint attribute, 144
Hop button, adding to looped animations, 232
horizontal constraints, 141
horizontalAccuracy property, 708
host method, 112
hugging, content, 543

I

iAd framework, 101
IBAction directive, 164, 166
IBOutlet directive, 164-165
IBOutlet keyword, 74, 174
icons
- storyboards, 128-129, 131
- universal apps, 766-767

icons. *See* images
id type, 76, 166
IDE (integrated development environment), 27
identifiers. *See* IDs
Identity Inspector, 142-144, 155
IDs
- bundle, 29
- company, 29-30
- developer registration, 9
- restoration, 492
- segues, 322
- view controllers, setting, 326-328

if-then-else statement, 88-89
Image I/O framework, 101
image picker, 635-638, 656-659
image resources
- adding to gestures, 576
- adding to Settings app, 509
- background, 715
- direction, 724
- Master-Detail Application templates, 471
- table views, 461

image views
- adding instance variables to sizes of, 577
- copying, 227-228
- described, 223
- looped animations, 226, 238-239
- removing constraints from, 575
- replacing, 587

image wells, dragging and dropping images into, 767
images
- adding to projects, 34-35
- dragging and dropping into Xcode image wells, 767
- high-resolution, loading for Retina displays, 228
- iOS devices, 4-5
- launch, universal apps, 766-767
- loading and displaying, 265-267
- setting
  - application icons, 53-54
  - for buttons, 205-206
  - launch images, 55-56
  - for looped animations, 226-227, 232-233
- tab bar, sizes of, 421
- tab bar item, 435
- *See also* pictures

imageView property, 576
imperative programming, 66-67
implementation files, 77-79
@implementation directive, 78
implicit preferences, flashlight app, 500-507
#import directive, 71-72, 77, 319
importing, interface files, 383, 397
incrementCount method, 432
incrementCountFirst action, 439
incrementCountSecond action, 439
incrementCountThird action, 439
incrementing
- counters, 432-433, 440-441
- integers, 90
- speed, looped animations, 242-243
- tab bar item badges, 441-443

Info Dark button type, 204

Info Light button type, 204
inheritance, 67
initial scene label, setting to editor scene field, 353
initializing
  Core Motion motion manager, 616-617
  interface properties, 558-559
  interfaces, when apps load, 561
  movie players, 646-647
  objects, 83-84
  strings with values, 109
  timers and counters, 757
initWithMediaTypes method, 662
initWithTitle parameter, 283, 286
input, iOS devices, 7
inspecting objects, 141
inspectors
  Accessibility, 145
  Attributes, 142, 204
  Connection, 151, 586
  Identity, 142-144, 155-
  Quick Help, 120
  Size, 138-139, 541-543
installing
  apps in iOS Simulator, 58
  development provisioning profile, 15-19
  Xcode, 12-13
  Xcode documentation updates, 119
instance methods, 68, 76
instance variables
  accessing directly, 75
  adding to image view sizes, 577
  declaring with properties, 74
  defining names of, 78
  described, 68
  @interface, 72-73
  methods, 73, 76
  private, 556
  properties compared with, 427
  @property directive, 73-75
  semicolons in, 72
  using instead of properties, 75
  using properties instead of, 377
instances
  described, 68
  location manager, 720-734
  tapRecognizer, 573
  See also classes; objects
instantiation, 68, 127
int data type, 111
integers
  incrementing, 90
  initializing strings with, 109
integrated development environment (IDE), 27
Interface Builder, 125
  creating user interfaces in
    adding objects to views, 134-135
    Auto Layout system, 139-141
    connecting to code, 146-156
    editing tools, 135-139
    Object Library, 131-133
  customizing appearances of user interfaces, 142-145
  described, 125-126
  outlets and actions, 148-153
  storyboards, 127-131
  zooming in and out in, 135
interface classes, 113-116
@interface directive, 72-73
interface files
  described, 71
  ending, 76, 79
  importing, 383, 397
interface properties, initializing, 558-559
interface rotation events, 603
interface update method, 559-560
interfaces. *See* user interfaces
iOS 6, launch images for, 55
iOS apps. *See* apps
iOS developers
  creating and installing development provisioning profiles, 15-19
  development possibilities for, 3
  equipment needed by, 8
  joining Apple Developer Program, 8-12
  qualifications for1-2
  technology used to create apps by, 22-23
iOS device preparation
  iOS platform tour, 3-7
  overview, 3
iOS platform
  application resource constraints, 6

connectivity of, 7
input and feedback, 7
life cycle, 105-106
overview, 3-4
screen resolution and graphical display, 4-5

iOS Simulator
Accessibility Inspector, 145
alternate names for, 46
as an Apple developer tool, 22
generating multitouch events in, 59
launching applications in, 57-59
limitations of, 8, 56
recovering from crashes in, 61
removing installed apps from, 58
resetting to defaults, 58
running first app in, 19-22
simulations with
accelerometers and gyroscopes, 600
esoteric conditions, 60-61
gestures, 571
interface rotations, 60
user interfaces, 145
testing applications with, 8, 15
as time-saver in development, 56

iOS technology layers
Cocoa Touch, 100-101
Core OS, 104
Core Services, 102-104
Media, 101-102

iPad app icons, 53
iPad Simulator. *See* iOS Simulator
iPad to iPhone targets, 777-778
iPad view, associating iPadViewController class with, 772-773
iPad view controllers, setting up in universal apps, 772-773
iPads
chips used in, 6
connectivity of, 7
developing projects on, 4
enforcement of popovers on, 638
input and feedback on, 7
resolution of, 4
split view controller, 456-459
tweaking interfaces of, 472-475
user interface guidelines for, 534
See also popovers

iPadViewController subclass, 772-775
iPhone Simulator. *See* iOS Simulator
iPhone to iPad targets, 776-777
iPhones
app icons for, 53
application resource constraints, 6
chips used in, 6
connectivity of, 7
default screen size, 131
developing projects on, 4
input and feedback on, 7
iPhone 5, launch images for, 55
launch images for, 55
resolution of, 4, 5
sizes of designs, 199
tweaking interfaces of, 474-476

iPod touch, 4
iPodMusicPlayer method, 662
isAvailableForServiceType method, 681
issue navigator, correcting errors and warnings in, 48-50
items
bar button, 365, 412
navigation and bar buttons, 412
tab bar, 418

J

joining Apple Developer Program, 8-12

K

key constants
adding to flashlight app, 501
adding to Settings app, 509

Keyboard trait, 197
keyboards
customizing displays with text input traits, 197-199
hardware, simulating, 61
hiding, 212-216
iOS devices, 7

keychains, 18

keys

audio recording, 635

background modes, adding to apps, 753-754

Main storyboard file base name, 172

keywords, IBOutlet, 74, 165, 174

L

Label attribute, 144

labels

described, 37

initial scene, setting to editor scene field, 343

purpose of, 113, 191

speed output, adding to looped animations, 232

symbol, 38

languages for programming, Objective-C, 22

launch images

setting, 55-56

universal apps, 766-767

launching

apps

drawing interfaces when, 561

in iOS Simulator, 57-59

images, universal apps, 766-767

Xcode, 27-28

layers, technology. *See* iOS technology layers

laying out interfaces. *See* user interfaces (UIs), creating

Layout Rectangle setting, 139

Left Detail table style, 452

libraries

code, location of in Xcode, 33

music, accessing and playing, 662-668

Object, Interface Builder, 131-133

photo, implementing in apps, 656-659

Xcode, searching for information in, 118-119

life cycle, apps, 105-106

life cycle methods, background-aware apps, 739-742

location accuracy and update filter, 712

location constants, adding to location-aware apps, 716

location manager delegate, 708-711, 720-723

location manager instance, 720-734

location services

Core Location framework, 103, 695-696, 707-714

creating apps with magnetic compass

building, 733

creating and connecting outlets, 726

setting up the project, 724-725

updating application logic, 726-732

updating the user interface, 725

creating location-aware apps

building the app, 723-724

creating and connecting outputs, 719

designing the view, 716-718

implementing application logic, 719-723

setting up the project, 715-716

locked devices, simulating, 61

logic. *See* application logic implementation

logical groupings, files, 32, 33

long pressing gesture, 572

long-running tasks, completing, 739, 754-760

looped animations

designing the interface for

adding Hop buttons, 232

adding image views, 226

adding speed output labels, 232

building the app, 243

changing speed with steppers, 231-232

controlling speed with sliders, 228-231

copying image views, 227-228

creating and connecting outlets and actions, 234-237

implementing application logic, 237-243

incrementing speed of, 242-243

setting background images and colors, 232-233
setting default images, 226-227
setting the speed, 240-242
starting and stopping, 239-240
setting up the project, 224-226

loops
condition-based, 90-91
count-based, 89
for, 90
repetition with, 89-91

M

machines for development, configuring with Xcode, 16-19

magnetic compass
creating apps with
building, 733
creating and connecting outlets, 726
recent location, 728
setting up the project, 724-725
updating application logic, 726-732
updating the user interface, 725
retrieving headings with, 712-714

magnification, adjusting in Interface Builder, 135

mail compose view, 700-701

mail compose view controller delegate, 680

main function, 105

Main storyboard file base name key, 172

MainStoryboard file, 172-173

Map Kit framework
BestFriend app
creating and connecting outlets and actions, 689-690
designing the interface, 688-689
implementing logic in, 695-699
setting up the project, 686-688
described, 100
integrating with other apps, 682-686

maps
annotations in, 684-686, 696-699
BestFriend app
building, 702
creating and connecting outlets and actions, 689-690
designing the interface, 688-689
setting up the project, 686-688
controlling display, 696
integrating with other apps, 682-686

master class, 69

Master-Detail Application template
creating apps for, tweaking the iPad interface, 472-475
creating apps with
building the app, 486
implementing application data source, 476-480
implementing master view controller, 480-484
setting up the project, 470-472
tweaking the iPhone interface, 474-476
described, 459

master scenes, updating, 472-475

master view controller, 480-484

matching sizes, 551-554

mathematical operations, with variables, 82

media files, adding to projects, 641-642

Media layer, 101-102

Media library, 133

media picker, 629-631, 662-668

media picker controller delegate, 630-631

Media Player framework, 102, 626-632, 646

MediaPlayground app
accessing and playing the music library, 662-668
creating and connecting outlets and actions, 644-645
designing the interface, 642
implementing audio recording and playback, 649-655

- implementing Core Image filter, 659-662
- implementing movie player, 646-650
- implementing photo library and camera, 656-659
- setting up the project, 640-642

membership, Apple Developer Program, 8-12

memory
- cleaning up objects in, 75
- iPhones, 6
- simulating low-memory event, 61

menus
- Pin, 554
- Xcode, 31-32

message parameter, 283

Message UI framework, 100, 700

messages
- alloc, 83
- described, 68
- email. *See* email
- nested, 86
- retain and release, 67-68
- sending to objects, 85-86

methods
- addGestureRecognizer, 573
- adding instance variables to, 80
- application:didFinishLaunching WithOptions, 741
- applicationDidBecomeActive, 741
- applicationDidEnterBack-ground, 740-741
- applicationWillEnterFore-ground, 741, 743-744
- applicationWillResignActive, 741
- applicationWillTerminate, 740, 741
- beginGeneratingDevice-OrientationNotifications, 603
- blocks in, 86-87
- buttonTitleAtIndex, 306
- canPerformUnwindSegue-Action, 327
- chooseImage, 656, 657
- class, 68
- class (+), 76
- clickedButtonAtIndex, 306
- colons in, 76, 85
- configureView, 485
- controlHardware, 617-618
- convenience, 83-84
- createStory, 193, 216-217
- date, 112
- declaring variables in, 81-83
- defining
  - Objective-C, 75-76
  - programmatically defined interfaces, 557-558
- detailItem, 484-485
- didRotateFromInterface Orientation, 561
- dismissal, 337-339, 360
- doAcceleration, 619-620
- doAlert, 745
- doRotation, 620-621
- doSound, 307
- earlierDate, 112
- finding in symbol navigator, 37-38
- foundPinch, 584, 589-590
- foundRotation, 584, 590-593
- foundSwipe, 583, 588
- foundTap, 573, 583, 588
- hideKeyboard, 214
- host, 112
- implementation of, 78-79
- incrementCount, 432
- initWithMediaTypes, 662
- instance, 68, 76
- interface update, 559-560
- iPodMusicPlayer, 662
- isAvailableForServiceType, 681
- life cycle, background-aware apps, 739-742
- messages and, 68
- newBFF, 691
- numberOfComponentsInPicker-View, 369
- numberOfSectionsInTableView, 453
- objectForKey, 110
- overriding with categories, 80
- pause, 662
- pickerView:didSelectRow: in-Component, 371-372
- pickerView:numberOfRowsIn-Component, 369
- pickerView:rowHeightFor-Component, 373
- pickerview:titleForRow: forComponent, 371, 373
- pickerView:widthFor-Component, 373

play, 662
playAudio, 654
playMusic, 667
prepareForSegue:sender, 330-331
private, 77-78
protocols and, 68
providing parameters for, 40
recordAudio, 652, 655
removeConstraints NSLayoutConstraint, 575
replacing objects with nil value in, 85
required, 81
resignFirstResponder, 212
self.presentingViewController, 331
semicolons in, 72
setIncrement, 226
setLightSourceAlphaValue, 504-505
setOutput, 183-184
setQueueWithItemCollection, 662
setSpeed, 226
setStartBarHidden, 636
sharing with #import and @class directives, 319
showFromRect, 305
showResults, 527
stopAccelerometerUpdates, 605
stopGyroUpdates, 605, 618
storeSurvey, 525-526
stub, 153-155
tableView:cellForRowAtIndex-Path, 453
tableView:numberOfRowsIn-Section, 452
tableView:titleForHeaderIn-Section, 453
toggleButtonAnimation, 226
typecasting objects for, 84
UIPopoverController, 342
valueForProperty, 632
viewDidAppear, 276, 587
viewDidLoad, 209-210, 653, 663, 750, 786-787
viewWillAppear:animated, 433
void type, 76

MFMailComposeResult value, 680

MFMailComposeViewController-Delegate protocol, 680

minus sign, 76

modal option, segues, 320

modal segues
- controlling manually, 324
- creating apps with
  - building the app, 354
  - creating and connecting outlets and actions, 351
  - creating the modal segue for, 349
  - designing the interface, 347-349
  - implementing application logic, 352-353
  - setting up the project, 343-347
  - unwinding back to initial scene, 350-351
- dismissing, 385

modal views, described, 313

model user interface elements, 282

Model-View-Controller (MVC)
- application design pattern
  - avoiding spaghetti code in, 162
  - described, 161
  - Single View Application template, 167-184
  - structured design with, 162-163
- described, 23
- implementing in Xcode, 163-167

models, data, 166-167

modifying. *See* editing

motion
- accessing data for, 603-606
- hardware for detecting. *See* accelerometers; gyroscopes
- tilt and rotation, creating app detecting, 611-622

movie player, 627-629, 646-650

movies. *See* videos

MPMediaItem class, 626, 632

MPMediaItemAlbumTitle property, 632

MPMediaItemCollection class, 626

MPMediaItemPropertyArtist property, 632

MPMediaItemPropertyGenre property, 632

MPMediaItemPropertyLyrics property, 632

MPMediaPickerController class, 626, 629-631, 635-636, 662-668

MPMediaTypeAnyAudio value, 630

MPMediaTypeAudioBook value, 630

MPMediaTypeMusic value, 630

MPMediaTypePodcast value, 630

MPMoviePlayerController class, 626, 627-629

MPMusicPlaybackStatePaused value, 631

MPMusicPlaybackStatePlaying value, 631

MPMusicPlaybackStateStopped value, 631

MPMusicPlayerController class, 626, 631

MSURL class, 112

multibutton alerts, creating, 298-299

multiscene storyboard, passing data between scenes, 329-331

multiscene storyboards

- creating projects with
  - controlling modal segues manually, 324
  - creating segues, 320-323
  - preparing the project, 314-319
- exit and unwind segues in, 325-327
- overview, 312-313

multitasking, application programming interfaces (APIs) for, 6

multitasking support, checking older devices for, 739-741

multitouch events, generating in iOS Simulator, 59

multitouch gestures. *See* gesture recognizer

music libraries, accessing and playing, 662-668

music player, 631-632, 663-664

mutators, 73

MVC. *See* Model-View-Controller

myLabel property, 165

myLabel variable, 165

N

names

- domain, using identifiers for, 29
- files
  - app icons, 54
  - category, 80
  - sharing with class names, 73
- instance variables, defining, 78
- products and organizations, 29-30
- scenes
  - keeping unique, 129
  - in multiscreen projects, 316-319

navigating

- code in Xcode
  - activating tabbed editing, 44
  - adding pragma marks, 42
  - assistant editor mode, 43-42
  - with code completion, 38-40
  - finding methods and properties, 37-38
  - finding with search navigator, 41
  - managing snapshots, 44-46
  - overview, 36-37
- Constraints objects, 539-540
- projects in the project navigator, 32-33
- Xcode user interface, 31-32

navigation bar item attributes, setting, 414-415

navigation bars and items, 412

navigation controller classes, 424

navigation controller delegate, 638

navigation controllers

- creating apps with, 423-433
- described, 411-416

navigation scenes, 413-416

navigator area, 31-33

nested messaging, 86

newBFF method, 691

Newsstand framework, 103

NeXTSTEP platform, 99

nil value, 85, 110

Nintendo Wii, 599

Non-Retina displays

- app icons for, 53
- launch images for, 55

notation, dot, 74

notifications

- common properties for, 745
- creating and scheduling, 745-747
- orientation, 603

NSArray class, 109-110
NSDate class, 112
NSDecimalNumber subclass, 111
NSDictionary class, 110, 634-635
NSLog function, 782-783, 788
NSMutableArray class, 109-110
NSMutableDictionary class, 110
NSMutableString class, 109
NSNotificationCenter class, 628
NSNumber class, 111
NSObject class, 69, 107
NSOperationQueue class, 605
NSString class, 109
NSURL class, 250-251
NSURLRequest class, 250-251
NSUserDefaults class, 493
numberOfComponentsInPickerView method, 369
numberOfSectionsInTableView method, 453
numberOfTapsRequired property, 573
numberOfTouchesRequired property, 573
numbers. ***See*** **values**

O

object data types, 82-83
Object Library, 132-133
object-oriented programming (OOP), 65-69
objectForKey method, 110
Objective-C
- automatic reference counting in, 92-93
- case sensitivity of, 70
- described, 22, 65, 69-70
- file structure of
  - automatic setup of, 79
  - categories and protocols, 79-81
  - defining methods, 75-76
  - implementation files, 77-79
  - #import directive, 71-72, 77
  - interface (header) files, 71
  - @interface directive and instance variables, 72-73
  - @property directive and instance variables, 73-75
- as object-oriented programming language, 65-69
- programming basics in
  - allocating and initializing objects, 83-84
  - declaring variables, 81-83
  - expressions and decision making, 87-91
  - sending messages to objects, 85-86
- readability of, 69
- retaining and releasing objects in, 91-92

objects
- adding to scroll view, 272
- adding to user interfaces, 176-178
- adding to views, 134-135
- aligning, 135-137
- allocating and initializing, 83-84
- application, 107
- cleaning up in memory, 75
- Constraints, 140-141, 539-540
- delegate, 106
- described, 68
- document outline, 130-131
- dragging and dropping, 579
- id type, 76
- inspecting, 141
- messages in, 68
- pointers to, 82-83
- replacing with nil value, 85
- retaining and releasing, 91-92
- retrieving properties from, 75
- self as reference to, 68
- sending messages to, 85-86
- sizing, 136-139
- switch, 113
- traits in, 144
- typecasting, 84
- UIAlertView, 282-285
- variables as pointers to, 82
- viewing distance between other objects and, 139
- window, 107
- *See also* classes; instances

onscreen controls, 108
OOP (object-oriented programming), 65-69
Open GL ES framework, 102
OpenGL standard, 5
opening
- projects in Xcode, 146
- Xcode, 27-28

operations, mathematical, with variables, 82

@optional directive, 80-81

organization names, 29-30

orientations

accessing data for, 603-606

enabling changes in, 557-543, 563

hardware for detecting. *See* accelerometers; gyroscopes

proper, rotating views into, 568

responsive user interfaces, 534-538

sensing, creating app for, 606-610

setting for devices, 51-53

tilt and rotation, creating app detecting, 611-622

updating interfaces when changing, 561

upside-down, disabling, 563

otherButtonTitles parameter, 283, 286

out signals, simulating, 61

outlets

buttons, toggling animations on and off with, 225-226

creating and connecting in apps

BestFriend app, 689-690

with custom pickers, 396-397

with the date picker, 381-383

detecting tilt and rotation, 614-616

flashlight app, 502-504

flower apps, 260-264

gesture apps, 583-586

Interface Builder, 149

location-aware apps, 719

long-running background tasks, 755

looped animations, 234-237

with magnetic compass, 726

with Master-Detail Application templates, 474-476

MediaPlayground app, 644-645

with modal segues, 351

with navigation controllers, 430-431

with orientation sensing, 607-608

with popovers, 358

scrolling views, 274-275

Settings app, 511

Single View Application template, 179-182

story creators, 206-209

survey app, 523-524

with tab bar controllers, 438-440

universal apps, 770, 774-775

with user alerts, 293-295

defining, 164

defining in GenericViewController class, 428

described, 148-149

identifying, 175

planning, in apps with magnetic compass, 725

referencing one scene's view controller from another, 410

for switches, 253

when needed, 193

See also instance variables

outlines for documents, storyboards, 127-131

output, viewing, NSLog function, 783-784

outputLabel outlet, 381, 397, 438

outputLabel property, 576

P

page curls, 323

page sheet presentation style, 323

paid developer programs

creating and installing development provisioning profiles, 15-19

joining, 10-12

panning gesture, 572

parameters

action sheets, 286

alert views, 283

colons in, 85

described, 68

providing for methods, 40

sender, 166

setSpeed, 243

parent classes. *See* superclasses

partial curl option, transition types, 323

passthrough views, popovers, 335-336

passwords, treating fields as, 198
paste process, customizing, 199
paths, data storage files, 498-499
pause method, 662
people picker, address books, 675-677, 691-695
photo libraries, implementing in apps, 656-659
picker view data source protocol, 369-371, 395, 400-401
picker view delegate protocol, 371-374, 395, 401-402
picker views, 369-374
pickers
  custom. *See* custom picker
  date. *See* date picker
  described, 115, 367-368
  image, 635-638, 656-659
  media, 629-631, 662-668
  people, 675-677, 691-695
  views in, 369-374
  See also choosing
pickerView:didSelectRow:inComponent method, 371, 372
pickerView:numberOfRowsInComponent method, 369
pickerView:rowHeightForComponent method, 373
pickerview:titleForRow:forComponent method, 371, 373
pickerView:viewForRow:viewForComponent:ReusingView method, 373
pickerView:widthForComponent method, 373
pictures, 4-6. *See also* images
Pin menu, 554
pinch recognizer, 582-583, 589-590
pinching gesture, 572
pinning, 141
pixels vs. points, 4
placeholder text, 195-196, 217
plain text fields, 196
platforms, NeXTSTEP, 99
play method, 662
playAudio method, 654
playback
  audio, 649-655
  implementing for movies, 648
playbackState property, 631
players
  AV audio, 633-634
  movie, 627-629, 646-650
  music, 631-632, 663-664
playing
  music, 666-668
  sounds, 289, 307-308, 639
  videos, 639
playMusic method, 667
plus sign, 76
PNG files, as app icons, 54
pointers, declaring variables as, 82-83
points vs. pixels, 4
pop segues, 411
popover option, segues, 321
popover segues, 333-336, 356, 411
popovers
  challenges with, 366, 383
  creating
    creating popover segues, 333-336
    displaying manually, 337
    overview, 333
    preparation for, 333
    programming popover displays, 340-342
  creating apps with, 355-360
  described, 115-116
  displaying image pickers in, 657-658
  enforcement of, 638
  responding to dismissals of, 337-339
  setting sizes of, 334, 356, 380-381, 396
  showing action sheets in, 305
  See also iPads
posts, preparing for Facebook, 681
pragma marks, adding to code, 42
preferences
  implicit, flashlight app, 500-507
  limiting for data storage, 489-491
prefixes, class, 30, 171
prepareForSegue:sender method, 330-331
presentation styles, 323, 326
preservation, state, 492
prices, Apple Developer Program, 8
primitive data types, 81-82
private instance variables, 556
private variables, properties, and methods, 77-78
procedural programming, 66

processing
- long-running background tasks, 758-760
- task-specific background, 739, 747-753

processors used in iOS devices, 6

product names, 29-30

Products subgroup, 33

profiles, development provisioning, 15-19

programmatically defined interfaces, creating, 555-562

programmers. *See* iOS developers

programming languages, Objective-C, 22

Project Code subgroup, 33

project development preparation
- iOS platform tour, 3-7
- overview, 3

project navigator
- adding resources to projects, 34-35
- described, 32-33
- editing code in, 36
- managing project properties in, 50-56

project properties, managing in project navigator, 50-56

projects
- adding empty files manually to, 34
- adding media files to, 641-642
- creating and managing with Xcode, 28-36
  - adding new code files, 34
  - adding resources, 34-35
  - choosing project types, 28-31
  - filtering information currently displayed, 33
  - looping animation, 223-245
  - navigating projects in, 32-33
  - opening Xcode, 27-28
  - removing files and resources, 35-36
  - user interface tour, 31-32
- managing properties of
  - overview, 50-51
  - setting application icons, 53-54
  - setting launch images, 55-56
  - setting supported device orientations, 51-53
- opening in Xcode, 146
- saving, 30
- selecting templates for, 28-29
- *See also* apps; user interfaces

prompt attribute, 415

prompt property, 662

properties
- accessing from popovers upon dismissal, 339
- action sheets, 286
- alertBody, 745
- alertViewStyle, 283
- allowsEditing, 636
- allowsPickingMultipleItems, 662
- animationDuration, 239
- animationImages, 239
- applicationIconBadgeNumber, 745
- cancelButtonTitle, 283
- compared with instance variables, 427
- contentViewController, 358, 384
- declaring instance variables with, 74
- defining, programmatically defined interfaces, 557-558
- delegate, 283, 383-384, 398
- described, 68
- destinationViewController, 384
- finding in symbol navigator, 37-38
- fireDate, 745
- gyroAvailable, 605
- horizontalAccuracy, 708
- imageView, 576
- including @synthesize line with, 78
- initWithTitle, 283
- interface, initializing, 558-559
- message, 283
- MPMediaItemAlbumTitle, 632
- MPMediaItemPropertyArtist, 632
- MPMediaItemPropertyGenre, 632
- MPMediaItemPropertyLyrics, 632
- myLabel, 165
- notification, 745
- numberOfTapsRequired, 573
- numberOfTouchesRequired, 573

otherButtonTitles, 283
outputLabel, 576
planning
  in apps detecting tilt and rotation, 612-613
  in apps with gestures, 576
  in apps with magnetic compass, 725
  in apps with Master-Detail Application templates, 472
  in apps with navigation controllers, 427-428
  in apps with popovers, 356
  in apps with tab bar controllers, 436
  in apps with the date picker, 377
  in BestFriend app, 687-688
  in flashlight app, 501
  in location-aware apps, 715-716
  in long-running background tasks, 754
  in looped animations, 225-226
  in MediaPlayground app, 642
  in modal segues, 346
  in programmatically defined interfaces, 556
  in scrolling views, 270
  in Settings app, 509
  in Single View Application templates, 173-175
  in story creators, 193
  swapping views on rotation, 563
  universal apps, 769
  in user alert apps, 292
playbackState, 631
private, 77-78
projects, managing in project navigator, 50-56
prompt, 662
push count, 432, 440
repeatInterval, 745
retrieving from objects, 75
semicolons in, 72
sharing with #import and @class directives, 319
soundName, 745
timeZone, 745
UIDeviceOrientation, 603
UIImagePickerController-SecureTypeSavedPhotos-Album, 636
UIImagePickerController-SourceTypeCamera, 636
UIImagePickerController-SourceTypePhotoLibrary, 636
underscores in, 74
using instance variables instead of, 75
using instead of instance variables, 377
window, 173

property value, 172

@property directive, 73-75, 174

protocols
  angle brackets in, 72
  AVAudioPlayerDelegate, 634
  CLLocationManagerDelegate, 708-710
  creating, 80-81
  described, 68, 73, 79
  MFMailComposeView-ControllerDelegate, 680
  picker view data source, 369-371, 395, 400-401
  picker view delegate, 371-374, 395, 401-402
  table view data source, 453-456, 465-467
  table view delegate, 468-469
  UIImagePickerController-Delegate, 637-638
  UIPopoverControllerDelegate, 337-339

prototype cell attributes, setting, 451-452

push count property, 432, 440

push option, segues, 321

push segues, 411, 415-416, 428

Q

Quartz Core framework, 102

Quick Help, Xcode, 119-122

Quick Help Inspector, 120-121

Quick Inspector, editing connections with, 153

Quick Look framework, 103

quitting
  apps, 184
  Xcode, 14

R

radian conversion constants, 725

radians constant, adding degrees to, 563

RAM (random access memory), iPhones, 6

random access memory (RAM), iPhones, 6

recognizers, gestures. *See* gesture recognizers

recordAudio method, 652, 655

recorders, AV audio, 634-635

recording, audio, 649-655

references

- circular, 75
- to objects, 82
- removing from projects, 35-36

registration

- development devices, 17, 18
- iOS developer, 9
- orientation updates, 608-609

relationships

- creating between controllers, 414, 421
- described, 313
- tab bar, creating, 436-437

release message, 67-68

remote content, loading, 250-251

removeConstraints NSLayoutConstraint method, 575

removing

- connections, 154
- constraints from image views, 575
- files and resources from projects, 35-36
- installed apps from iOS Simulator, 58

repeatInterval property, 745

repetition, with loops, 89-91

replace option, segues, 321

replacing, image views, 587

requestWithURL class, 250-251

@required directive, 80-81

resetting iOS Simulator to defaults, 58

resignFirstResponder method, 212

resistance, content compression, 543

resolutions, iOS devices, 4-5

resources

- adding to projects, 34-35
- animation, adding to looped animations, 225
- image
  - adding to gestures, 576
  - adding to Settings app, 509
  - background, 715
  - direction, 724
  - Master-Detail Application templates, 471
  - table views, 461
- removing from projects, 35-36
- sound, adding to apps, 291
- *See also* files

Resources group, 33

responders, described, 108

responsive interfaces, programmatically defined, 555-562

responsive user interfaces

- described, 533-534
- designing rotatable and resizable interfaces, 536-538
- enabling interface rotation, 534-535
- *See also* Auto Layout system

restoration, 492

retain message, 91-92

Retina displays

- app icons for, 53
- image views, 223
- launch images for, 54-55
- locating high-resolution images for, 228
- resolution of, 4

Return Key trait, 198

rich media

- AV Foundation framework, 632-635
- creating MediaPlayground app
  - accessing and playing the music library, 662-668
  - creating and connecting outlets and actions, 644-645
  - designing the interface, 642-643
  - implementing audio recording and playback, 649-655
  - implementing Core Image filter, 659-662
  - implementing movie player, 646-650
  - implementing the photo library and camera, 656-659
  - setting up the project, 640-642

Media Player framework, 626-632
supported file formats for, 628
Right Detail table style, 452
root class, 107
rotating gesture, 572
rotation recognizer, 583, 590-593
rotations
creating app detecting tilt and, 611-622
interface rotation events, 603
into proper orientations, 568
responsive user interfaces, 534-538
simulating, 60
swapping views on, 562-569
Rounded Rect button type, 204
rows
changing sizes of, 402-403
counting, 465
selecting, 468
running apps
compiling and, 211
consequences of skipping steps in this book and, 39
detecting devices, 764
first, trying out, 19-22
troubleshooting, 105
in Xcode, 46
See also iOS Simulator

S

saving projects, 30
scales, reversing in sliders, 229
scaling, effects of, 260
scaling factors, 4-5
scene segue logic, implementing, 383-386, 397-399
scenes
adding to storyboards, 314, 344-346
associating view controllers with
date picker, 376-377
navigation controller, 425-426
tab bar controller, 385-436
creating in tab bar controllers, 418-422
date chooser, 380, 384-386
described, 164, 313
detail, updating, 474-476
loading, 172-173
master, updating, 472-475
multiscene storyboards. *See* multiscene storyboards
naming, 316-319
navigation, 413-416
passing data between, 329-331
in storyboard outlines, 127-129
tab bar item buttons as part of, 420
See also storyboards
scheduling notifications, 745-747
schemes, build, choosing, 46
screen resolutions, iOS devices, 4-5
screens, Welcome to Xcode, 27
scrolling options, setting, 203
scrolling views
described, 251
designing the interface for, 270-277
SDKs. *See* software development kits
search, in Xcode library, 118-119
search navigator, finding code with, 40-41
search results, filtering, 41
searching. *See* finding
Secure trait, 198
Security framework, 104
segmented controls
adding to apps, 254
described, 114, 248-249
using in flower apps, 254, 255-256
segments, adding and configuring, 254-255
segues
challenges with, 366
creating
in apps with custom pickers, 396
in apps with the date picker, 379-381
in multiscene storyboards, 320-323
described, 313
exit and unwind, 324-325, 353
implementing logic for, 383-386, 397-399
modal. *See* modal segues
popover, 333-336, 356, 411
push, 411, 415-416, 428
referencing one scene's view controller from another, 410
selecting. *See* choosing

selection handles, 136-137
selection process, customizing, 199
self, 68
self.presentingViewController method, 331
semicolons, in code, 72
sender parameter, 166
sender variable, 212-213
setDateTime action, 382
setIncrement method, 226
setLightSourceAlphaValue method, 504-505
setOutput method, 183-184
setQueueWithItemCollection method, 662
setSpeed method, 226
setSpeed parameter, 243
setStartBarHidden method, 636
setters (mutators), 73
Settings app
- building, 520
- creating and connecting outlets, 511
- creating the settings bundles, 494-496, 511-517
- designing the interface, 509-510
- implementing application logic, 518-520
- setting up the project, 507-509

settings bundles, 494-496, 511-517
setValueFromPreferences value, 519
shake recognizer, 593
shaking gestures, 61, 572, 576, 594-595
sharing
- classes, 78
- data
  - between navigation scenes, 416
  - between tab bar scenes, 422
- properties and methods with #import and @class directives, 319

sheets, action, 285-287, 303-306
ShowAnimalChooser action, 397
showDateChooser action, 381
showFromRect method, 305
showing, functional areas, Xcode, 32
showResults method, 527
signals, TV out, simulating, 61
simulated sizes, view controllers, 271-274
simulators.
Sina Weibo, integrating with other apps, 680-682
Single View Application template, 167-184, 414, 419
singletons, 67
Size Inspector, 138-139, 541-543
sizes
- components and rows, changing, 402-403
- image view, adding instance variables to, 577
- iPhone designs, 199
- launch images, 767
- popovers, setting, 334, 356, 380-381, 396
- simulated, view controllers, 271-274
- tab bar images, 421
- variable and matching, 551-554

sizing objects, 136-139
skeletons, 155
SLComposeViewController class, 680-681
slider range attributes, setting in looped animations, 229-231
sliders
- controlling looped animation speed with, 228-231
- described, 114, 221-222
- reversing scales in, 229

snapshots, managing in Xcode, 44-46
Social framework, 103
social networking
- BestFriend app
  - building, 702
  - creating and connecting outlets and actions, 689
  - designing the interface, 688-689
  - implementing logic in, 701-702
  - setting up the project, 686-688
- integrating with other apps, 680-682

software development kits (SDKs)
- access to beta versions of, 8
- classes in
  - core application, 107-108
  - data type, 109-112
  - interface, 113-116
- versions available, 13

sound. ***See*** **audio**

soundID variable, 307

soundName property, 745

spaghetti code, 162

speed

looped animations

changing with steppers, 231-232

controlling with sliders, 228-231

incrementing, 242-243

setting, 240-242

speed output labels, adding to looped animations, 232

split view controller, 456-459, 471-472

standard developer program, 8

standard editor mode, 42

state preservation, 492

statements

if-then-else, 88-89

readability of, 69

switch, 88-89

status bar settings, 52-53

status bars, toggle in-call, simulating, 61

stepper range attributes, setting for looped animations, 231-232

steppers

changing looped animation speed with, 231-232

described, 114, 222-223

stepping through code, 786-796

stopAccelerometerUpdates method, 605

stopGyroUpdates method, 605, 618

storage of data

approaches to

direct file system access, 496-500

settings bundles, 494-496

user defaults, 493-494

implementing for file systems, 520-528

implementing system settings, 507-520

limiting preferences for, 489-491

Store Kit framework, 104

storeSurvey method, 525-526

storing, data, 30

story creators

building the app, 217

creating and connecting outlets and actions, 206-209

designing the interface for

adding styled buttons, 203-204

adding text fields, 193-194

adding text views, 199

editing button attributes, 204-205

editing text field attributes, 194-196

editing text view attributes, 199-202

setting custom button images, 205-206

setting scrolling options, 203

using attributed and plain text, 196

hiding keyboards, 212-216

implementing application logic, 216-217

implementing button templates, 209-212

setting up the project, 192-193

storyboard files, 171-173

storyboards

adding scenes to, 314

anatomy of, 127-131

creating universal apps with, 767-776

described, 313

multiscene

controlling modal segues manually, 324

creating segues, 320-323

exit and unwind segues in, 325-327

overview, 312-313

passing data between scenes, 329-331

preparing the project, 314-319

navigation controllers in, 413-416

reverting to old layout, 141

scenes compared to, 164

tab bar controllers in, 418-421

See also scenes

stretching objects, 136-137

strings

date format, 387

described, 109

finding in files, 41

strong attribute, 75-76

stub methods, 155

styled buttons, adding to user interfaces, 203-204

styles

action sheets, 286

cells, 452

presentation, 323, 326

segues, choosing, 321, 323

tables, 452

subclasses

described, 67

generic view controllers, 424

iPadViewController, 772-773, 775

master class for, 69

navigation controllers, 424

NSDecimalNumber, 111

similarities in, 775

view controller, 316-319

subgroups

project navigator, 33

See also folders

Subtitle table style, 452

superclasses

described, 67

UIControl, 108

Supporting Files group, 33, 54

survey, implementing application logic, 525-528

survey app

creating and connecting outlets and actions, 523-524

designing the interface, 521-522

setting up the project, 521

survey results, viewing, 527-528

suspension, background, 743-744

swapping views, 537, 562-569

swipe recognizer, 580-581, 588

swiping gesture, 572

switch objects, purpose of, 113

switch statement, 88-89

switches

adding to flower apps, 257-258

described, 248

outlets for, 253

symbol label, 38

symbol navigator, finding methods and properties in, 37-38

@synthesize line, 78

System Configuration framework, 104

System framework, 104

system preparation

iOS platform tour, 3-7

overview, 3

System Sound Services, 288-290, 307, 632

T

tab bar controller classes, 435

tab bar controllers

creating apps with, 433-443

described, 417-422

tab bar images, sizes of, 421

tab bar item attributes, setting, 420-434

tab bar item badges, incrementing, 441-443

tab bar item images, 435

tab bar relationships, creating, 436-437

tab bars and items, 418

tabbed editing, activating, 44

tabbed templates, 418

table attributes, setting, 450-451

table section constants, adding to apps with table views, 461

table view data source protocol, 453-456, 465-467

table view delegate protocol, 468-469

table views

adding tables to, 450-456

appearance of, 448-449

cells in, 448

creating apps with

building the app, 469

designing the interface, 461-463

implementing application logic, 463-469

setting up the project, 460-461

described, 448

tableView:cellForRowAtIndexPath method, 453

tableView:numberOfRowsIn-Section method, 452

tableView:titleForHeaderInSection method, 453

tap recognizer, 579-580, 588

tapping gesture, 572

tapRecognizer instance, 573

targets, universal apps, 776-778

task completion, long-running tasks, 739, 754-760

task-specific background processing, 739, 747-753

Team Provisioning Profile, 18

technology layers. *See* iOS technology layers

televisions. *See* TVs

templates

- buttons, implementing, 209-212
- frameworks in, 100
- Master-Detail Application
  - creating apps with, 469-486
  - described, 459
- selecting for iOS programming projects, 28-29
- Single View Application, 167-184, 414, 419
- tabbed, 418
- universal apps, 764-765

testing

- apps in iOS Simulator
  - generating multitouch events, 59
  - launching, 57-59
  - simulating esoteric conditions, 60-61
  - simulating rotations, 60
- unit, 30

testing applications, on iOS Simulator, 8, 15

text, placeholder, 195-196, 217

text entry fields, alertViewStyle property, 283

text field attributes

- editing, 194-196
- traits of, 197-199

text fields

- adding to user interfaces, 193-194
- alert view, 302-303
- attributed, 196
- editing attributes of, 194-196
- plain, 196
- purpose of, 115, 190

text input traits, 197-199

text views

- adding to user interfaces, 199
- purpose of, 193

three-dimensional (3D) displays, 5

tilt, creating app detecting rotation and, 611-622

time, viewing dates and, 386-387

Time mode, 369

Timer mode, 369

timers, initializing, 757

timeZone property, 745

Tinting setting, 53

title attribute, 415

titles, bar button, setting, 485-486

toggle in-call status bar, simulating, 61

toggleButtonAnimation method, 226

Toolbar area, 31

toolbars

- challenges with, 366
- described, 363-366

touch gestures. *See* gestures

Touch Up Inside event, 153, 189

touching backgrounds, hiding keyboards by, 214-215

tracing apps

- feedback from, 782-783
- with Xcode debugger, 784-799

traits, text input, 197-199

Traits attribute, 144

transition types, segues, 323, 326

troubleshooting

- app tracing and debugging
  - feedback from, 782-783
  - with Xcode debugger, 784-799
- location errors, 710-711
- running apps, 105

tutorial apps, 18

TVs

- Apple, 4
- out signals, simulating, 61

Twitter

- compose view, 60-61
- integrating with other apps, 680-682

two-dimensional (2D) displays, 5

typecasting, 84

types

- button, 204-205
- content, web views, 250
- data
  - float, 111
  - int, 111
  - object, 82-83
  - primitive, 81-82
- described, 81
- id, 76, 166
- transition, segues, 323, 326
- void, 76

typing, hiding characters while, 198

U

UI image picker controller delegate, 637-638
UIActionSheetStyleAutomatic property, 286
UIActionSheet class, 285-287
UIActionSheetStyleBlackOpaque property, 286
UIActionSheetStyleBlack-Translucent property, 286
UIActionSheetStyleDefault property, 286
UIAlertView object, 282-285
UIAlertViewStyleDefault field, 283
UIAlertViewStyleLoginAnd-PasswordInput field, 283
UIAlertViewStylePlainTextInput field, 283
UIAlertViewStyleSecureTextInput field, 283
UIApplication class, 107, 170
UIApplicationMain function, 105
UIBarButtonItem class, 412
UIButton class, 82, 113, 189
UIControl class, 108
UIDatePicker class, 115, 368-369
UIDevice class, 603
UIDeviceOrientation property, 603
UIDeviceOrientationFaceDown value, 603
UIDeviceOrientationFaceUp value, 603
UIDeviceOrientationLandscapeLeft value, 603
UIDeviceOrientationLandscape-Right value, 603
UIDeviceOrientationPortrait value, 603
UIDeviceOrientationPortraitUpside Down value, 603
UIImagePickerController class, 635-638
UIImagePickerControllerDelegate protocol, 637-638
UIImagePickerControllerSecure-TypeSavedPhotosAlbum property, 636
UIImagePickerControllerSource-TypeCamera property, 636
UIImagePickerControllerSource-TypePhotoLibrary property, 636
UIImageView class, 223, 239, 576, 660
UIKit framework, 100
UILabel class, 113, 165, 174, 191, 576
UINavigationBar class, 412
UINavigationController class, 411-416, 424
UINavigationItem class, 412
UIPicker class, 115
UIPickerView class, 369-374
UIPickerViewDataSource protocol, 369-371
UIPopoverArowDirectionDown constant, 342
UIPopoverArrowDirectionAny constant, 342
UIPopoverArrowDirectionLeft constant, 342
UIPopoverArrowDirectionRight constant, 342
UIPopoverArrowDirectionUp constant, 342
UIPopoverController class, 115-116, 340-341
UIPopoverController method, 342
UIPopoverControllerDelegate protocol, 337-339
UIResponder class, 108, 171
UIResponderStandardEditActions class, 199
UIs. *See* user interfaces
UIScrollView class, 251
UISegmentedControl class, 114
UISlider class, 114, 221
UIStepper class, 114, 222
UISwitch class, 113
UITabBar class, 418
UITabBarController class, 435
UITabBarItem class, 418, 420-421
UITapGestureRecognizer class, 573
UITextField class, 115, 190
UITextView class, 115, 190
UIView class, 108, 130, 171
UIViewController class, 108, 316, 331, 424, 775
UIViewController subclass, 316-319
UIWindow class, 107
underscore character, 72, 74
uniform resource locators. *See* URLs
unit testing, 30
universal apps
  creating, 767-776
  development overview, 763
  setup information changes in, 766-767

templates, 765
using multiple targets on, 776-778

unwind, described, 313

unwind segues, 325-327

updateinterface method, 559-560

upside-down orientation, disabling, 563

URLs (uniform resource locators)
allocating and initializing, 83-84
manipulating, 112

user alerts
action sheets, 285-287, 303-306
alert views, 282-285, 296-303
creating apps with
designing the interface, 293-294
implementing action sheets, 303-306
implementing alert sounds and vibrations, 307-308
implementing alert views, 296-303
setting up the project, 291-292
System Sound Services, 288-290, 307
vibrations, 288, 290

user IDs, developer registration, 9

user interfaces (UIs)
Address Book, 674-675
connecting to code, 146-156, 164
creating
adding objects to views, 134-135
Auto Layout system, 139-141
editing tools for, 135-139
Object Library, 131-133
creating apps with custom pickers
building the app, 406
creating and connecting outlets and actions, 396-397
creating segues, 396
designing the interface, 394-395
implementing custom picker view, 399-405
setting up the project, 392-393
creating apps with gestures
adding gesture recognizers, 579-583
building the app, 595
creating and connecting outlets and actions, 583-586
designing the interface, 577-579
implementing application logic, 587-595
setting up the project, 576-579
creating apps with magnetic compass
building, 733
creating and connecting outlets, 726
setting up the project, 724-725
updating application logic, 726-732
updating the user interface, 725
creating apps with Master-Detail Application templates
building the app, 486
implementing application data source, 476-480
implementing master view controller, 480-484
setting up the project, 470-472
tweaking the iPad interface, 472-475
tweaking the iPhone interface, 474-476
creating apps with modal segues
building the app, 354
creating and connecting outlets and actions, 351
creating the modal segue for, 349
designing the interface, 347-349
implementing application logic, 352-353
setting up the project, 343-347
unwinding back to initial scene, 350-351
creating apps with navigation controllers, 423-433
creating apps with orientation sensing, 606-610

creating apps with popovers, 355-360
creating apps with tab bar controllers, 433-443
creating apps with table views
  building the app, 469
  designing the interface, 461-463
  implementing application logic, 463-469
  setting up the project, 460-461
creating apps with the date picker
  building the app, 390
  creating and connecting outlets and actions, 381-383
  creating segues, 379-381
  designing the interface, 378-380
  implementing date calculation logic, 386-390
  implementing scene segue logic, 383-386
  setting up the project, 376-377
creating apps with user alerts
  creating and connecting outlets and actions, 293-295
  designing the interface, 293
  implementing action sheets, 303-306
  implementing alert sounds and vibrations, 307-308
  implementing alert views, 296-303
  setting up the project, 291-292
creating BestFriend app
  building, 702
  creating and connecting outlets and actions, 689-690
  designing the interface, 688-689
  implementing logic in, 690-702
  setting up the project, 686-688
creating flashlight app
  building the app, 506
  creating and connecting outlets and actions, 502-504
  designing the interface, 501-502
  implementing application logic, 504-506
  setting up the project, 501
creating FlowerWeb app
  adding and configuring segments, 254-255
  adding switches, 257-258
  adding web views, 259-260
  building the app, 268
  creating and connecting outlets and actions, 260-264
  implementing application logic, 248-268
  segmented controls, 254-256
  setting up the project, 253
creating location-aware apps
  building the app, 723-724
  creating and connecting outputs, 719
  designing the view, 716-718
  implementing application logic, 719-723
  setting up the project, 715-716
creating looped animations
  adding Hop buttons, 232
  adding image views, 226
  adding speed output labels, 232
  building the app, 243
  changing speed with steppers, 231-232
  controlling speed with sliders, 228-231
  copying image views, 227-228
  creating and connecting outlets and actions, 234-237
  implementing application logic, 237-243
  incrementing speed of, 242-243
  setting background images and colors, 232-233
  setting default images, 226-227
  setting the speed, 240-242
  setting up the project, 224-226
  starting and stopping, 239-240

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

creating MediaPlayground app
  accessing and playing the music library, 662-668
  creating and connecting outlets and actions, 644-645
  designing the interface, 642-643
  implementing audio recording and playback, 649-655
  implementing Core Image filter, 659-662
  implementing movie player, 646-650
  implementing the photo library and camera, 656-659
  setting up the project, 640-642
creating multiscene projects
  controlling modal segues manually, 324
  creating segues, 320-323
  exit and unwind segues in, 325-327
  passing data between scenes, 329-331
  preparing the project, 314-319
creating popovers
  creating popover segues, 333-336
  displaying manually, 337
  preparation for, 333
  programming popover displays, 340-342
  responding to dismissals of, 337-339
creating scenes in tab bar controllers, 418-422
creating scrolling views, 269-277
creating Settings app
  building the app, 520
  creating and connecting outlets, 511
  creating the settings bundles, 494-496, 511-517
  designing the interface, 509-510
  implementing application logic, 518-520
  setting up the project, 507-509
creating story creators
  adding styled buttons, 203-204
  adding text fields, 193-194
  adding text views, 199
  with attributed and plain text, 196
  building the app, 217
  creating and connecting outlets and actions, 206-209
  editing button attributes, 204-205
  editing text field attributes, 194-196
  editing text view attributes, 199-202
  hiding keyboards, 212-216
  implementing application logic, 216-217
  implementing button templates, 209-212
  setting custom button images, 205-206
  setting scrolling options, 203
  setting up projects, 192-193
creating survey app
  creating and connecting outlets and actions, 523-524
  designing the interface, 521-522
  implementing application logic, 525-528
  setting up the project, 521
creating with Single View Application templates, 176-178
customizing appearances of, 142-145
described, 27
disabling editing of, 483
Message, 100, 700
model elements, 282
responsive
  described, 533-534
  designing rotatable and resizable interfaces, 536-538
  enabling interface rotation, 534-535
  programmatically defined, 555-562
  swapping views on rotation, 562-569
  See also Auto Layout system

reusing in other applications, 162
simulating, 145
simulating rotations of, 60
Xcode, tour of, 31-32
See also apps; projects
utility area, 31-32

V

valueForProperty method, 632
values
in CGRectMake function, 49
floating-point
float data type, 111
initializing strings with, 109
height, 276
initializing strings with, 109
inputting on iPads, 7
media picker filters, 630
MFMailComposeResult, 680
nil, 85, 110
numbers, referring by, 393
playbackState property, 631
property, 172
retrieving and setting, 73
setValuesFromPreferences, 518-519
UIDeviceOrientation property, 603
width, 276
working with, 111
variable lists, accessing, 796
variable sizes, 551-554
variable states, examining and changing, 791
variables
assignments with, 82
audio, adding to apps, 749
declaring as pointers, 82-83
described, 68
instance. *See* instance variables
mathematical operations with, 82
myLabel, 165
planning
in apps with Master-Detail Application templates, 472
in apps with table views, 461
as pointers to objects, 82
private, 77-78
private instance, 556
sender, 212-213
soundID, 307
states of, examining and changing, 791
version control, snapshots as, 46
versions
devices, simulating apps on each, 61
iOS, checking for multitasking support in, 739-741
vertical constraints, 141
vibrations, 288, 290, 308
videos
capturing on iOS devices, 7
playing, 639
supported file formats for, 628
View Controller icon, 128
view controller subclasses, 316-319
view controllers
associating with projects, 344-346
associating with scenes
date picker, 376-377
navigation controller, 425-426
tab bar controller, 435-436
becoming first responders, 593-594
changing simulated size of, 271-272
Cocoa Touch, 106
described, 164-166, 313
displaying, 329
generic, classes of, 424
instantiating, 326
iPad, setting up in universal apps, 772-773
preparing for exits, 325, 347
referencing one scene's view controller from another, 410
setting identifiers for, 326-328
stacks of, 411
View icon, 128
view-rotation logic, 567-568
ViewController class, 108, 171-173, 331
ViewController file, 237
ViewController files, 170, 174-175
viewDidAppear method, 276, 587
viewDidLoad method, 209-210, 653, 663, 750, 786-787

viewing
- cells in table views, 466
- constraints in Size Inspector, 541-543
- counters, 432-433, 440-441
- dates and time, 386-387
- detail view, 484-485
- development provisioning profiles, 18
- distances between selected objects and other objects, 139
- functional areas, Xcode, 32
- icons in storyboards, 131
- libraries in Interface Builder, 133
- snapshots, 45-46
- storyboards, 127
- survey results, 527-528
- text field attributes, 194
- variable states, 791
- view controllers, 329
- web views, 264-265

views
- adding gesture recognizers to, 579-583
- adding objects to, 134-135
- alert, 282-285, 296-303
- annotation, 699
- compose, 60-61
- connecting through segues, 429
- date and time settings, 369
- described, 108, 164, 313
- designing, location-aware apps, 716-718
- detail, 484-485
- image
  - adding instance variables to sizes of, 577
  - copying, 227-228
  - described, 223
  - looped animations, 226, 238-239
  - removing constraints from, 575
  - replacing, 587
- instantiating, 326
- iPad, associating iPadViewController class with, 772-773
- mail compose, 700-701
- modal, described, 313
- passthrough, popovers, 335-336
- pickers, 369-374
- popover, setting sizes of, 356
- scrolling
  - described, 251
  - designing the interface for, 269-277
- setting button templates after loading, 209-210
- swapping, 537, 562-569
- table
  - creating apps with, 460-469
  - described, 448-456
- text
  - adding to user interfaces, 199
  - editing attributes of, 199-202
  - purpose of, 193
- web
  - adding to flower apps, 259-260
  - described, 249-251
  - hiding and showing, 264-265

viewWillAppear:animated method, 433

Voiceover, 142

void type, 76

W

warnings
- correcting, 48-50
- low-memory, simulating, 61

watchpoints, setting, 795

weak attribute, 75-76

web addresses. *See* URLs

web view attributes, setting, 259

web views
- adding to flower apps, 258-260
- described, 249-251
- hiding and showing, 264-265

Welcome app, 22

Welcome to Xcode screen, 27

wells, image, 767

width value, 276

WiFi, locating devices with, 44

Wii, 599

window objects, 107

window property, 173

Word Lens, 7

X

X button, 145

Xcode

adding device support components to, 14

as an Apple developer tool, 22

application delegate class, 105

building apps in, 46-50

configuring development machine with, 16-19

correcting errors and warnings in, 48-50

creating and managing projects with, 28

adding new code files, 34

adding resources, 34-35

choosing project types, 28-31

filtering information currently displayed, 33

navigating projects in, 32-33

opening Xcode, 27-28

removing files and resources, 35-36

in Single View Application templates, 168-169

user interface tour, 31-32

creating groups in, 33, 35

creating projects in, 168-169, 192-193

Mad Libs-style story creator, 192-193

creating the right code in, 183

documentation for, 117-119

documentation for Cocoa Touch in, 98

dragging and dropping images in image wells, 767

editing and navigating code in

activating tabbed editing, 44

adding pragma marks, 42

assistant editor mode, 43-42

with code completion, 38-40

finding code with search navigator, 41

finding methods and properties, 37-38

managing snapshots, 44-46

overview, 36-37

implementing Model-View-Controller in, 163-167

installing, 12-13

Interface Builder

connecting user interfaces to code, 146-156

creating user interfaces in, 131-141

customizing appearances of user interfaces, 142-145

described, 125-126

storyboards, 127-131

zooming in and out in, 135

managing project properties in

overview, 50-51

setting application icons, 53-54

setting launch images, 55-56

setting supported device orientations, 51-53

opening projects in, 146

Quick Help for, 119-122

running first iOS app with, 19-22

Single View Application template, 167-184

Xcode debugger, troubleshooting with, 784-799

Y-Z

zooming in and out, Interface Builder, 135