John Ray

In **Full Color**

Figures and code appear as they do in Xcode

Covers **iOS 4.0 and up, and iPhone 4**

Additional files and updates available online

Sams **Teach Yourself**

iPhone® Application Development

Second Edition

in **24 Hours**

SAMS

John Ray

Sams **Teach Yourself**

iPhone® Application Development in 24 Hours

Second Edition

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

Sams Teach Yourself iPhone Application Development in 24 Hours Second Edition

ISBN-13: 978-0-672-33220-3
ISBN-10: 0-672-33220-5

Library of Congress Cataloging-in-Publication Data:

Ray, John, 1971-

Sams teach yourself iPhone application development in 24 hours / John Ray. — 2nd ed.

p. cm.

ISBN 978-0-672-33220-3

1. iPhone (Smartphone)—Programming. 2. Application software—Development. I. Title. II. Title: Teach yourself iPhone application development in 24 hours. III. Title: iPhone application development in 24 hours.

QA76.8.I64R39 2011

005.26—dc22

2010035798

Printed in the United States of America

First Printing October 2010

Trademarks

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

Warning and Disclaimer

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

Bulk Sales

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

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

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

International Sales
international@pearson.com

Associate Publisher
Greg Wiegand

Acquisitions Editor
Laura Norman

Development Editor
Keith Cline

Managing Editor
Sandra Schroeder

Senior Project Editor
Tonya Simpson

Copy Editor
Keith Cline

Indexer
Brad Herriman

Proofreader
Language Logistics, LLC

Technical Editor
Matthew David

Publishing Coordinator
Cindy Teeters

Designer
Gary Adair

Compositor
TnT Design, Inc.

Contents at a Glance

Introduction

HOUR 1 Preparing your System and iPhone for Development
2 Introduction to Xcode and the iPhone Simulator
3 Discovering Objective-C: The Language of Apple Platforms
4 Inside Cocoa Touch
5 Exploring Interface Builder
6 Model-View-Controller Application Design
7 Working with Text, Keyboards, and Buttons
8 Handling Images, Animation, and Sliders
9 Using Advanced Interface Objects and Views
10 Getting the User's Attention
11 Making Multivalue Choices with Pickers
12 Implementing Multiple Views with Toolbars and Tab Bars
13 Displaying and Navigating Data Using Table Views
14 Reading and Writing Application Data
15 Building Rotatable and Resizable User Interfaces
16 Using Advanced Touches and Gestures
17 Sensing Orientation and Motion
18 Working with Rich Media
19 Interacting with Other Applications
20 Implementing Location Services
21 Building Background-aware Applications
22 Building Universal Applications
23 Application Debugging and Optimization
24 Distributing Applications Through the App Store

Index

Table of Contents

Introduction 1

Who Can Become an iPhone Developer? 1

Who Should Use This Book? 2

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

HOUR 1: Preparing Your System and iPhone for Development 3

Welcome to the iOS Platform 3

Becoming an iOS Developer 7

Creating a Development Provisioning Profile 12

Developer Technology Overview 23

Summary 25

Q&A 25

Workshop 26

HOUR 2: Introduction to Xcode and the iPhone Simulator 27

Using Xcode 27

Using the iPhone Simulator 45

Further Exploration 50

Summary 50

Q&A 51

Workshop 51

HOUR 3: Discovering Objective-C: The Language of Apple Platforms 53

Object-Oriented Programming and Objective-C 53

Exploring the Objective-C File Structure 58

Objective-C Programming Basics 64

Memory Management 74

Further Exploration 77

Summary 77

Q&A 78

Workshop 79

HOUR 4: Inside Cocoa Touch 81
What Is Cocoa Touch? 81
Exploring the iOS Technology Layers 83
Tracing the iPhone Application Life Cycle 88
Cocoa Fundamentals 90
Exploring the iOS Frameworks with Xcode 98
Summary 102
Q&A 102
Workshop 103

HOUR 5: Exploring Interface Builder 105
Understanding Interface Builder 105
Creating User Interfaces 110
Customizing Interface Appearance 115
Connecting to Code 119
Further Exploration 126
Summary 127
Q&A 127
Workshop 128

HOUR 6: Model-View-Controller Application Design 129
Understanding the Model-View-Controller Paradigm 129
How Xcode and Interface Builder Implement MVC 131
Using the View-Based Application Template 135
Further Exploration 148
Summary 149
Q&A 149
Workshop 150

HOUR 7: Working with Text, Keyboards, and Buttons 151
Basic User Input and Output 151
Using Text Fields, Text Views, and Buttons 153
Setting Up the Project 154
Further Exploration 176

Summary ... 177
Q&A ... 177
Workshop ... 178

HOUR 8: Handling Images, Animation, and Sliders 179
User Input and Output ... 179
Creating and Managing Image Animations and Sliders ... 181
Further Exploration ... 196
Summary ... 197
Q&A ... 197
Workshop ... 198

HOUR 9: Using Advanced Interface Objects and Views 199
User Input and Output (Continued) ... 199
Using Switches, Segmented Controls, and Web Views ... 204
Using Scrolling Views ... 221
Further Exploration ... 227
Summary ... 227
Q&A ... 228
Workshop ... 228

HOUR 10: Getting the User's Attention 231
Exploring User Alert Methods ... 231
Generating Alerts ... 235
Using Action Sheets ... 245
Using Alert Sounds and Vibrations ... 249
Further Exploration ... 253
Summary ... 254
Q&A ... 254
Workshop ... 255

HOUR 11: Making Multivalue Choices with Pickers 257
Understanding Pickers ... 257
Using Date Pickers ... 261

Implementing a Custom Picker View ... 270
Further Exploration ... 289
Summary ... 290
Q&A ... 290
Workshop ... 291

HOUR 12: Implementing Multiple Views with Toolbars and Tab Bars **293**

Exploring Single Versus Multi-View Applications ... 293
Creating a Multi-View Toolbar Application ... 295
Building a Multi-View Tab Bar Application ... 307
Further Exploration ... 326
Summary ... 327
Q&A ... 327
Workshop ... 328

HOUR 13: Displaying and Navigating Data Using Table Views **329**

Understanding Table Views and Navigation Controllers ... 329
Building a Simple Table View Application ... 332
Creating a Navigation-Based Application ... 344
Further Exploration ... 359
Summary ... 359
Q&A ... 360
Workshop ... 360

HOUR 14: Reading and Writing Application Data **363**

Design Considerations ... 363
Reading and Writing User Defaults ... 366
Understanding the iPhone File System Sandbox ... 381
Implementing File System Storage ... 384
Further Exploration ... 404
Summary ... 405
Q&A ... 405
Workshop ... 406

HOUR 15: Building Rotatable and Resizable User Interfaces 407
Rotatable and Resizable Interfaces 407
Creating Rotatable and Resizable Interfaces with Interface Builder 411
Reframing Controls on Rotation 416
Swapping Views on Rotation 423
Further Exploration 429
Summary 430
Q&A 430
Workshop 431

HOUR 16: Using Advanced Touches and Gestures 433
Multitouch Gesture Recognition 434
Using Gesture Recognizers 435
Further Exploration 448
Summary 449
Q&A 449
Workshop 449

HOUR 17: Sensing Orientation and Motion 451
Understanding iPhone Motion Hardware 451
Accessing Orientation and Motion Data 454
Sensing Orientation 458
Detecting Tilt and Rotation 462
Further Exploration 471
Summary 472
Workshop 473

HOUR 18: Working with Rich Media 475
Exploring Rich Media 475
Preparing the Media Playground Application 478
Using the Movie Player 482
Creating and Playing Audio Recordings 486
Using the Photo Library and Camera 492

Accessing and Playing the iPod Library 495
Further Exploration 501
Summary 502
Q&A 502
Workshop 503

HOUR 19: Interacting with Other Applications 505
Extending Application Integration 505
Using Address Book, Email, and Maps… Oh My! 509
Further Exploration 526
Summary 527
Q&A 527
Workshop 527

HOUR 20: Implementing Location Services 529
Understanding Core Location 529
Creating a Location-Aware Application 534
Understanding the Magnetic Compass 541
Further Exploration 549
Summary 550
Q&A 550
Workshop 551

HOUR 21: Building Background-Aware Applications 553
Understanding iOS 4 Backgrounding 554
Disabling Backgrounding 558
Handling Background Suspension 559
Implementing Local Notifications 561
Using Task-Specific Background Processing 564
Completing a Long-Running Background Task 570
Further Exploration 576
Summary 577
Q&A 577
Workshop 577

HOUR 22: Building Universal Applications 579

Universal Application Development 579
Understanding the Universal Window-Based Application Template 581
Other Universal Application Tools 596
Further Exploration 598
Summary 599
Q&A 599
Workshop 599

HOUR 23: Application Debugging and Optimization 601

Debugging in Xcode 601
Monitoring with Instruments 614
Profiling with Shark 620
Further Exploration 627
Summary 627
Q&A 627
Workshop 628

HOUR 24: Distributing Applications Through the App Store 629

Preparing an Application for the App Store 630
Submitting an Application for Approval 642
Promoting Your Application 649
Exploring Other Distribution Methods 655
Summary 657
Q&A 657
Workshop 657

Index 659

About the Author

John Ray is currently serving as a Senior Business Analyst and Development Team Manager for the Ohio State University Research Foundation. He has written numerous books for Macmillan/Sams/Que, including *Using TCP/IP: Special Edition*, *Teach Yourself Dreamweaver MX in 21 Days*, *Mac OS X Unleashed*, and *Teach Yourself iPad 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://teachyourselfiphone.com or follow him on Twitter at #iPhoneIn24.

Dedication

This book is dedicated to everyone who makes me smile, even if only on occasion. Thanks for keeping me stay sane during long nights of typing.

Acknowledgments

Thank you to the group at Sams Publishing—Laura Norman, Sandra Schroeder, Keith Cline, Matthew David—for providing amazing support during the creation of this book. Your thoroughness and attention to detail make the difference between a book that works and one that bewilders.

Thanks to my friends, family, and pets. Deepest apologies to my fish tank. I swear I'll get you working right soon.

We Want to Hear from You!

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

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

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

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

E-mail: feedback@quepublishing.com

Mail: Greg Wiegand
Associate Publisher
Sams Publishing
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

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

Introduction

Over the past four years, Apple has changed the way we think about mobile computing. The iOS Platform has changed the way that we, the public, think about our mobile computing devices. With full-featured applications and an interface architecture that demonstrates that small screens can be effective workspaces, the iPhone has become the smartphone of choice for users and developers alike.

Part of what makes the iPhone such a success is the combination of an amazing interface and an effective software distribution method. With Apple, the user experience is key. The iOS is designed to be controlled with your fingers rather by using a stylus or keypad. The applications are "natural" and fun to use, instead of looking and behaving like a clumsy port of a desktop app. Everything from interface to application performance and battery life has been considered. The same cannot be said for the competition.

Through the App Store, Apple has created the ultimate digital distribution system for developers. 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, an audience exists.

In 2010, Apple introduced the iPad and iPhone 4 platforms—bringing larger, faster, and higher-resolution capabilities to the iOS. Although these devices will only be a few months "old" by the time you read this, they will already be in the hands of millions of users, eagerly awaiting the next great app.

My hope is that this book will bring iOS development to a new generation of developers. *Teach Yourself iPhone Development in 24 Hours* provides a clear natural progression of skills development, from installing developer tools and registering 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 iPhone 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 Snow Leopard, you have everything you need to begin developing for the iPhone.

Developing an application for the iPhone 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 iPhone 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 apps that are useful and unique, 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 the iPhone 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 authors do 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. There is a vast amount of information on iPhone development available, and only limited space in this book. 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 4. Much of what you'll be learning is common to all the iOS releases, but this book also covers several important advances in 4, such as Gestures, embedded video playback, multitasking, universal (iPhone/iPad) applications, and more!

Unfortunately, this is not a complete reference for the iPhone 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'll be downloading in Hour 1, "Preparing Your System and iPhone for Development." In many hours, you'll find a section titled "Further Exploration." This will identify additional related topics of interest. Again, a willingness to explore is an important quality in becoming a successful iPhone 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 the book's website at http://teachyourselfiphone.com.

In addition to the support website, you can follow along on Twitter! Search for #iPhoneIn24 on Twitter to receive official updates and tweets from other readers. Use the hashtag #iPhoneIn24 in your tweets to join the conversation. To send me messages via Twitter, begin each tweet with @johnemeryray.

HOUR 21

Building Background-Aware Applications

What You'll Learn in This Hour:

- How iOS 4 supports background tasks
- What types of background tasks are supported
- How to disable backgrounding
- How to suspend applications
- How to execute code in the background

"The ability to run multiple applications in the background" mocks the Verizon commercial. "Why can't a modern operating system run multiple programs at once?" question the discussion groups. As a developer and a fan of the iPhone, I've found these threads amusing in their naiveté and somewhat confusing. The iPhone has always run multiple applications simultaneously in the background, but they were limited to Apple's applications. This restriction has been to preserve the user experience of the device as a phone. Rather than an "anything goes" approach, Apple has taken steps to ensure that the phone remains responsive at all times.

With the release of iOS 4.x, Apple answers the call from the competition by opening up background processing to third-party applications. Unlike the competitors, however, Apple has been cautious in how it approached backgrounding—opening it up to a specific set of tasks that users commonly encounter. In this hour's lesson, you learn several of the multitasking techniques that you can implement in iOS 4.

Understanding iOS 4 Backgrounding

If you've been working in iOS 4.x or later as you've built the tutorials in this book, you may have noticed that when you quit the applications on your phone or in the iPhone Simulator, they still show up in the iOS task manager, and, unless you manually stop them, they tend to pick up right where they left off. The reason for this is that projects created in iOS 4.x are background-ready as soon as you click the Build and Run button. That doesn't mean that they will run in the background, just that they're aware of the new iOS 4 background features and will take advantage with a little bit of help.

Before we examine how to enable backgrounding (also called multitasking) in our projects, let's first identify exactly what it means to be a background-aware application, starting with the types of backgrounding supported, then the application life cycle methods.

Types of Backgrounding

We explore four primary types of backgrounding in iOS 4.x: application suspension, local notifications, task-specific background processing, and task completion.

Suspension

When an application is suspended, it will cease executing code but be preserved exactly as the user left it. When the user returns to the application, it appears to have been running the whole time. In reality, all tasks will be stopped, keeping the app from using up your iPhone's resources. Any application that you compile against iOS 4.x will, by default, support background suspension. You should still handle cleanup in the application if it is about to be suspended (see "The Background-Aware Application Life Cycle" section, later in this chapter), but beyond that, it "just works."

In addition to performing cleanup as an application is being suspended, it will be your responsibility to recover from a background suspended state and update anything in the application that should have changed while it was suspended (time/date changes and so on).

Local Notifications

The second type of background processing is the scheduling of local notifications (`UILocalNotification`). If you've ever experienced a push notification, local notifications are the same but are generated by the applications that you write. An application, while running, can schedule notifications to appear onscreen at a point in

time in the future. For example, the following code initializes a notification (`UILocationNotification`) configures it to appear in five minutes, and then uses the application's `scheduleLocalNotification` method to complete the scheduling:

```
UILocalNotification *futureAlert;
futureAlert = [[[UILocalNotification alloc] init] autorelease];
futureAlert.fireDate = [NSDatedateWithTimeIntervalSinceNow:300];
futureAlert.timeZone = [NSTimeZonedefaultTimeZone];
[[UIApplication sharedApplication] scheduleLocalNotification:futureAlert];
```

These notifications, when invoked by iOS, can show a message, play a sound, and even update your application's notification badge. They cannot, however, execute arbitrary application code. In fact, it is likely that you will simply allow iOS to suspend your application after registering your local notifications. A user who receives a notification can click View button in the notification window to return to your application.

Task-Specific Background Processing

Before Apple decided to implement background processing, they did some research on how users worked with their handhelds. What they found was that there were specific types of background processing that people needed. First, they needed audio to continue playing in the background; this is necessary for applications like Pandora. Next, location-aware software needed to update itself in the background so that users continued to receive navigation feedback. Finally, VoIP applications like Skype needed to operate in the background to handle incoming calls.

These three types of tasks are handled uniquely and elegantly in iOS 4.x. By declaring that your application requires one of these types of background processing, you can, in many cases, enable your application to continue running with little alteration. To declare your application capable of supporting any (or all) of these tasks, you will add the Required Background Modes (`UIBackgroundModes`) key to the project's plist file, and then add values of App Plays Audio (Audio), App Registers for Location Updates (Location), or App Provides Voice over IP Services (VoIP).

Task Completion for Long-Running Tasks

The fourth type of backgrounding that we'll be using in iOS 4.x is task completion. Using task-completion methods, you can "mark" the tasks in your application that will need to finish before it can be safely suspended (file upload/downloads, massive calculations, and so on).

For example, to mark the beginning of a long running task, first declare an identifier for the specific task:

```
UIBackgroundTaskIdentifier myLongTask;
```

Then use the application's `beginBackgroundTaskWithExpirationHandler` method to tell iOS that you're starting a piece of code that can continue to run in the background:

```
myLongTask = [[UIApplicationsharedApplication]
              beginBackgroundTaskWithExpirationHandler:^{
                 // If you're worried about exceeding 10 minutes, handle it here
              }];
```

And, finally, mark the end of the long-running task with the application `endBackgroundTask` method:

```
[[UIApplication sharedApplication] endBackgroundTask:myLongTask];
```

Each task you mark will have roughly 10 minutes (total) to complete its actions, plenty of time for most common uses. After the time completes, the application is suspended and treated like any other suspended application.

The Background-Aware Application Life Cycle Methods

In Hour 4, "Inside Cocoa Touch," you started learning about the application life cycle, as shown in Figure 21.1. You learned that, in iOS 4.x, applications should clean up after themselves in the `applicationDidEnterBackground` delegate method. This replaces `applicationWillTerminate` in earlier versions of the OS, or as you'll learn shortly, in applications that you've specifically marked as not capable (or necessary) to run in the background.

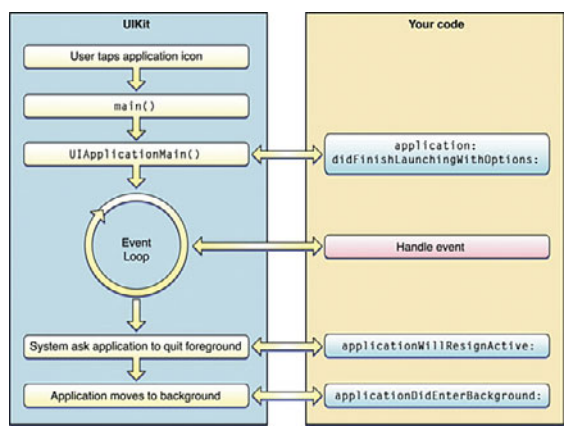

FIGURE 21.1
The iOS 4.x application life cycle.

In addition to `applicationDidEnterBackground`, there are several other methods that you should implement to be a proper background-aware iOS citizen. For many small applications, you won't need to do anything with these, other than leave them as is in the application delegate. As your projects increase in complexity, however, you'll want to make sure that your apps move cleanly from the foreground to background (and vice versa), avoiding potential data corruption and creating a seamless user experience.

Watch Out!

It is important to understand that iOS can terminate your applications, even if they're backgrounded, if it decides that the device is running low on resources. You can expect that your applications will be fine, but plan for a scenario where they are forced to quit unexpectedly.

The methods that Apple expects to see in your background-aware apps are as follows:

- **`application:didFinishLaunchingWithOptions`:** Called when your application first launches. If your application is terminated while suspended, or purged from memory, needs to restore its previous state manually. (You did save it your user's preferences, right?)
- **`applicationDidBecomeActive`:** Called when an application launches or returns to the foreground from the background. This method can be used to restart processes and update the user interface, if needed.
- **`applicationWillResignActive`:** Invoked when the application is requested to move to the background or to quit. This method should be used to prepare the application for moving into a background state, if needed.
- **`applicationDidEnterBackground`:** Called when the application has become a background application. This replaces `applicationWillTerminate` in iOS 4.x. You should handle all final cleanup work in this method. You may also use it to start long-running tasks and use task-completion backgrounding to finish them.
- **applicationWillEnterForeground:** Called when an application returns to an active state after being backgrounded.
- **applicationWillTerminate:** Invoked when an application on a nonmultitasking version of iOS is asked to quit, or when iOS determines that it needs to shut down an actively running background application.

Method stubs for all of these exist in your iOS 4.x application delegate implementation files. If your application needs additional setup or teardown work, just add the code to the existing methods. As you'll see shortly, many applications, such as the majority of those in this book, require few changes.

Watch Out!

The assumption in this hour's lesson is that you are using iOS 4.x or later. If you are not, using background-related methods and properties on earlier versions of the OS will result in errors. To successfully target both iOS 4.x and earlier devices, check to see whether backgrounding is available, and then react accordingly in your apps.

Apple provides the following code snippet in the *iPhone Application Programming Guide* for checking to see (regardless of OS version) whether multitasking support is available:

```
UIDevice* device = [UIDevice currentDevice];
BOOL backgroundSupported = NO;
if ([device respondsToSelector:@selector(isMultitaskingSupported)])
    backgroundSupported = device.multitaskingSupported;
```

If the resulting `backgroundSupported` Boolean is `YES`, you're safe to use background-specific code.

Now that you have an understanding of the background-related methods and types of background processing available to you, let's look at how they can be implemented. To do this, we'll reuse tutorials that we've built throughout the book (with one exception). We won't be covering how these tutorials were built, so be sure to refer to the earlier hours if you have questions on the core functionality of the applications.

Disabling Backgrounding

We start with the exact opposite of enabling backgrounding: disabling it. If you think about it, there are many different "diversion" apps that don't need to support background suspension or processing. These are apps that you use and then quit. They don't need to hang around in your task manager afterward.

For example, consider the HelloNoun application in Hour 6, "Model-View-Controller Application Design." There's no reason that the user experience would be negatively affected if the application started from scratch each time you ran it. To implement this change in the project, follow these steps:

1. Open the project in which you want to disable backgrounding (such as HelloNoun).
2. Open the project's plist file in the resources group (HelloNoun-Info.plist).

3. Add an additional key to the property list, selecting Application Does Not Run in Background (`UIApplicationExitsOnSuspend`) from the Key pop-up menu.
4. Click the check box beside the key, as shown in Figure 21.2.
5. Save the changes to the plist file.

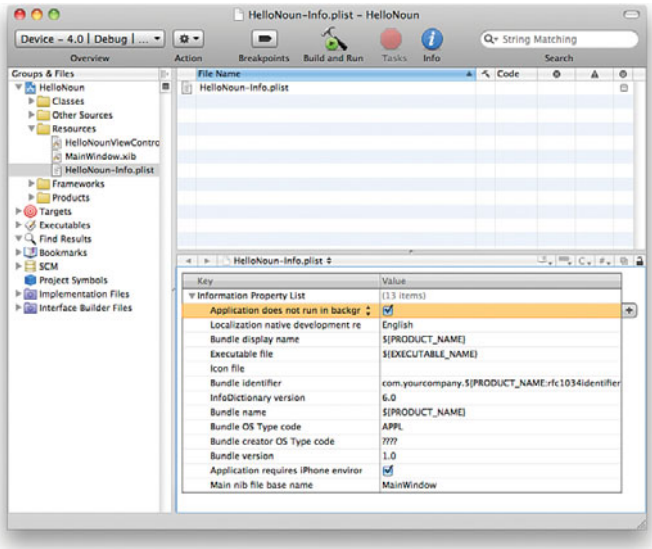

FIGURE 21.2
Add the Application Does Not run in Background (`UIApplicationExitsOnSuspend`) key to the project.

Build and run the application on your iPhone or in the iPhone simulator. When you exit the application with the Home button, it will not be suspended, nor will not show in the task manager, and it will restart fresh when you launch it the next time.

Handling Background Suspension

In the second tutorial, we handle background suspension. As previously noted, you don't have to do anything to support this other than build your project with the iOS 4.x development tools. That said, we use this example as an opportunity to prompt users when they return to the application after it was backgrounded.

For this example, we update the ImageHop application from Hour 8, "Handling Images, Animation, and Sliders." It is conceivable (work with me here, folks!) that a user will want to start the bunny hopping, exit the application, and then return to exactly where it was at some time in the future.

To alert the user when the application returns from suspension, we'll edit the application delegate method `applicationWillEnterForeground`. Recall that this method is invoked only when an application is returning from a backgrounded state. Open ImageHopAppDelegate.m and implement the method, as shown in Listing 21.1.

LISTING 21.1

```
- (void)applicationWillEnterForeground:(UIApplication *)application {
    UIAlertView *alertDialog;
    alertDialog = [[UIAlertView alloc]
                   initWithTitle: @"Yawn!"
                   message:@"Was I asleep?"
                   delegate: nil
                   cancelButtonTitle: @"Welcome Back"
                   otherButtonTitles: nil];
    [alertDialog show];
    [alertDialog release];
}
```

Within the method, we declare, initialize, show, and release an alert view, exactly as we did in the "Getting Attention" tutorial in Hour 10, "Getting the User's Attention." After updating the code, build and run the application. Start the ImageHop animation, and then use the Home button to background the app.

After waiting a few seconds (just for good measure), open ImageHop again using the task manager or its application icon (not Build and Run!). When the application returns to the foreground, it should pick up exactly where it left off and present you with the alert shown in Figure 21.3.

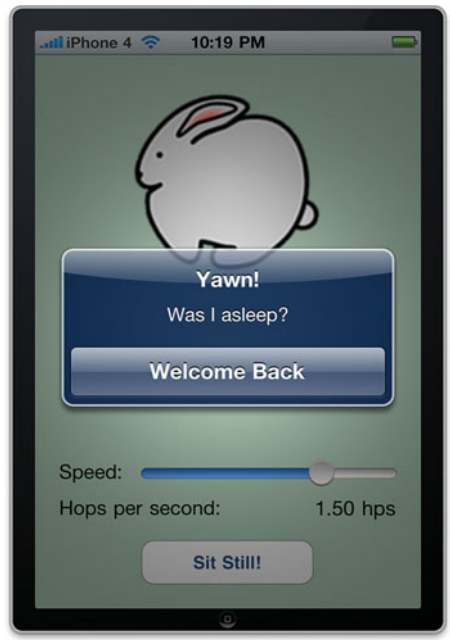

FIGURE 21.3
The `applicationWillEnterForeground` method is used to display an alert upon returning from the background.

Implementing Local Notifications

Earlier in this lesson, you saw a short snippet of the code necessary to generate a local notification (`UILocalNotification`). As it turns out, there's not much more you'll need beyond those few lines! To demonstrate the use of local notifications, we'll be updating Hour 10's "Getting the User's Attention" `doAlert` method. Instead of just displaying an alert, it will also show a notification 5 minutes later and then schedule local notifications to occur every day thereafter.

Common Notification Properties

You want to configure several properties when creating notifications. A few of the more interesting of these include the following:

- **`applicationIconBadgeNumber`:** An integer that is displayed on the application icon when the notification is triggered
- **`fireDate`:** An `NSDate` object that provides a time in the future for the notification to be triggered
- **`timeZone`:** The time zone to use for scheduling the notification
- **`repeatInterval`:** How frequently, if ever, the notification should be repeated
- **`soundName`:** A string (`NSString`) containing the name of a sound resource to play when the notification is triggered
- **`alertBody`:** A string (`NSString`) containing the message to be displayed to the user

Creating and Scheduling a Notification

Open the GettingAttention application and edit the `doAlert` method so that it resembles Listing 21.2. (Bolded lines are additions to the existing method.) Once the code is in place, we'll walk through it together.

LISTING 21.2

```
1: -(IBAction)doAlert:(id)sender {
2:     UIAlertView *alertDialog;
3:     UILocalNotification *scheduledAlert;
4:
5:     alertDialog = [[UIAlertView alloc]
6:                    initWithTitle: @"Alert Button Selected"
7:                  message:@"I need your attention NOW (and in alittle bit)!"
8:                    delegate: nil
9:                    cancelButtonTitle: @"Ok"
```

LISTING 21.2 continued

```
10:                     otherButtonTitles: nil];
11:
12:     [alertDialog show];
13:     [alertDialog release];
14:
15:
16:     [[UIApplication sharedApplication] cancelAllLocalNotifications];
17:     scheduledAlert = [[[UILocalNotification alloc] init] autorelease];
18:     scheduledAlert.applicationIconBadgeNumber=1;
19:     scheduledAlert.fireDate = [NSDate dateWithTimeIntervalSinceNow:300];
20:     scheduledAlert.timeZone = [NSTimeZone defaultTimeZone];
21:     scheduledAlert.repeatInterval =  NSDayCalendarUnit;
22:     scheduledAlert.soundName=@"soundeffect.wav";
23:     scheduledAlert.alertBody = @"I'd like to get your attention again!";
24:
25:     [[UIApplication sharedApplication]
        ➥scheduleLocalNotification:scheduledAlert];
26:
27: }
```

First, in line 3, we declare `scheduledAlert` as an object of type `UILocalNotification`. This local notification object is what we set up with our desired message, sound, and so on, and then pass off to the application to display sometime in the future.

In Line 16, we use `[UIApplication sharedApplication]` to grab our application object, and then call the `UIApplication` method `cancelAllLocalNotifications`. This cancels any previously scheduled notifications that this application may have made, giving us a clean slate.

Line 17 allocates and initializes the local notification object `scheduledAlert`. Because the notification is going to be handled by iOS rather than our GettingAttention application, we can use `autorelease` to release it.

In line 18, we configure the notification's `applicationIconBadgeNumber` property so that when the notification is triggered, the application's badge number is set to 1 to show that a notification has occurred.

Line 19 uses the `fireDate` property along with the `NSDate` class method `dateWithTimeIntervalSinceNow` to set the notification to be triggered 300 seconds in the future.

Line 20 sets the `timeZone` for the notification. This should almost always be set to the local time zone, as returned by `[NSTimeZone defaultTimeZone]`.

Line 21 sets the `repeatInterval` property for the notification. This can be chosen from a variety of constants, such as `NSDayCalendarUnit` (daily), `NSHourCalendarUnit`

(hourly), and `NSMinuteCalendarUnit` (every minute). The full list can be found in the `NSCalendar` class reference in the Xcode developer documentation.

In Line 22, we set a sound to be played along with the notification. The `soundName` property is configured with a string (`NSString`) with the name of a sound resource. Because we already have soundeffect.wav available in the project, we can use that without any further additions.

Line 23 finishes the notification configuration by setting the `alertBody` of the notification to the message we want the user to see.

When the notification object is fully configured, we schedule it using the `UIApplication` method `scheduleLocalNotification` (line 25). This finishes the implementation!

Choose Build and Run to compile and start the application on your iPhone or in the iPhone Simulator. After GettingAttention is up and running, click the Alert Me! button. After the initial alert is displayed, click the Home button to exit the application. Go get a drink, and come back in about 4 minutes and 59 seconds. At exactly 5 minutes later, you'll receive a local notification, as shown in Figure 21.4.

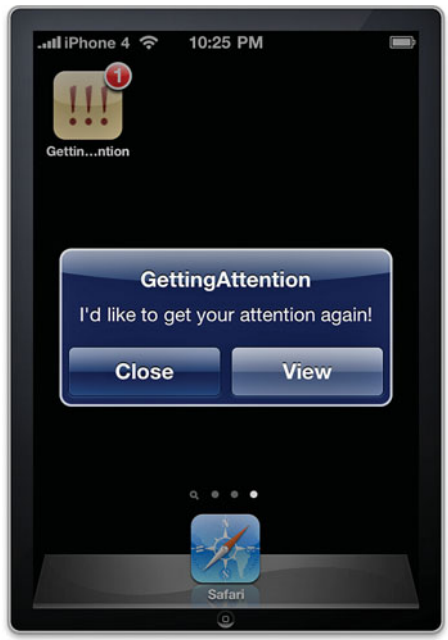

FIGURE 21.4
Local notifications are displayed onscreen even when the application isn't running.

Using Task-Specific Background Processing

So far, we haven't actually done any real background processing! We've suspended an application and generated local notifications, but, in each of these cases, the application hasn't been doing any processing. Let's change that! In our final two examples, we'll execute *real* code behind the scenes while the application is in the background. Although it is well beyond the scope of this book to generate a VoIP application, we can use our Cupertino application from last hour's lesson, with some minor modifications, to show background processing of location and audio!

Preparing the Cupertino Application for Audio

When we finished off the Cupertino application in the last hour, it told us how far away Cupertino was, and presented straight, left, and right arrows on the screen to indicate the direction the user should be traveling to reach the Mothership. We can update the application to audio using `SystemSoundServices`, just as we did in Hour 10's GettingAttention application.

The only tricky thing about our changes is that we won't want to hear a sound repeated if it was the same as the last sound we heard. To handle this requirement, we'll use a constant for each sound: 1 for straight, 2 for right, and 3 for left, and store this in a variable called `lastSound` each time a sound is played. We can then use this as a point of comparison to make sure that what we're about to play isn't the same thing we did just play!

Adding the AudioToolbox Framework

To use System Sound Services, we need to first add the AudioToolbox framework. Open the Cupertino (with Compass implementation) project in Xcode. Right-click the Frameworks group and choose Add, Existing Frameworks. Choose AudioToolbox.framework from the list that appears, and then click Add, as shown in Figure 21.5.

Adding the Audio Files

Within the Cupertino Audio Compass - Navigation and Audio folder included with this hour's lesson, you'll find an Audio folder. Drag the files from the audio folder (straight.wav, right.wav, and left.wav) to the Resources group within the Xcode project. Choose to copy the files into the application when prompted, as shown in Figure 21.6.

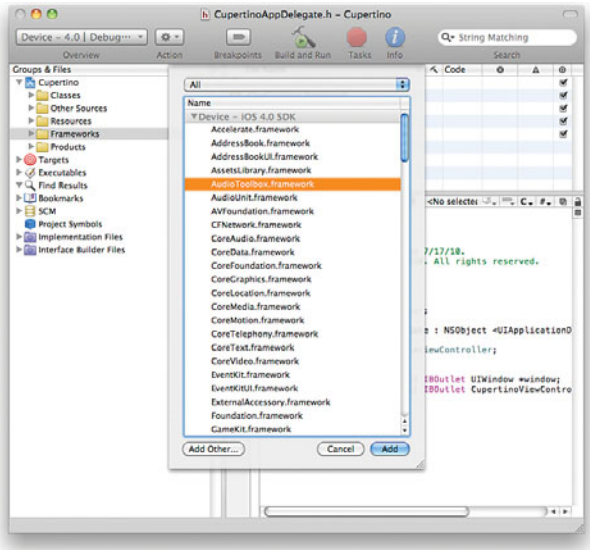

FIGURE 21.5
Add the AudioToolbox.framework to the project.

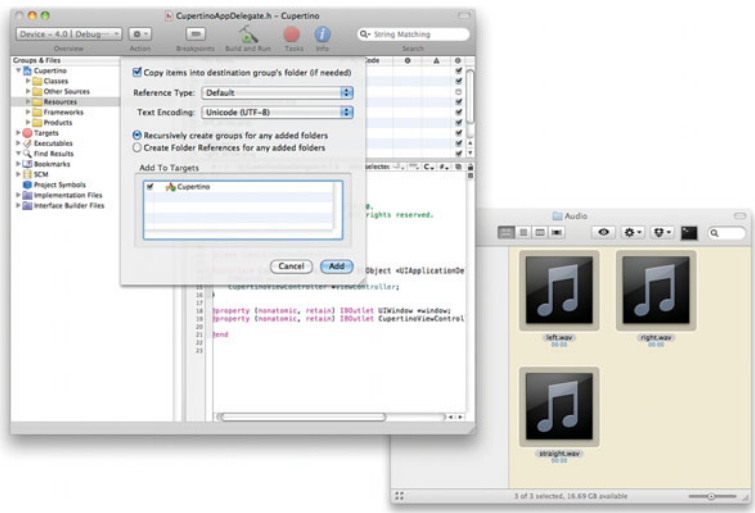

FIGURE 21.6
Add the necessary sound resources to the project.

Updating the CupertinoViewController.h Interface File

Now that the necessary files are added to the project, we need to update the CupertinoViewController interface file. Add an `#import` directive to import the AudioToolbox interface file, and then declare instance variables for three `SystemSoundIDs` (`soundStraight`, `soundLeft`, and `soundRight`) and an integer

`lastSound` to hold the last sound we played. Remember that these aren't objects, so there's no need to declare the variables as pointers to objects, add properties for them, or release them!

The updated CupertinoViewController.h file should resemble Listing 21.3.

LISTING 21.3

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

@interface CupertinoViewController : UIViewController
<CLLocationManagerDelegate> {

    CLLocationManager *locMan;
    CLLocation *recentLocation;
    IBOutlet UILabel *distanceLabel;
    IBOutlet UIView  *distanceView;
    IBOutlet UIView  *waitView;
    IBOutlet UIImageView *directionArrow;
    SystemSoundID soundStraight;
    SystemSoundID soundRight;
    SystemSoundID soundLeft;
    int lastSound;

}

@property (assign, nonatomic) CLLocationManager *locMan;
@property (retain, nonatomic) CLLocation *recentLocation;
@property (retain, nonatomic) UILabel *distanceLabel;
@property (retain, nonatomic) UIView *distanceView;
@property (retain, nonatomic) UIView *waitView;
@property (retain, nonatomic) UIView *directionArrow;

-(double)headingToLocation:(CLLocationCoordinate2D)desired
                   current:(CLLocationCoordinate2D)current;

@end
```

Adding Sound Constants

To help keep track of which sound we last played, we declared the `lastSound` instance variable. Our intention is to use this to hold an integer representing each of our three possible sounds. Rather than remembering that 2 = right, and 3 = left, and so on, let's add some constants to the CupertinoViewController.m implementation file to keep these straight.

Insert these three lines following the existing constants we defined for the project:

```
#define straight 1
#define right 2
#define left 3
```

With the setup out of the way, we're ready to implement the code to generate the audio directions for the application.

Implementing the Cupertino Audio Directions

To add sound playback to the Cupertino application, we need to modify two of our existing `CupertinoViewController` methods. The `viewDidLoad` method will give us a good place to load all three of our sound files and set the `soundStraight`, `soundRight`, `soundLeft` references appropriately. We'll also use it to initialize the `lastSound` variable to `0`, which doesn't match any of our sound constants. This ensures that whatever the first sound is, it will play.

Edit CupertinoViewController.m and update `viewDidLoad` to match Listing 21.4.

LISTING 21.4

```
- (void)viewDidLoad {
    [super viewDidLoad];

    NSString *soundFile;

    soundFile = [[NSBundle mainBundle]
                  pathForResource:@"straight" ofType:@"wav"];
    AudioServicesCreateSystemSoundID((CFURLRef)
                [NSURL fileURLWithPath:soundFile]
                ,&soundStraight);
    [soundFile release];

    soundFile = [[NSBundle mainBundle]
                  pathForResource:@"right" ofType:@"wav"];
    AudioServicesCreateSystemSoundID((CFURLRef)
                [NSURL fileURLWithPath:soundFile]
                ,&soundRight);
    [soundFile release];

    soundFile = [[NSBundle mainBundle]
                  pathForResource:@"left" ofType:@"wav"];
    AudioServicesCreateSystemSoundID((CFURLRef)
                [NSURL fileURLWithPath:soundFile]
                ,&soundLeft);
    [soundFile release];
    lastSound=0;

    locMan = [[CLLocationManager alloc] init];
    locMan.delegate = self;
    locMan.desiredAccuracy = kCLLocationAccuracyThreeKilometers;
    locMan.distanceFilter = 1609; // a mile
    [locMan startUpdatingLocation];
    if ([CLLocationManager headingAvailable]) {
        locMan.headingFilter = 15;
        [locMan startUpdatingHeading];
    }
}
```

Did you Know?

> Remember, this is all code we've used before! If you are having difficulties understanding the sound playback process, refer back to the Hour 10 tutorial.

The final logic that we need to implement is to play each sound when there is a heading update. The CupertinoViewController.m method that implements this is `locationManager:didUpdateHeading`. Each time the arrow graphic is updated in this method, we'll prepare to play the corresponding sound with the `AudioServicesPlaySystemSound` function. Before we do that, however, we'll check to make sure it isn't the same sound as `lastSound`; this will help prevent a Max Headroom stuttering effect as one sound file is played repeatedly over top of itself. If `lastSound` doesn't match the current sound, we'll play it and update `lastSound` with a new value.

Edit the `locationManager:didUpdateHeading` method as described. Your final result should look similar to Listing 21.5.

LISTING 21.5

```
- (void)locationManager:(CLLocationManager *)manager
       didUpdateHeading:(CLHeading *)newHeading {


    if (self.recentLocation != nil && newHeading.headingAccuracy >= 0) {
        CLLocation *cupertino = [[[CLLocation alloc]
                                  initWithLatitude:kCupertinoLatitude
                                  longitude:kCupertinoLongitude] autorelease];
        double course = [self headingToLocation:cupertino.coordinate
                                        current:recentLocation.coordinate];
        double delta = newHeading.trueHeading - course;
        if (abs(delta) <= 10) {
            directionArrow.image = [UIImage imageNamed:@"up_arrow.png"];
           if (lastSound!=straight) AudioServicesPlaySystemSound(soundStraight);
            lastSound=straight;
        } else {
            if (delta > 180) {
                directionArrow.image = [UIImage imageNamed:@"right_arrow.png"];
                if (lastSound!=right) AudioServicesPlaySystemSound(soundRight);
                lastSound=right;
            } else if (delta > 0) {
                directionArrow.image = [UIImage imageNamed:@"left_arrow.png"];
                if (lastSound!=left) AudioServicesPlaySystemSound(soundLeft);
                lastSound=left;
            } else if (delta > -180) {
                directionArrow.image = [UIImage imageNamed:@"right_arrow.png"];
                if (lastSound!=right) AudioServicesPlaySystemSound(soundRight);
                lastSound=right;
            } else {
                directionArrow.image = [UIImage imageNamed:@"left_arrow.png"];
                if (lastSound!=left) AudioServicesPlaySystemSound(soundLeft);
                lastSound=left;
            }
```

```
        }
        directionArrow.hidden = NO;
    } else {
        directionArrow.hidden = YES;
    }

}
```

The application is now ready for testing. Use Build and Run to install the updated Cupertino application on your iPhone, and then try moving around. As you move, it will speak "Right," "Left," and "Straight" to correspond to the onscreen arrows. Try exiting the applications and see what happens. Surprise! It won't work! That's because we haven't yet updated the project's plist file to contain the Required Background Modes (`UIBackgroundModes`) key.

Did you Know?

If, while you're testing the application, it still seems a bit "chatty" (playing the sounds too often), you may want to update the `locMan.headingFilter` to a larger value (like 15 or 20) in the `viewDidLoad` method. This will help cut down on the number of heading updates.

Adding the Background Modes Key

Our application performs two tasks that should remain active when in a background state. First, it tracks our location. Second, it plays audio to give us a general heading. We need to add both audio and location background mode designations to the application for it to work properly. Update the Cupertino project plist by following these steps:

1. Click to open the project's plist file in the resources group (Cupertino-Info.plist).
2. Add an additional key to the property list, selecting Required Background Modes (`UIBackgroundModes`) from the Key pop-up menu.
3. Expand the key and add two values within it: App Plays Audio (Audio) and App Registers for Location Updates (Location), as shown in Figure 21.7. Both values will be selectable from the pop-up menu in the Value field.
4. Save the changes to the plist file.

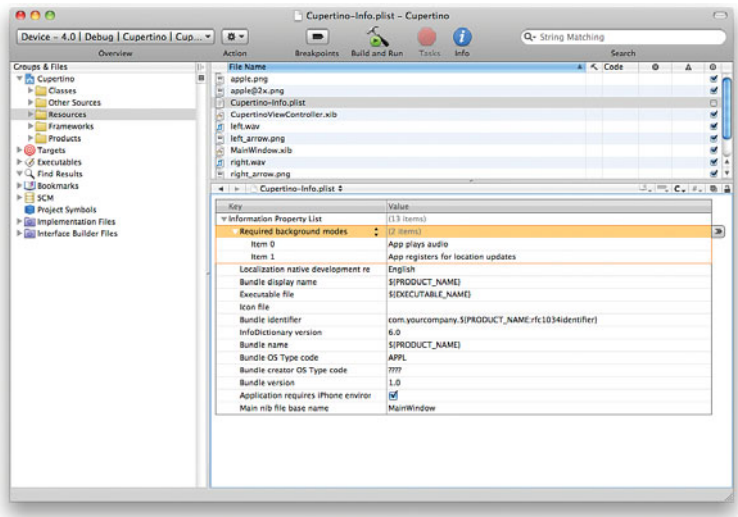

FIGURE 21.7
Add the background modes that are required by your application.

After updating the plist, install the updated application on your iPhone and try again. This time, when you exit the application, it will continue to run! As you move around, you'll hear spoken directions as Cupertino continues to track your position behind the scenes.

By the Way

By declaring the location and audio background modes, your application is able to use the full services of Location Manager and the iOS's many audio playback mechanisms when it is in the background.

Completing a Long-Running Background Task

In our final tutorial of the hour, we need to create a project from scratch. Our book isn't about building applications that require a great deal of background processing. Sure, we could demonstrate how to add code to an existing project and allow a method to run in the background, but we don't have any long running methods that could make use of it.

To demonstrate how we can tell iOS to allow something to run in the background, we'll create a new application, SlowCount, that does nothing but count to 1,000—slowly. We'll use the task-completion method of background to make sure that, even when the application is in the background, it continues to count until it reaches 1,000—as shown in Figure 21.8.

FIGURE 21.8
To simulate a long-running task, our application will count. Slowly.

Preparing the Project

Create a new view-based iPhone application named **SlowCount**. We'll move through development fairly quickly because, as you can imagine, this application is pretty simple.

The application will have a single outlet, a `UILabel` named `theCount`, which we'll use to present the counter onscreen. In addition, it will need a an integer to use as a counter (`count`), an `NSTimer` object that will trigger the counting at a steady interval (`theTimer`), and a `UIBackgroundTaskIdentifier` variable (not an object!) that we'll use to reference the task we have running in the background (`counterTask`).

> Every task that you want to enable for background task completion will need its own `UIBackgroundTaskIdentifier`. This is used along with the `UIApplication` method `endBackgroundTask` to identify which background task has just ended.

By the Way

Open the SlowCountViewController.h file and implement it as shown in Listing 21.6.

LISTING 21.6

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

@interface SlowCountViewController : UIViewController {
    int count;
    NSTimer *theTimer;
    UIBackgroundTaskIdentifier counterTask;
    IBOutlet UILabel *theCount;
}

@property (nonatomic,retain) UILabel *theCount;

@end
```

By the Way

The `UILabel theCount` is the only object we'll be accessing and modifying properties of in the application; therefore, it is the only thing that needs a `@property` declaration.

Next, clean up after the `UILabel` object in the SlowCountViewController.m `dealloc` method. The other instance variables either aren't objects (`count`, `counterTask`) or will be allocated and released elsewhere in the application (`NSTimer`):

```
- (void)dealloc {
    [theCount release];
    [super dealloc];
}
```

Creating the User Interface

It's a bit of a stretch to claim that this application has a "user interface," but we still need to prepare the SlowCountViewController.xib to show the `theCount` label on the screen.

Open the XIB file in Interface Builder, and drag a `UILabel` into the center of the view. Set the label's text to read **0**. With the label selected, use the Attributes Inspector (Command+1) to set the label alignment to center and the font size to 36. Finally, align the right and left sides of the label with the right and left sizing guides. You've just created a UI masterpiece, as shown in Figure 21.9.

Finish the view by Control-dragging from the File's Owner icon in the Document window to the `UILabel` in the view. Choose `theCount` when prompted to make the connection.

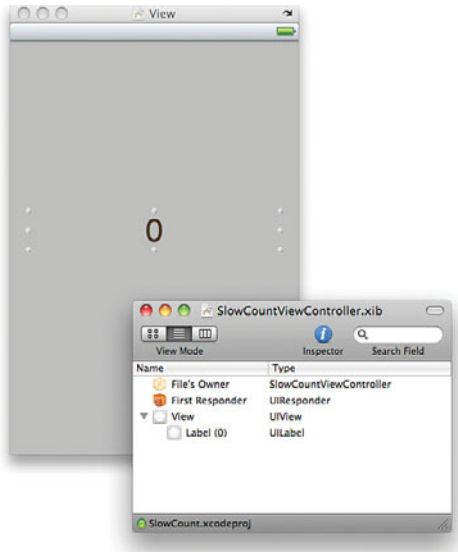

FIGURE 21.9
Add a `UILabel` to the view to hold the current count.

Implementing the Counter Logic

To finish our applications core functionality (counting!), we need to do two things. First, we need to set the counter (`count`) to **0** and allocate and initialize `NSTimer` that will fire at a regular interval. Second, when the timer fires, we will ask it to invoke a second method, `countUp`. In the `countUp` method, we'll check to see whether `count` is `1000`. If it is, we'll turn off the timer and we're done, if not, we'll update `count` and display it in our `UILabel theCount`.

Initializing the Timer and Counter

Let's start with initializing the counter and timer. What better place to do this than in SlowCount.m's `viewDidLoad` method. Implement `viewDidLoad` as shown in Listing 21.7.

LISTING 21.7

```
- (void)viewDidLoad {
    [super viewDidLoad];
    count=0;
    theTimer=[NSTimer scheduledTimerWithTimeInterval:0.1
                    target:self
                    selector:@selector(countUp)
                    userInfo:nil
                    repeats:YES];
}
```

Line 3 initializes our integer counter, `count`, to 0.

Lines 4–8 initialize and allocate the `theTimer` `NSTimer` object with an interval of `0.1` seconds. The `selector` is set to use the method `countUp`, which we'll be writing next. The timer is set to keep repeating with `repeats:YES`.

All that remains is to implement `countUp` so that it increments the counter and displays the result.

Updating the Counter and Display

Add the `countUp` method, as shown in Listing 21.8, before the `viewDidLoad` method in SlowCountViewController.m. This should be quite straightforward—if the count equals `1000`, we're done and it's time to clean up—otherwise, we count!

LISTING 21.8

```
 1: - (void)countUp {
 2:     if (count==1000) {
 3:         [theTimer invalidate];
 4:         [theTimer release];
 5:     } else {
 6:         count++;
 7:         NSString *currentCount;
 8:         currentCount=[[NSString alloc] initWithFormat:@"%d",count];
 9:         theCount.text=currentCount;
10:         [currentCount release];
11:     }
12: }
```

Lines 2–4 handle the case that we've reached the limit of our counting (`count==1000`). If it has, we use the timer's `invalidate` method to stop it, and then `release` it.

Lines 5–11 handle the actual counting and display. Line 5 updates the `count` variable. Line 7 declares the `currentCount` string, which is then allocated and populated in line 8. Line 9 updates our `theCount` label with the `currentCount` string. Line 10 releases the string object.

Build and Run the application—it should do exactly what you expect—count slowly until it reaches 1,000. Unfortunately, if you background the application, it will suspend. The counting will cease until the application returns to the foreground.

Enabling the Background Task Processing

To enable the counter to run in the background, we need to mark it as a background task. We'll use this code snippet to mark the beginning of the code we want to execute in the background:

```
counterTask = [[UIApplication sharedApplication]
               beginBackgroundTaskWithExpirationHandler:^{
                  // If you're worried about exceeding 10 minutes, handle it here
               }];
```

And we'll use this code snippet to mark the end:

```
[[UIApplication sharedApplication] endBackgroundTask:counterTask];
```

By the Way

If we were worried about the application not finishing the background task before it was forced to end (roughly 10 minutes), we could implement the optional code in the `beginBackgroundTaskWithExpirationHandler` block. You can always check to see how much time is remaining by checking the `UIApplication` property `backgroundTimeRemaining`.

Let's update our `viewDidLoad` and `countUp` methods to include these code additions. In `viewDidLoad`, we'll start the background task right before we initialize the counter. In `countUp`, we end the background task after `count==1000` and the timer is invalidated and released.

Update `viewDidLoad` as shown in Listing 21.9.

LISTING 21.9

```
- (void)viewDidLoad {
    [super viewDidLoad];

    counterTask = [[UIApplication sharedApplication]
               beginBackgroundTaskWithExpirationHandler:^{
                  // If you're worried about exceeding 10 minutes, handle it here
               }];
    count=0;
    theTimer=[NSTimer scheduledTimerWithTimeInterval:0.1
                     target:self
                     selector:@selector(countUp)
                     userInfo:nil
                     repeats:YES];
}
```

Then make the corresponding additions to `countUp`, demonstrated in Listing 21.10.

LISTING 21.10

```
- (void)countUp {
    if (count==1000) {
        [theTimer invalidate];
        [theTimer release];
        [[UIApplication sharedApplication] endBackgroundTask:counterTask];
    } else {
        count++;
```

LISTING 21.10 continued

```
        NSString *currentCount;
        currentCount=[[NSString alloc] initWithFormat:@"%d",count];
        theCount.text=currentCount;
        [currentCount release];
    }
}
```

Save your project files, then Build and Run the application on your iPhone or in the simulator. After the counter starts counting, pressing the Home button to move the application to the background. Wait a minute or so, and then re-open the application through the task manager or the application icon. The counter will have continued to run in the background!

Obviously, this isn't a very compelling project itself, but the implications for what can be achieved in real-world apps is definitely exciting!

Further Exploration

When I sat down to write this lesson, I was torn. Background tasks/multitasking is definitely the "must have" feature of iOS 4.0, but it's a challenge to demonstrate anything meaningful in the span of a dozen or two pages. What I hope we've achieved is a better understanding of how iOS multitasking works and how you might implement it in your own applications. Keep in mind that this is not a comprehensive guide to background processing—there are many more features available, and many ways that you can optimize your background-enabled apps to maximize iPhone battery life and speed.

As a next step, you should read the following sections in Apple's *iPhone Application Programming Guide* (available through the Xcode documentation): "Executing Code in the Background," "Preparing Your Application to Execute in the Background," and "Initiating Background Tasks."

As you review Apple's documentation, pay close attention to the tasks that your application should be completing as it enters the background. There are implications for games and graphic-intensive applications that are well beyond the scope of what we can discuss here. How well you adhere to these guidelines will determine whether Apple accepts your application or kicks it back to you for optimization.

Summary

Background applications on iOS devices are not the same as background applications on your Macintosh. There are a well-defined set of rules that iOS background-enabled applications you must follow to be considered "good citizens" of iOS 4.x. In this hour's lesson, you learned about the different types of backgrounding available in iOS and the methods available to support background tasks. Over the course of five tutorial applications, you put these techniques to the test, creating everything from notifications triggered when an application isn't running to a simple navigation app that provides background voice prompting.

You should now be well prepared to create your own background-aware apps and take full advantage of the latest and greatest feature of iOS 4.0.

Q&A

Q. ***Why can't I run any code I want in the background?***

A. Someday, I suspect you will, but for now the platform is constrained to the specific types of background processing we discussed. The security and performance implications of running anything and everything on a device that is always connected to the Internet is enormous. Remember that the iPhone is a *phone*. Apple intends to ensure that when your iPhone needs to be used as a phone, it functions as one!

Q. ***What about timeline-based background processing, like IM clients?***

A. Timeline-based processing (reacting to events that occur over time) is currently not allowed in iOS. This is a disappointment but ensures that there aren't dozens of apps sitting on your phone, eating up resources, waiting for something to happen.

Workshop

Quiz

1. Background tasks can be anything you want in iOS 4.0. True or false?
2. Any application you compile for iOS 4.0 will continue to run when the user exits it. True or false?
3. Only a single long-running background task can be marked background completion. True or false?

Answers

1. False. Apple has a well-defined set of rules for implementing background processing in iOS 4.0.
2. False. Applications will suspend in the background by default. To continue processing, you must implement background tasks as described in this hour's lesson.
3. False. You can mark as many long-running tasks as you'd like, but they must all complete within a set period of time (around 10 minutes).

Activities

1. Return to a project in an earlier hour and properly enable it for background processing.

Index

#import directive
implementation files, 63
interface files, 59
#pragma mark directive, 39
%@ string format specifier, 602
%f string format specifier, 602-603
%i string format specifier, 602
148apps.com, 651
@implementation directive, implementation files, 63
@interface directive, interface files, 59-60
@property directive, 133
interface files, 61-62
@synthesize directive, 133
implementation files, 63

A

ABPeoplePickerNavigationControllerDelegate protocol, 513
ABPersonHasImageData function, 517
ABRecordCopyVal method, 517
ABRecordCopyValue function, 515
Accelerate framework, Core OS layer, 88
accelerometer, 452-453, 472
API, 456
gravity unit, 452
measurable axes, 452
orientation, sensing, 458-461
reading, 456-458
sensing movement, 469
tilt, detecting, 462-471
updates
managing, 467-468
reacting to, 468-469
Accessibility Inspector, 118-119
***Accessibility Programming Guide for iPhone OS*, 127**
Accessibility settings (Interface Builder), 116-117
accessing text fields, alerts, 244-245
action sheets, 245, 255
button presses, responses, 248-249
buttons, 247
changing appearance, 247-248
displaying, 245-247
actions
BestFriend application, connecting, 512
buttons, connecting, 171
ColorTilt application, adding, 463-464
connecting, 190
GettingAttention application, 234
date pickers, connecting to, 265
FieldButtonFun application, 155-156
Flashlight application, connecting, 368
FlowerWeb application, preparing, 205-206
GetFlower, 121
ImageHop application
connecting outlets, 190
preparing, 182-184
Internet Builder application, 120-121
connections, 123-124
MediaPlayground application, connecting, 480-482
MultiViews application
adding to, 302-303
connecting to, 303
newBFF, 510
segmented controls, connecting, 210
sendEmail, 510
switches, connecting to, 211-212

TabbedCalculation application
adding, 313-314
connecting, 317
view controllers, 140-141
connection points, 144-146
Active Configuration setting, building applications, 40
active device, universal applications, detecting and displaying, 588-590
Activity Monitor instrument, 619
ad hoc deployment, applications, 655-656
Add Contact button, 166
Add Horizontal Guide command (Layout menu), 112
Add Vertical Guide command (Layout menu), 112
Address Book framework, 505-506
Core Services layer, 86
***Address Book Programming Guide for iPhone OS*, 526**
Address Book UI framework, 505-506
BestFriend application, accessing, 512-518
Cocoa Touch layer, 85
addTextField method, 254
advertising applications, 649-655
iAds, 653-655
pricing, 653
social networks, 650-652
updates, 652-653
websites, 650-652
alertBody notification property, 561
alertDialog variable, 237
alerts, 231-232, 249
action sheets, 245
button press responses, 248-249
changing appearance, 247-248
displaying, 245-247
buttons, adding, 238-241
displaying, 236-237
fields, adding, 241-245
generating, 235-245
multi-option alerts, creating, 238-241
notification interfaces, creating, 233
playing sounds, 250-253
prepping notification files, 232-233
sounds, 254
System Sound Services, 250
vibrations, 253
AlertViewDelegate protocol, 240-241
alignment (IB layout tool), 113-114
Alignment command (Layout menu), 113
allocation, objects, 67-68
Anderson, Fritz, 627
animation resources, adding, 182
animation speed, setting, 193-195
animationDuration property, 189
animations
image views, 186-187
startAnimating, 188
starting, 187-188
stopping, 187-188
API (application programming interface), accelerometer, 456
App IDs
choosing, Development Provisioning Assistant, 14
setting, 636-637
App Store, 629
App IDs, setting, 636-637
applications
distributing, 629, 655-656
preparing for, 630-639, 642
promoting, 649-655
submitting for approval, 642-649
uploading, 647-649
distribution certificates, creating, 634-636
unique applications, 642
AppAdvice.com, 651
AppCraver.com, 651
appearance
action sheets, changing, 247-248
segmented controls, choosing, 208
Appearance text input trait, 159
Apple Developer Program, 7-10
costs, 8
registration, 8-9
Apple Developer Suite, 23-24
Interface Builder, 105-106
connecting interfaces to code, 119-124
Identity Inspector, 125-126
user interfaces, 110-117
XIB files, 107-110
iPhone Simulator, 45
esoteric conditions, 49-50
generating multitouch events, 48
launching applications, 46-47
rotation simulation, 48
Xcode, 27-28
building applications, 39-42
editing code, 34-39
modifying project properties, 42, 45
navigating code, 34-39
project management, 28-32
removal of files and resources, 33-34
Apple IDs, 8
Apple iPhone Dev Center, 8-10
Apple tutorials, 177
Apple website, 8
AppleiPhoneApps.com, 652
application icons, adding, 631-633

application logic, FlashCards application, implementing, 394-399

application objects, UIApplication class, 91

Application Preferences in the iPhone Application Programming tutorial, 405

application resource constraints, iOS platform, 5

application:didFinishLaunching-WithOptions method, 586, 592

applicationIconBadgeNumber notification property, 561

applications

App IDs, setting, 636-637

App Store

preparing for, 630-639, 642

submitting for approval, 642-649

art work, creating for, 630-633

attaching Shark profiler, 621-624

background-aware applications, 553-556, 576-577

background suspension, 559-560

disabling backgrounding, 558-559

implementing local notifications, 561-563

life cycle methods, 556-558

long-running background tasks, 570-576

task-specific background processing, 564-570

BestFriend, 509

Address Book framework, 512-518

connecting actions and outlets, 512

creating UI, 511-512

map objects, 518-523

Message UI, 523-525

setting up, 510-511

built-in capabilities, 505

Core Location framework, 529

ColorTilt, 462

adding actions and outlets, 463-464

CoreMotion framework, 463

motion events, 466-471

preparing interface, 464-465

setting up, 462-465

Contacts, 381

Cupertino, 534

audio directions, 567-569

background image resources, 534

background modes key, 569-570

Core Location framework, 534

creating UI, 536-537

location manager, 538-540

outlets, 535-536

preparing for audio, 564-567

properties, 535-536

protocols, 535-536

task-specific background processing, 564-570

Cupertino Compass, 541-549

calculating heading, 547

direction image resources, 543

heading updates, 545-549

location manager headings, 541-542

outlets, 543-544

properties, 543-544

setting up, 543-544

updating UI, 544-545

data, storage locations, 382-383

DateCalc, 261

adding date pickers, 263-265

finishing interface, 266-267

setting up, 262-263

view controller logic, 267-270

DebuggerPractice, 604-606, 612-614

Instruments, 614-620

profiling, 620-626

setting breakpoints, 606-607

setting watchpoints, 611-612

stepping through code, 608-611

variable states, 608

decision making, 70

expressions, 70-71

if-then-else statements, 71

repetition with loops, 72-74

switch statements, 72

design, 363-365

MVC structure, 130-131

device capability requirements, defining, 634

distributing, 629, 655-656

distribution, configuring for, 638-639, 642

distribution certificates, creating, 634-636

distribution provisioning profiles, creating, 638

FieldButtonFun

actions, 155-156

adding text fields, 156-161

adding text views, 161-164

creating styled buttons, 164-171

hiding keyboard, 171-174

outlets, 155-156

setting up, 154

file cleanup, 383

FlashCards, 384

application logic, 394-399

archiving flash cards, 402-404

applications

class logic, 385-386
CreateCardViewContoller, 391-393
creating interface, 384, 387-391
object archiving, 400-402
preparing interface, 386-387
Flashlight, 366-372
connecting actions and outlets, 368
creating interface, 367
logic, 369-370
reading preferences, 371-372
setting up, 366-367
storing preferences, 370-371
FlowerColorTable, 332-333
adding outlets, 334
adding table views, 335-337
data source methods, 338-340
populating cells, 340-342
providing data to, 337-342
row touch events, 342-343
setting up, 333-337
FlowerInfoNavigator, 344-345
adding outlets and properties, 351
adding web view, 352-353
detail view, 350-353
detail view controller logic, 351-352
navigation events, 356-357
providing data to, 346-350
root view table controllers, 353-356
setting up, 345-346
table data source methods, 354
UI, 357-358
FlowerWeb, 204
finishing interface, 215
preparing actions and outlets, 205-206
releasing objects, 220-221
segmented controls, 206-210
setting up, 205
switches, 210-212
view controller logic, 216-220
web views, 212-214
Gestures, 435
connecting outlets, 439
creating interface, 437-439
pinch recognizer, 443-445
rotation recognizer, 445-447
setting up, 436-437
shake recognizer, 447-448
swipe recognizer, 441-443
tap recognizer, 439-443
GettingAttention, 249
action sheets, 245-249
connecting actions, 234
connecting outlets, 234
creating notification interface, 233
generating alerts, 235-245
local notifications, 561-563
playing sounds, 250-253
prepping notification files, 232-233
System Sound Services, 250
vibrations, 253
ImageHop, 181-182
actions, 182-184
adding animation resources, 182
adding hop button, 191-192
adding image views, 184-188
adding labels, 191
adding sliders, 188-190
background suspension, 559-560
connecting actions, 190
connecting outlets, 190
finishing interface, 190-192
outlets, 182-184
releasing objects, 195
setting up, 182
view controller logic, 193-195
integration, 505, 526-527
Address Book frameworks, 505-506
Map Kit framework, 508
mapping, 508-509
Message UI framework, 507
launch images, adding, 633
life cycle, 88-90
location-aware applications, creating, 534-540
Mac OS X Installer application, launching, 11
MatchPicker, 271
adding picker views, 273-274
configuring UI, 284-289
connecting outlets, 275
data structures, 276-278
finishing interface, 274-275
outlets, 272
output labels, 275
protocols, 271
providing data to, 275-281
reacting to choices, 281-284
releasing objects, 272-273
setting up, 271-273
MediaPlayground, 478
adding media files, 483
adding Media Player framework, 482
connecting actions and outlets, 480-482
creating audio recordings, 486-490
creating interface, 480
handling cleanup, 485-486
Image Picker, 492-495

Media Picker, 495-501
movie playback, 483-485
movie player, 482-486
music player, 499-501
playing audio recordings, 490-491
receiving notifications, 485
setting up, 478-480
memory usage, 615
multi-view applications, 293-295
benefits, 294
static interface elements, 294-295
MultipleViews, 295
adding actions and outlets, 302-303
adding toolbar controls, 300-302
adding view controllers, 296-297
adding views, 296-297
connecting actions and outlets, 303
instantiating view controllers, 298-299
setting up, 296-297
view switch methods, 303-305
multitouch gesture recognition, 434, 448-449
Notes, 381
Orientation
determining orientation, 461
orientation changes, 460
preparing interface, 459-460
setting up, 458
preferences, 363, 366-372
setting up, 366-367
profiles, preparing, 643-647
profiling, Shark profiler, 620-626
promoting, 649-655
iAds, 653-655
pricing, 653
social networks, 650-652
updates, 652-653
websites, 650-652
Reframe, 416
adding outlets and properties, 416-417
connecting outlets, 421-422
creating interface, 417-422
disabling Autosizing, 418
laying out, 418-421
reframing logic, 422-423
releasing objects, 417
setting up, 416-417
ReturnMe, 372
creating interface, 374
setting up, 373-374
settings bundles, 375-381
sales, monitoring, 649-650
sandbox, 381-384
Scroller, 221
adding scroll views, 223-225
preparing outlets, 222-223
releasing objects, 226
scrolling behavior, 225-226
setting up, 222
SimpleSpin, 411-416
Autosizing, 413-416
setting up, 411-412
testing, 412-413
single-view applications, 293-295
SlowCount
counter logic, 573-574
creating UI, 572
long-running background tasks, 570-576
Swapper, 423
adding outlets and properties, 423-424
connecting outlets, 426
creating interface, 425-426
enabling rotation, 424
releasing objects, 424
setting up, 423-425
view-swapping logic, 426-429
TabbedCalculation, 307
adding actions and outlets, 313-314
adding tab bar controller, 310-312
adding view controllers, 308-309
area calculation logic, 317-319
area view, 313-319
connecting actions, 317
connecting outlets, 317
setting up, 307-310
summary view, 323-326
volume calculation logic, 325-326
volume view, 319-323
testing
Interface Builder, 117
iPhone Simulator, 45-50
View-Based Application template, 148
tracing, Instruments tool, 614-619
Universal, 583
active devices, 588-590
device-specific view controllers, 584-588
setting up, 584
universal applications, 579-580, 590, 598-599
converting interfaces, 597
GenericViewController view controller class, 591-596
upgrading iPhone target, 596-597
Window-based template, 581-590
UniversalToo, 590, 596
GenericViewController, 590-592, 595
instantiating view controllers, 592-593

setting up, 590
views, 595-596
XIB files, 593-595
updating, 653
uploading, 647-649
Xcode, 27-28
building applications, 39, 41-42
editing code, 34-39
modifying project properties, 42, 45
navigating code, 34-39
project management, 28-32
removal of files and resources, 33-34
applicationWillEnterForeground method, 560
Apprater.com, 652
approvals, applications, submitting for, 642-649
AppSafari.com, 652
AppShopper.com, 652
AppStoreApps.com, 651
Apptism.com, 652
***Archives and Serializations Programming Guide for Cocoa*, 404**
area calculation logic, TabbedCalculation application, 317-319
area view, multi-view applications, implementing, 313-319
arrays, 94
artwork, applications, creating for, 630-633
attributes, 116
Accessibility settings, 116-117
buttons, editing, 166-167
date pickers, setting, 264-265
nonatomic, 62
retain, 62
text views, editing, 162
web views, setting, 212-213
Attributes Inspector, 115-116
Autosizing, disabling, 418
button attributes, editing, 166-167
Attributes Inspector command (Tools menu), 115
audio, Cupertino application
audio directions, 567-569
background modes key, 569-570
preparing for, 564-567
audio formats, Apple, 483
audio recordings
creating, 486-490
playing, 490-491
AudioToolbox framework, adding, 251, 564
Auto-Enable Return Key text input trait, 159
autocompletion, Xcode editor, 35-37
Automation instrument, 619
autorelease method, releasing objects, 75
Autosizing (Size Inspector), 115, 413-416
disabling, 418
masks, 597
AV Foundation framework, 477
Media layer, 85
availability, Quick Help results, 102
AVAudioPlayer versus MPMusicPlayerController, 502
axes, accelerometer, 452

B

background image resources, adding, 534
background modes key, Cupertino application, adding, 569-570
background touch, keyboard, hiding, 173-174
background-aware applications, 553-556, 576-577
background suspension, 559-560
backgrounding, disabling, 558-559
life cycle methods, 556-558
local notifications, implementing, 561-563
long-running background tasks, completing, 570-576
task-specific background processing, 564-570
backgrounding, 554-556, 576-577
disabling, 558-559
local notifications, 554-555
implementing, 561-563
long-running background tasks
completing, 570-576
task completion, 555-556
suspension, 554
handling, 559-560
task-specific background processing, 555, 564-570
BestFriend application, 509
actions, connecting, 512
Address Book framework, 512-518
map objects, 518-523
Message UI, 523-525
outlets, connecting, 512
setting up, 510-511
UI, creating, 511-512
blocks, 70
handler blocks, 456
Bluetooth, supplementation, 6
bookmarks, 38
breakpoints, 604, 606-607
Build and Run button, 40-41
Build command (Build menu), 40
build configurations (Xcode), 604
Build menu commands, Build, 40
building applications, 39-42
Active Configuration setting, 40
Build and Run button, 40-41
errors and warnings, 41-42
built-in capabilities, 505
Address Book frameworks, 505-506

Core Location framework, 529
Map Kit framework, 508
Message UI framework, 507
Bundle Identifiers, setting, 636-637
button bars, 120
buttons, 96, 152-154
action sheets, 247
actions, connecting, 171
Add Contact, 166
alerts, adding to, 238-241
attributes, editing, 166-167
Build and Run, 40-41
Check for Leaks Now, 618
Custom, 166
Detail Disclosure, 166
Done, 171
hiding keyboard, 172-173
Export Developer Profile, 21
images, setting custom, 167-170
Import Developer Profile, 21
Info Dark, 166
Info Light, 166
outlets, 183
overlap, 430
radio buttons, 200
Rounded Rect, 166
styled buttons, creating, 164-171
toolbars, adding and editing, 301-302

C

CA (certificate authority), 635-636
calculate method, 318, 322
calculating headings, Cupertino Compass application, 547
camera
controlling, 502
Image Picker, 492-495
cancelButtonTitle parameter (actionSheet), 247
cancelButtonTitle parameter (alertDialog), 237
capability requirements, applications, defining, 634
Capitalize text input trait, 159
cells
images, populating, 354-356
rows, 360
table view controllers, populating, 340-342
text, populating, 354-356
cellular technology, 529
centerMap method, 519
Certificate Assistant, 16-17
Certificate Assistant (Development Provisioning Assistant), 16-17
certificate authority (CA), 635-636
Certificate Revocation List (CRL), 635
certificate signing requests, generating and uploading, 16-17
CFNetwork framework, Core Services layer, 86
changes, orientation, reacting to, 460
check boxes, 200
Check for Leaks Now button, 618
child plane preference type, 376
chooseImage method, 492
ChosenColor outlet, 121
class files, text comments, adding, 64
class logic, FlashCards application, 385-386
class methods, definition, 56
classes. *see also* objects
core, 91-93
NSObject, 91
UIApplication, 91
UIControl, 92
UIResponder, 92
UIView, 92
UIViewController, 93
UIWindow, 92
data type, 93-96
NSArray, 94
NSDate, 95
NSDecimalNumber, 94-95
NSDictionary, 94
NSMutableArray, 94
NSMutableDictionary, 94
NSMutableString, 93
NSNumber, 94-95
NSString, 93
NSURL, 95-96
definition, 55
files, 31
interface, 96-98
UIButton, 96
UIDatePicker, 97
UILabel, 96
UIPicker, 97
UISegmentedControl, 97
UISlider, 97
UISwitch, 96
UITextField, 97
UITextView, 97
iPadViewController, 584
iPhoneViewController, 585
NSObject, 56
single, limitations, 130
singletons, 456
UIDevice, 588
View-Based Application template, 136, 138
classes subgroup (project groups), 30
cleanup, handling, 485-486
clearColor, web views, 220
clearView method, 305
CLLocation, 531
Cocoa, Cocoa Touch, compared, 83
Cocoa Touch, 24, 81-82, 90
Cocoa, compared, 83
core classes, 91-93
NSObject, 91
UIApplication, 91
UIControl, 92
UIResponder, 92
UIView, 92
UIViewController, 93
UIWindow, 92

data type classes, 93-96
NSArray, 94
NSDate, 95
NSDecimalNumber, 94-95
NSDictionary, 94
NSMutableArray, 94
NSMutableDictionary, 94
NSMutableString, 93
NSNumber, 94-95
NSString, 93
NSURL, 95-96
functionality, 82-83
interface classes, 96-98
UIButton, 96
UIDatePicker, 97
UILabel, 96
UIPicker, 97
UISegmentedControl, 97
UISlider, 97
UISwitch, 96
UITextField, 97
UITextView, 97
origins, 83
Cocoa Touch layer frameworks
Address Book UI, 85
Game Kit, 85
Map Kit, 84
Message UI, 85
UIKit, 84
code
adding to projects, 31-32
connection to user interfaces, 119
actions, 120-124
implementation, 120
launching IB from Xcode, 119-120
outlets, 120-123
spaghetti code, 130
stepping through, 608-611
code snapshots, 37-38
codecs, Apple support, 483
ColorChoice outlet, 121
ColorTilt application, 462
actions, adding, 463-464
CoreMotion framework, adding, 463
interface, preparing, 464-465
motion events, implementing, 466-471
outlets, adding, 463-464
properties, adding, 463-464
setting up, 462-465
commands
Build menu, Build, 40
File menu
Make Snapshot, 37
New Project, 28
Simulate Interface, 117-118
Snapshots, 37
Help menu
Developer Documentation, 98
Quick Help, 100
Layout menu
Add Horizontal Guide, 112
Add Vertical Guide, 112
Alignment, 113
project menu, New Smart Group, 31
Project menu, Set Active Build Configuration, Debug, 604
Run menu, Run, 40
Tools menu
Attributes Inspector, 115
Identity Inspector, 126
Library, 110
Size Inspector, 114
Xcode menu, Preferences, 100
comments, class files, adding to, 64
component numbers, constants, 282
componentsSeparatedByString method, 521
condition-based loops, 73
configuration
BestFriend application, 510-511
ColorTilt application, 462-465
Cupertino Compass application, 543-544
DateCalc application, 262-263
distribution, 638-639, 642
FieldButtonFun, 154
Flashlight application, 366-367
FlowerColorTable application, 333-337
FlowerInfoNavigator application, 345-346
FlowerWeb application, 205
Gestures application, 436-437
ImageHop, 182
MatchPicker application, 271-273
MediaPlayground application, 478-480
MultipleViews application, 296-297
Orientation application, 458
Reframe application, 416-417
ReturnMe application, 373-374
Scroller application, 222
segmented controls, 207-208
SimpleSpin application, 411-412
Swapper application, 423-425
TabbedCalculation application, 307-310
Universal application, 584
UniversalToo application, 590
view controller classes, 312
connections
actions, 190
BestFriend application, 512
buttons, 171
date pickers, 265
Flashlight application, 368
GettingAttention application, 234
MediaPlayground application, 480-482
switches, 211-212

outlets, 190
BestFriend application, 512
Flashlight application, 368
Gestures application, 439
GettingAttention application, 234
MatchPicker application, 275
MediaPlayground application, 480-482
Reframe application, 421-422
Swapper application, 426
text views, outlets, 164, 214
Connections Inspector, 123, 265
connectivity, iOS platform, 6
constants, component numbers, 282
contacts, Address Book frameworks, 513
Contacts application, 381
content types, web views, 202
Continue icon (debugger), controlling program execution, 609
Continue to Here option (gutter context menu), 610
controlHardware method, 467
controllers
multiple views, 149
MVC structure, 131-132
IBAction directive, 133-134
IBOutlet directive, 132
view, UIControl class, 93
controls
rotatable applications, reframing, 416-423
segmented, 201, 258
FlowerWeb application, 204-210
UISegmentedControl class, 97
segmented controls
choosing appearance, 208
configuring, 207-208
connecting to actions, 210
connecting to outlets, 209
sizing, 208
toolbars, adding to, 300-307
UIControl class, 92
convenience methods, 67-68
converting interfaces, universal applications, 597
copy and paste, text entry areas, 161
Core Animation instrument, 620
Core Audio framework, Media layer, 85
core classes, 91-93
NSObject, 91
UIApplication, 91
UIControl, 92
UIResponder, 92
UIView, 92
UIViewController, 93
UIWindow, 92
Core Data framework, 404-406
Core Services layer, 87
Core Data instrument, 619
Core Data Tutorial for iOS tutorial, 405
Core Foundation framework, Core Services layer, 87
Core Graphics framework, Media layer, 85
Core Location, 529, 550
Cupertino application, adding framework, 534
location manager, 530
Compass, 541-542
delegate protocol, 530-533
handling location errors, 532-533
location accuracy, 533-534
update filter, 533
location-aware applications, creating, 534-540
Core Location framework, Core Services layer, 87
Core Motion framework
accelerometer, reading, 456-458
ColorTilt application, framework, 463
Core Services layer, 87
gyroscope, reading, 456-458
motion manager, initializing, 466-467
***Core Motion Framework Reference*, 471**
Core OS layer, frameworks, 88
Core Services layer, frameworks, 86-88
Core Text framework, Media layer, 86
CoreGraphics framework, 84
Correction text input trait, 159
costs, Apple Developer Program, 8
count-based loops, 72
counter logic, SlowCount application, implementing, 573-574
countUp method, 574-575
CPU Sampler instrument, 619
Create iPhone/iPod Touch Version (Interface Builder), 597
CreateCardDelegate protocol, 398
CreateCardViewContoller, FlashCards application, adding to, 391-393
CRL (Certificate Revocation List), 635
Cupertino application, 534
audio
directions, 567-569
preparing for, 564-567
background image resources, adding, 534
background modes key, 569-570
Core Location framework, adding, 534
location manager delegate, implementing, 538-540
outlets, adding, 535-536
properties, adding, 535-536
protocols, adding, 535-536
task-specific background processing, 564-570
UI, creating, 536-537

Cupertino Compass application, 541-549
calculating heading, 547
direction image resources, 543
heading updates, 545-549
location manager headings, 541-542
outlets, adding, 543-544
properties, adding, 543-544
setting up, 543-544
UI, updating, 544-545
currentDevice method, 588
Custom button, 166
custom images, buttons, setting, 167-170
customization, user interfaces, 115
Accessibility settings, 116-117
Attributes Inspector, 115-116

D

data detectors, 164
data models, MVC structure, 134
data source methods
pickers, 278-279
table view controllers, 338-340
data source protocol, picker views, 260
data structures, MatchPicker application, 276-278
data type classes, 93-96
NSArray, 94
NSDate, 95
NSDecimalNumber, 94-95
NSDictionary, 94
NSMutableArray, 94
NSMutableDictionary, 94
NSMutableString, 93
NSNumber, 94-95
NSString, 93
NSURL, 95-96
data type objects, C language, 93
data types, 93
primitive data types, 78
datatip, variable examination, 608
date formats, strings, 268
date pickers, 258-263, 266-270
actions, connecting to, 265
adding, 263-265
attributes, setting, 264-265
calculating difference between two dates, 268
displaying date and time, 267-268
getting date, 267
DateCalc application, 261
adding date pickers, 263-265
interface, finishing, 266-267
setting up, 262-263
view controller logic, 267-270
dates, 95
dealloc method, 76, 147, 417, 424
Debug build configuration, 604
Debugger Console, 602
Debugger view (GNU Debugger), 612-614
DebuggerPractice application, 604-606, 612-614
breakpoints, setting, 606-607
Instruments, monitoring with, 614-620
profiling, Shark profiler, 620-626
stepping through code, 608-611
variable states, 608
watchpoints, setting, 611-612
debugging, 627
Xcode, 601
GNU Debugger, 603-614
Instruments tool, 614-619
NSLog function, 602-603
Shark profiler, 620-626
debugging tools, 601
***Debugging with GDB: The GNU Source-Level Debugger*, 627**
DebugPractice application, 615
decision making, 70
expressions, 70-71
if-then-else statements, 71
repetition with loops, 72-74
switch statements, 72
declaration
Quick Help results, 101
variables, 65
object data types, 66
primitive data types, 65-66
declination, 551
default state, switches, setting, 211
degrees, radians and rotation, 445
delegate parameter (actionSheet), 246
delegate parameter (alertDialog), 237
delegate protocol
location manager, 530-533
picker views, 260-261
describeInteger method, 605, 609
design
applications, 363-365
interfaces, 410-411
MVC structure, 130-131
destructiveButtonTitle parameter (actionSheet), 247
Detail Disclosure button, 166
detail view controller logic, FlowerInfoNavigator application, 350-353
detecting tilt, 462-471
Dev Center, 13
Developer Documentation command (Help menu), 98
Developer Program (Apple), 7-10
costs, 8
registration, 8-9
Developer Suite, 23-24
Interface Builder, 105-106
connecting interfaces to code, 119-124
Identity Inspector, 125-126
user interfaces, 110-117
XIB files, 107-110
iPhone Simulator, 45
esoteric conditions, 49-50
generating multitouch events, 48

launching applications, 46-47
rotation simulation, 48
Xcode, 27-28
building applications, 39, 41-42
editing code, 34-39
modifying project properties, 42, 45
navigating code, 34-39
project management, 28-32
removal of files and resources, 33-34
developer tools, installing, 10-11
Developer/Applications folder, 11
developers, 7
Apple Developer Program, 7-10
costs, 8
registration, 8-9
development provisioning profiles, 12
iOS developer tools, installing, 10-11
paid developer programs, joining, 10
technologies, 23-24
development devices, assigning, 14-15
development paradigms
imperative development, 54
OOP (object-oriented programming), 54-55
terminology, 55-57
Development Provisioning Assistant, 12-21
App ID, choosing, 14
Certificate Assistant, 16-17
certificate signing requests
generating, 16
uploading, 17
development devices, assigning, 14-15
installing provisioning profile, 20
launching, 13
multiple devices, 21
provisioning profiles
downloading, 18-19
installing, 20-21
naming and generating, 17
unique device identifiers, 12-13
development provisioning profiles
generation and installation, 12-21
testing, 21-22
device capability requirements, applications, defining, 634
Device feature (iPhone Simulator), 49
device identifiers, 12-13
device IDs, 14-15
device-specific view controllers, Universal application, adding, 584-588
deviceType outlet, 589
dictionaries, 94
dimensions, launch images, 583
direction image resources, Cupertino Compass application, adding, 543
directives, 59
#import
implementation files, 63
interface files, 59
@implementation, implementation files, 63
@interface, interface files, 59
@property, 133
interface files, 61-62
@synthesize, 133
implementation files, 63
IBAction, 133-134
IBOutlet, 132
display, iOS platform, 4-5
displaying
action sheets, 245-247
alerts, 236-237
images, 493-494
distribution, applications, 655-656
configuring for, 638-639, 642
distribution certificates, creating, 634-636
distribution profiles, 12
distribution provisioning profiles, creating, 638
doAccelerometer method, 469
doActionSheet method, 246
doAlert method, 561
Document icons (XIB files), 109-110
document sets, 100
Document window (XIB file), 107-109
documentation system
Cocoa Touch, 81, 90
core classes, 91-93
data type classes, 93-96
functionality, 82-83
interface classes, 96-98
origins, 83
Xcode, 45
documentation window, 98-99
doMultiButtonAlert method, 239
Done button, 171
hiding keyboard, 172-173
doRotation method, 470
doSound method, 251
double primitive data type, 65
downloading provisioning profiles, 18-19

E

editing
button attributes, 166-167
code, 34-39
text views, attributes, 162
toolbar buttons, 301-302
editor, Xcode, autocompletion, 35-37
electromagnetic compass, 529
email, built-in capabilities, 523-525
Message UI framework, 507

ending
implementation files, 64
interface files, 62
Energy Diagnostics instrument, 620
enterprise program (Apple Developer Program), 8
Enterprise provisioning, 656
errors
building applications, 41-42
definition, 41
***Event Handling Guide for iPhone OS*, 471**
Event Kit framework, Core Services layer, 87
events, motion events, implementing, 466-471
existing resources, adding to projects, 32
Export Developer Profile button, 21
expressions, 70-71
External Accessory framework, Core OS layer, 88

F

Facebook, applications, promoting, 651
feedback
iOS platform, 6
Xcode, errors and warnings, 41-42
fees, Apple Developer Program, 8
FieldButtonFun, 154
actions, 155-156
keyboard, hiding, 171-174
objects, releasing, 175
outlets, 155-156
setting up, 154
styled buttons, creating, 164-171
text fields, adding, 156-161
text views, adding, 161-164
view controller logic, implementing, 174-175
fields, alerts, adding to, 241-245
File Activity instrument, 620
File menu commands
Make Snapshot, 37
New Project, 28
Simulate Interface, 117-118
Snapshots, 37
file paths, 383-384
file structure, Objective-C, 58
file system
file paths, 383-384
sandbox, 381-384
storage, 384-399
archiving flash cards, 402-404
object archiving, 400-402
File's Owner icon (XIB files), 107
files
adding to projects, 32
header, 31
#import directive, 59
@interface directive, 59-60
@property directive, 61-62
ending, 62
method declaration, 60-61
implementation, 31, 62
#import directive, 63
@implementation directive, 63
@synthesize directive, 63
ending, 64
method implementation, 64
locating methods and properties, 35
project management, 28
adding existing resources to files, 32
adding new code files, 31-32
editing/navigating code, 34-39
identifying project type, 28-29
project groups, 30-31
removal of files from project, 33-34
removal from projects, 34
XIB (Interface Builder), 107
Document icons, 109-110
Document window, 107-109
View-Based Application template, 138-139
fireDate notification property, 561
first responder icon (XIB files), 108
FlashCards application, 384, 405
application logic, implementing, 394-399
class logic, implementing, 385-386
CreateCardViewContoller, adding, 391-393
flash cards, archiving, 402-404
interface, creating, 384-391
object archiving, implementing, 400-402
Flashlight application, 366-372
actions, connecting, 368
creating interface, 367
logic, 369-370
outlets, connecting, 368
preferences, 370-372
setting up, 366-367
float primitive data type, 65
floatValue method, 318
flow of program execution, GNU Debugger, 608-611
FlowerColorTable application, 332-333
adding outlets, 334
data source methods, 338-340
populating cells, 340-342
providing data to, 337-342
row touch events, 342-343
setting up, 333-337
table views, adding, 335-337
FlowerInfoNavigator application, 344-345
adding outlets, 351
adding properties, 351

adding web view, 352-353
detail view, 350-353
detail view controller logic, 351-352
navigation events, 356-357
providing data to, 346-350
root view table controllers, 353-356
setting up, 345-346
table data source methods, 354
UI, 357-358
FlowerView outlet, 121
FlowerWeb application, 204
actions, preparing, 205-206
adding segmented controls, 206-210
adding switches, 210-212
adding web views, 212-214
finishing interface, 215
outlets, preparing, 205-206
releasing objects, 220-221
setting up, 205
view controller logic, 216-220
format specifiers (strings), 602
Foundation framework, 84
Core Services layer, 87
foundPinch method, 444
foundRotation method, 446
frameworks
Address Book UI, 505-506
AudioToolbox, 251
adding, 564
AV Foundation, 477
Core Location, 518-523, 529
Map Kit, 508, 518-523
Media Player, 476-477
adding, 482
Message UI, 507, 523-525
technology layers
Cocoa Touch layer, 84-85
Core OS layer, 88
Core Services layer, 86-88
CoreGraphics, 84
Foundation, 84
Media layer, 85
UIKit, 84
Xcode documentation, 98-100
Quick Help, 100-101
frameworks subgroup (project groups), 31
functionality, Cocoa Touch, 82-83
functions. *see also* methods
ABPersonHasImageData, 517
ABRecordCopyValue, 515
NSLog, 603, 627

G

g (gravity) unit, accelerometer, 452
Game Kit framework, Cocoa Touch layer, 85
games, preferences, 405
gdb (GNU Debugger), 603-604
breakpoints, 604-607
Debugger view, 612-614
flow of program execution, 608-611
variable states, datatip, 608
watchpoints, 611-612
***GDB Pocket Reference*, 627**
generating provisioning profiles, 12-21
GenericViewController view controller class
creating universal applications, 591-596
adding device-specific views, 591
adding to application delegates, 591-592
implementation, 595
instantiating view controller, 592-593
iPhone and iPad views, 595-596
XIB files, 593, 595
UniversalToo application, adding to, 590-592
gesture recognizers, 434-439
pinch recognizer, 443-445
rotation recognizer, 445-447
shake recognizer, 447-448
swipe recognizer, 441-443
tap recognizer, 439-443
gestures, 434
gesture recognizers, 435-439
pinch recognizer, 443-445
rotation recognizer, 445-447
shake recognizer, 447-448
swipe recognizer, 441-443
tap recognizer, 439-443
multitouch gesture recognition, 434, 448-449
Gestures application, 435
interface, creating, 437-439
outlets, connecting, 439
pinch recognizer, implementing, 443-445
rotation recognizer, implementing, 445-447
setting up, 436-437
shake recognizer, implementing, 447-448
swipe recognizer, implementing, 441-443
tap recognizer, implementing, 439-443
GetFlower action, 121
getter methods, 61
GettingAttention application, 249
action sheets, 245
button press responses, 248-249
changing appearance, 247-248
displaying, 245-247
alerts
generating, 235-245
playing sounds, 250-253
System Sound Services, 250
vibrations, 253

connecting actions, 234
connecting outlets, 234
local notifications, implementing, 561-563
notification files, prepping, 232-233
notification interfaces, creating, 233
GNU Debugger (gdb), 603-604
breakpoints, 604-607
Debugger view, 612-614
flow of program execution, 608-611
variable states, datatip, 608
watchpoints, 611-612
Google Analytics, 652
Google Maps/Google Earth API, Map Kit map tiles, 508
GPS technology, 529
graphics, iOS platform, 4-5
gravity (g) unit, accelerometer, 452
group preference type, 376
grouped tables, 330-331
groups (projects), Xcode, 30-31
guides (IB layout tool), 112
gutter (Xcode), 604
gyroscope, 453-454, 472
reading, 456-458
updates
managing, 467-468
reacting to, 470-471

H

handler blocks, 70, 456
handling
background suspension, 559-560
location errors, location manager, 532-533
navigation events, 356-357
hardware
motion hardware, 451
accelerometer, 452-453
gyroscope, 453-458
requirements, 7
header files, 31, 58
#import directive, 59
@interface directive, 59-60
@property directive, 61-62
ending, 62
method declaration, 60-61
heading updates, Cupertino Compass application, 545-549
headings, Cupertino Compass application
calculating, 547
location manager, 541-542
updates, 545
Heavy view, Shark profiler results, 624
HelloNoun
object release, 147-148
setting up, 136
classes, 136-138
XIB files, 138-139
testing, 148
view controller
outlets and actions, 140-141
implementing, 146-147
Help menu commands
Developer Documentation, 98
Quick Help, 100
hideKeyboard method, 172-174
hiding keyboard, 171-174
hop button, ImageHop applications, adding, 191-192

I

iAD framework, 653
applications, 654-655
Cocoa Touch layer, 85
IBAction directive, 133-134
IBAction method, 513
IBOutlet directive, 132
icon files, universal applications, 582-583
Icon Files property, adding multiple items to, 44
icons, application icons, adding, 631-633
id return type (methods), 61
IDE (integrated development environment). *See* Xcode
Identity Inspector, 125-126
Identity Inspector command (Tools menu), 126
if-then-else statement, 71, 78
iLounge.com, 651
Image I/O framework, Media layer, 86
Image Picker, 477, 492-495
image views
animating, 186-187
sliders, 180
ImageHop application, 181-182
actions, 182-184
connecting, 190
animation resources, adding, 182
background suspension, handling, 559-560
hop button, adding, 191-192
image views, adding, 184-188
labels, adding, 191
outlets, 182-184
connecting, 190
releasing objects, 195
setting up, 182
sliders, adding, 188-190
user interface, finishing, 190-192
view controller logic, 193-195
images
buttons, setting, 167-170
cells, populating, 354-356
displaying, 493-494
imageWithData method, 515
imperative development, 54
implementation
GenericViewController class, 595
methods
convenience methods, 67-68

declaration of variables, 65-66
expressions and decision making, 70-74
messaging syntax, 68-70
object allocation and initialization, 67
view controller logic, 146-147
View-Based Application template, 135
implementation files, 31, 62
#import directive, 63
@implementation directive, 63
@synthesize directive, 63
ending, 64
method implementation, 64
implicit preferences, creating, 366-372
Import Developer Profile button, 21
Info Dark button, 166
Info Light button, 166
Info property list resource, 42
inheritance, OOP and, 55
initialization, objects, 67-68
initWithContentsOfURL:encoding: error method, 521
initWithQuestion:answer method, 385
initWithTitle parameter (actionSheet), 246
initWithTitle parameter (alertDialog), 237
input, 151, 179, 199-200
buttons, 152-154
iOS platform, 6-7
keyboard, hiding, 171-174
labels, 152-153
scrolling views, 203
segmented controls, 201
FlowerWeb application, 204
sliders, 180
adding, 188-190
image views, 180
styled buttons, creating, 164-171
switches, 200
FlowerWeb application, 204
text fields, 152-156
text views, 153-154, 161-164
views, 152
web views, 202-204
installation
development provisioning profile, 12-21
iOS developer tools, 10-11
provisioning profile, 20
provisioning profiles, 20-21
instance methods, definition, 56
instance variables, 59-60
declaration, 65
object data types, 66
primitive data types, 65-66
definition, 56
releasing, 76
text fields, alerts, 242-243
instances
definition, 56
MKMapView, 508
navigationController, 357
instantiation
definition, 56, 107
view controllers, 298-299
GenericViewController class, 592-593
universal applications, 586-588
Instruments Library, 619
Instruments tool, 614
available instruments, 619
leak detector, 614-618
int primitive data type, 65
integers, 73
integrated development environment (IDE). *See* Xcode
integration, 505, 526-527
Address Book frameworks, 505-506
BestFriend application, 509
Address Book framework, 512-518
connecting actions and outlets, 512
creating UI, 511-512
map objects, 518-523
Message UI, 523-525
setting up, 510-511
Core Location framework, 529
Map Kit framework, 508
mapping, 508-509
Message UI framework, 507
Interface Builder, 23, 51, 105-106
connecting interfaces to code, 119
actions, 120-124
implementation, 120
launching IB from Xcode, 119-120
outlets, 120-123
Create iPhone/iPod Touch Version, 597
Identity Inspector, 125-126
rotatable interfaces
Autosizing, 413-416
creating, 411-416
reframing controls, 416-423
setting up, 411-412
swapping views, 423-429
testing, 412-413
scrolling views, 228
user interfaces, 110
customization, 115-117
layout tools, 112-115
Objects Library, 110-111
simulation, 117
Xcode 4, 106
XIB files, 107-110
***Interface Builder User Guide*, 126**
interface classes, 96-98
UIButton, 96
UIDatePicker, 97
UILabel, 96
UIPicker, 97
UISegmentedControl, 97
UISlider, 97

interface classes

UISwitch, 96
UITextField, 97
UITextView, 97
interface files, 58
#import directive, 59
@interface directive, 59-60
@property directive, 61-62
ending, 62
method declaration, 60-61
interfaces
BestFriend application, creating, 511-512
ColorTilt application, preparing, 464-465
connection to code, 119
actions, 120-124
implementation, 120
launching IB from Xcode, 119-120
outlets, 120-123
converting (universal applications), 597
creating with Interface Builder, 110
layout tools, 112-115
Objects Library, 110-111
Cupertino application, creating, 536-537
Cupertino Compass application, updating, 544-545
customization, 115
Accessibility settings, 116-117
Attributes Inspector, 115-116
DateCalc application, finishing, 266-267
FlashCards application
creating, 384, 387-391
preparing, 386-387
Flashlight application, creating, 367
FlowerInfoNavigator application, finishing, 357-358
FlowerWeb application, finishing, 215
Gestures application, creating, 437-439
MatchPicker application, finishing, 274-275, 284-289
MediaPlayground application, creating, 480
notification interfaces, creating, 233
Orientation application, preparing, 459-460
Reframe application
creating, 417-422
reframing logic, 422-423
resizable, 407-408, 429-430
Autosizing, 413-416
creating, 411-416
designing, 410-411
setting up, 411-412
ReturnMe application, creating, 374
rotatable, 407-408, 429-430
Autosizing, 413-416
creating, 411-416
designing, 410-411
enabling, 408-409
reframing controls, 416-423
setting up, 411-412
swapping views, 423-429
testing, 412-413
simulation, 117
SlowCount application, creating, 572
static interface elements, 294-295
Swapper application, creating, 425-426
IOS developer tools
installing, 10-11
iOS platform, 3
application resource constraints, 5
backgrounding, 554-556
connectivity, 6
display and graphics, 4-5
feedback, 6
frameworks, Xcode documentation, 98-102
input, 6-7
"retain" count, 76
iOS SDK (Software Development Kit), 7
iPad Human Interface Guidelines, 127
iPad view, GenericViewController class, 595-596
iPadViewController class, 584
***iPhone Application Programming Guide*, 449, 558, 576**
iPhone Dev Center, 8, 10, 13
iPhone distribution certificates, creating, 634-636
iPhone OS
frameworks, Xcode documentation, 100
technology layers, 83
Cocoa Touch, 84-85
Core OS, 88
Core Services, 86-88
Media, 85
iPhone Provisioning Portal link, 13
iPhone Simulator, 23, 51, 433
Accessibility Inspector, 118
running applications in, 382
testing applications, 45, 148
esoteric conditions, 49-50
generating multitouch events, 48
Interface Builder, 117
launching applications, 46-47
rotation simulation, 48
iPhone target, upgrading, 596-597
iPhone view, GenericViewController class, 595-596
iPhoneApplicationList.com, 652
iPhoneApps.co.uk, 652
iPhoneViewController class, 585
iPod library, Media Picker, 495-501
iTunes. *see also* App Store
artwork, adding to applications, 630-631
iTunes Connect, sales, monitoring, 649-650

K

keyboard, hiding, 171-174
keyboard displays, customizing, text input traits, 159
Keyboard text input trait, 159
Keychain Access Certificate Assistant (Development Provisioning Assistant), 16-17
Keychain Access utility, 635
keychains, 14
keys, Launch image (iPad), 583

L

labels, 96, 152-153
ImageHop applications, adding, 191
lastAction, 272
matchResult, 272
landscape orientation, 408
lastAction label, 272
Launch image (iPad) key, 583
modifying project properties, 44-45
launch images
adding, 633
dimensions, 583
universal applications, 583
launching
applications in iPhone Simulator, 46-47
Development Provisioning Assistant, 13
Mac OS X Installer application, 11
layers, iPhone OS, 83
Cocoa Touch, 84-85
Core OS, 88
Core Services, 86-88
Media, 85
laying out Reframe application, 418-421
Layout menu commands
Add Horizontal Guide, 112
Add Vertical Guide, 112
Alignment, 113
layout tools, Interface Builder, 112
alignment, 113-114
guides, 112
selection handles, 112
Size Inspector, 114-115
leak detector (Instruments tool), 614-618
Leaks instrument, 619
"Learning Objective-C: A Primer" document, 77
Library command (Tools menu), 110
life cycle (applications), 88-90
life cycle methods, background-aware applications, 556-558
limitations, iOS platform, 5
loadFirstView method, 305-306
loading remote content, NSURL and requestWithURL, 202-203
loadView methods, 304
local notification, creating, 561-563
local notifications
backgrounding, 554-555
implementing, 561-563
scheduling, 561-563
location accuracy, location manager, 533-534
location errors, location manager, handling, 532-533
location manager (Core Location), 530
Cupertino application, implementing, 538-540
delegate protocol, 530-533
headings, Compass, 541-542
location accuracy, 533-534
location errors, handling, 532-533
update filter, 533
location services
Core Location framework, 518-523, 529
Map Kit framework, 508, 518-523
location-aware applications, creating, 534-540
locationManager:didUpdateHeading method, 568
locationManager:didUpdateToLocation:fromLocation method, 531
Lock feature (iPhone Simulator), 49
logic
Flashlight application, 369-370
view controllers, 146-147
implementing, 174-175
long pressing (gesture), 434
long-running background tasks, completing, 555-556, 570-576
loops, 72-74

M

Mac OS templates, 51
***Mac OS X Advanced Development Techniques*, 77**
Mac OS X Installer application, launching, 11
magnetic compass, 541-549
location manager headings, 541-542
Make Snapshot command (File menu), 37
***Making Your Application Location-Aware*, 549**
map display, 518, 522
Map Kit framework, 508, 518-523
Cocoa Touch layer, 84
Google Maps/Google Earth API, 508
map objects, BestFriend application, accessing, 518-523
map views, configuring, 512
mapping, 508-509
marketing applications, 649-655
iAds, 653-655
pricing, 653
social networks, 650-652
updates, 652-653
websites, 650-652
masks, autosizing, 597

MatchPicker application, 271
connecting outlets, 275
data structures, 276-278
finishing interface, 274-275
outlets, adding, 272
output labels, 275
picker views
adding, 273-274
providing data to, 275-281
reacting to choices, 281-284
protocols, conforming to, 271
releasing objects, 272-273
setting up, 271-273
UI, configuring, 284-289
matchResult label, 272
measurable axes, accelerometer, 452
media, rich media, 475-476, 501-502
AV Foundation framework, 477
Image Picker, 477
Media Player framework, 476-477
MediaPlayground application, 478-482
media files, MediaPlayground application, adding to, 483
Media layer frameworks, 85-86
Media Picker, 495-501
music player, 499-501
Media Player
framework, 476-477
adding, 482
movie player, 482-486
Media Player framework, Media layer, 86
MediaPlayground application, 478
actions, connecting, 480-482
audio recordings
creating, 486-490
playing, 490-491
cleanup, handling, 485-486
Image Picker, 492-495
interface, creating, 480
media files, adding, 483
Media Picker, 495-501
Media Player framework, adding, 482
movie playback, implementing, 483-485
movie player, 482-486
music player, 499-501
notifications, receiving, 485
outlets, connecting, 480-482
setting up, 478-480
memory, object release, 147-148
memory management
releasing instance variables, 76
releasing objects, 74-75
releasing rules, 76-77
retaining objects, 75-76
memory usage, applications, 615
menus, Overview drop-down, 604
message parameter (alertDialog), 237
Message UI framework, 507
BestFriend application, accessing, 523-525
Cocoa Touch layer, 85
messages, 56
messaging syntax, objects, 68-70
methods. *see also* functions
ABRecordCopyVal, 517
addTextField, 254
application:didFinishLaunching-WithOptions, 586, 592
applicationWillEnter-Foreground, 560
autorelease, 75
calculate, 318, 322
centerMap, 519
chooseImage, 492
clearView, 305
componentsSeparatedBy-String, 521
controlHardware, 467
countUp, 574-575
currentDevice, 588
data source methods, pickers, 278-279
dealloc, 76, 147, 417, 424
declaration in interface files, 60-61
definition, 35
describeInteger, 605, 609
doAccelerometer, 469
doActionSheet, 246
doAlert, 561
doMultiButtonAlert, 239
doRotation, 470
doSound, 251
floatValue, 318
foundPinch, 444
foundRotation, 446
getters, 61
hideKeyboard, 172-174
IBAction, 513
imageWithData, 515
implementation
convenience methods, 67-68
declaration of variables, 65-66
expressions and decision making, 70-74
messaging syntax, 68-70
object allocation and initialization, 67
implementation files, 64
initWithContentsOfURL:encod-ing:error, 521
initWithQuestion:answer, 385
loadFirstView, 305-306
loadView, 304
locating, 35
locationManager:didUpdate-Heading, 568
locationManager:didUpdateTo-Location:fromLocation, 531
motionEnded:withEvent, 448
orientationChanged, 461
pickerView:didSelectRow: inComponent, 282
pickerView:numberOfRowsIn-Component, 279

pickerView:titleForRow:forComponent, 280
presentModalViewController:animated, 398
recordAudio:, 487
release, object release, 147-148
return types
id, 61
void, 61
sendEmail, 524
setLightSourceAlphaValue, 370
setRegion:animated, 519
setters, 61
setToRecipients, 524
showDate, 265, 269
showNextCard, 396
startUpdatingLocation, 538
stopUpdatingLocation, 532
tableView:cellForRowAtIndexPath, 341
tableView:didSelectRowAtIndexPath method, 356
tableView:titleForHeaderInSection, 339
timeIntervalSinceDate:, 269
toggleFlowerDetail, 205, 218
updateRightWrongCounters, 396
viewDidLoad, 168-169, 186, 396, 440, 460, 488, 545, 567, 573-575, 605, 611, 617, 621

missing 20 points, orientation, 428
MKMapView instance, 508, 527
Mobile Safari, 651
Model-View-Controller (MVC), 24
models, MVC structure, 131
data models, 134
modifying project properties
launch image, 44-45
setting application icon, 43
status bar display, 45
motion data, accessing, 454-458
motion events, ColorTilt application, 466-471
motion hardware, 451
accelerometer, 452-453
gravity unit, 452
measurable axes, 452
reading, 456-458
gyroscope, 453-454
reading, 456-458
motion manager, Core Motion, initializing, 466-467
motion updates, receiving, 456
motionEnded:withEvent method, 448
movement, sensing, 469
movie playback, implementing, 483-485
movie player, 482-486
MPMusicPlayerController versus AVAudioPlayer, 502
multi-option alerts, creating, 238-241
multi-view applications
benefits, 294
static interface elements, 294-295
tab bars, 307
adding to, 310-312
adding view controllers, 308-309
area view, 313-319
setting up, 307-310
summary view, 323-326
volume view, 319-323
toolbars, 295-307
versus single-view applications, 293-295
multiple views, view controllers, 149
MultipleViews application, 295
actions, adding, 302-303
actions, connecting, 303
adding view controllers, 296-297
adding views, 296-297
outlets, adding, 302-303
outlets, connecting, 303
setting up, 296-297
toolbar controls, adding, 300-307
view controllers, instantiating, 298-299
view switch methods, 303-305
multitouch events, iPhone Simulator, 48
multitouch gesture recognition, 434, 448-449
gesture recognizers
pinch recognizer, 443-445
rotation recognizer, 445-447
shake recognizer, 447-448
swipe recognizer, 441-443
tap recognizer, 439-443
multivalue options, pickers, 257
multivalue preference type, 376
MVC (Model-View-Controller), 24, 129-131
application design, 130-131
controllers, 131-132
data models, 134
models, 131
View-Based Application template, 135
creating views, 141-146
implementation, 135
implementation of view controller logic, 146-147
object release, 147-148
project setup, 136-139
testing application, 148
view controller outlets and actions, 140-141
views, 131-132

N

naming provisioning profiles, 17
NativeiPhoneApps.com, 652
navigating code, 34-39
navigation controllers, 329-331, 360
navigation events, FlowerInfoNavigator application, handling, 356-357

navigation-based applications, 344-345. *see also* FlowerInfoNavigator application
navigationController instance, 357
nested messaging, 69
New project command (File menu), 28
New Smart Group command (Project menu), 31
newBFF action, 510
NeXTSTEP platform, 83
nil value, 69
nonatomic attribute, 62
Notes application, 381
notification files, prepping, 232-233
notification interfaces, creating, 233
notification properties, 561
notifications, 231-232
  alerts, 249
    action sheets, 245-248
    button press responses, 248-249
    generating, 235-245
    playing sounds, 250-253
    System Sound Services, 250
    vibrations, 253
  creating, 561-563
  local notifications
    backgrounding, 554-555
    implementing, 561-563
  notification interfaces, creating, 233
  prepping notification files, 232-233
  receiving, 485
  scheduling, 561-563
NSArray class, 94
NSDate class, 95
NSDecimalNumber class, 94-95
NSDictionary class, 94
NSLog function (debugging tool), 602-603, 627
NSMutableArray class, 94
NSMutableDictionary class, 94
NSMutableString class, 93
NSNumber class, 94-95
NSObject class, 56, 91
NSString class, 93
NSUInteger properties, 385
NSURL class, 95-96
  remote content, loading, 202-203
NSUserDefaults API, 371
numbers, 94-95

O

Object Allocations instrument, 619
object archiving, implementing, 400-402
object data types, declaration of variables, 66
"Object-Oriented Programming with Objective-C" document, 77
object-oriented programming (OOP). *See* OOP (object-oriented programming)
Objective-C, 24, 53, 57-58, 64, 78
  decision-making
    expressions, 70-71
    if-then-else statements, 71
    repetition with loops, 72-74
    switch statements, 72
  file structure, 58
    header files, 58-62
    implementation files, 62-64
  integers, 73
  memory management
    releasing instance variables, 76
    releasing objects, 74-75
    releasing rules, 76-77
    retaining objects, 75-76
  messaging syntax, 68-69
    blocks, 70
    nested messaging, 69
  method implementation, declaration of variables, 65-66
  object allocation and initialization, convenience methods, 67-68
  statements, 57
"Objective-C 2.0 Programming Language" document, 77
objects. *see also* classes
  adding to views, 141-144
  allocation and initialization, 67-68
  application, UIApplication class, 91
  definition, 56
  instantiation, 107
  messaging syntax, 68-70
  Reframe application, releasing, 417
  releasing, 67, 78, 147-148
    convenience methods, 67-68
    FieldButtonFun, 175
    FlowerWeb application, 220-221
    ImageHop application, 195
    MatchPicker application, 272-273
    memory management, 74-75
  retaining, 75-76
  Scroller application, releasing, 226
  scrolling views, adding, 223-224
  Swapper application, releasing, 424
  switch, UISwitch class, 96
  window, UIWindow class, 92
Objects Library (Interface Builder), 110-111
Online Certificate Status Protocol (OSCP), 635
onscreen controls, UIControl class, 92
OO programs, 130

OOP (object-oriented programming), 53-55, 130
Objective-C, 24, 53, 57-58, 64
blocks, 70
decision-making, 70-74
declaration of variables, 65-66
file structure, 58-64
memory management, 74-77
messaging syntax, 68-69
object allocation and initialization, 67-68
terminology, 55-57
Open GL ES instrument, 620
OpenGL ES framework, Media layer, 86
OpenStep platform, 83
orientation
accessing data, 454-458
changes, reacting to, 460
determining, 461
sensing, 458-461
tilt, detecting, 462-471
Orientation application
changes, reacting to, 460
interface, preparing, 459-460
orientation, determining, 461
setting up, 458
orientation notifications, requesting, UIDevice, 455
orientationChanged method, 461
orientations (screens), 408
origins, Cocoa Touch, 83
OSCP (Online Certificate Status Protocol), 635
other sources subgroup (project groups), 30
otherButtonTitles parameter (actionSheet), 247
otherButtonTitles parameter (alertDialog), 237
outlets, 121
BestFriend, connecting, 512
buttons, 183
ChosenColor, 121
ColorTilt, adding, 463-464
connecting, 190
GettingAttention application, 234
MatchPicker application, 275
Cupertino, adding, 535-536
Cupertino Compass, adding, 543-544
deviceType, 589
FieldButtonFun, 155-156
Flashlight, connecting, 368
FlowerColorTable, adding to, 334
FlowerInfoNavigator, adding, 351
FlowerView, 121
FlowerWeb, preparing, 205-206
Gestures, connecting, 439
ImageHop
adding hop button, 191-192
adding image views, 184-188
adding labels, 191
adding sliders, 188-190
connecting outlets, 190
finishing interface, 190-192
preparing, 182-184
Internet Builder, 120-121
connections, 122-123
MatchPicker, adding to, 272
MediaPlayground, connecting, 480, 482
MultiViews, adding to, 302-303
padViewController, 585
Reframe application
adding, 416-417
adding properties, 416-417
connecting, 421-422
laying out, 418-421
Scroller, preparing, 222-223
segmented controls, connecting, 209
Swapper application
adding, 423-424
connecting, 426
switches, 205
TabbedCalculation application
adding, 313-314
connecting, 317
text views, connecting, 164
view controllers, 140-141
connection points, 144-146
web views, connecting to, 214
output, 151, 179, 199-200
buttons, 152-154
image views, adding, 184-188
keyboard, hiding, 171-174
labels, 152-153
scrolling views, 203
segmented controls, 201
FlowerWeb application, 204
styled buttons, creating, 164-171
switches, 200
FlowerWeb application, 204
text fields, 152-154
actions, 155-156
adding, 156-161
outlets, 155-156
text views, 153-154
adding, 161-164
views, 152
web views, 202-203
FlowerWeb application, 204
output labels, 120
MatchPicker application, 275
overlap, buttons, 430
Overview drop-down menu, 604

P

padViewController outlet, 585
paid developer programs, joining, 10
panning (gesture), 434
parameters
definition, 56
Quick Help results, 101
parent classes, 56

patterns, 366
photo library, Image Picker, 492-495
photographs, displaying, 493-494
picker views, 259, 270-275, 284-289
adding, 273-274
choices, reactions, 281-284
outlets, 272
output labels, 275
protocols, 259-260, 271
data source protocol, 260
delegate protocol, 260-261
providing data to, 275-281
pickers, 257-258, 289-290
data source methods, 278-279
date pickers, 258-259, 261-263, 266-270
adding, 263-265
calculating difference between two dates, 268
connecting to actions, 265
displaying date and time, 267-268
getting date, 267
setting attributes, 264-265
multivalue options, 257
picker views, 259, 270-275, 284-289
adding, 273-274
outlets, 272
output labels, 275
protocols, 259-261, 271
providing data to, 275-281
reacting to choices, 281-284
UIDatePicker/UIPicker class, 97
pickerView:didSelectRow:inComponent method, 282
pickerView:numberOfRowsInComponent method, 279
pickerView:titleForRow:forComponent method, 280
pin annotation view, creating, 522
pinch gesture recognizer, implementing, 443-445
pinching (gesture), 434
placeholder text, 158
plain tables, 330
playback, movie, implementing, 483-485
playing alerts, 250-253
sounds with vibrations, 253
system sounds, 251-252
plist files, 371
universal applications, 581
icon files, 582-583
launch images, 583
pointers, 66
populating data structures, 277-278
portrait orientation, 408
portrait upside-down orientation, 408
Position setting (Size Inspector), 114
pragma marks, adding, 39
preferences, 363
applications, 366-372
games, 405
implicit preferences, creating, 366-372
reading, 371-372
storing, 370-371
system settings, 372-374
settings bundles, 375-381
Preferences command (Xcode menu), 100
premature optimization, 621
presentModalViewController:animated method, 398
pressing (gesture), 434
pricing applications, 653
primitive data types, 78
declaration of variables, 65-66
procedural programming, 54
products subgroup (project groups), 31
profiles
applications, preparing, 643-647
development provisioning
generation and installation, 12-21
testing, 21-22
development provisioning profiles, 12
"distribution" profiles, 12
profiling applications, Shark profiler, 620-626
program execution, GNU Debugger, 608-611
***Programming in Objective-C 2.0*, Second Edition, 77**
programs. *See* applications
project groups, subgroups, 30
project management, Xcode, 28
adding existing resources, 32
adding new code files, 31-32
creating a new project, 28-29
project groups, 30-31
Project menu commands, New Smart Group, 31
projects. *see also* applications
BestFriend application, 509
Address Book framework, 512-518
connecting actions and outlets, 512
creating UI, 511-512
map objects, 518-523
Message UI, 523-525
setting up, 510-511
ColorTilt, 462
adding actions and outlets, 463-464
CoreMotion framework, 463
motion events, 466-471
preparing interface, 464-465
setting up, 462-465

Cupertino, 534
audio directions, 567-569
background image resources, 534
background modes key, 569-570
Core Location framework, 534
creating UI, 536-537
location manager, 538-540
outlets, 535-536
preparing for audio, 564-567
properties, 535-536
protocols, 535-536
task-specific background processing, 564-570
Cupertino Compass
calculating heading, 547
direction image resources, 543
heading updates, 545-549
outlets, 543-544
properties, 543-544
setting up, 543-544
updating UI, 544-545
DateCalc, 261
adding date pickers, 263-265
finishing interface, 266-267
setting up, 262-263
view controller logic, 267-270
Debugger Practice, 604-606, 612-614
Instruments, 614-620
profiling, 620-626
setting breakpoints, 606-607
setting watchpoints, 611-612
stepping through code, 608-611
variable states, 608
distribution, configuring for, 638-639, 642
FieldButtonFun
actions, 155-156
adding text fields, 156-161
adding text views, 161-164
creating styled buttons, 164-171
hiding keyboard, 171-174
outlets, 155-156
releasing objects, 175
setting up, 154
view controller logic, 174-175
FlashCards, 384
application logic, 394-399
archiving flash cards, 402-404
class logic, 385-386
CreateCardViewContoller, 391-393
creating interface, 384, 387-391
object archiving, 400-402
preparing interface, 386-387
Flashlight, 366-372
connecting actions and outlets, 368
creating interface, 367
logic, 369-370
reading preferences, 371-372
setting up, 366-367
storing preferences, 370-371
FlowerColorTable, 332-333
adding outlets, 334
adding table views, 335-337
data source methods, 338-340
populating cells, 340-342
providing data to, 337-342
row touch events, 342-343
setting up, 333-337
FlowerInfoNavigator, 344-345
adding outlets and properties, 351
adding web view, 352-353
detail view, 350-353
detail view controller logic, 351-352
navigation events, 356-357
providing data to, 346-350
root view table controllers, 353-356
setting up, 345-346
table data source methods, 354
UI, 357-358
FlowerWeb, 204
finishing interface, 215
preparing actions outlets, 205-206
releasing objects, 220-221
segmented controls, 206-210
setting up, 205
switches, 210-212
view controller logic, 216-220
web views, 212-214
Gestures, 435
connecting outlets, 439
creating interface, 437-439
pinch recognizer, 443-445
rotation recognizer, 445-447
setting up, 436-437
shake recognizer, 447-448
swipe recognizer, 441-443
tap recognizer, 439-443
GettingAttention, 249
action sheets, 245-249
connecting actions and outlets, 234
creating notification interface, 233
generating alerts, 235-245
local notifications, 561-563

playing sounds, 250-253
prepping notification files, 232-233
System Sound Services, 250
vibrations, 253
HelloNoun
creating views, 141-146
object release, 147-148
setting up, 136-139
testing, 148
view controller logic, 146-147
view controller outlets and actions, 140-141
ImageHop, 181-182
actions, 182-184
adding animation resources, 182
adding hop button, 191-192
adding image views, 184-188
adding labels, 191
adding sliders, 188-190
background suspension, 559-560
connecting actions, 190
connecting outlets, 190
finishing interface, 190-192
outlets, 182-184
releasing objects, 195
setting up, 182
view controller logic, 193-195
MatchPicker, 271
adding picker views, 273-274
configuring UI, 284-289
connecting outlets, 275
data structures, 276-278
finishing interface, 274-275
outlets, 272
output labels, 275
protocols, 271
providing data to, 275-281
reacting to choices, 281-284
releasing objects, 272-273
setting up, 271-273
MediaPlayground, 478
adding media files, 483
adding Media Player framework, 482
connecting actions and outlets, 480-482
creating audio recordings, 486-490
creating interface, 480
handling cleanup, 485-486
Image Picker, 492-495
Media Picker, 495-501
movie playback, 483-485
movie player, 482-486
music player, 499-501
playing audio recordings, 490-491
receiving notifications, 485
setting up, 478-480
MultipleViews, 295
adding actions and outlets, 302-303
adding toolbar controls, 300-307
adding view controllers, 296-297
adding views, 296-297
connecting actions, 303
connecting outlets, 303
instantiating view controllers, 298-299
setting up, 296-297
view switch methods, 303-305
Orientation
determining orientation, 461
orientation changes, 460
preparing interface, 459-460
setting up, 458
Reframe, 416
adding outlets and properties, 416-417
connecting outlets, 421-422
creating interface, 417-422
disabling Autosizing, 418
laying out, 418-421
reframing logic, 422-423
releasing objects, 417
setting up, 416-417
ReturnMe, 372
creating interface, 374
setting up, 373-374
settings bundles, 375-381
Scroller, 221
adding scroll views, 223-225
preparing outlets, 222-223
releasing objects, 226
scrolling behavior, 225-226
setting up, 222
SimpleSpin, 411-416
setting up, 411-412
testing, 412-413
SlowCount
counter logic, 573-574
creating UI, 572
long-running background tasks, 570-576
Swapper, 423
adding outlets and properties, 423-424
connecting outlets, 426
creating interface, 425-426
enabling rotation, 424
releasing objects, 424
setting up, 423-425
view-swapping logic, 426-429
TabbedCalculation, 307
adding actions and outlets, 313-314
adding tab bar controller, 310-312
adding view controllers, 308-309

area calculation logic, 317-319
area view, 313-319
connecting actions, 317
connecting outlets, 317
setting up, 307-310
summary view, 323-326
volume calculation logic, 325-326
volume view, 319-323
Universal application, 583
active devices, 588-590
device-specific view controllers, 584-588
setting up, 584
UniversalToo application, 590, 596
GenericViewController, 590-592, 595
setting up, 590
view controllers, 592-593
views, 595-596
XIB files, 593-595
View-Based Application template, 135
promoting applications, 649-655
iAds, 653-655
pricing, 653
social networks, 650-652
updates, 652-653
websites, 650-652
properties
animationDuration, 189
ColorTilt application, adding, 463-464
Cupertino application, adding, 535-536
Cupertino Compass application, adding, 543-544
definition, 56
FlowerInfoNavigator application, adding, 351
locating, 35
modifying, 42, 45
launch image, 44-45
setting application icon, 43
status bar display, 45
NSUInteger, 385
Swapper application, adding, 423-424
Property List Editor, 371
protocols
ABPeoplePickerNavigationControllerDelegate, 513
CreateCardDelegate, 398
Cupertino application, adding, 535-536
definition, 60
MatchPicker application, conforming to, 271
picker views, 259-261
UIPickerView, 290
Provisioning Portal link, 13
provisioning profiles, 12
Development Provisioning Assistant
downloading, 18-19
installing, 20-21
naming and generating, 17
generation and installation, 12-21
testing, 21-22
push buttons, 120

Q

Quartz Core framework, Media layer, 86
Quick Help (Xcode), 100-101
Quick Look framework, Core Services layer, 87

R

radians, degrees, 445
radio buttons, 200
reactions, orientation changes, 460
reading
accelerometer, 456-458
gyroscope, 456-458
preferences, 371-372
recordAudio: method, 487
recordings (audio)
creating, 486-490
playing, 490-491
Reframe application, 416
Autosizing, disabling, 418
Interface, creating, 417-422
laying out, 418-421
objects, releasing, 417
outlets
adding, 416-417
connecting, 421-422
properties, adding, 416-417
reframing logic, 422-423
setting up, 416-417
reframing
controls, rotatable applications, 416-423
interfaces, 410
reframing logic, implementing, 422-423
registration, Apple Developer Program, 8-9
related API, Quick Help results, 102
related documents, Quick Help results, 102
Release build configuration, 604
release method, object release, 147-148
release of objects, 67
convenience methods, 67-68
memory management, 74-75
releasing objects, 78
FieldButtonFun application, 175
ImageHop application, 195
instance variables, memory management, 76
MatchPicker application, 272-273
objects, FlowerWeb application, 220-221
Reframe application, 417
rules, 76-77
Scroller application, 226
Swapper application, 424

remote content, loading, NSURL and requestWithURL, 202-203
removing breakpoints, 606
repeatInterval notification property, 561
repetition, loops, 72-74
requesting orientation notifications, UIDevice, 455
requestWithURL, remote content, loading, 202-203
requirements, hardware, 7
resources
adding to projects, 32
removal from projects, 33-34
Resources subgroup (project groups), 31
responders, UIResponder class, 92
responses, action sheets, button presses, 248-249
results, Shark profiler, 624-626
retain attribute, 62
"retain" count, 76
retaining objects, memory management, 75-76
Return Key text input trait, 159
return types (methods), 61
return value, Quick Help results, 102
ReturnMe application, 372, 405
interface, creating, 374
setting up, 373-374
settings bundles, creating, 375-381
reverse geocoding, 508
rich media, 475-476, 501-502
AV Foundation, framework, 477
Image Picker, 477
Media Player, framework, 476-477
MediaPlayground application, 478
adding media files, 483
adding Media Player framework, 482
cleanup, 485-486
connecting outlets, 480, 482
creating audio recordings, 486-490
creating interface, 480
Image Picker, 492-495
Media Picker, 495-501
movie playback, 483-485
movie player, 482-486
music player, 499-501
playing audio recordings, 490-491
receiving notifications, 485
setting up, 478-480
Robbin, Arnold, 627
root class, NSObject, 91
root view table controllers, FlowerInfoNavigator application, implementing, 353-356
rotatable interfaces, 407-408, 429-430
Autosizing, 413-416
controls, reframing, 416-423
creating, 411-416
designing, 410-411
enabling, 408-409
setting up, 411-412
swapping views, 423-429
testing, 412-413
rotating (gesture), 434
rotation
degrees, 445
testing with iPhone Simulator, 48
rotation gesture recognizer, implementing, 445-447
Rounded Rect buttons, 166
row touch events, FlowerColorTable application, 342-343
rows, cells, 360
rules, releasing, 76-77
Run command (Run menu), 40

S

sales, monitoring, iTunes Connect, 649-650
sample code, Quick Help results, 102
sandbox (Apple), 381-384
scaling factors, 4
scaling web pages, 214
scheduling notifications, 561-563
screen orientations, 408
Scrolling application, 221
adding scroll views, 223-225
preparing outlets, 222-223
releasing objects, 226
scrolling behavior, implementing, 225-226
setting up, 222
scrolling behavior, Scroller application, 225-226
scrolling options, text views, setting up, 163-164
scrolling views, 203, 221
Interface Builder, 228
objects, adding, 223-224
Scroller application, 221
adding, 223-225
preparing outlets, 222-223
setting up, 222
width, 226
SDK (Software Development Kit), 7
search results, Xcode documentation, 100
Secure text input trait, 159
Security framework, Core OS layer, 88
segmented controls, 120, 201, 258
choosing appearance, 208
configuring, 207-208
connecting to actions, 210
connecting to outlet, 209
FlowerWeb application, 204
adding, 206-210
sizing, 208
UISegmentedControl class, 97
selection handles (IB layout tool), 112
self.view, 304
sendEmail action, 510
sendEmail method, 524
sender variable, 172
sensing movement, 469

Set Active Build Configuration, Debug command (Project menu), 604
setLightSourceAlphaValue method, 370
setRegion:animated method, 519
setter methods, 61
setting
animation speed, 193-195
default state, switches, 211
images, buttons, 167-170
web view attributes, 212-213
Setting Application Schema References in the iPhone Reference Library tutorial, 405
settings bundles, creating, 375-381
setToRecipients method, 524
Shake Gesture feature (iPhone Simulator), 49
shake gesture recognizer, implementing, 447-448
shaking (gesture), 434
Shark profiler, 620
attaching to an application, 621-624
interpretation of results, 624-626
showDate action method, 265
showDate method, 269
showNextCard controller, 395
showNextCard method, 396
SimpleSpin application, 411-416
Autosizing, 413-416
setting up, 411-412
testing, 412-413
Simulate Hardware Keyboard feature (iPhone Simulator), 49
Simulate Interface command (File menu), 117-118
Simulate Memory Warning feature (iPhone Simulator), 49
simulation, user interfaces, 117
Simulator, testing applications, 148
simulators, 46, 148
single classes, limitations, 130
single-view applications versus multi-view applications, 293-295
singletons, 56, 366, 456
sizable interfaces, 407-408, 429-430
Autosizing, 413-416
creating, 411-416
designing, 410-411
setting up, 411-412
Size Inspector (IB layout tool), 114-115
Autosizing, 413-416
Size Inspector command (Tools menu), 114
Size setting (Size Inspector), 114
sizing segmented controls, 208
slider preference type, 376
sliders, 180
image views, 180
UISlider class, 97
vertical, 197
SlowCount application
counter logic, implementing, 573-574
long-running background tasks, completing, 570-576
UI, creating, 572
smart groups, 31
snapshots, 37-38
Snapshots command (File menu), 37
social networks, applications, promoting, 650-652
Software Development Kit (SDK), 7
sound constants, Cupertino application, adding, 566-567
soundName notification property, 561
sounds
alerts, playing, 250-253
system sounds, creating and playing, 251-252
vibrations, playing with, 253
spaghetti code, 130
speed, animations, setting, 193-195
Stallman, Richard, 627
standard program (Developer Program), 8
startAnimating method, 188
starting animations, 187-188
startUpdatingLocation method, 538
statements
if-then-else, 71, 78
Objective-C, 57
switch, 72, 339
static interface elements, 294-295
status bar, 428
status bar display, modifying project properties, 45
Step Into icon (debugger), controlling program execution, 609
Step Out icon (debugger), controlling program execution, 609
Step Over icon (debugger), controlling program execution, 609
stepping through code, 608-611
stopping animations, 187-188
stopUpdatingLocation method, 532
storage
application data, 382-383
file system, 384-399
flash cards, 402-404
object archiving, 400-402
preferences, 370-371
Store Kit framework, Core Services layer, 88
***String Programming Guide for Cocoa*, 602**
strings, 93
date formats, 268
format specifiers, 602
structure, MVC, 130-131
styled buttons, creating, 164-171
subclasses, 55
subgroups, project groups, 30
submissions, applications, 642-649
subviews, text fields, alerts, 243-244
summary view, multi-view applications, implementing, 323-326

superclasses, 56
supplementation, WiFi, 6
supported content types, web views, 202
suspension backgrounding, 554
Swapper application, 423
interface, creating, 425-426
objects, releasing, 424
outlets
adding, 423-424
connecting, 426
properties, adding, 423-424
rotation, enabling, 424
setting up, 423-425
view-swapping logic, implementing, 426-429
swapping views, 410
rotatable applications, 423-429
Swapper application, 426-429
swipe gesture recognizer, implementing, 441-443
swiping (gesture), 434
switch objects, UISwitch class, 96
switch statements, 72, 339
switches, 200
actions, connecting to, 211-212
default state, setting, 211
FlowerWeb application, 204
adding, 210-212
outlets, 205
syntax, expressions, 71
System Configuration framework, Core Services layer, 88
System framework, Core OS layer, 88
system settings, 372-374
settings bundles, creating, 375-381
System Sound Services, 250
system sounds, creating and playing, 251-252
System Usage instrument, 620

T

tab bars, 295, 326-327
multi-view applications, 307
adding to, 310-312
adding view controllers, 308-309
area view, 313-319
setting up, 307-310
summary view, 323-326
volume view, 319-323
versus toolbars, 328
TabbedCalculation application, 307
actions
adding, 313-314
connecting, 317
area calculation logic, 317-319
area view, 313-319
outlets
adding, 313-314
connecting, 317
setting up, 307-310
summary view, 323-326
tab bar controllers, adding, 310-312
view controllers, adding, 308-309
volume calculation logic, 325-326
volume view, 319-323
table data source methods, FlowerInfoNavigator application, 354
table views, 329-330
FlowerColorTable application, 332-333
adding outlets, 334
adding table views, 335-337
adding to, 335-337
data source methods, 338-340
populating cells, 340-342
providing data to, 337-342
row touch events, 342-343
setting up, 333-337
tables, 330
grouped, 330-331
plain, 330
providing data to, 360
rows, cells, 360
tableView:cellForRowAtIndexPath method, 341
tableView:didSelectRowAtIndexPath method, 356
tableView:titleForHeaderInSection method, 339
tap gesture recognizer, implementing, 439-443
tapping (gesture), 434
targets, 580
task completion, long-running tasks, 555-556
task-specific background processing, 555, 564-570
technologies
Apple Developer Suite, 23-24
Interface Builder, 105-126
iPhone Simulator, 45-50
Xcode, 27-42, 45
Cocoa Touch, 81, 90
core classes, 91-93
data type classes, 93-96
functionality, 82-83
interface classes, 96-98
origins, 83
developers, 23-24
MVC structure, 129-131
application design, 130-131
controllers, 132-134
data models, 134
View-Based Application template, 135-148
views, 132
Objective-C, 53, 57-58, 64
blocks, 70
decision-making, 70-74
declaration of variables, 65-66
file structure, 58-64

memory management, 74-77
messaging syntax, 68-69
object allocation and initialization, 67-68
technology layers, iPhone OS, 83
Cocoa Touch, 84-85
Core OS, 88
Core Services, 86, 88
Media, 85
templates
Mac OS, 51
View-Based Application template, 135
creating views, 141-146
implementation, 135
implementation of view controller logic, 146-147
object release, 147-148
project setup, 136-139
testing application, 148
view controller outlets and actions, 140-141
Xcode, 28
testing
development provisioning profile, 21-22
SimpleSpin application, 412-413
testing applications
iPhone Simulator, 45
esoteric conditions, 49-50
generating multitouch events, 48
interface simulation, 117
launching applications, 46-47
rotation simulation, 48
View-Based Application template, 148
text, cells, populating, 354-356
text comments, class files, adding to, 64
text entry areas, copy and paste, 161
text field preference type, 376
text fields, 152-154
alerts
accessing, 244-245
instance variables, 242-243
subviews, 243-244
FieldButtonFun application
actions, 155-156
adding, 156-161
outlets, 155-156
setting up, 154
UITextField/UITextView class, 97
text input traits, keyboard displays, customizing, 159
text views, 152-154, 177
attributes, editing, 162
FieldButtonFun, adding, 161-164
outlets, connecting, 164
scrolling options, setting up, 163-164
tilt, detecting, 462-471
Time Profiler instrument, 619
timeIntervalSinceDate: method, 269
timer mode, UIDatePicker, 290
timeZone notification property, 561
title preference type, 376
Toggle In-Call Status Bar feature (iPhone Simulator), 49
toggle switch preference type, 376
toggleFlowerDetail, 205
toggleFlowerDetail method, 205, 218
toolbars, 294, 326-327
buttons, adding and editing, 301-302
multi-view applications, 295-307
MultipleViews application
adding actions, 302-303
adding controls, 300-307
adding outlets, 302-303
connecting actions, 303
connecting outlets, 303
view switch methods, 303-305
versus tab bars, 328
tools
Apple Developer Suite, 23-24
Interface Builder, 105-126
iPhone Simulator, 45-50
Xcode, 27-42, 45
Cocoa Touch, 24, 81, 90
core classes, 91-93
data type classes, 93-96
functionality, 82-83
interface classes, 96-98
origins, 83
debugging, 601
Instruments, 614-619
MVC (Model-View-Controller), 24, 129-131
application design, 130-131
controllers, 132-134
data models, 134
View-Based Application template, 135-148
views, 132
Objective-C, 24, 53, 57-58, 64
blocks, 70
decision-making, 70-74
declaration of variables, 65-66
file structure, 58-64
memory management, 74-77
messaging syntax, 68-69
object allocation and initialization, 67-68
universal applications, 596-597
Tools menu commands
Attributes Inspector, 115
Identity Inspector, 126
Library, 110
Size Inspector, 114

tracing applications, Instruments tool, 614-619
Tree view, Shark profiler results, 624
tutorials (Apple), 177
TV Out feature (iPhone Simulator), 49
Twitter, applications, promoting, 651

U

UIAlertView class, 237
UIApplication class, 91
UIButton class, 96, 152-154
UICatalog class, 177, 197
UIControl class, 92
UIDatePicker class, 97
  timer mode, 290
UIDevice class, 588
  orientation notifications, requesting, 455
UIImage class, 197
UIImageView class, 197, 290
UIKit framework, 84
UILabel class, 96, 152, 156, 177
UIPicker class, 97
UIPickerView class, protocols, 290
UIResponder class, 92
UIs (user interfaces). *see also* interfaces
  Address Book framework, 512-518
  BestFriend application, creating, 511-512
  ColorTilt application, preparing, 464-465
  Cupertino application, creating, 536-537
  Cupertino Compass application, updating, 544-545
  FlashCards application
    creating, 384, 387-391
    preparing, 386-387
  Flashlight application, creating, 367
  FlowerInfoNavigator application, 357-358
  FlowerWeb application, finishing, 215
  Gestures application, creating, 437-439
  ImageHop applications, finishing, 190-192
  MediaPlayground application, creating, 480
  Message UI, BestFriend application, 523-525
  Orientation application, preparing, 459-460
  Reframe application
    creating, 417-422
    reframing logic, 422-423
  resizable, 407-408, 429-430
    Autosizing, 413-416
    creating, 411-416
    designing, 410-411
    setting up, 411-412
  ReturnMe application, creating, 374
  rotatable, 407-408, 429-430
    Autosizing, 413-416
    creating, 411-416
    designing, 410-411
    enabling, 408-409
    reframing controls, 416-423
    setting up, 411-412
    swapping views, 423-429
    testing, 412-413
  sliders, 180
  SlowCount application, creating, 572
  Swapper application, creating, 425-426
UISegmentedControl class, 97
UISlider class, 97, 197
UISwitch class, 96
UITextField class, 97, 152-153, 156
UITextView class, 97, 152-154, 161, 177
UIView class, 92
UIViewController class, 93
UIWebView class, 228
UIWindow class, 92
unique device identifiers, 12-13
Universal application, 583
  active devices, detecting and displaying, 588-590
  device-specific view controllers, adding, 584-588
  setting up, 584
universal applications, 579-580, 590, 598-599
  GenericViewController view controller class, 591-596
    adding device-specific views, 591
    adding to application delegates, 591-592
    implementation, 595
    instantiating view controller, 592-593
    iPhone and iPad views, 595-596
    XIB files, 593, 595
  tools
    converting interfaces, 597
    upgrading iPhone target, 596-597
  Window-based template, 581, 583
    adding view controllers to application delegates, 585-586
    detecting and displaying active device, 588-590
    device-specific view controllers and views, 584
    instantiating view controllers, 586, 588
    plist files, 581-583
    project preparation, 584
UniversalToo application, 590, 596
  GenericViewController
    adding, 590-592
    implementing, 595
  setting up, 590

view controllers, instantiating, 592-593
views, 595-596
XIB files, setting up, 593-595
update filter, location manager, 533
updateRightWrongCounters method, 396
updates
accelerometer
managing, 467-468
reacting to, 468-469
gyroscope
managing, 467-468
reacting to, 470-471
headings, Cupertino Compass application, 545-549
motion updates, receiving, 456
updating
applications, 652-653
UI, Cupertino Compass application, 544-545
upgrading iPhone target, 596-597
uploading
applications, 647-649
Certificate Assistant, 17
upside-down portrait mode, 430
URLs (uniform resource locators), 95-96
user alerts, 231-232, 249
action sheets, 245
button press responses, 248-249
changing appearance, 247-248
displaying, 245-247
buttons, adding, 238-241
displaying, 236-237
fields, adding, 241-245
generating, 235-245
multi-option alerts, creating, 238-241
notification interfaces, creating, 233
playing sounds, 250-253
prepping notification files, 232-233
System Sound Services, 250
vibrations, 253
user defaults, 366-372
games, 405
implicit preferences, creating, 366-372
reading, 371-372
setting up, 366-367
storing, 370-371
system settings, 372-374
settings bundles, 375-381
user input, 179, 199-200
scrolling views, 203
segmented controls, 201
FlowerWeb application, 204
sliders, 180
adding, 188-190
image views, 180
switches, 200
FlowerWeb application, 204
web views, 202-203
FlowerWeb application, 204
user interfaces
connection to code, 119
actions, 120-121, 123-124
implementation, 120
launching IB from Xcode, 119-120
outlets, 120-123
creating with Interface Builder, 110
layout tools, 112-115
Objects Library, 110-111
customization, 115
Accessibility settings, 116-117
Attributes Inspector, 115-116
simulation, 117
user output, 179, 199-200
scrolling views, 203
segmented controls, 201
FlowerWeb application, 204
switches, 200
FlowerWeb application, 204
web views, 202-203
FlowerWeb application, 204

V

variables
alertDialog, 237
declaration, 65
object data types, 66
primitive data types, 65-66
definition, 56
GNU Debugger, datatip, 608
instance variables, 59-60
sender, 172
versions, testing with iPhone Simulator, 49
vibrations, alerts, 253
view controller logic
DateCalc application, implementing, 267-270
FlowerWeb application, implementing, 216-220
ImageHop application, 193-195
view controllers
configuring, 312
creating universal applications with Window-based template, 585-586
FieldButtonFun
actions, 155-156
adding text fields, 156-161
adding text views, 161-164
outlets, 155-156
setting up, 154
instantiating, 592-593
logic implementation, 146-147, 174-175
multiple views, 149
MultipleViews application
adding, 296-297
instantiating, 298-299

MVC structure, 132
IBAction directive, 133-134
IBOutlet directive, 132
outlets and actions, 140-141
TabbedCalculation application, adding, 308-309
UIViewController class, 93
Universal application, adding, 584-588
universal applications, GenericViewController, 591-596
view icon (XIB files), 108
view switch methods, MultiViews application, implementing, 303-305
View-Based Application template, 135
creating views, 141
addition of objects, 141-144
outlet and action connection, 144-146
FieldButtonFun
actions, 155-156
adding text fields, 156-161
adding text views, 161-164
creating styled buttons, 164-171
hiding keyboard, 171-174
outlets, 155-156
releasing objects, 175
setting up, 154
view controller logic, 174-175
FlowerWeb, 204
finishing interface, 215
preparing actions, 205-206
preparing outlets, 205-206
releasing objects, 220-221
segmented controls, 206-210
setting up, 205
switches, 210-212
view controller logic, 216-220
web views, 212-214
implementation, 135
implementation of view controller logic, 146-147
object release, 147-148
project setup, 136
classes, 136-138
XIB files, 138-139
Scroller, 221
adding scroll views, 223-225
preparing outlets, 222-223
releasing objects, 226
scrolling behavior, 225-226
setting up, 222
testing application, 148
view controllers, outlets and actions, 140-141
viewDidLoad method, 168-169, 186, 396, 440, 460, 488, 545, 567, 573-575, 605, 611, 617, 621
views, 109. *see also* table views
Debugger (GNU Debugger), 612-614
map, configuring, 512
MultipleViews application, adding views, 296-297
MVC structure, 131-132
picker views, 259, 270-275, 284-289
adding, 273-274
outlets, 272
output labels, 275
protocols, 259-261, 271
providing data to, 275-281
reacting to choices, 281-284
pin annotation view, creating, 522
scrolling, 203, 221
Scroller, 221-222
Scroller application, 223-226
width, 226
swapping, 410
rotatable applications, 423-429
Swapper application, 426-429
table views, 329-330
text views, 152-154
UIView class, 92
UniversalToo application, 595-596
view controllers, multiple controllers, 149
View-Based Application template
creating views, 141-146
implementation of view controller logic, 146-147
object release, 147-148
web views, 202-203
FlowerWeb application, 204
void return type (methods), 61
volume view, multi-view applications, implementing, 319-323

W

warnings, building applications, 41-42
watchpoints, GNU Debugger, 611-612
web pages, scaling, 214
web views, 120, 202-203
attributes, setting, 212-213
clearColor, 220
FlowerInfoNavigator application, adding, 352-353
FlowerWeb application, 204, 212-214
outlets, connecting to, 214
supported content types, 202
websites
Apple, 8
applications, promoting, 650-652
width, scroll views, 226

WiFi technology, 529
supplementation, 6
wildcard IDs, 637
window objects, UIWindow class, 92
Window-based templates, universal applications, 581-583
adding view controllers to application delegates, 585-586
detecting and displaying active device, 588-590
device-specific view controllers and views, 584
instantiating view controllers, 586-588
plist files, 581-583
project preparation, 584

X

Xcode, 23, 27-28, 50
build configurations, 604
building applications, 39-42
Active Configuration setting, 40
Build and Run button, 40-41
errors and warnings, 41-42
debugging, 601
GNU Debugger, 603-614
Instruments tool, 614-619
NSLog function, 602-603
Shark profiler, 620-626
documentation system, 45
Cocoa Touch, 81-83, 90-98
exploration of frameworks, 98-101
editing, 34-39
editor, autocompletion, 35-37
gutter, 604
launching IB from, 119-120
modifying project properties, 42, 45
launch image, 44-45
setting application icon, 43
status bar display, 45
navigating, 34-39
project management, 28
adding existing resources, 32
adding new code files, 31-32
creating a new project, 28-29
project groups, 30-31
removal of files and resources, 33-34
Xcode editor, 51
***Xcode 3 Unleashed*, 77, 627**
Xcode 4
Interface Builder, 106
preview, 24
***Xcode Debugging Guide, Shark User Guide*, 627**
Xcode editor, 51
Xcode menu commands, Preferences, 100
***Xcode Workspace Guide*, 50**
XIB (Interface Builder) files, 107
Document icons, 109-110
Document window, 107-109
universal applications, 593-595
UniversalToo application, setting up, 593-595
View-Based Application template, 138-139