

John Ray



In Full Color

Figures  
and code  
appear as they  
do in Xcode

Covers iOS 3.2  
and up

Sams Teach Yourself

# iPad Application Development

in **24**  
**Hours**



SAMS

John Ray

Sams **Teach Yourself**

**iPad**<sup>TM</sup>

Application Development

in **24**  
**Hours**



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

## **Sams Teach Yourself iPad™ Application Development in 24 Hours**

### **Copyright © 2011 by Pearson Education, Inc.**

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

ISBN-13: 978-0-672-33339-2

ISBN-10: 0-672-33339-2

*Library of Congress Cataloging-in-Publication Data:*

Ray, John, 1971-

Sams teach yourself iPad application development in 24 hours / John Ray.

p. cm.

Includes index.

ISBN 978-0-672-33339-2

1. iPad (Computer)—Programming. 2. Application software—Development. I. Title.

QA76.8.I863R392 2011

005.3—dc22

2010023693

Printed in the United States of America

First Printing July 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](mailto:corpsales@pearsontechgroup.com)**

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

**International Sales**

**[international@pearson.com](mailto:international@pearson.com)**

**Associate  
Publisher**

*Greg Wiegand*

**Acquisitions Editor**

*Laura Norman*

**Development  
Editor**

*Keith Cline*

**Managing Editor**

*Kristy Hart*

**Project Editor**

*Lori Lyons*

**Indexer**

*Angela Martin*

**Proofreader**

*Kathy Ruiz*

**Technical Editor**

*Matthew David*

**Publishing  
Coordinator**

*Cindy Teeters*

**Multimedia  
Developer**

*Dan Scherf*

**Book Designer**

*Gary Adair*

**Compositor**

*Gloria Schurick*

# Contents at a Glance

Introduction.....	1
<b>HOUR 1</b> Preparing Your System and iPad for Development.....	5
<b>2</b> Introduction to XCode and the iPhone Simulator .....	29
<b>3</b> Discovering Objective-C: The Language of Apple Platforms.....	55
<b>4</b> Inside Cocoa Touch .....	83
<b>5</b> Exploring Interface Builder.....	107
<b>6</b> Model-View-Controller Application Design.....	133
<b>7</b> Working with Text, Keyboards, and Buttons.....	157
<b>8</b> Handling Images, Animation, and Sliders.....	187
<b>9</b> Using Advanced Interface Objects and Views .....	211
<b>10</b> Getting the User's Attention.....	241
<b>11</b> Presenting Options with Popovers and Toolbars.....	261
<b>12</b> Making Multivalue Choices with Pickers and Action Sheets.....	283
<b>13</b> Focusing on Tasks with Modal Views .....	325
<b>14</b> Implementing Multipleview Applications.....	341
<b>15</b> Navigating Information Using Table Views and Split View-based Applications .....	379
<b>16</b> Reading and Writing Application Data .....	415
<b>17</b> Building Rotatable and Resizable User Interfaces.....	461
<b>18</b> Extending the Touch Interface.....	489
<b>19</b> Sensing Movement with Accelerometer Input .....	509
<b>20</b> Working with Rich Media .....	527
<b>21</b> Interacting with Other Applications .....	557
<b>22</b> Building Universal Applications.....	579
<b>23</b> Application Debugging and Optimization .....	601
<b>24</b> Distributing Applications Through the App Store .....	631
Index.....	659

# Table of Contents

<b>Introduction</b>	<b>1</b>
<b>HOUR 1: Preparing Your System and iPad for Development</b>	<b>5</b>
Welcome to the iPad Platform .....	5
Becoming an iPad Developer .....	8
Creating a Development Provisioning Profile .....	13
Developer Technology Overview .....	25
Summary.....	27
Q&A .....	27
Workshop .....	28
<b>HOUR 2: Introduction to Xcode and the iPhone Simulator</b>	<b>29</b>
Using Xcode .....	29
Using the iPhone Simulator .....	47
Further Exploration.....	52
Summary.....	52
Q&A .....	53
Workshop .....	53
<b>HOUR 3: Discovering Objective-C: The Language of Apple Platforms</b>	<b>55</b>
Object-Oriented Programming and Objective-C .....	55
Exploring the Objective-C File Structure .....	60
Objective-C Programming Basics .....	67
Memory Management .....	76
Further Exploration.....	78
Summary.....	79
Q&A .....	79
Workshop .....	80
<b>HOUR 4: Inside Cocoa Touch</b>	<b>83</b>
What Is Cocoa Touch? .....	83
Exploring the iPhone OS Technology Layers .....	85

**Contents**

Tracing the iPad Application Life Cycle.....	89
Cocoa Fundamentals .....	91
Exploring the iPhone OS Frameworks with Xcode .....	100
Summary.....	104
Q&A .....	104
Workshop .....	104
<b>HOUR 5: Exploring Interface Builder</b>	<b>107</b>
Understanding Interface Builder .....	107
Creating User Interfaces .....	112
Customizing Interface Appearance .....	117
Connecting to Code .....	122
Further Exploration.....	130
Summary.....	131
Q&A .....	131
Workshop .....	132
<b>HOUR 6: Model-View-Controller Application Design</b>	<b>133</b>
Understanding the Model-View-Controller Paradigm .....	133
How Xcode and Interface Builder Implement MVC .....	136
Using the View-Based Application Template .....	139
Further Exploration.....	153
Summary.....	154
Q&A .....	154
Workshop .....	154
<b>HOUR 7: Working with Text, Keyboards, and Buttons</b>	<b>157</b>
Basic User Input and Output .....	157
Using Text Fields, Text Views, and Buttons .....	159
Further Exploration.....	184
Summary.....	185
Q&A .....	185
Workshop .....	186

<b>HOUR 8: Handling Images, Animation, and Sliders</b>	<b>187</b>
User Input and Output .....	187
Creating and Managing Image Animations and Sliders .....	188
Further Exploration.....	207
Summary.....	208
Q&A .....	208
Workshop .....	209
<b>HOUR 9: Using Advanced Interface Objects and Views</b>	<b>211</b>
User Input and Output (Continued) .....	211
Using Switches, Segmented Controls, and Web Views.....	216
Using Scrolling Views .....	232
Further Exploration.....	238
Summary.....	239
Q&A .....	239
Workshop .....	240
<b>HOUR 10: Getting the User's Attention</b>	<b>241</b>
Exploring User Alert Methods.....	241
Generating Alerts .....	245
Using Alert Sounds .....	255
Further Exploration.....	258
Summary.....	259
Q&A .....	259
Workshop .....	260
<b>HOUR 11: Presenting Options with Popovers and Toolbars</b>	<b>261</b>
Understanding Popovers and Toolbars .....	262
Using Popovers with Toolbars.....	264
Further Exploration.....	279
Summary.....	280
Q&A .....	280
Workshop .....	281

<b>HOUR 12: Making Multivalue Choices with Pickers and Action Sheets</b>	<b>283</b>
Popover-centric UI Elements .....	283
The PopoverPlayground Project .....	289
Using Date Pickers .....	289
Implementing a Custom Picker View.....	299
Using Action Sheets .....	316
Further Exploration.....	321
Summary.....	322
Q&A .....	322
Workshop .....	323
<b>HOUR 13: Focusing on Tasks with Modal Views</b>	<b>325</b>
Modal Views .....	325
Using Modal Views .....	328
Further Exploration.....	339
Summary.....	339
Q&A .....	339
Workshop .....	340
<b>HOUR 14: Implementing Multiview Applications</b>	<b>341</b>
Exploring Single Versus Multiview Applications .....	341
Creating a Multiview Application .....	342
Building a Multiview Tab Bar Application .....	354
Further Exploration.....	374
Summary.....	376
Q&A .....	376
Workshop .....	376
<b>HOUR 15: Navigating Information Using Table Views and Split View-Based Applications</b>	<b>379</b>
Understanding Table Views and Split Views .....	380
Building a Simple Table View Application .....	383
Creating a Split View-Based Application .....	396
Further Exploration.....	411

**Sams Teach Yourself iPad™ Application Development in 24 Hours**

Summary.....	411
Q&A .....	412
Workshop .....	412
<b>HOUR 16: Reading and Writing Application Data</b>	<b>415</b>
Design Considerations .....	415
Reading and Writing User Defaults .....	418
Understanding the iPad File System Sandbox .....	433
Implementing File System Storage.....	436
Further Exploration.....	457
Summary.....	458
Q&A .....	458
Workshop .....	459
<b>HOUR 17: Building Rotatable and Resizable User Interfaces</b>	<b>461</b>
Rotatable and Resizable Interfaces .....	461
Creating Rotatable and Resizable Interfaces with Interface Builder .....	465
Reframing Controls on Rotation .....	471
Swapping Views on Rotation .....	479
Further Exploration.....	485
Summary.....	486
Q&A .....	486
Workshop .....	487
<b>HOUR 18: Extending the Touch Interface</b>	<b>489</b>
Multitouch Gesture Recognition.....	490
Using Gesture Recognizers .....	491
Further Exploration.....	506
Summary.....	507
Q&A .....	507
Workshop .....	508

<b>HOUR 19: Sensing Movement with Accelerometer Input</b>	<b>509</b>
Accelerometer Background.....	510
Sensing Orientation .....	513
Detecting Tilt.....	518
Detecting Movement.....	522
Further Exploration.....	523
Summary.....	524
Workshop .....	524
<b>HOUR 20: Working with Rich Media</b>	<b>527</b>
Exploring Rich Media .....	527
Preparing the Media Playground Application.....	529
Using the Movie Player.....	534
Creating and Playing Audio Recordings .....	539
Using the iPad Photo Library .....	544
Accessing and Playing the iPod Library .....	548
Further Exploration.....	554
Summary.....	555
Q&A .....	555
Workshop .....	556
<b>HOUR 21: Interacting with Other Applications</b>	<b>557</b>
Extending Application Integration .....	557
Using Address Book, Email, and Maps... Oh My! .....	561
Further Exploration.....	577
Summary.....	577
Q&A .....	577
Workshop .....	578
<b>HOUR 22: Building Universal Applications</b>	<b>579</b>
Universal Application Development .....	579
Understanding the Universal Window Application Template .....	581
Other Universal Application Tools .....	597
Further Exploration.....	599

**Sams Teach Yourself iPad™ Application Development in 24 Hours**

Summary.....	.599
Q&A .....	.599
Workshop .....	.600
<b>HOUR 23: Application Debugging and Optimization</b>	<b>601</b>
Debugging in Xcode .....	.602
Monitoring with Instruments.....	.615
Profiling with Shark .....	.622
Further Exploration.....	.629
Summary.....	.629
Workshop .....	.630
<b>HOUR 24: Distributing Applications Through the App Store</b>	<b>631</b>
Preparing an Application for the App Store .....	.632
Submitting an Application for Approval.....	.642
Promoting Your Application.....	.649
Exploring Other Distribution Methods.....	.654
Summary.....	.656
Q&A .....	.656
Workshop .....	.657
<b>Index</b>	<b>659</b>

## 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*, *Sams Teach Yourself Dreamweaver MX in 21 Days*, *Mac OS X Unleashed*, and *Sams Teach Yourself iPhone 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 on the Macintosh and its applications—and have garnered numerous positive reviews for its straightforward approach and accessibility to beginning and intermediate users.

## Dedication

*This book is dedicated to everyone who can see beyond the count of USB ports, RAM slots, and technical jargon to recognize the beauty of a platform as a whole. I'm excited to see what you create.*

## Acknowledgments

Thank you to the group at Sams Publishing—Laura Norman, Kristy Hart, Lori Lyons, Keith Cline, Matthew David, Gloria Schurick, Kathy Ruiz—for recognizing the importance of the iPhone OS/iPad platform, and helping to create this book. Skilled editors make authors coherent.

As always, thanks to my family and friends for keeping me sane for the duration of the project. It wasn't that bad, was it?

## 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@samspublishing.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](http://informit.com/register) for convenient access to any updates, downloads, or errata that might be available for this book.

# Introduction

*“It’s just a big iPod Touch.”*

Few words have puzzled me more during the weeks leading up to the iPad launch. Let’s break down exactly what it means to be a “big iPod Touch.”

First, it means a large, bright, colorful display, coupled with an amazingly thin enclosure and amazing battery life. Second, it means a user experience based on the world’s most popular portable Internet device.

Perhaps the most important aspect of being a “big iPod touch” is that it is a device designed to be controlled by human fingers. Every aspect of development is centered on touch interactions. Quite simply, the iPad is a multitouch device that is 100% dedicated to running applications that you control with your fingers.

Terrible, isn’t it?

Less than a month after the iPad launched, Apple has sold more than a million units. It’s reassuring that people still recognize and embrace innovation. It also means that there is no end to the opportunity that the iPad affords to you, the developer.

The iPad is an open canvas. On the iPhone, there are plenty of apps, but less of an opportunity to experiment with user interfaces. On the iPad, apps take on new life. The display begs to be touched, and complex gestures are fun and easy to implement. Computing truly becomes a personal experience, similar to curling up with a good book.

Our hope is that this book will bring iPad development to a new generation of developers who want to create large-scale multitouch applications. *Sams Teach Yourself iPad Application 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 just 24 hour-long lessons.

## Who Can Become an iPad 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, you have everything you need to begin developing for the iPad.

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

## Who Should Use This Book?

This book targets individuals who are new to development for the iPhone OS 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 this book does not assume that you've coded before.

That said, some things are expected from 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 covered in this book. A vast amount of information on iPhone development is available, but only limited space is available in this book. However, this book does cover what you need to forge your own path forward.

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

This book targets the initial release of the iPad OS 3.2. Much of what you'll be learning is common to all iPhone OS releases (the iPad is built on top of the iPhone OS), but we also cover several important advances, such as popovers, modal views, and more!

Unfortunately, this is not a complete reference for the iPhone OS application programming interfaces. Some topics require much more space than the format of 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 iPad for Development." Many lessons include a section titled "Further Exploration"

that will guide you toward additional related topics of interest. Again, a willingness to explore is an important quality in becoming a successful developer!

Each coding lesson is accompanied by project files that include everything you need to compile and test an example or, preferably, follow along and build the application yourself. Be sure to download the project files from the book's website at <http://teachyourselfipad.com>.

In addition to the support website, you can follow along on Twitter! Search for #iPadIn24 on Twitter to receive official updates and tweets from other readers. Use the hashtag #iPadIn24 in your tweets to join the conversation. To send me messages via Twitter, begin each tweet with @johnemeryray.

*This page intentionally left blank*

## HOUR 11

# Presenting Options with Popovers and Toolbars

---

### ***What You'll Learn in This Hour:***

- ▶ How to add toolbars and toolbar buttons to your projects
- ▶ The role of popovers in the iPhone OS
- ▶ How to generate custom popover views in your projects
- ▶ Tricks for checking to see whether a popover is already displayed

On the iPhone, what you see is typically what you get. The user interface elements either show the options that are available to you, offer the ability to scroll to additional options, or swap out the current screen for another view that displays more information. The multiple-window model used in Mac OS X is gone. Although you might encounter an occasional alert dialog, windowing is not a standard in iPhone interfaces. On the iPad, things have changed. Apple has introduced the popover: a user interface element that can present views on top of other views.

In this hour, we explore how to prepare views for use in popovers, including adding toolbars and toolbar buttons (the most frequent UI element used to invoke a popover). You'll also configure the different display attributes associated with popovers, and communicate information between popover views and your main application view. Popovers are such a prevalent and important UI element that we'll be focusing on them for the next few hours, so be sure to work through this lesson carefully.

## Understanding Popovers and Toolbars

Popovers are everywhere in the iPad interface, from Mail to Safari, as demonstrated in Figure 11.1. Using a popover enables you to display new information to your users without leaving the screen you are on, and to hide the information when the user is done with it. There are few desktop counterparts to popovers, but they are roughly analogous to tool palettes, inspector panels, and configuration dialogs. In other words, they provide user interfaces for interacting with content on the iPad screen, but without eating up permanent space in your UI.

**FIGURE 11.1**

Popovers are unique to the iPad UI.



Popovers, although capable of being displayed when a user interacts with any onscreen object, are most often shown when the user presses a toolbar button (`UIBarButtonItem`) from within a toolbar object (`UIToolbar`). This is exactly the scenario shown in Figure 11.1. Because of this relationship, we will be presenting both of these objects within this hour's lesson. Let's quickly review what we need for each before we get started coding.

### Popovers

Unlike other UI elements, popovers aren't something you just drag into a view from the Interface Builder Library. They are, in fact, entirely independent views, designed just like your main application view. The display of the views is governed by a popover controller (`UIPopoverController`). The controller displays the popover

when a user event is triggered, such as touching a toolbar button. When the user is done with the popover, touching outside of its visible rectangle automatically closes the view.

To create a popover, we'll need to cover three different requirements. First, we need to make a view and view controller specially designed for the popover's contents. Second, when the proper event occurs in the user interface, we need to allocate and initialize an instance of popover controller. Third, when the user is done with the popover, we want to make sure that any changes made in the popover are reflected in the main application.

## Popover Views

You've been developing views and view controllers for the past several hours, so you'll feel right at home working with a popover view. It uses the same `UIViewController` that we've been using all along, but with the addition of one unique property: `contentSizeForViewInPopover`.

This property should be set to the width and height of the popover to be displayed. Apple allows popovers up to 600 pixels wide and the height of the iPad screen, but recommends that they be kept to 320 pixels wide, or less. For example, to set the content size of 320 pixels by 200 pixels for a view controller that will be displaying a popover, we might add the following to the `viewDidLoad` method:

```
self.contentSizeForViewInPopover=CGSizeMake(320.0,200.0);
```

In fact, that's exactly what we're going to be doing in the tutorial shortly.

## Popover Controller

Like views need view controllers, popovers need popover controllers (`UIPopoverController`). Popover controllers take care of all the hard work of rendering popovers on the screen in the right place. We'll focus on two methods of the popover controller:

`initWithContentViewController`—Initializes the popover with the contents of a view controller. When the popover is displayed, whatever the view controller's view is, is displayed.

`presentPopoverFromBarButtonItem:permittedArrowDirections:animated`—Invokes the display of the popover so that it appears to emerge from (and point to) a toolbar button. The parameters for this method allow fine-tuning of the arrow from the popover to the UI element it is appearing from, and whether its display is animated.

The popover controller will also need the `delegate` property set to an object that will take care of all the "cleanup" when the popover is dismissed by the user. This

includes releasing the popover controller and updating the contents of the main application to reflect the user's actions in the popover. This leads us to the final popover requirement: the `UIPopoverControllerDelegate` protocol.

### **Popover Controller Delegate Protocol**

To make the full use of a popover, we'll need an additional protocol method added to one of our classes. In our sample application, we'll be using our main application's view controller class for this purpose. This means we need to add a line to our main view controller's interface (.h) file to state that we're conforming to the `UIPopoverControllerDelegate` protocol. Second, we'll be adding the protocol method `popoverControllerDidDismissPopover` to our application's view controller implementation file. That's it.

When the popover is dismissed by the user touching outside of its display, the `popoverControllerDidDismissPopover` method is invoked and we can react appropriately.

### **Toolbars**

Toolbars (`UIToolbar`) are, comparatively speaking, one of the simpler UI elements that you have at your disposal. A toolbar is implemented as a solid bar, either at the top or bottom of the display, with buttons (`UIBarButtonItem`) that correspond to actions that can be performed in the current view. The buttons provide a single selector action, which works nearly identically to the typical Touch Up Inside event that you've encountered before.

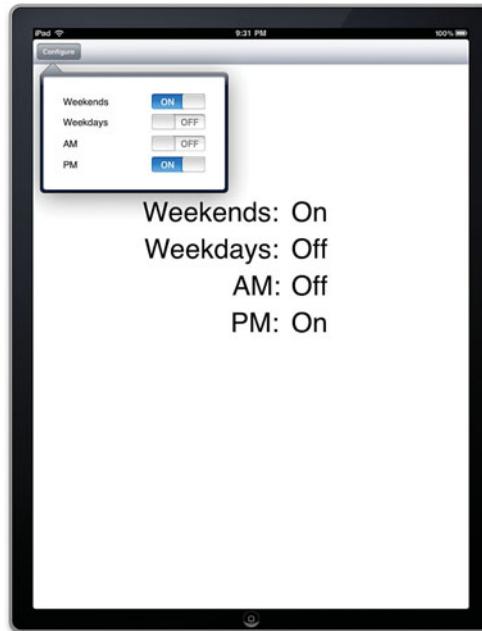
Toolbars, as their name implies, are used for providing a set of static choices to the user—interface options that should be visible regardless of whether the application's primary content is changing. As you'll see, they can be implemented almost entirely visually and are the de facto standard for triggering the display of a popover on the iPad.

Although implementing popovers might be sounding a bit convoluted at this point, hang in there. After you've created one, the process will seem incredibly simple, and you'll want to use them everywhere!

## **Using Popovers with Toolbars**

Popovers are used to display interface elements that configure how your application behaves but that don't need to be visible all the time. Our sample implementation will display a toolbar, complete with a Configure button, that invokes a popover. The popover will display configuration four switches (`UISwitch`) for a hypothetical time-based application: Weekends, Weekdays, AM, and PM.

The user will be able to update these switches in the popover, and then touch outside the popover to dismiss it. Upon dismissal, four labels in the main application view will be updated to show the user's selections. The final application will resemble Figure 11.2.



**FIGURE 11.2**  
This application will display a popover and update the main application view to reflect a user's actions in the popover.

## Implementation Overview

The implementation of this project is simpler than it may seem at the onset. You'll be creating a View-based iPad application that includes a toolbar with the Configure button and four labels that will display what a user has chosen in the popover. The popover will require its own view controller and view. We'll add these to the main project, but they'll be set up almost entirely independently from the main application view.

Building the connection between the main view and the popover will require surprisingly few lines of code. We need to be careful that touching the Configure button doesn't continue to add popovers to the display if one is already shown, but you'll learn a trick that keeps it all under control.

## Setting Up the Project

This project will start with the View-Based Application template; we'll be adding in another view and view controller to handle the popover. Let's begin. Launch Xcode

(Developer/Applications), and then create a new View-based iPad project called **PopoverConfig**.

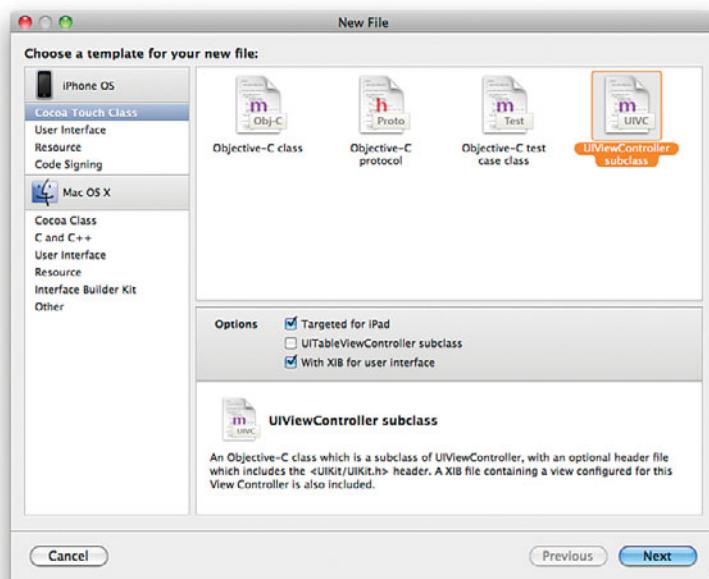
Xcode will create the basics structure for your project, including the **PopoverConfigViewController** classes. We'll refer to this as the *main application view controller* (the class the implements the view that the user sees when the application runs). For the popover content itself, we need to add a new view controller and XIB file to the **PopoverConfig** project.

### Adding an Additional View Controller Class

With the Classes group selected in your Xcode project, choose File, New File, from the menu bar. Within the New File dialog box, choose the Cocoa Touch Class within the iPhone OS category, and then the **UIViewController subclass** icon, as shown in Figure 11.3.

**FIGURE 11.3**

Create the popover's content view controller and XIB file.



Be sure that Targeted for iPad and With XIB for user interface are selected, and then choose Next. When prompted, name the new class **PopoverContentViewController** and click Finish.

The **PopoverContentViewController** implementation and interface files are added to the Classes group.

Depending on your version of Xcode, the XIB file may also be added to the folder you had selected when creating the class files. If this is the case, drag it to the Resources group.

**Did you  
Know?**

## Preparing the Popover Content

This hour's project is unique in that most of your interface work takes place in a view that is only onscreen occasionally when the application is running—the popover's content view. The view will have four switches (`UISwitch`), which we'll need to account for.

We only need to be able to read values from the popup view, not invoke any actions, so we'll just add four `IBOutlets`.

### Adding Outlets

Open the `PopoverContentViewController.h` interface file and add outlets for four `UISwitch` elements: `weekendSwitch`, `weekdaySwitch`, `amSwitch`, `pmSwitch`. Be sure to also add `@property` directives for each switch. The resulting interface file is shown in Listing 11.1.

---

**LISTING 11.1**

---

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

@interface PopoverContentViewController : UIViewController {
    IBOutlet UISwitch *weekendSwitch;
    IBOutlet UISwitch *weekdaySwitch;
    IBOutlet UISwitch *amSwitch;
    IBOutlet UISwitch *pmSwitch;
}

@property (nonatomic,retain) UISwitch *weekendSwitch;
@property (nonatomic,retain) UISwitch *weekdaySwitch;
@property (nonatomic,retain) UISwitch *amSwitch;
@property (nonatomic,retain) UISwitch *pmSwitch;

@end
```

---

For each of the properties we've declared, we need to add a `@synthesize` directive in the implementation (`popoverContentViewController.m`) file. Open this file and make your additions following the `@implementation` line:

```
@synthesize weekdaySwitch;
@synthesize weekendSwitch;
@synthesize amSwitch;
@synthesize pmSwitch;
```

## Setting the Popover Content Size

Our next step is easy to overlook, but amazingly important to the final application. For an application to present an appropriately sized popover, you must manually define the popover's content size. The easiest (and most logical) place to do this is within the popover's view controller.

Continue editing the `popoverContentViewController.m` file to uncomment its `viewDidLoad` method and add a size definition:

```
- (void)viewDidLoad {
    self.contentSizeForViewInPopover=CGSizeMake(320.0,200.0);
    [super viewDidLoad];
}
```

For this tutorial project, our popover will be 320 pixels wide and 200 pixels tall. Remember that Apple supports values up to 600 pixels wide and a height as tall as the iPad's screen area allows.

## Releasing Objects

Even though this view controller sits outside of our main application, we still need to clean up memory properly. Finish up the implementation of the `popoverContentViewController` class by releasing the four switch instance variables in the `dealloc` method:

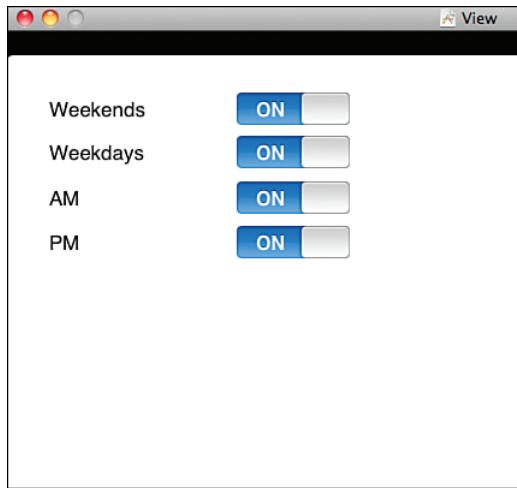
```
- (void)dealloc {
    [weekdaySwitch release];
    [weekendSwitch release];
    [amSwitch release];
    [pmSwitch release];
    [super dealloc];
}
```

That finishes the `popoverContentViewController` logic! Although we still have a little bit of work to do in Interface Builder, the rest of the programming efforts will take place in the main `popoverConfig` view controller class.

## Preparing the View

Building a popover's view is *identical* to building any other view with one small difference: You can only use the portion of the view that fits within the size of the popover you're creating. Open the `popoverContentViewController` XIB file in Interface Builder and add four labels (Weekends, Weekdays, AM, and PM) and four corresponding switches (`UISwitch`) from the library.

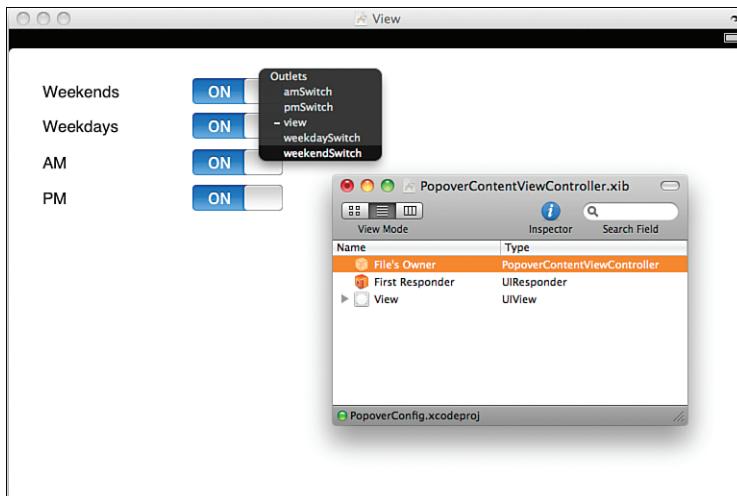
Position these in the upper-left corner of the view to fit within the 320x200 dimensions we've defined, as shown in Figure 11.4.

**FIGURE 11.4**

Add four configuration switches and corresponding labels to the popover content view.

## Connecting the Outlets

After creating the view, connect the switches to the IBOutlets. Control-drag from the File's Owner icon in the Document window to the first switch in the view (the Weekends switch in my implementation) and choose the `weekendSwitch` outlet when prompted, as shown in Figure 11.5.

**FIGURE 11.5**

Connect the switches to their outlets.

Repeat these steps for the other three switches, connecting them to the `weekdaySwitch`, `amSwitch`, and `pmSwitch` outlets.

The popover content is now complete. Let's move to the main application.

## Preparing the Application View

With the popover content under control, we'll build out the main application view/view controller. There are only a few "gotchas" here, such as declaring that we're going to conform to the `UIPopoverControllerDelegate` protocol, and making sure that we create an instance of the popover content view.

### Conforming to a Protocol

To conform to the popover controller delegate, open the `popoverConfigViewController.h` interface file, and modify the `@interface` line to include the name of the protocol, enclosed in `<>`. The line should read as follows:

```
@interface PopoverConfigViewController : UIViewController  
<UIPopoverControllerDelegate> {
```

We still need to implement a method for the protocol within the view controller, but more on that a bit later.

### Adding Outlets, Actions, and Instance Variables

We need to keep track of quite a few things within the main application's view controller. We're going to need an instance variable for the popover's controller (`popoverController`). This will be used to display the popover, and to check whether the popover is already onscreen. We'll also need an `IBAction` defined (`showPopover`) for displaying the popover.

In addition, five `IBOutlets` are required—four for `UILabels` that will display the values the user enters in the popover (`weekdayOutput`, `weekendOutput`, `amOutput`, `pmOutput`), and the last for the popover's view controller (`popoverContent`).

Sound like enough? Not quite! Because we're going to be using the `popoverContentViewController` class within the main application, we need to import its interface file, too.

### Did you know?

#### What About the Configure Button?

If you're following closely, you might wonder whether we need an instance variable for the Configure button. When we initialize the popover controller, we need to tell it what onscreen object it should point to (that is, the Configure button).

Thankfully, the button passes a reference of itself to the action it calls when pressed, so we can use that reference rather than keeping track of the button separately.

Edit the interface file so that it matches Listing 11.2.

**LISTING 11.2**

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

@interface PopoverConfigViewController : UIViewController
    <UIPopoverControllerDelegate> {
    UIPopoverController *popoverController;
    IBOutlet UILabel *weekdayOutput;
    IBOutlet UILabel *weekendOutput;
    IBOutlet UILabel *amOutput;
    IBOutlet UILabel *pmOutput;
    IBOutlet PopoverContentViewController *popoverContent;
}

@property (retain,nonatomic) UILabel *weekdayOutput;
@property (retain,nonatomic) UILabel *weekendOutput;
@property (retain,nonatomic) UILabel *amOutput;
@property (retain,nonatomic) UILabel *pmOutput;
@property (retain,nonatomic) PopoverContentViewController *popoverContent;

-(IBAction)showPopover:(id)sender;

@end
```

For each `@property` directive, there needs to be a corresponding `@synthesize` in the `PopoverConfigViewController.m` file. Edit the file now, adding these lines following the `@implementation` line:

```
@synthesize popoverContent;
@synthesize weekdayOutput;
@synthesize weekendOutput;
@synthesize amOutput;
@synthesize pmOutput;
```

This gives us everything we need to build and connect the main application interface elements, but before we do, let's make sure that everything we've added here is properly released.

## Releasing Objects

Edit `PopoverConfigViewController.m`'s `dealloc` method to release the `UILabels`, and the instance of the popover content view controller (`popoverContent`):

```
- (void)dealloc {
    [weekdayOutput release];
    [weekendOutput release];
    [amOutput release];
    [pmOutput release];
    [popoverContent release];
    [super dealloc];
}
```

Nicely done! All that's left now is to edit the popoverConfigViewController XIB file to create the main application interface and write the methods for showing and handling the subsequent dismissal of the popover.

### **Creating the View**

Open the popoverConfigViewController XIB file in Interface Builder. We need to add a toolbar, a toolbar button, and some labels to display our application's output.

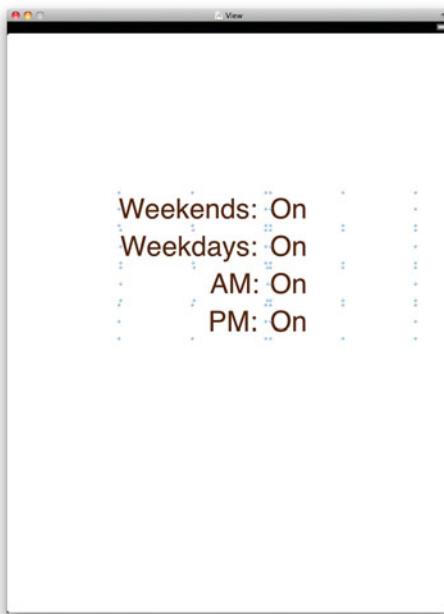
Let's start with the labels, because we've got plenty of experience with them. Drag a total of eight `UILabel` objects to the screen. Four will hold the application's output, and four will just be labels (fancy that!).

Arrange the labels near the center of the screen, forming a column with Weekends:, Weekdays:, AM:, and PM: on the left, and On, On, On, On aligned with them on the right. The On labels are the labels that will map to the `IBOutlet` output variables; they've been set to a default value of On because the switches in the popover content view default to the On position.

If desired, use the Attributes Inspector (Command+1) to resize the labels to something a bit larger than the default. I've used a 48pt font in my interface, as shown in Figure 11.6.

**FIGURE 11.6**

Add a total of eight labels to the view.



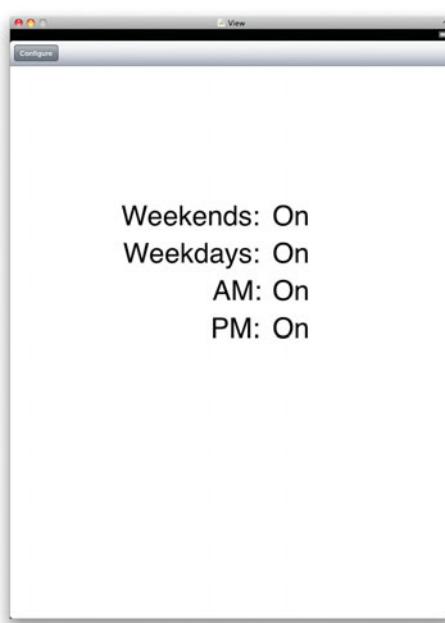
## Adding a Toolbar and Toolbar Button

Using the Interface Builder Library, drag an instance of a toolbar (`UIToolbar`) to top of the view. The toolbar object includes, by default, a single button called `Item`.

Double-click the button to change its title to `Configure`; the button will automatically resize itself to fit the label.

In this application, the single button is all that is needed. If your project needs more, you can drag Bar Button Items from the library into the toolbar. The buttons are shown as subviews of the toolbar within the Interface Builder Document window.

Figure 11.7 shows the final interface and the Document window showing the interface hierarchy.



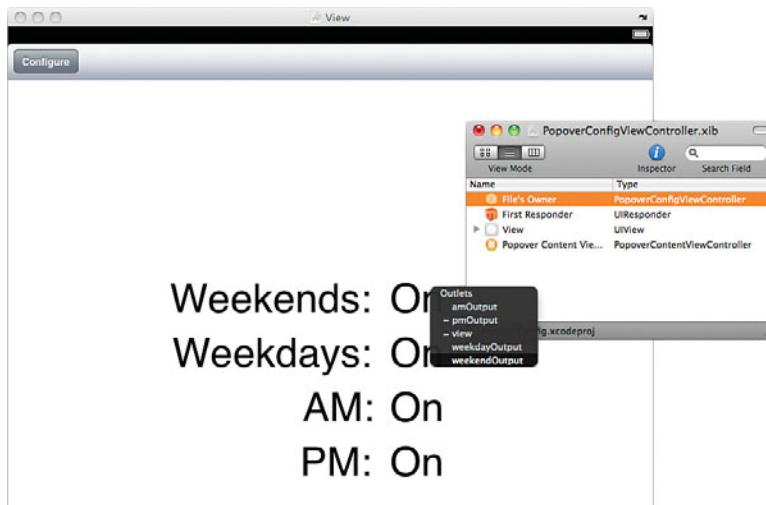
**FIGURE 11.7**  
Labels, and  
toolbar, and a  
toolbar button  
complete the  
interface.

## Connecting the Outlets and Actions

It's time to connect the interface we've built to the outlets and actions we defined in the view controller. Control-drag from the File's Owner icon in the IB Document window to the first `On` label, connecting to the `weekendOutput` outlet, as shown in Figure 11.8. Repeat for the other three labels, connecting to `weekdayOutput`, `amOutput`, and `pmOutput`.

**FIGURE 11.8.**

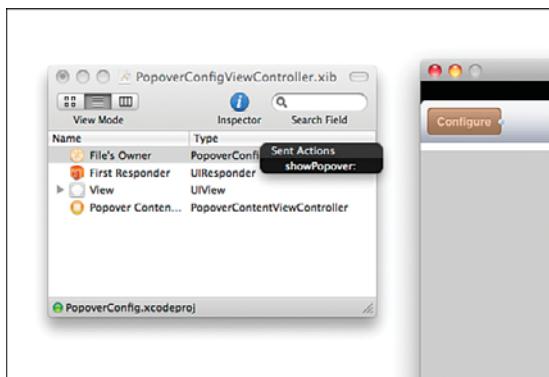
Connect each output label to its outlet.



Next, Control-drag from the Configure toolbar button to the File's owner icon. Choose showPopover when prompted, as shown in Figure 11.9. Note that we didn't have to worry about connecting from a Touch Up Inside event because toolbar buttons have only one event that can be used.

**FIGURE 11.9**

Connect the configure button to the showPopover action.



Only one step remains to be completed in interface builder: instantiating the popover content view controller.

### Instantiating the Popover Content View Controller

Earlier in the tutorial, we developed the popover content view controller and view (`popoverContentViewController`). What we haven't done, however, is actually use it anywhere. We can take two approaches to creating an instance of the controller so that we can use it in the application:

1. The content view controller is instantiated whenever the popover is invoked, and released when the popover is dismissed.
2. The content view controller is instantiated when main application view loads and is released when the application is finished.

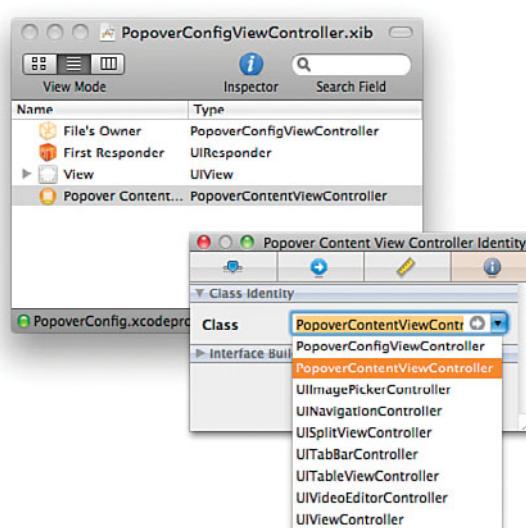
I've chosen to go with approach number 2. By instantiating the popover's view controller when the main application view loads, we can use it repeatedly without reloading the view. This means that if the user displays the popover and updates the switches, those changes will be visible no matter how many times the user dismisses or opens the popover.

If you are creating an application with *many* popovers, go with method 1; otherwise, all the views will be kept in memory simultaneously.

**Did you  
Know?**

Without adding any code, we can instantiate `popoverContentViewController` when the `popoverConfigViewController.xib` file is loaded:

1. Open the `popoverConfigViewController.xib` file's document window in Interface Builder.
2. Drag a View Controller object from the Library into the document window.
3. Select the view controller in the Document window, and press Command+4 to open the Identity Inspector.
4. Set the class to the `popoverContentViewController` rather than the generic `UIViewController` class set by default. This can be seen in Figure 11.10.

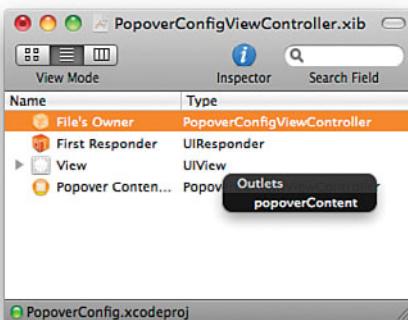


**FIGURE 11.10**  
Set the object to be an instance of `popoverContentViewController`.

5. Switch to the Attributes Inspector (Command+1) and set the NIB name field to **popoverContentViewController** so that the view controller knows where its view is stored.
6. Close the Inspector window.
7. Control drag from the File's Owner icon to the popover content view controller icon within the Document window. Choose **popoverContent** when prompted, as shown in Figure 11.11.
8. Save the XIB file.

**FIGURE 11.11**

Connect the popover content view controller to the `popoverContent` outlet.



Our views and view controllers are completed. All that remains is writing the code that handles the application logic.

## Implementing the Application Logic

We need to implement two methods to complete this tutorial. First, we need to implement `showPopover` to display the popover and allow the user to interact with it. Second, the popover controller delegate method `popoverControllerDidDismissPopover` must be built to take care of cleaning up the popover when the user is done with it, and to update the application's view with any changes the user made within the popover.

### Displaying the Popover

Open the `PopoverConfigViewController.m` file and add the `showPopover` method, shown in Listing 11.3, immediately following the `@synthesize` directives.

**LISTING 11.3**

```
1: - (IBAction)showPopover:(id)sender {
2:     if (popoverController==nil) {
3:         popoverController=[[UIPopoverController alloc]
4:             initWithContentViewController:popoverContent];
5:         [popoverController presentPopoverFromBarButtonItem:sender
6:             permittedArrowDirections:UIPopoverArrowDirectionAny animated:YES];
7:         popoverController.delegate=self;
8:     }
9: }
```

There are three steps to displaying and configuring the popover.

First, in lines 3–4, the popover controller, `popoverController`, is allocated and initialized with the popover's content view, `popoverContent`.

Second, lines 5 and 6 display the popover using the (very verbose) method `presentPopoverFromBarButtonItem:permittedArrowDirections:animated`. The bar button item (our toolbar button) can be referenced through the `sender` variable, which is passed to `showPopover` when the button is pressed. The `permittedArrowDirections` parameter is passed the constant `UIPopoverArrowDirectionAny`, meaning the popover can be drawn with an arrow that points in any direction (as long as it points to the specified interface element). The `animated` parameter gives the iPad the go-ahead to animate the appearance of the popover (currently a nice fade-in effect).

Third, line 7 sets the popover controller's delegate to the same object that is executing the code (`self`)—in other words, the `popoverConfigViewController`. By doing this, the popover controller will automatically call the method `popoverControllerDidDismissPopover` within `popoverConfigViewController.m` when the user is done with it.

Nothing too scary, right? Right. But what about lines 2 and 8? The entire display of the popover is wrapped in an `if-then` statement. The reason for this can be easily demonstrated by removing the `if-then` and running the application. Without the conditional, multiple copies of the popover will be displayed (one on top of the other) each time the Configure button is pressed. This is a large memory leak and would make the application behave very strangely for the user. To get around the problem, we perform a simple comparison: `popoverController==nil`. When the popover controller hasn't been initialized, it will have a value of `nil` (that is, no value at all). In this case, the statements to initialize the controller and show the popover are executed. Once the popover is displayed, however, the `popoverController` has a value and will no longer equal `nil`, keeping any further instances of it from being displayed.

Of course, we want the user to be able to dismiss and redisplay the popover, so we need to release the popoverController and set it back to nil when we hide the popover again. Let's look at that implementation now.

## Did you know?

### What Constants Can I Provide for a Popover's Arrow Direction?

You can force the popover's arrow (and subsequent onscreen positioning) by using one of five different constants:

UIPopoverArrowDirectionUp—The popover points up toward the interface element.

UIPopoverArrowDirectionDown—The popover points down toward the interface element.

UIPopoverArrowDirectionLeft—The popover points left toward the interface element.

UIPopoverArrowDirectionRight—The popover points right toward the interface element.

UIPopoverArrowDirectionAny—The popover can be oriented in whatever position the iPhone OS finds most appropriate.

Apple recommends using the “Any” option whenever possible in your applications.

### Reacting to the Popover Dismissal

When the user gets rid of the popover by touching outside of its content area, we want our application to react and display any changes the user made within the popover view. We also want to prepare the popover's controller to show the popover again. Enter the popover controller delegate method `popoverControllerDidDismissPopover` as shown in Listing 11.4.

#### LISTING 11.4

```
1: - (void)popoverControllerDidDismissPopover:
2: {
3:     UIPopoverController *controller =
4:     weekdayOutput.text=@"On";
5:     weekendOutput.text=@"On";
6:     amOutput.text=@"On";
7:     pmOutput.text=@"On";
8:     if (!popoverContent.weekdaySwitch.on) {
9:         weekdayOutput.text=@"Off";
10:    }
11:    if (!popoverContent.weekendSwitch.on) {
12:        weekendOutput.text=@"Off";
13:    }
14:    if (!popoverContent.amSwitch.on) {
15:        amOutput.text=@"Off";
16:    }
}
```

```
17:     if (!popoverContent.pmSwitch.on) {  
18:         pmOutput.text=@"Off";  
19:     }  
20:     [popoverController release];  
21:     popoverController=nil;  
22: }
```

Most of the display logic used in this method should be familiar to you by now. Lines 3–6 set the four output labels to On, because this is the default state of our switches. Lines 8–19 are simple if-then statements which check to see whether a switch is *not* set to on, and, if so, sets the corresponding output label to Off.

Because we have an instance variable for the popover's view controller (popoverContent) and have defined the UISwitches as properties, we can access the individual state of a given switches using its on property in a single line:  
popoverContent.<switch instance variable>.on.

In the final two lines, 20 and 23, the popover controller is released and its instance variable (popoverController) set to nil. This prepares it for the next time the user presses the Configure button.

It might surprise you to learn that releasing an object does not automatically set its instance variable to nil. In fact, the instance variable is *not* changed at release and will reference a nonexistent object, potentially causing major problems if you attempt to use it.

### Did you Know?

You might be wondering why we didn't just use the controller reference rather than popoverController instance variable. The answer is that we need to be able to set the popoverController variable to nil. If we use controller, we reference the same object as popoverController, but setting controller to nil doesn't change the value of popoverController.

### Watch Out!

The application is now complete. Use Build and Run to test the popover's display on your iPad. You've just implemented one of the most important and flexible UI features available on the iPad platform!

## Further Exploration

In this hour's sample project, you attached a toolbar to a "bar button" (toolbar button) using the presentPopoverFromBarButtonItem:permittedArrowDirections:animated method. This, granted, is a very popular approach, but you can create popovers

anywhere within your view by using the `UIPopoverController` method `presentPopoverFromRect:inView:permittedArrowDirections:animated`. With this method you can present the popover so that it appears from any rectangular areas, within any view. In addition, popover content does not need to be static! If you'd like, your popover's view controller can update its content on-the-fly, and the popover will update dynamically to display the changes. You'll need to manually update the `popoverContentSize` property of the controller so that all of your content fits, but size changes are animated smoothly for the end user.

To learn more about popovers, be sure to review Apple's `UIPopoverController` class reference within the developer documentation to get a complete picture of this important class and UI element.

## Summary

Popovers provide a canvas for creating a range of unique interface elements that can be displayed virtually anywhere in your application. The approach that we took in this hour's lesson (creating a popover that is displayed when a toolbar button is pressed) is the most common implementation that you'll encounter.

You've learned not only how a popover is designed and displayed, but how to access data from within its view, and ways of keeping the popover controller from getting out of hand. In the next hour's lesson, you'll learn about several UI elements that Apple will allow *only* if they are displayed from within a popover. So even if you can't think of any uses for them yet, chances are, you will!

## Q&A

**Q. Can I have multiple popover's within a single application view?**

**A.** Yes, you can, but keep in mind that the example here uses a single delegate for handling the dismissal of a popover. There are a number of ways to get around this, including structuring your code so that changes within a popover are immediately reflected in the application, or you can segment your application so that each popover has a different delegate.

**Q. You told me to drag the toolbar to the top of the window. The developer docs say to drag it to the bottom. What gives?**

**A.** At the time of this writing, Apple has not yet updated all the descriptions of the toolbar UI element to state that it can be used at the top and bottom of the iPad screen.

# Workshop

## Quiz

1. What class is a toolbar button?
2. How do you set where a popover appears?
3. Why do we need to compare the popover controller to `nil` before initializing it?

## Answers

1. A toolbar button is an instance of the bizarrely named class `UIBarButtonItem`.
2. The iPhone OS determines where a popover appears onscreen. Setting the `permittedArrowDirections` parameter when displaying the popover, however, limits where the OS may position the popover so that it can be drawn with an arrow pointing to the UI element invoking it.
3. If the popover controller is *not* `nil`, that means the popover is visible onscreen and a new copy of it should not be created.

## Activities

1. Explore the possibilities of popovers outside of toolbars. Implement an additional button (`UIButton`) within the `popoverConfig` application that displays the same popover, but located in the center of the screen.
2. Implement a second toolbar-based popover within the `popoverConfig` application. If you choose to use a single delegate for each, you can check to see which popover is being dismissed by comparing each controller instance variable to the controller variable passed to the `popoverControllerDidDismissPopover` method.

*This page intentionally left blank*

# Index

## Symbols

#import directive, 61, 65  
 #pragma mark directive, 41  
 %@ string format specifier, 602  
 %f string format specifier, 602  
 %i string format specifier, 602  
 @implementation directive, 65  
 @interface directive, 61-62  
 @property directive, 3-64, 6137  
 @synthesize directive, 66, 137

## A

A4 processor, 7  
 About group (ReturnMe preferences), 428  
 ABPeoplePickerNavigation-ControllerDelegate protocol, 564  
 ABPersonHasImageData function, 569  
 ABRecordCopyVal method, 568  
 ABRecordCopyValue function, 567  
 abstract, Quick Help results, 103  
 accelerometer, 8  
     API, 511-513  
     background, 510-511  
     ColorTilt application, 518-520  
     Orientation application, 513-516  
     sensing movement, 522-523  
 accelerometer:didAccelerate method, 512, 515  
 Accessibility Programming Guide for iPhone OS, 131  
 Accessibility settings (Interface Builder), 119-120  
 action sheets, 287, 316  
     animated versus non-animated, 288  
     changing appearance and behavior, 321  
     implementation, 316  
     interface, 317-319  
     project setup, 316-317

response to user, 320  
 UIActionSheetDelegate protocol, 288  
 view controller logic, 319  
 actionMessage string, 310  
 actions, 124-129, 161-162  
     action sheets, 318-320  
     area view (multiview applications), 361-362, 365  
     built-in capabilities, 563  
     connection to  
         buttons, 178-179  
         notification project interface, 243-245  
         sliders, 200  
         switches, 223  
     flashlight application, 420-421  
 FlowerWeb application, 217-218  
 GetFlower, 125, 221  
 image views, 190-192  
 IsSetLightSourceAlpha, 418  
 main view (modal views), 333  
 Media Playground application, 533-534  
 multiview application  
     toolbars, 349-350  
 newBFF, 561  
 popover application view, 270-273  
 segmented controls, 221-222  
 sendEmail, 561  
 view controllers, 144-145, 149-150  
 volume view (multiview applications), 367-370  
 actionSheet:clickedButtonAtIndex method, 288, 320  
 Active Configuration setting, 43  
 active device (universal applications), 588-590  
 Active SDK setting, 42  
 Activity Monitor instrument, 621  
 ad hoc deployment of applications, 654-655  
 Add Contact button, 173  
 Add Horizontal Guide command (Layout menu), 115  
 Add Vertical Guide command (Layout menu), 115  
 addButtonWithTitle method, 320  
 Address Book framework, 557-558, 563-565  
     Cocoa Touch layer, 87  
     contact selection, 565  
     delegate methods, 566  
     displaying contact information, 566-569  
 addSubview method, 386  
 Alert View Delegate protocol, 250-251  
 alertDialog variable, instantiation, 246-247  
 alerts (user notifications), 241  
     alert sounds, 255-258  
     connecting to outlets and actions, 243-245  
     creating notification project interface, 243  
     generating, 245-246  
         multi-option alerts, 248-255  
         simple alerts, 246-248  
     prepping project files, 242-243  
 alignment (IB layout tool), 115-116  
 Alignment command (Layout menu), 115  
 allocation of objects, 69-70  
 Anderson, Fritz, Xcode 3 Unleashed, 629  
 animalNames array, 304  
 animalSounds array, 304  
 animated action sheets, 288  
 animation, 188  
     action sheets, 288  
     image views, 190, 195-197  
     implementation, 189-190  
     starting/stopping, 197

## animationDuration property

- animationDuration property, 196**
- API accelerometer, 511-513**
- App ID, 16, 636-637**
- App store**
  - distribution of applications, 631-642
  - promotion of applications, 649-653
  - submitting applications for approval, 642-649
- appearance**
  - action sheets, 321
  - segmented controls, 220
  - table views, 396
  - text input trait, 165
- Apple Developer Program, 8**
  - costs, 9
  - registration, 9-12
- Apple Developer Suite, 25-26.**
  - See also* Interface Builder; iPhone Simulator; Xcode
- Apple ID, 10**
- Apple tutorials, 185**
- Apple website, 9**
- application view**
- application:DidFinishLaunchingWithOptions method, 357, 586, 592**
- applications**
  - building, 42-45
    - user input/output, 183-184
  - built-in capabilities, 557
    - Address Book frameworks, 557-558, 563-569
    - connecting to outlets and actions, 563
    - Core Location framework, 560, 569-573
    - creating app interface, 562-563
    - implementation, 561
    - Map Kit framework, 560, 569-573
    - Message UI framework, 559, 573-577
    - project setup, 561-562
  - charging for, 653-654
  - ColorTilt, 518-520
  - DebugPractice, 616
  - decision making, 72
  - expressions, 72-73
  - if-then-else statements, 73
  - repetition with loops, 74-76
  - switch statements, 73
- distribution, 631-632**
  - ad hoc deployment, 654-655
  - App ID, 636-637
  - artwork, 632-634
  - Distribution Certificates, 634-636
  - Distribution Provisioning Profile, 638
  - Enterprise Deployment, 655-656
  - project configuration, 639-642
- FlowerWeb. See FlowerWeb application**
- life cycles, 89-91**
- logic**
  - flash card application, 447-449
  - flashlight application, 421-422
  - popovers, 276-279
- Media Playground, 529-534**
- multiview. See multiview applications**
- Orientation, 513-516**
- OS X Installer, 13**
- preferences, 415**
  - design considerations, 415-417
  - file system storage implementation, 436-457
  - iPad file system sandbox, 433-436
  - reading and writing, 418-423
  - Settings application, 424-433
  - popovers, 270-276
  - promotion, 649-653
- Property List Editor, 423**
- resizable interfaces, 461-462**
  - design, 464-465
  - implementation of reframing logic, 477-478
- Interface Builder, 465-471**
  - reframing controls, 471-477
  - swapping views, 479-485
- ReturnMe, 424**
- rotatable interfaces, 461-462**
  - design, 464-465
  - implementation of reframing logic, 477-478
- Interface Builder, 465-471**
  - orientation constants, 463
  - reframing controls, 471-477
  - swapping views, 479-485
- Shark profiler, 622-625**
- Split View-based Application template. See Split View-based Application template**
- submitting for approval, 642-649**
- table views, 383**
  - appearance, 396
  - implementation, 384
  - project setup, 384-388
  - providing data to, 389-394
  - reacting to a row touch event, 394-395
- testing**
  - Interface Builder, 120
  - iPhone Simulator, 47-52
  - View-Based Application template, 152-153
- tracing, 615-621**
- UIApplication class, 92**
- universal. See universal applications**
- updates, 653**
- user input/output, 183-184**
- Xcode, 29-30**
  - building applications, 42-45
  - editing code, 36-42
  - modifying project properties, 45-47
  - navigating code, 36-42
  - project management, 31-35
  - removal of files and resources, 35-36
- Applications Library (iTunes), 632**

- approval, submitting applications, 642-643
    - binary upload, 648-649
    - profile preparation, 643-648
  - archiveFlashCards** method, 456
  - archiveRootObject:toFile** method, 456
    - archiving Flash cards, 455-457
  - area calculation logic, 365-367
  - area view (multiview applications), 361
    - area calculation logic, 365-367
    - creating the view, 362-364
    - outlets and actions, 361-365
  - arrays**, 95
    - animalNames, 304
    - animalSounds, 304
    - CFBundleIconFiles, 583
    - flowerSections, 403
    - NSMutableArray, 402
  - arrow constants (popovers), 278
  - artwork, distribution of applications, 632-634
  - attributes**, 117-119
    - Accessibility settings, 119-120
    - buttons, 173-174
    - Date Pickers, 292-293
    - non-atomic, 64
    - retain, 64
    - sliders, 198-199
    - text fields, 163-165
    - text views, 168-169
    - web views, 225
  - Attributes Inspector**, 117-119, 163, 225
  - Attributes Inspector command (Tools menu)**, 117, 163
  - audio playback, 539
    - cleanup, 543-544
    - control, 541-543
    - implementation, 540-541
  - Audio Toolbox framework**
    - Media layer, 87
    - playing alert sounds, 257-258
  - audioPlayerDidFinishPlaying:** successfully: method, 539, 542-543
    - Active Configuration setting, 43
  - AudioServicesCreateSystemSoundID** function, 257
  - AudioServicesPlaySystemSound** function, 257
  - Auto-Enable Return Key (text input trait), 165
  - Autocompletion (Xcode editor), 38-39
  - autorelease method, 76
  - autoresizing, 464, 473
  - autorotation, 464
  - Autoresizing settings (Size Inspector), 117
  - AV Foundation framework**, 528-529
    - audio playback, 539
    - cleanup, 543-544
    - control, 541-543
    - implementation, 540-541
  - AVAudioPlayer** class, 529
  - AVAudioPlayerDelegate** protocol, 539, 542
  - AVAudioRecorder** class, 529, 539
  - axes, accelerometer, 510
- B**
- background**
    - accelerometer, 510-511
    - graphics/color, 202-203
    - touch, hiding keyboard, 181-182
  - Background menu**, 175
  - behavior**, action sheets, 321
  - binary upload, submitting applications for approval, 648-649
  - BlueTooth supplementation, 7
  - bookmarks, 40-41
  - boolForKey** method, 423
  - bounds property, 483
  - breakpoints, 605-608
  - Build and Run button, 43-44
  - Build command (Build menu), 43
  - build configurations (Xcode), 604
  - Build menu commands, 43
  - building applications, 42-45
    - Active Configuration setting, 43
  - Active SDK setting, 42
  - Build and Run button, 43-44
  - errors and warnings, 44-45
  - user input/output, 183-184

**built-in capabilities**, 557

- Address Book frameworks, 557-558, 563-565
- contact selection, 565
- delegate methods, 566
- displaying contact information, 566-569
- connecting to outlets and actions, 563
- Core Location framework, 560, 569-573
- creating app interface, 562-563
- implementation, 561
- Map Kit framework, 560, 569-573
- Message UI framework, 559, 573-577
- project setup, 561-562

**Bundle (Settings application)**, 416, 427-431

- Bundle Identifier**, 637
- button bars**, 123
- buttons**, 97, 158
  - action sheets, 317-318
  - Add Contact, 173
  - Build and Run, 43-44
  - Check for Leaks Now, 619
  - Clear, 164
  - Configure, 270
  - connection to actions, 178-179
  - Custom, 173
  - customization, 172
  - Detail Disclosure, 173
  - Done, 180-181
  - editing attributes, 173-174
  - Export Developer Profile, 23
  - FlowerWeb application, 226-227
  - Generate Story, 175
  - Hop, 201-202
  - Import Developer Profile, 23
  - Info Dark, 173

## buttons

Info Light, 173  
 multi-option alerts, 248-249  
 multiview application  
     toolbars, 347-348  
 popovers, 273  
 radio, 212  
 Rounded Rect, 172-173  
 setting images, 174-178  
 toolbar buttons, 262  
**buttonTitleAtIndex method,**  
 250, 320

## C

**CA** (certificate authority), 635  
**calculate method**, 365  
**cancelButtonIndex method**, 320  
**cancelButtonTitle parameter**, 247  
 capacitive multitouch screen, 7  
 Capitalize (text input trait), 165  
 card view controller (flash card  
     application), 444-445  
 cards (flash card application),  
 450-453  
 cells, table views, 391-394  
 cellular technology, 560  
 centerMap method, 571  
 Certificate Assistant, 17-18  
 certificate authority (CA), 635  
 Certificate Revocation List  
     (CRL), 635  
 CFBundleIconFiles array, 583  
 CFNetwork framework, 89  
 CGAffineTransformMakeRotation  
     function, 502  
 CGRectMake( ) function, 478  
 changing state, 174  
 charging for applications,  
 653-654  
 check boxes, 212  
 Check for Leaks Now button, 619  
 chooseImage method, 544  
 choosePod: method, 549  
 ChosenColor outlet, 125  
 class methods  
     definition, 58  
     imagenamed, 177  
 classes, 33. *See also objects*  
     AVAudioPlayer, 529  
     AVAudioRecorder, 529, 539  
     core, 91-94

data type, 94-97  
 definition, 58  
 DetailViewController, 383  
 files, 33  
 FirstViewController, 355  
 FlashCard, 437  
 gesture recognizers, 490  
 interface, 97-99  
 iPadViewController, 584  
 iPhoneViewController, 585  
 MPMediaItem, 528  
 MPMediaItemCollection, 528  
 MPMediaPickerController,  
 528, 548  
 MPMoviePlayerController,  
 528, 534  
 MPMusicPlayerController,  
 528, 548  
 NSNotificationCenter, 537  
 NSObject, 59  
 NSURL, 214  
 NSURLRequest, 214  
 NSUserDefaults, 418  
 PopoverConfigViewController,  
 266  
 single, 134  
 UIActionSheet, 287-288  
 UIAlertView, 245-255  
 UIDevice, 588  
 UIImagePickerController, 544  
 View-Based Application  
     template, 142  
**cleanup**  
     audio playback, 543-544  
     Image Picker, 546-547  
     Media Picker, 551  
     movie playback, 537-538  
**Clear button**, 164  
**Cocoa versus Cocoa Touch**, 85  
**Cocoa Touch**, 26  
     Cocoa Touch layer,  
         frameworks, 86-87  
     Cocoa versus, 85  
     core classes, 91-94  
     data type classes, 94-97  
     functionality, 84-85  
     interface classes, 97-99  
     origins, 85

**Cocoa Touch layer,**  
**frameworks**, 86  
     Address Book UI, 87  
     Game Kit, 87  
     Map Kit, 86  
     Message UI, 87  
     UIKit, 86  
**code**  
     adding to projects, 34  
     connection to user  
         interfaces, 122  
         implementation, 123  
         launching IB from  
             Xcode, 122  
         outlets and actions,  
             124-129  
**code snapshots**, 39-40  
**codecs**, 535  
**color, background**, 202-203  
**ColorChoice outlet**, 125, 217, 221  
**ColorTilt application**, 518  
     interface, 519  
     project setup, 518  
     UIAccelerometerDelegate  
         implementation, 519-520  
**commands**  
     Build menu, Build, 43  
     Edit menu, Duplicate, 194  
     File menu  
         Make Snapshot, 40  
         New Project, 31  
         Simulate Interface,  
             120, 471  
         Snapshots, 40  
     Help menu  
         Developer Documentation,  
             100  
         Quick Help, 102  
     Layout menu, 115  
     Project menu  
         New Smart Group, 34  
         Set Active Build  
             Configuration,  
             Debug, 604  
     Run menu, Run, 43  
     Tools menu  
         Attributes Inspector,  
             117, 163  
         Connections Inspector, 178  
         Identity Inspector, 129

- Library, 112, 162
- Size Inspector, 116
- Window menu, Document, 192
- Xcode menu, Preferences, 101
- components, 284**
- componentsSeparatedByString method, 572**
- condition-based loops, 75**
- configuration**
  - map view, 563
  - segments (segmented controls), 219
  - view controller classes, 360
- Configure button, 270**
- configureView method, 410**
- connections**
  - buttons to actions, 178-179
  - outlets to image views, 194-195
  - popover content to outlets, 269
  - preferences to applications, 432-433
  - scrolling views to outlets, 237
  - segmented controls to actions/outlets, 221-222
  - sliders to actions/outlets, 199-200
  - switches to actions, 223
  - text fields to outlets, 166-167
  - text views to outlets, 171-172
  - web views to outlets, 225
- Connections Inspector, 127, 178, 223**
- Connections Inspector command (Tools menu), 178**
- connectivity (platform), 7**
- constants**
  - popover arrow, 278
  - swipe directions, 499
- constraints (platform), 7**
- Contact Information group (ReturnMe preferences), 428**
- contacts, Address Book frameworks, 565-569**
- content**
  - loading remote content, 214-215
  - multiview applications, 344-345
  - popovers, 267
    - connection to outlets, 269
    - object release, 268
    - outlets, 267
    - size, 268
    - views, 268
    - web view support, 214
- contentSize property, 233**
- ContentViewController view controller, 333**
- Continue icon (debugger), 609**
- Continue to Here option (gutter context menu), 611**
- controllers**
  - modal view, 333-334
  - MVC structure, 135-136
    - IBAction directive, 138
    - IBOutlet directive, 137
  - navigation, 383, 398
  - popovers, 263-264
  - root view table controller, 406-408
  - tab bars, 357
    - adding, 358-359
    - item images, 359-360
  - UIPopoverController, 262
  - UIViewController, 263
  - view
    - multiview applications, 343-344
    - UIControl class, 94
- controls**
  - audio, 541-543
  - reframing, 471-477
  - segmented, 213
    - FlowerWeb application, 218-222
  - UISegmentedControl class, 98
  - UIControl class, 93
- convenience methods, 69-70**
- converting interfaces, universal applications, 598**
- copies, image views, 194**
- Core Animation instrument, 621**
- core classes, 91-94**
  - NSObject, 92
  - UIApplication, 92
  - UIControl, 93
  - UIResponder, 93
  - UIView, 92
  - UIViewController, 94
  - UIWindow, 92
- Core Data framework, 88**
- Core Data instrument, 621**
- Core Foundation framework, 88**
- Core Graphics framework, 87**
- Core Location framework, 88, 560, 569-573**
- Core OS layer, frameworks, 89**
- Core Services layer, frameworks, 88-89**
- CoreGraphics framework, 86**
- Correction (text input trait), 165**
- costs, Apple Developer Program, 9**
- count-based loops, 74**
- CPU Sampler instrument, 621**
- Create iPhone/iPod Touch Version (Interface Builder), 598**
- CreateCardDelegate protocol, 444, 451**
- createFlowerData method, 402**
- createStory method, 182**
- creating**
  - projects, Xcode, 31-32
  - universal applications
    - GenericViewController view controller class, 590-596
    - Window-based template, 583-590
- CRL (Certificate Revocation List), 635**
- currentCard method, 447**
- currentDevice method, 588**
- Custom button, 173**
- custom picker views, 299**
  - adding picker views, 302-303
  - implementation, 299
  - interface, 303-304
  - project setup, 300-301

## custom picker views

- providing data to, 304
  - application data structures, 304-305
  - data source methods, 306-307
  - populating data structures, 305-306
  - populating picker display, 307-308
- response to user, 309-311
- `UIPickerViewDelegate` optional methods, 311-315
- customization**
  - application preferences, 415
    - design considerations, 415-417
  - file system storage implementation, 436-457
  - iPad file system sandbox, 433-436
  - reading and writing, 418-423
  - Settings application, 424-433
  - buttons, 172
  - keyboard display, 165-166
  - user interfaces, 117
    - Accessibility settings, 119-120
    - Attributes Inspector, 117-119
- D**
- data**
  - custom picker views, 304-308
  - detectors, 171
  - models, MVC structure, 138-139
  - providing to Split View-based Application template, 401-405
  - providing to table view, 389-394
  - storage, iPad file system sandbox, 433-434
    - file paths, 435-436
  - implementation, 436-457
- storage locations, 434-435
  - structures, 402-405
- data source methods**
  - pickers, 306-307
  - root view table controller, 406
  - table views, 390-391
- data type classes, 94-97**
  - `NSArray`, 95
  - `NSDate`, 96
  - `NSDecimalNumber`, 95-96
  - `NSDictionary`, 95
  - `NSMutableArray`, 95
  - `NSMutableDictionary`, 95
  - `NSMutableString`, 94
  - `NSNumber`, 95-96
  - `NSString`, 94
  - `NSURL`, 96-97
- datatip, variable examination, 608-609**
- Date Pickers, 285**
  - adding, 291
    - attributes, 292-293
    - connection to actions, 293-294
  - implementation, 289
  - interface, 294-295
  - project setup, 290-291
  - view controller logic, 295
    - calculating difference between dates, 297
    - current date, 296
    - displaying date and time, 296
    - implementing date calculation, 297-299
- dates, 96**
- dealloc method, 77-78, 152, 183, 206, 232, 472, 480**
- Debug build configuration, 604**
- Debugger Console, 602**
- Debugger view (GNU Debugger), 613-615**
- debugging Xcode, 601**
  - GNU Debugger, 603-615
  - Instruments tool, 615-621
  - `NSLog` function, 602-603
  - Shark profiler, 622-629
- Debugging with GDB: The GNU Source-Level Debugger, 629**
- DebugPractice application, 616**
- decision making, 72**
  - expressions, 72-73
  - if-then-else statements, 73
  - repetition with loops, 74-76
  - switch statements, 73
- declaration**
  - Quick Help results, 103
  - variables, 67-69
- default image (image views), 193**
- default state, switches, 223**
- delegate methods, Address Book frameworks, 566**
- delegate parameter, 247**
- describeInteger method, 605, 610**
- design**
  - application preferences, 415-417
  - rotatable and resizable interfaces, 464-465
- destructiveButtonIndex method, 320**
- Detail Disclosure button, 173**
- Detail view**
  - controller, 409-410
  - Split View-based Application template, 399-401
- detail web view, 228-229**
- detailURLString string, 230**
- DetailViewController class, 383**
- detecting tilt, 518**
  - interface, 519
  - project setup, 518
  - `UIAccelerometerDelegate` implementation, 519-520
- Developer Documentation command (Help menu), 100**
- Developer Program (Apple), 8**
  - costs, 9
  - registration, 11-12
- Developer Suite, 25-26. See also Interface Builder; iPhone Simulator; Xcode**
- developer tools (iPhone OS), 12-13**
- Developer/Applications folder, 13**
- development paradigms, 56-57**
- Development Provisioning Assistant, 14**
  - App ID, 16

Certificate Assistant, 17-18  
device ID, 16-17  
provisioning profile, 13-14  
  downloading, 20-21  
  generation and  
    installation, 14-23  
  installing, 21-22  
  naming, 18-20  
  testing, 24-25  
launching, 15  
unique device identifiers, 14  
device identifiers, 14-17  
deviceType outlet, 589  
dictionaries, 95  
didCancelCardCreation method, 444  
didCreateCardWithQuestion: answer method, 444  
digital compass, 8  
directives  
  #import, 61, 65  
  @implementation, 65  
  @interface, 61-62  
  @property, 63-64, 137  
  @synthesize, 66, 137  
  definition, 61  
  IBAction, 138  
  IBOutlet, 137  
directories  
  Documents, 435  
  Library/Caches, 435  
  Library/Preferences, 434  
  tmp, 435  
disabling autoresizing, 473  
disclosure indicators, 408  
dismissal  
  modal views, 337-338  
  popovers, 278-279  
dismissModalViewControllerAnimated method, 326, 337  
dismissPopoverAnimated method, 546  
display  
  platform, 6  
  popovers, 262  
**Distribution Certificates**, 634-636

**distribution**  
  applications, 631-632  
  ad hoc deployment,  
    654-655  
  App ID, 636-637  
  artwork, 632-634  
  Distribution Certificates,  
    634-636  
  Distribution Provisioning Profile, 638  
  Enterprise Deployment,  
    655-656  
  project configuration,  
    639-642  
  profiles, 14, 638  
**Distribution Provisioning Profiles**, 638  
doActionSheet method, 319  
Document command (Window menu), 192  
Document icons (XIB files), [111-112](#)  
document sets, 101  
Document window (XIB file), [109-111](#)  
documentation system (Cocoa Touch), 83  
  core classes, 91-94  
  data type classes, 94-97  
  functionality, 84-85  
  interface classes, 97-99  
  origins, 85  
Documents directory, 435  
Done button, [180-181](#)  
double primitive data type, 67  
downloading provisioning profile, 20-21  
Duplicate command (Edit menu), 194

**E**

Edit menu commands, Duplicate, 194  
editing  
  button attributes, 173-174  
  code, 36-42  
  text field attributes, 163-165

text view attributes, 168-169  
toolbar control buttons, 347-348  
editor (Xcode), 38-39  
email, built-in capabilities, 559, 573-577  
encodeObjectForKey method, 454  
encodeWithCoder method, 454  
ending  
  implementation files, 66  
  interface files, 64  
**Enterprise Deployment**, 655-656  
enterprise program (Developer Program), 9  
errors, 44-45  
existing resources, 35  
Export Developer Profile button, 23  
expressions, 72-73  
External Accessory framework, 89

**F**

feedback  
  mechanisms, 7-8  
Xcode errors and warnings, 44-45  
fees, Apple Developer Program, 9  
fields  
  adding to alerts, 251-255  
  Minimum, 198  
File menu commands  
  Make Snapshot, 40  
  New Project, 31  
  Simulate Interface, 120, 471  
  Snapshots, 40  
File's Owner icon (XIB files), 109  
files, 60  
  adding to projects, 34  
  data storage, 433  
    file paths, 435-436  
    implementation, 436-457  
    storage locations, 434-435  
  header, 33, 60-64  
  implementation, 33, 65-66  
  locating methods and properties, 37

## files

- project management, 31
  - adding existing resources to files, 35
  - adding new code files, 34
  - editing/navigating code, 36-42
  - identifying project type, 31-32
- project groups, 32-34
  - removal of files from project, 35-36
- removal from projects, 36
- XIB (Interface Builder), 108
  - Document icons, 111-112
  - Document window, 109-111
  - View-Based Application template, 142-144
- finances, charging for applications**, 653-654
- first responder icon (XIB files), 109
- first responders, 179, 505
- FirstViewController class, 355
- flash card application
  - application logic, 447-449
  - archiving Flash cards, 455-457
  - card view controller, 444-445
  - cards, 450-453
  - interface, 438-446
  - object archiving, 453-455
  - project setup, 436-438
- Flash cards, archiving, 455-457
- FlashCard class, 437
- flashlight application, 418-423
- flexibility, rotatable and resizable interfaces, 466-471
- float primitive data type, 67
- floatForKey method, 423
- flow of program execution, GNU Debugger, 609-612
- flowerData structure, 403
- flowerDetailView outlet, 225
- flowerSections array, 403
- FlowerView outlet, 125, 225
- FlowerWeb application, 216
  - buttons, 226-227
  - object release, 232, 238
  - outlets and actions, 217-218
- project setup, 217
  - scrolling views, 232-234
    - adding objects, 235-236
    - implementation, 233, 237-238
    - outlets, 234, 237
  - project setup, 234
- segmented controls, 218
  - appearance selection, 220
  - configuration of segments, 219
  - connection to actions, 221-222
  - connection to outlets, 221
  - sizing controls, 220
- switches, 222-223
- testing, 232, 238
  - view controller logic implementation
    - detail web view, 228-229
    - loading/displaying details, 229-231
    - running application, 231-232
    - web views, 224-225
- format specifiers (strings), 602
- Foundation framework, 86-88
- foundPinch method, 501
- foundRotation method, 504
- foundSwipe method, 500
- foundTap method, 499
- frame property, 464
- frameworks, 33
  - Address Book, 557-558
  - contact selection, 565
  - delegate methods, 566
  - displaying contact information, 566-569
  - AudioToolbox, 257-258
  - AV Foundation, 528-529, 539-544
  - Core Location, 560, 569-573
  - Map Kit, 560, 569-573
  - Media Player, 528, 535-538
  - Message UI, 559, 573-577
  - technology layers, 86-89
  - Xcode documentation, 100-103
- Freeverse, Postman, 652
- functionality, Cocoa Touch, 84-85
- functions. *See also methods; protocols*
  - ABPersonHasImageData, 569
  - ABRecordCopyValue, 567
  - AudioServicesCreateSystemSoundID, 257
  - AudioServicesPlaySystemSound, 257
  - CGAffineTransformMakeRotation, 502
  - CGRectMake( ), 478
- G**
- g (gravity) unit, accelerometer, 510
- Game Kit framework, 87
- gdb (GNU Debugger), 604
  - breakpoints, 605-608
  - Debugger view, 613-615
  - flow of program execution, 609-612
  - variable states, 608-609
  - watchpoints, 612-613
- GDB Pocket Reference**, 629
- Generate Story button, 175
- generating alerts, 245-246
  - multi-option alerts, 248-255
  - simple alerts, 246-248
- GenericViewController view
  - controller class, creating
  - universal applications, 590-596
    - adding device-specific views, 590-591
    - adding to application delegates, 591-592
    - implementation, 595-596
    - instantiating view controller, 592-594
    - iPhone and iPad views, 596
    - XIB files, 594-595
- geocoding, 560
- gesture-recognition capabilities, 489-491
  - implementation, 491
  - interface, 494-497
  - pinch recognizer, 500-502
  - project setup, 492-494
  - rotation recognizer, 503-505
  - shake recognizer, 505-506
  - swipe recognizer, 499-500
  - tap recognizer, 497-499

## Info Light button

**GetFlower action**, 125, 221  
**getter methods**, 63  
**GNU Debugger (gdb)**, 603-604  
  breakpoints, 605-608  
  Debugger view, 613-615  
  flow of program execution, 609-612  
  variable states, 608-609  
  watchpoints, 612-613  
**Google Analytics**, 653  
**Google Maps/Google Earth API**, 560  
**GPS technology**, 8, 560  
**graphics**  
  background, 202-203  
  OpenGL ES implementation, 6  
  platform, 6  
**gravity (g) unit, accelerometer**, 510  
**Greeked text**, 167  
**grouped table views**, 380-381  
**groups (projects)**, Xcode, 32-34  
**guides (IB layout tool)**, 114-115  
**gutter (Xcode)**, 605

## H

**hardware requirements**, 8  
**header files**, 33, 60  
  #import directive, 61  
  @interface directive, 61-62  
  @property directive, 63-64  
  ending, 64  
  method declaration, 62-63  
**Heavy view**, 626  
**Help menu commands**  
  Developer Documentation, 100  
  Quick Help, 102  
**hideKeyboard method**, 180, 364  
**hideModal method**, 335  
**hiding keyboard**, 179-180  
  background touch, 181-182  
  Done button, 180-181  
**Hop button**, 201-202

## I

**IBAction directive**, 138  
**IBOutlet directive**, 137

**icon files, universal applications**, 582-583  
**id return type (methods)**, 63  
**IDE (integrated development environment)**. See **Xcode**  
**identifiers, unique device**, 14  
**Identity Inspector**, 129-130  
**Identity Inspector command (Tools menu)**, 129  
**if-then-else statements**, 73  
**image animations**, 188-190  
**Image Picker**, 529  
  implementation, 544  
  cleanup, 546-547  
  displaying chosen images, 545-546  
**image resources**  
  gesture recognition project setup, 493-494  
  Split View-based Application template, 405  
**image views**, 188  
  animation, 190, 195-197  
  default image, 193  
  implementation, 189-190  
  making copies, 194  
  outlets and actions, 190-195  
  project setup, 190  
**imagenamed method**, 177  
**imagePickerController:didFinishPickingMediaWithInfo method**, 544-546  
**imagePickerControllerDidCancel delegate method**, 546  
**images, buttons**, 174-178  
**imageURLString string**, 230  
**imageWithData method**, 567  
**imperative development**, 56  
**implementation**  
  action sheets, 316  
  audio recording, 540-541  
  built-in capabilities, 561  
  connecting interface to code, 123  
  custom picker views, 299  
  Date Pickers, 289  
  file system storage, 436-457  
  flashlight application logic, 421-422  
  GenericViewController class, 595-596  
  gesture recognition, 491  
  image animations, 189-190  
  Image Picker, 544  
    cleanup, 546-547  
    displaying chosen images, 545-546  
  input techniques, 159-160  
  Media Picker, 549-550  
  Media Playground application, 529-530  
  methods, 67-76  
  modal view logic, 336-338  
  modal views, 329  
  movie playback, 535-538  
  multiview applications, 342-343, 354-355  
  Music Player, 552-553  
  reframing logic, 477-478  
  scrolling views, FlowerWeb application, 233  
  Settings application, 424-433  
  Split View-based Application template, 397  
  table views, 384  
  UIAccelerometerDelegate, 515-516, 519-520  
  using popovers with toolbars, 265  
  view controller logic, 151-152, 203  
    FlowerWeb application, 228-232  
    starting/stopping animation, 204  
  View-Based Application template, 139-140  
**implementation files**, 33, 65-66  
**implicit preferences**, 418-423  
**Import Developer Profile button**, 23  
**indexed tables**, 380  
**Info Dark button**, 173  
**Info Light button**, 173

## Info property list resource

- Info property list resource**, 45
- inheritance**, 57
- initialization of objects**, 69-70
- initWithCoder** method, 454
- initWithContentsOfURL:encoding:error** method, 572
- initWithContentURL:** method, 534
- initWithContentViewController** method, 263
- initWithFormat:** method, 231
- initWithFrame:** method, 314
- initWithMediaTypes:** method, 548
- initWithObjects** instance method, 196
- initWithQuestion:answer** method, 437
- initWithString** class method, 231
- initWithTitle** parameter, 247
- initWithURL:settings:error** method, 539
- input techniques**, 187
  - buttons, 158, 226-227
  - entering text, 159
    - application building, 183-184
    - buttons, 172-179
    - hiding keyboard, 179-182
    - implementation, 159-160
    - object release, 183
    - preparation of outlets and actions, 161-162
    - project setup, 160-161
    - text fields, 162-167
    - text views, 167-172
    - view controller logic, 182-183
  - labels, 159
  - platform, 7-8
  - scrolling views, 215, 232-238
  - segmented controls, 213, 218-222
  - sliders, 188, 197-200
  - switches, 212, 222-223
  - text fields, 158
  - view controller logic implementation, 228-232
  - views, 158
  - web views, 213-215, 224-225
- insertSubviewAtIndex:method**, 350
- installation**
  - development provisioning profile, 14-23
  - iPhone OS developer tools, 12-13
  - provisioning profile, 21-22
- instance methods**
  - `buttonTitleAtIndex`, 250
  - definition, 58
  - `initWithObjects`, 196
  - `isOn`, 212
  - `setTitle:forState`, 204
  - `stretchableImageWithLeftCapWidth:topCapHeight`, 177
  - `stringByReplacingOccurrencesOfString:WithString`, 182
  - `titleForSegmentAtIndex`, 213
- instance variables**
  - `@interface` directive, 61
  - declaration, 67-69
  - definition, 58
  - popover application view, 270-271
  - releasing, 77-78
  - text fields, 252
- instances**
  - definition, 58
  - `MKMapView`, 560
- instantiation**
  - `alertView` variable, 246-247
  - definition, 58, 109
  - modal view controllers, 333-334
  - multiview application view controllers, 345-347
  - popover view controller, 274-276
  - view controllers
    - `GenericViewController` class, 592-594
    - universal applications, 586-588
- Instruments Library**, 620
- Instruments tool**, 615
  - available instruments, 620-621
  - leak detector, 615-619
- Instruments User Guide**, 629
- int primitive data type**, 67
- integrated development environment (IDE)**. See `Xcode` integration, 557
- Address Book frameworks**, 557-565
  - contact selection, 565
  - delegate methods, 566
  - displaying contact information, 566-569
- Core Location framework**, 560, 569-573
- Map Kit framework**, 560, 569-573
- Message UI framework**, 559, 573-577
- Interface Builder**, 25, 107-108
  - connecting interfaces to code, 122
  - implementation, 123
  - launching IB from Xcode, 122
  - outlets and actions, 124-129
- Create iPhone/iPod Touch Version**, 598
- Identity Inspector**, 129-130
- rotatable/resizable interfaces**
  - flexibility, 466-471
  - project setup, 465
- user interfaces**, 112
  - customization, 117-120
  - layout tools, 114-117
  - Objects Library, 112-114
  - simulation, 120
- XIB files**, 108-112
- Interface Builder User Guide**, 130
- interface classes**, 97-99
- interface files**, 60
  - `#import` directive, 61
  - `@interface` directive, 61-62
  - `@property` directive, 63-64
  - ending, 64
  - method declaration, 62-63
- interfaces**
  - action sheets, 317-319
  - background graphics/color, 202-203
  - built-in capabilities, 562-563
  - ColorTilt application, 519
  - connection to code, 122-129

**life cycle (applications)**

- converting (universal applications), 598  
 creating with Interface Builder, 112-117  
 custom picker views, 303-304  
 customization, 117-120  
**Date Pickers**, 294-295  
 flash card application, 438-446  
 flashlight application, 419-420  
 gesture recognition, 494-497  
**Hop button**, 201-202  
 input/output techniques. See input techniques; output techniques  
**labels**, 200  
 main view (modal views), 331-332  
 Media Playground application, 532-533  
 modal UI elements, 245  
 object release, 206  
 Orientation application, 513-515  
 popovers. See popovers  
 resizable. See resizable interfaces  
 ReturnMe application, 426-427  
 rotatable. See rotatable interfaces  
 simulation, 120  
 Split View-based Application template. See Split-View based Application template  
 table views. See table views  
 user notifications. See user notifications  
 view controller logic  
     animation speed, 204-206  
     implementation, 203-204
- iPad Human Interface Guidelines**, 130
- iPad view (GenericViewController class)**, 596
- iPadViewController class**, 584
- iPhone Dev Center (Apple website)**, 9
- iPhone OS**  
     developer tools, 12-13  
     frameworks, 100-103  
     SDK (Software Development Kit), 8  
     technology layers, 85  
         Cocoa Touch, 86-87  
         Core OS, 89  
         Core Services, 88-89  
         Media, 87-88
- iPhone Simulator, testing applications**, 47, 152-153  
     esoteric conditions, 51-52  
     generating multitouch events, 50  
     Interface Builder, 120  
     launching applications, 48-49  
     rotation simulation, 50
- iPhone target**, 597-598
- iPhone view (GenericViewController class)**, 596
- iPhoneViewController class**, 585
- iPod Library, Media Picker**, 548-549  
     cleanup, 551  
     implementation, 549-550  
     Music Player, 552-553  
     playlists, 551
- iPodMusicPlayer method**, 548
- isAnimating property**, 197
- isOn method**, 212, 228
- iTunes Applications Library**, 632
- iTunes Connect, application promotion**, 650-651
- ## J-K
- keyboard**  
     customization, 165-166  
     hiding, 160, 179-182  
     input process, 159  
         buttons, 172-179  
         implementation, 159-160  
         preparation of outlets and actions, 161-162
- project setup**, 160-161  
 text fields, 162-167  
 text views, 167-172
- Keyboard (text input trait)**, 165
- keychain**, 16
- Keychain Access utility**, 635
- keys, Launch image**, 583
- ## L
- labels**, 97, 159, 200  
     action sheets, 317-318  
     adding to views, 163  
     SimpleSpin, 469
- landscape left orientation**, 463
- landscape right orientation**, 463
- landscapeView outlet**, 482
- lastAction outlet**, 303
- launch images**  
     Launch image (iPad) key, 583  
     modifying project properties, 47  
     universal applications, 583
- launching**  
     applications in iPhone Simulator, 48-49  
     Development Provisioning Assistant, 15  
     Mac OS X Installer, 13
- layers (iPhone OS)**, 85  
     Cocoa Touch, 86-87  
     Core OS, 89  
     Core Services, 88-89  
     Media, 87-88
- Layout menu commands**, 115
- layout tools (Interface Builder)**  
     alignment, 115-116  
     guides, 114-115  
     selection handles, 115  
     Size Inspector, 116-117
- leak detector (Instruments tool)**, 616-619
- Leaks instrument**, 621
- Library command (Tools menu)**, 112, 162
- Library/Caches directory**, 435
- Library/Preferences directory**, 434
- life cycle (applications)**, 89-91

## lightSource view

**lightSource view**, 421  
**limitations**  
 platform, 7  
 rotation, 480  
 single classes, 134  
**loadFirstView method**, 353  
**loadHTMLString:baseURL method**, 215  
 loading remote content, 214-215  
**loadRequest method**, 231, 409  
**loadSecondView method**, 351  
**loadThirdView method**, 351  
**location services**  
 Core Location framework, 560, 569-573  
 Map Kit framework, 560, 569-573  
**lock feature, iPhone Simulator**, 51  
**logic, view controllers**, 151-152  
 action sheets, 319  
 Date Pickers, 295-299  
 FlowerWeb application, 228-232  
 implementation, 203-206  
 multiview applications, 360  
 text entry, 182-183  
 using popovers with toolbars, 266  
**loops, repetition**, 74-76

## M

*Mac OS X Advanced Development Techniques*, 79  
**Mac OS X Installer application**, 13  
**main view, modal views**, 330-331  
 instantiating modal view controller, 333-334  
 interface, 331-332  
 outlets and actions, 333  
**Make Snapshot command (File menu)**, 40  
**managing sales, iTunes Connect**, 650-651  
**map display**, 570-573  
**Map Kit framework**, 86, 560, 569-573  
**map view, configuration**, 563  
**marketing applications**, 631-632  
 ad hoc deployment, 654-655  
 App ID, 636-637

artwork, 632-634  
**Distribution Certificates**, 634-636  
**Distribution Provisioning Profile**, 638  
**Enterprise Deployment**, 655-656  
 project configuration, 639-642  
 promotion, 649-653  
 submitting for approval, 642-643-649  
**matchResult outlet**, 303  
**measurable axes**, accelerometer, 510  
**media files**, 535  
**Media layer (frameworks)**, 87-88  
**Media Picker**, 548-549  
 cleanup, 551  
 implementation, 549-550  
 Music Player, 552-553  
 playlists, 551  
**Media Player framework**, 528, 534  
 media files, 535  
 Media layer, 87  
 movie playback, 535-538  
**Media Playground application**, 529  
 implementation, 529-530  
 interface, 532-533  
 outlets and actions, 533-534  
 project setup, 530-532  
**mediaPicker:didPickMediaItems: protocol method**, 551  
**mediaPickerController protocol method**, 551  
**memory**  
 limitations, 7  
 management, 76-78  
 object release, 152  
 warning, testing with iPhone Simulator, 51  
**menus**  
 Background, 175  
 Overview drop-down, 604  
 State Configuration, 174  
 State pop-up, 223  
 Style drop-down, 220  
**message parameter, instantiation of alertDialog variable**, 247  
**Message UI framework**, 87, 559, 573-577  
**messages, definition**, 58  
**messaging syntax**, 70-72  
**methods**. See also functions; protocols  
 ABRecordCopyVal, 568  
 accelerometer:didAccelerate, 512, 515  
 actionSheet:clickedButtonAtIndex, 288  
 addButtonWithTitle, 320  
 addSubview, 386  
 application:DidFinishLaunchingWithOptions, 357, 586, 592  
 archiveFlashCards, 456  
 archiveRootObject:toFile, 456  
 audioPlayerDidFinishPlaying: successfully:, 539, 542-543  
 autorelease, 76  
 boolForKey, 423  
 buttonTitleAtIndex, 320  
 calculate, 365  
 cancelButtonTitle, 320  
 centerMap, 571  
 chooseImage, 544  
 choosePod:, 549  
 componentsSeparatedByString, 572  
 configureView, 410  
 createFlowerData, 402  
 createStory, 182  
 currentCard, 447  
 currentDevice, 588  
 dealloc, 78, 152, 183, 206, 232, 472, 480  
 declaration in interface files, 62-63  
 definition, 37  
 dequeueReusableCellWithIdentifierWithIdentifier UITableView, 392  
 describeInteger, 605, 610  
 destructiveButtonIndex, 320  
 didCancelCardCreation, 444  
 didCreateCardWithQuestion: answer, 444  
 dismissModalViewControllerAnimated, 326, 337

**methods**

**dismissPopoverAnimated**,  
 546  
**doActionSheet**, 319  
**encodeObject:forKey**, 454  
**encodeWithCoder**, 454  
**floatForKey**, 423  
**foundPinch**, 501  
**foundRotation**, 504  
**foundSwipe**, 500  
**foundTap**, 499  
**getters**, 63  
**hideKeyboard**, 180, 364  
**hideModal**, 335  
**imagenamed**, 177  
**imagePickerController:didFinishPickingMediaWithInfo**,  
 545-546  
**imagePickerControllerDidCancel**, 546  
**imageWithData**, 567  
**implementation**  
 convenience methods,  
 69-70  
 declaration of variables,  
 67-69  
 expressions and decision  
 making, 72-76  
 files, 66  
 messaging syntax, 70-72  
 object allocation and  
 initialization, 69  
**initWithCoder**, 454  
**initWithContentsOfURL**:  
 encoding:error, 572  
**initWithContentURL:**, 534  
**initWithFormat**, 231  
**initWithFrame:**, 314  
**initWithMediaTypes:**, 548  
**initWithObjects**, 196  
**initWithQuestion:answer**, 437  
**initWithString**, 231  
**initWithURL:settings:error:**,  
 539  
**insertSubviewAtIndex**, 350  
**iPodMusicPlayer**, 548  
**isOn**, 212, 228  
**loadFirstView**, 353  
**loadHTMLString:baseURL**, 215  
**loadRequest**, 231, 409  
**loadSecondView**, 351  
**loadThirdView**, 351  
**locating**, 37  
**mediaPicker:didPickMediaItems**: protocol, 551  
**mediaPickerDidCancel**: protocol, 551  
**motionEnded:withEvent**, 505  
**MPMediaPickerControllerDelegate** protocol, 550  
**numberOfComponentsInPickerView**, 286, 306  
**pause**, 548  
**pickerView:didSelectRow**:  
 inComponent, 287,  
 309-310  
**pickerView:numberOfRowsInComponent**, 286, 306-307  
**pickerView:rowHeightForComponent**, 315  
**pickerView:titleForRow**:  
 forComponent, 287, 307  
**pickerView:viewForRow**:  
 forComponent:reusingView:,  
 313  
**pickerView:widthForComponent**, 315  
**play**, 534, 548  
**playAudio**, 542  
**playMedia**, 535  
**playMediaFinished**, 537  
**popover controllers**, 263-264  
**popoverControllerDidCancel**,  
 547  
**popoverControllerDidDismiss**:  
 Popover, 264, 278, 551  
**presentModalViewController**:  
 animated, 326  
**record**, 539  
**recordAudio**, 540  
**registerDefaults**, 432  
**release**, 152  
**removeFromSuperview**, 353  
**requestWithURL**, 214-215  
**resignFirstResponder**, 180  
**return types**, 63  
**selectedRowInComponent**,  
 309  
**sendEmail**, 575  
**setBool**, 422  
**setDelegate**, 543  
**setFloat**, 422  
**setFullscreen:animated**, 534  
**setLightSourceAlphaValue**,  
 420  
**setQueueWithItemCollection**,  
 548  
**setRegion:animated**, 570  
**setSpeed**, 191, 200  
**setters**, 63  
**setText**, 392  
**setTitle:forState**, 204  
**setToRecipients**, 574  
**setValuesFromPreferences**,  
 432  
**shouldAutorotateToInterfaceOrientation**, 462  
**showDate**, 293, 297  
**showFromRect:inView**:  
 animated, 321  
**showInView**, 319, 321  
**showModal**, 330, 336  
**showNextCard**, 447  
**standardUserDefaults**, 422  
**startAnimating**, 197  
**stop**, 539  
**stretchableImageWithLeftCapWidth:topCapHeight**, 177  
**stringByAppendingPathComponent**, 435  
**stringByReplacingOccurrencesOfString:WithString**, 182  
**stringFromDate**, 296  
**System Sound Services**, 241  
**tableView:cellForRowAtIndex**,  
 Path, 391  
**tableView:heightForRowAtIndexPath**, 408  
**tableView:titleForHeaderInSection**,  
 Section, 391, 406  
**timeIntervalSinceDate**,  
 289, 297  
**titleForSegmentAtIndex**, 213

## methods

- toggleAnimation, 191, 204
- toggleFlowerDetail, 217, 223, 228
- UIAlertView, 241
- UIPickerViewDelegate, 311-315
- unarchiveObjectWithFile, 456
- updateRightWrongCounters, 449
- updateTotal, 371
- valueForProperty:, 552
- viewDidLoad, 176, 195, 268, 305, 423, 605
- viewWillDisappear, 422
- MKMapView instance**, 560
- Mobile Safari**, 651
- modal UI elements**, 245
- modal views**, 325
  - controllers, 333-334
  - implementation, 329
  - logic, 336-338
  - main view, 330-334
  - preparing the view, 334-336
  - project setup, 329-330
  - styles and transitions, 326-328
- modalContent outlet**, 330
- modalContent view controller**, 337
- modalPresentationStyle property**, 327
- Model-View-Controller structure**.
  - See **MVC structure**
- models**, **MVC structure**, 135, 138-139
- modifying project properties**, 45-47
- monitoring sales**, iTunes Connect, 650-651
- motionEnded:withEvent method**, 505
- movement sensing**, 522-523
- movie playback**, 535-538
- MPMediaItem class**, 528
- MPMediaItemCollection class**, 528, 551
- MPMediaPickerController**, 528-529, 548-550
- MPMoviePlayerController class**, 528, 534
- MPMoviePlayerPlaybackDidFinishNotification notification**, 538
- MPMusicPlayerController class**, 528, 548
- multi-option alerts**, 248
  - adding fields to alerts, 251-255
  - Alert View Delegate protocol, 250-251
  - buttons, 248-249
- multitouch events**, iPhone Simulator, 50
- multitouch screens**, 7, 489
  - gesture recognition, 490-491
  - implementation, 491
  - interface, 494-497
  - pinch recognizer, 500-502
  - project setup, 492-494
  - rotation recognizer, 503-505
  - shake recognizer, 505-506
  - swipe recognizer, 499-500
  - tap recognizer, 497-499
- multiview applications**, 342
  - area view, 361
  - area calculation logic, 365-367
  - creating the view, 362-364
  - outlets and actions, 361-365
  - configuring view controller classes, 360
  - implementation, 342-343
  - instantiating view controllers, 345-347
  - project setup, 343-345
  - summary view, 371-374
  - tab bars, 354
    - implementation, 354-355
    - project setup, 355-357
    - tab bar controllers, 357-360
  - toolbar controls, 347
    - adding/editing buttons, 347-348
    - clearing current view, 352-354
  - implementing view switch methods, 350-351
- outlets and actions, 349-350
- setting view with application start, 352
- versus single-view**, 341-342
- volume view**, 367
  - creating the view, 368-370
  - outlets and actions, 367-370
  - volume calculation logic, 370-371
- MultiViewsViewController object**, 349
- Music Player**, 552-553
- musicPickerController object**, 550
- MVC structure (Model-View-Controller)**, 26, 133
  - application design, 134-135
  - controllers, 136-138
  - data models, 138-139
  - View-Based Application template. See View-Based Application template, 136
- myHTML string**, 215

## N

- naming provisioning profiles**, 18-20
- navigating code**, 36-42
- navigation controllers**, 383, 398
- navigation events (Split View-based Application template)**, 408-409
- nested messaging**, 71-72
- New project command (File menu)**, 31
- New Smart Group command (Project menu)**, 34
- newBFF action**, 561
- NeXTSTEP platform**, 85
- nil value**, 71
- non-animated action sheets**, 288
- non-atomic attribute**, 64
- notifications**. See user notifications
- NSArray class**, 95
- NSCoder object**, 454

**NSCoding protocol**, 454  
**NSDate class**, 96  
**NSDateFormatter object**, 289, 296, 299  
**NSDecimalNumber class**, 95-96  
**NSDictionary class**, 95  
**NSIndexPath object**, 392  
**NSLog function (debugging tool)**, 602-603  
**NSMutableArray class**, 95, 402  
**NSMutableDictionary class**, 95, 403  
**NSMutableString class**, 94  
**NSNotificationCenter class**, 537  
**NSNumber class**, 95-96  
**NSObject class**, 59, 92  
**NSSearchPathForDirectoriesInDomains C function**, 435  
**NSString class**, 94, 182  
**NSTemporaryDirectory C function**, 436  
**NSURL class**, 96-97, 214  
**NSURLRequest class**, 214  
**NSUserDefaults class**, 418  
**numberOfComponentsInPickerView method**, 286, 306  
**numberOfTapsRequired property**, 498  
**numberOfTouchesRequired property**, 498  
**numbers**, 95-96

**O**

**Object Allocations instrument**, 621  
**object archiving**, 453-455  
**object data types, declaration of variables**, 68-69  
**object graphs**, 453  
**Object-Oriented Programming with Objective-C document**, 79  
**object-oriented programming. See OOP**  
**Objective-C**, 26, 55-60  
  decision-making, 72-76  
  file structure, 60-66  
  memory management, 76-78  
  messaging syntax, 70-72

method implementation, 67-69  
  object allocation and initialization, 69-70  
**Objective-C 2.0 Programming Language (document)**, 79  
**objects. See also classes**  
  adding to scrolling views, 235-236  
  adding to views, 145-149  
  allocation and initialization, 69-70  
  application, 92  
  definition, 58  
  instantiation, 109  
  messaging syntax, 70-72  
  MPMediaItemCollection, 551  
  MultiViewsViewController, 349  
  musicPickerPopoverController, 550  
  NSCoder, 454  
  NSDateFormatter, 289, 296, 299  
  NSIndexPath, 392  
  release, 152, 183, 206  
  convenience methods, 69-70  
  custom picker views, 301  
  dealloc method, 472, 480  
  FlowerWeb application, 232, 238  
  memory management, 76  
  popover application view, 271  
  popovers, 268  
  retaining, 77  
  SplitViewController, 398-399  
  switch, 98  
  UIAcceleration, 512  
  UIBarButton, 347  
  UIBarButtonItem, 264  
  UIDatePicker, 285, 289-294  
  UIImagePickerController, 529  
  UIImageView, 363  
  UINavigationBar, 398  
  UINavigationController, 398  
  UINavigationItem, 399  
  UIPickerView, 285-287, 299-301  
  UIPopoverController, 531  
  UISwitch, 264  
  UITabBar, 354  
  UITabBarController, 354, 357  
  UITableView, 380, 383-388  
  UITableViewController, 380  
  UIToolbar, 264, 273, 347  
  UIViewController, 342-343  
  window, 92  
**Objects Library (Interface Builder)**, 112-114  
**Online Certificate Status Protocol (OSCP)**, 635  
**onscreen controls (UIControl class)**, 93  
**OOP (object-oriented programming)**, 55  
  definition, 56-57  
  Objective-C, 26, 55-60  
  decision-making, 72-76  
  declaration of variables, 67-69  
  file structure, 60-66  
  memory management, 76-78  
  messaging syntax, 70-72  
  object allocation and initialization, 69-70  
  terminology, 57-58  
**Open GL ES**  
  framework, 87  
  implementation, 6  
  instrument, 621  
**OpenStep platform**, 85  
**Orientation application**, 513  
  interface, 513-515  
  project setup, 513  
  UIAccelerometerDelegate implementation, 515-516  
**orientation constants**, 463  
**origins, Cocoa Touch**, 85  
**OSCP (Online Certificate Status Protocol)**, 635  
**Other Sources (code files)**, 33  
**otherButtonTitles parameter**, 247-248  
**outlets**, 124-129, 161-162

## outlets

action sheets, 303, 318-319  
area view (multiview applications), 361-365  
built-in capabilities, 563  
ChosenColor, 125  
colorChoice, 217, 221  
connection to  
    image views, 194-195  
notification project  
    interface, 243-245  
popovers, 269  
scrolling views, 237  
sliders, 199  
text fields, 166-167  
text views, 171-172  
web views, 225  
custom picker views, 301  
deviceType, 589  
flashlight application, 420-421  
flowerDetailView, 225  
FlowerView, 125, 225  
FlowerWeb application, 217-218  
gesture recognition interface, 496-497  
image views, 190-192  
landscapeView, 482  
lastAction, 303  
main view (modal views), 333  
matchResult, 303  
Media Playground application, 533-534  
modalContent, 330  
multiview application  
    toolbars, 349-350  
padViewController, 585  
pinchView, 496  
popover application view, 270-273  
popover content, 267  
portraitView, 482  
presentationStyle, 333  
Reframe application project, 471-472  
rotateView, 496  
scrolling views, 234  
segmented controls, 221  
swipeView, 496  
tabBarController, 358

table view applications, 385-386  
tapView, 496  
theScroller, 237  
transitionStyle, 333  
userOutput, 242  
view controllers, 144-145, 149-150  
volume view (multiview applications), 367-370  
**output labels, 123**  
    action sheets, 303  
    Date Pickers, 294-295  
**output techniques (image views), 187, 211. See also input techniques**  
    animation, 195-197  
    animation resources, 190  
    default image, 193  
    implementation, 189-190  
    making copies, 194  
    outlets and actions, 190-195  
    project setup, 190  
Overview drop-down menu, 604

## P

**padViewController outlet, 585**  
**paid developer programs, 11**  
**parameters**  
    definition, 58  
    instantiation of alertDialog variable, 247-248  
    Quick Help results, 103  
**parent classes, 58**  
**parentViewController**  
    property, 338  
**paste, 167**  
**pause method, 548**  
**Photo Library, 544-547**  
**picker views, 285-287, 299**  
    implementation, 299  
    interface, 303-304  
    project setup, 300-301  
    protocols, 302-303  
    providing data to, 304-308  
    response to users, 309-311  
    UIPickerViewDataSource protocol, 286  
    UIPickerViewDelegate protocol, 287, 311-315

**pickers, 284**  
    Date Pickers. See Date Pickers  
    Image Picker, 529, 544-547  
    Media Picker, 548-553  
    picker views. See picker views  
    UIDatePicker/UIPicker class, 99  
**pickerView:didSelectRow:inComponent method, 287, 309-310**  
**pickerView:numberOfRowsInComponent method, 286, 306-307**  
**pickerView:rowHeightForComponent: method, 287, 315**  
**pickerView:titleForRow:forComponent: method, 307**  
**pickerView:viewForRow:forComponent:reusingView: method, 313**  
**pickerView:widthForComponent: method, 315**  
**pinch gesture recognizer, 500-502**  
**pinchView outlet, 496**  
**placeholder text, 164**  
**plain table views, 380-381**  
**platform, 5**  
    connectivity, 7  
    display and graphics, 6  
    feedback mechanisms, 7-8  
    input mechanisms, 7-8  
    limitations, 7  
    NeXTSTEP, 85  
    OpenStep, 85  
**play method, 534, 548**  
**playAudio: method, 542**  
**playing alert sounds, 256-258**  
**playlists (Media Picker), 551**  
**playMedia: method, 535**  
**playMediaFinished: method, 537**  
**plist files, universal applications, 582-583**  
**pointers, 68-69**  
**PopoverConfigViewController classes, 266**  
**popoverControllerDidCancel method, 547**

- popoverControllerDidDismiss-**  
  Popover method, 264, 278, 551
- PopoverPlayground – Skeleton**  
  project (Date Pickers), 289  
    adding, 291-294  
    implementation, 289  
    interface, 294-295  
    project setup, 290-291  
    view controller logic, 295-299
- popovers, 261**  
  action sheets, 287, 316  
    animated versus  
      non-animated, 288  
    changing appearance and  
      behavior, 321  
    implementation, 316  
    interface, 317-319  
    project setup, 316-317  
    response to user, 320  
    UIActionSheetDelegate  
      protocol, 288  
    view controller logic, 319
- arrow constants, 278
- controllers, 264-264
- display, 262
- MPMediaPickerController, 529
- pickers, 284  
  Date Pickers, 285,  
    289-299  
  picker views, 285-287,  
    299-315
- toolbars, 264  
  additional view controller  
    classes, 266  
  application logic, 276-279  
  implementation  
    overview, 265  
  preparing application view,  
    270-276  
  preparing content, 267-269  
  project setup, 265
- UIPopoverController class, 99
- views, 263
- populating**  
  data structures,  
    305-306, 405  
  table view cells, 391-394
- portrait orientation, 463**
- portrait upside-down**  
  orientation, 463
- portraitView outlet, 482**
- Position setting (Size Inspector), 116**
- Postman (Freeverse), 652**
- pragma marks, 41-42**
- preferences, 415**  
  design considerations,  
    415-417  
  file system storage  
    implementation, 436-457  
  iPad file system sandbox,  
    433-436  
  reading and writing, 418-423  
  Settings application, 424-433
- Preferences command (Xcode menu), 101**
- PreferencesSpecifiers**  
  property, 428
- premature optimization, 622**
- presentation**  
  modal views, styles and  
    transitions, 326-328  
  segmented controls, 220
- presentationStyle outlet, 333**
- presentModalViewControllerAnimated:**  
  animated method, 326
- presentPopoverFromBarButtonItem:permittedArrowDirections:**  
  animated method, 263
- pricing applications, 653-654**
- primitive data types, 67-68**
- procedural programming, 56**
- profiles**  
  development provisioning, 13  
    generation and  
      installation, 14-23  
    testing, 24-25  
    distribution, 14
- program execution, GNU Debugger, 609-612**
- programming**  
  Objective-C, 26, 55-60  
    decision-making, 72-76  
    declaration of variables,  
      67-69
- file structure, 60-66
- memory management, 76-78
- messaging syntax, 70-72
- object allocation and initialization, 69-70
- OOP, 56-57
- definition, 56-57
- terminology, 57-58
- Programming in Objective-C 2.0, Second Edition, 79**
- Project menu commands**  
  New Smart Group, 34  
  Set Active Build  
    Configuration, Debug, 604
- projects**  
  configuration, 639-642  
  management (Xcode), 31-35  
  setup  
    action sheets, 316-317  
    built-in capabilities, 561-562  
    ColorTilt application, 518  
    creating rotatable and resizable interfaces, 465  
    custom picker views, 300-301  
    Date Pickers, 290-291  
    entering text, 160-161  
    flash card application, 436-438  
    flashlight application, 418-419  
    FlowerWeb application, 217  
    gesture recognition, 492-494  
    image views, 190  
    Media Playground  
      application, 530-532  
    modal views, 329-330  
    multiview applications, 343-345, 355-357  
    Orientation application, 513  
    reframing controls on rotation, 471-477  
    ReturnMe application, 424-425

## projects

- scrolling views, 234
- Split View-based Application template, 398-401
- swapping views on rotation, 479-480
- table views, 384-388
- using popovers with toolbars, 265
- View-Based Application template. See View-Based Application template
- promotion of applications, 649**
  - iTunes Connect, 650-651
  - websites and social networks, 651-653
- properties**
  - animationDuration, 196
  - bounds, 483
  - contentSize, 233
  - definition, 58
  - frame, 464
  - isAnimating, 197
  - locating, 37
  - modalPresentationStyle, 327
  - modifying, 45-47
  - parentViewController, 338
  - PreferencesSpecifiers, 428
  - Reframe application project, 471-472
  - scale, 500
  - startAnimating, 197
  - stopAnimating, 197
  - tap gesture recognizer, 498
  - transform, 483
  - velocity, 500
- Property List Editor, 423, 427**
- protocols. See also functions; methods**
  - ABPeoplePickerNavigationControllerDelegate, 564
  - Alert View Delegate, multi-option alerts, 250-251
  - AVAudioPlayerDelegate, 539, 542
  - CreateCardDelegate, 444, 451
  - definition, 62
- imagePickerController:didFinishPickingMediaWithInfo, 544
- NSCoding, 454
- UIAccelerometerDelegate, 513, 518
- UIAccelerometerDelegate-Protocol, 511
- UIActionSheetDelegate, 288, 317
- UIPickerViewDataSource, 286, 300-303
- UIPickerViewDelegate, 287, 300-303
- UIPopoverControllerDelegate, 264
- UITabBarControllerDelegate, 356
- UITableViewDataSource, 389-390
- UITableViewDelegate, 389
- provisioning profiles, 13**
  - generation and installation, 14-23
  - testing, 24-25
- push buttons, 123**
- Q-R**
  - Quartz Core framework, 88
  - Quick Help assistant (Xcode), 102-103
  - Quick Help command (Help menu), 102
  - radio buttons, 212
  - RAM limitations, 7
  - reactions, shake gesture recognizer, 506
  - reading application preferences, 418-423
  - recognizers (gestures), 491
    - implementation, 491
    - interface, 494-497
    - pinch recognizer, 500-502
    - project setup, 492-494
    - rotation recognizer, 503-505
    - shake recognizer, 505-506
    - swipe recognizer, 499-500
    - tap recognizer, 497-499
  - record method, 539
  - recordAudio: method, 540
  - reframing, 464
    - controls, 471-477
    - implementation of reframing logic, 477-478
  - registerDefaults method, 432
  - registration**
    - Apple Developer Program, 11-12
    - Apple's website, 9
  - related API, Quick Help results, 103**
  - related documents, Quick Help results, 103**
  - Release build configuration, 604**
  - release**
    - instance variables, 77-78
    - objects, 183, 206
      - convenience methods, 69-70
    - custom picker views, 301
    - dealloc method, 472, 480
    - FlowerWeb application, 232, 238
    - memory management, 76
    - popover application view, 271
    - popovers, 268
    - release method, 152
    - rules, 78
  - remote content, 214-215**
  - removeFromSuperview instance method, 353**
  - removing breakpoints, 607**
  - repetition, loops, 74-76**
  - Request Promotional Codes feature (iTunes Connect), 651**
  - requestWithURL method, 214-215**
  - requirements, hardware, 8**
  - resignFirstResponder method, 180**
  - resizable interfaces, 461-462**
    - design, 464-465
    - implementation of reframing logic, 477-478
  - Interface Builder, 465**
    - flexibility, 466-471
    - project setup, 465
  - reframing controls, 471-477**

## settings (application preferences)

- swapping views, 479
    - interface creation, 481-482
    - project setup, 479-480
    - view-swapping logic, 483-485
  - resources**
    - adding to projects, 35
    - removal from projects, 35-36
  - Resources group files**, 33
  - responders**
    - first, 179
    - UIResponder class, 93
  - response to gesture recognizers**
    - pinch gesture, 501-502
    - rotation gesture, 504-505
    - swipe gesture, 500
    - tap gesture, 499
  - results, Shark profiler**, 626-629
  - retain attribute**, 64
  - retaining objects**, 77
  - Return Key (text input trait)**, 165
  - return types (methods)**, 63
  - return value, Quick Help**
    - results, 103
  - ReturnMe application**, 424
  - reverse geocoding**, 560
  - Rich Media**, 527
    - AV Foundation framework, 528-529, 539-544
    - Image Picker, 529, 544-547
    - Media Picker, 548-553
    - Media Player framework, 528, 534-538
    - Media Playground application, 529-534
  - Robbin, Arnold, GDB Pocket Reference**, 629
  - root class, NSObject**, 92
  - root view table controllers**, 406-408
  - rotatable interfaces**, 461-462
    - design, 464-465
    - implementation of reframing logic, 477-478
  - Interface Builder, 465
    - flexibility, 466-471
    - project setup, 465
  - orientation constants, 463
    - reframing controls, 471-477
    - swapping views, 479-485
  - rotateView outlet**, 496
  - rotation**
    - gesture recognizer, 503-505
    - testing with iPhone Simulator, 50
  - Rounded Rect button**, 172-173
  - rows (table views)**, 394-396
  - rules, releasing**, 78
  - Run command (Run menu)**, 43
- S**
- sales management, iTunes Connect**, 650-651
  - sample code, Quick Help**
    - results, 103
  - sandbox**, 433
  - scale property**, 500
  - Schema Reference (Settings application)**, 428
  - screen rotation**, 461-462
    - designing rotatable interfaces, 464
    - auto-rotating, 464
    - implementation of reframing logic, 477-478
    - Interface Builder, 465-471
    - reframing, 464
    - reframing controls, 471-477
    - swapping views, 465, 479-485
    - orientation constants, 463
  - scrolling options (text views)**, 170-171
  - scrolling views, FlowerWeb application**, 215, 232
    - adding objects, 235-236
    - connection to outlets, 237
    - implementation, 233
    - implementing scrolling behavior, 237-238
    - outlets, 234
    - project setup, 234
  - SDK (Software Development Kit)**, 8
  - search results, Xcode documentation**, 101
  - Secure (text input trait)**, 165
  - Security framework**, 89
  - segmented controls**, 123, 213
    - appearance selection, 220
    - configuration of segments, 219
    - connection to actions, 221-222
    - connection to outlets, 221
    - FlowerWeb application, 218-222
    - sizing, 220
  - UISegmentedControl class**, 98
  - selectedRowInComponent: method**, 309
  - selection handles (IB layout tool)**, 115
  - sendEmail method**, 561, 575
  - sender variable**, 180
  - sensing movement**, 522-523
  - Set Active Build Configuration, Debug command (Project menu)**, 604
  - setBool method**, 422
  - setDelegate method**, 543
  - setFloat method**, 422
  - setFullscreen:animated method**, 534
  - setLightSourceAlpha action**, 418
  - setLightSourceAlphaValue method**, 420
  - setQueueWithItemCollection method**, 548
  - setRegion:animated method**, 570
  - setSpeed method**, 191, 200-204
  - setter methods**, 63
  - setText method**, 392
  - setting**
    - button images, 174-178
    - text view scrolling options, 170-171
  - settings (application preferences)**, 415

## settings (application preferences)

- design considerations, 415-417
- file system storage implementation, 436-457
- iPad file system sandbox, 433-436
- reading and writing, 418-423
- Settings application, 416-417
  - application preferences, 424-433
  - Schema Reference, 428
  - Settings Bundle, 416, 427-431
- Settings Bundle**, 416, 427-431
- setTitle:forState** instance method, 204
- setToRecipients** method, 574
- setValuesFromPreferences** method, 432
- shake gesture**
  - testing with iPhone Simulator, 51
  - recognizer, 505-506
- Shark profiler**, 622
  - attaching to an application, 622-625
  - interpretation of results, 626-629
- Shark User Guide**, 629
- shouldAutorotateToInterfaceOrientation:** Orientation method, 462
- showDate:** method, 293, 297
- showFromRect:inView:animated:** method, 321
- showInView:** method, 319-321
- showModal** method, 330, 336
- showNextCard** method, 447
- simple alerts, 246-248
- SimpleSpin** label, 469
- Simulate Interface** command (File menu), 120, 471
- simulation, user interfaces, 120
- Simulator**, testing applications, 152-153
- single classes, 134
- single-view applications, 341-342
- singleton classes, 418
- Singleton** pattern, 418
- singletons
  - definition, 58
- UIAccelerometer**, 511
- Size Inspector**, 220, 467
  - IB layout tool, 116-117
  - Size setting, 116
  - Tools menu command, 116
- sizing controls**, 220
- sliders**, 188, 197
  - connection to actions, 200
  - connection to outlets, 199
  - range attributes, 198-199
  - UISlider class, 98
- smart groups**, 34
- snapshots**, 39-40
- Snapshots** command (File menu), 40
- Snow Leopard**, launching Mac OS X Installer, 13
- social networks**, application promotion, 651-653
- Software Development Kit (SDK)**, 8
- soundID** variable, 257
- soundRecorder** (audio recorder), 541
- sounds**, alerts, 255-258
- Split View-based Application template**, 382-383, 396
  - Detail view controller, 409-410
  - implementation, 397
  - navigation events, 408-409
  - project setup, 398-401
  - providing data to, 401-405
  - root view table controller, 406-408
- SplitViewController** object, 398-399
- Stallman, Richard**, *Debugging with GDB: The GNU Source-Level Debugger*, 629
- standard program** (Developer Program), 9
- standardUserDefaults** method, 422
- startAnimating** method, 197
- startAnimating** property, 197
- starting animation**, 197
- state changing**, 174
- State Configuration** menu, 174
- State pop-up** menu, 223
- statements**
  - if-then-else, 73
  - Objective-C, 59
  - switch, 73
- status bar display**, modifying project properties, 46
- Step Into** icon (debugger), 609
- Step Out** icon (debugger), 610
- Step Over** icon (debugger), 609
- stop** method, 539
- stopAnimating** property, 197
- stopping animation**, 197
- storage of data**, iPad file system
  - sandbox, 433
  - file paths, 435-436
  - implementation, 436-457
  - storage locations, 434-435
- Store Kit** framework, 88
- stretchableImageWithLeftCapWidth:topCapHeight** instance method, 177
- String Programming Guide for Cocoa**, 602
- stringByAppendingPath-** Component method, 435
- stringFromDate:** method, 296
- strings**, 94
  - actionMessage**, 310
  - Date Format**, 297
  - detailURLString**, 230
  - format specifiers**, 602
  - imageURLString**, 230
  - myHTML**, 215
- structure**, MVC (Model-View-Controller)
  - application design, 134-135
  - controllers, 136-138
  - data models, 138-139
  - View-Based Application** template. See **View-Based Application** template
  - views, 136
- style**
  - modal views, 326-328
  - table views, 396
- Style** drop-down menu, 220
- subclasses**
  - definition, 58
- UIViewController**, 343

## tilt, ColorTilt application

**submitting applications for approval**, 642-643  
 binary upload, 648-649  
 profile preparation, 643-648

**summary view, multiview applications**, 371-374

**superclasses**, 58

**supported codecs**, 535

**swapping views, rotatable /resizable interfaces**, 465  
 interface creation, 481-482  
 project setup, 479-480  
 view-swapping logic, 483-485

**swipe gesture recognizer**, 499-500

**swipeView outlet**, 496

**switch methods, multiview applications**, 350-351

**switch objects**, 98

**switch statements**, 73, 391

**switches**, 212, 222-223

**Sympathy Image group (ReturnMe preferences)**, 428

**System Configuration framework**, 89

**System framework**, 89

**System Sound Services C-style interface**, 255-256

**System Sound Services method**, 241

**System Usage instrument**, 621

## T

**tab bar controllers, multiview applications**, 357-360

**tab bars, multiview applications**  
 implementation, 354-355  
 project setup, 355-357  
 tab bar controllers, 357-360

**tabBarController outlet**, 358

**table view controllers**, 387-388

**table views**, 380  
 appearance, 396  
 implementation, 384  
 plain versus grouped, 380-381

project setup, 384-388  
 providing data to, 389-394  
 reacting to a row touch event, 394-395

**tableView:cellForRowAtIndexPath method**, 391

**tableView:heightForRowAtIndexPath method**, 408

**tableView:titleForHeaderInSection method**, 391, 406

**tap gesture recognizer**, 497-499

**tapView outlet**, 496

**targets**, 580

**technologies**, 25  
 Apple Developer Suite, 25-26  
 Interface Builder, 107-130  
 iPhone Simulator, 47-52  
 Xcode, 29-47

Cocoa Touch, 26  
 core classes, 91-94  
 data type classes, 94-97  
 functionality, 84-85  
 interface classes, 97-99  
 origins, 85

MVC structure, 26, 133  
 application design, 134-135  
 controllers, 136-138  
 data models, 138-139

**View-Based Application template**, 139-153  
 views, 136

Objective-C, 26, 55-60  
 decision-making, 72-76  
 declaration of variables, 67-69  
 file structure, 60-66  
 memory management, 76-78  
 messaging syntax, 70-72  
 object allocation and initialization, 69-70

**technology layers (iPhone OS)**, 85  
 Cocoa Touch, 86-87  
 Core OS, 89  
 Core Services, 88-89  
 Media, 87-88

**templates, Xcode**, 31. *See also View-Based Application template*

**testing applications**  
 development provisioning profile, 24-25  
 FlowerWeb application, 232, 238  
 iPhone Simulator, 47-52, 120  
 rotation, 467  
 View-Based Application template, 152-153

**text**  
 Greeked, 167  
 input process, 159  
 application building, 183-184  
 buttons, 172-179  
 hiding keyboard, 179-182  
 implementation, 159-160  
 object release, 183  
 preparation of outlets and actions, 161-162  
 project setup, 160-161  
 text fields, 162-167  
 text views, 167-172  
 view controller logic, 182-183  
 input traits, 165  
 placeholder, 164

**text fields**, 158, 162  
 access, 254-255  
 attributes, 163-165  
 connection to outlets, 166-167  
 instance variables, 252  
 keyboard customization, 165-166  
 subviews, 253-254  
**UITextField/UITextView class**, 98

**text views**, 167  
 connection to outlets, 171-172  
 editing attributes, 168-171

**theScroller outlet**, 237

**tilt, ColorTilt application**, 518-520

## timeIntervalSinceDate: method

**timeIntervalSinceDate:** method, 289, 297

**timestamp**, **UIAcceleration object**, 513

**titleForSegmentAtIndex** instance method, 213

**tmp directory**, 435

**toggleAnimation** method, 191, 204

**toggleFlowerDetail** method, 217, 223, 228

**toolbars**

  buttons, 262

  controls, multiview

    applications, 347-354

  popovers, 264

    additional view controller classes, 266

    application logic, 276-279

    application view, 273

    implementation overview, 265

    preparing application view, 270-276

    preparing content, 267-269

    project setup, 265

**tools**, 25

  Apple Developer Suite, 25-26  
    Interface Builder, 107-130  
    iPhone Simulator, 47-52  
    Xcode, 29-47

  Cocoa Touch, 26, 83

    core classes, 91-94

    data type classes, 94-97

    functionality, 84-85

    interface classes, 97-99

    origins, 85

  debugging, 601, 615-621

  iPhone OS, 12-13

  MVC structure, 26, 133  
    application design, 134-135

    controllers, 136-138

    data models, 138-139

    View-Based Application template, 139-153

    views, 136

  Objective-C, 26, 55-60

    decision-making, 72-76

declaration of variables, 67-69

file structure, 60-66

memory management, 76-78

messaging syntax, 70-72

object allocation and initialization, 69-70

universal applications, 597-598

### Tools menu commands

  Attributes Inspector, 117, 163

  Connections Inspector, 178

  Identity Inspector, 129

  Library, 112, 162

  Size Inspector, 116

### tracing applications

#### traits, text input

#### transform property

#### transitions, modal views

#### transitionStyle outlet

#### Tree view, Shark profiler results

#### tutorials, user interface controls

## U

**UIAcceleration object**, 512

**UIAccelerometer singleton**, 511

**UIAccelerometerDelegate implementation**, 515-516, 519-520

**UIAccelerometerDelegate protocol**, 511, 513, 518

**UIActionSheet class**, 287-288

**UIActionSheetDelegate protocol**, 288, 317

**UIAlertView class, generating alerts**, 245

  adding fields to alerts, 251-255

  multi-option alerts, 248-251

  simple alerts, 246-248

**UIAlertView method**, 241

**UIApplication class**, 92

**UIBarButtonItem object**, 347

**UIBarButtonItem toolbar button**, 262

**UIBarButtonItem object**, 264

**UIButton class**, 97, 226

**UIControl class**, 93

**UIDatePicker object**, 99, 285

  adding Date Pickers, 291-294

  implementation, 289

  project setup, 290-291

**UIDevice class**, 588

**UIEvent class**, 490

**UIImage**, 195

**UIImagePickerController object**, 529, 544

**UIImageView object**, 188, 190-192, 363

**UIKit framework**, 86

**UILabel class**, 97, 200

**UIModalTransitionStyleCover-Vertical transition**, 328

**UIModalTransitionStyleCross-Dissolve transition**, 328

**UIModalTransitionStyleFlip-Horizontal transition**, 328

**UIModalTransitionStylePartialCurl transition**, 328

**UINavigationBar object**, 398

**UINavigationController object**, 398

**UINavigationItem object**, 399

**UIPanGestureRecognizer class**, 490

**UIPicker class**, 99

**UIPickerView object**, 285

  implementation, 299

  project setup, 300-301

  UIPickerViewDataSource protocol, 286, 300-303

  UIPickerViewDelegate protocol, 287, 300-303, 311-315

**UIPickerViewDataSource protocol**, 286, 300-303

**UIPickerViewDelegate protocol**, 287, 300-303, 311-315

**UIPinchGestureRecognizer class**, 490

**UIPopoverController class**, 99, 262-264, 531

**UIPopoverControllerDelegate protocol**, 264

**UIPrerenderedIcon key**, 634

**UIPressGestureRecognizer class**, 490

**view controllers**

- UIResponder class**, 93  
**UIRotationGestureRecognizer class**, 490  
**UIScrollView**, 215, 233  
**UISegmentedControl class**, 98, 213  
**UISlider class**, 98, 188, 197  
**UISwipeGestureRecognizer class**, 490  
**UISwitch class**, 98, 212, 222, 264  
**UITabBar object**, 354  
**UITabBarController object**, 354, 357  
**UITabBarControllerDelegate protocol**, 356  
**UITableView object**, 380, 383  
  implementation, 384  
  project setup, 384-388  
**UITableViewController object**, 380  
**UITableViewDataSource protocol**, 389-390  
**UITableViewDelegate protocol**, 389  
**UITapGestureRecognizer class**, 490  
**UITextField class**, 98  
**UITextView class**, 98, 167  
**UIToolbar object**, 264, 273, 347  
**UIView class**, 92, 213  
**UIViewController class**, 94, 263, 342-343  
**UIWebView**, 214, 224  
**unarchiveObjectWithFile method**, 456  
**unique device identifiers**, 14  
**universal applications**, 579-580  
  GenericViewController view  
  controller class, 590-596  
  tools, 597-598  
  Window-based template, 581-590  
**updateRightWrongCounters method**, 449  
**updates, applications**, 653  
**updateTotal method**, 371  
**upgrading iPhone target**, 597-598  
**URLs**, 96-97
- user defaults. See preferences**  
**user input/output**, 187  
  buttons, 158, 226-227  
  image views, 188, 192  
  animation, 195-197  
  animation resources, 190  
  default image, 193  
  implementation, 189-190  
  making copies, 194  
  outlets and actions, 190-195  
  project setup, 190  
  labels, 159  
  scrolling views, 215, 232-238  
  segmented controls, 213, 218  
  appearance selection, 220  
  configuration of segments, 219  
  connection to actions, 221-222  
  connection to outlets, 221  
  sizing controls, 220  
  sliders, 188, 197-200  
  switches, 212, 222-223  
  text, 159  
  application building, 183-184  
  buttons, 172-179  
  hiding keyboard, 179-182  
  implementation, 159-160  
  object release, 183  
  preparation of outlets and actions, 161-162  
  project setup, 160-161  
  text fields, 162-167  
  text views, 167-172  
  view controller logic, 182-183  
  text fields, 158  
  view controller logic implementation, 228-232  
  views, 158  
  web views, 213-215, 224-225
- user interfaces**  
  connection to code, 122-129  
  creating with Interface Builder, 112-117
- customization, 117-120  
  simulation, 120
- user notifications**, 241  
  alert methods, 241  
  connecting to outlets and actions, 243-245  
  creating notification project interface, 243  
  prepping project files, 242-243  
  alert sounds  
  playing sounds, 256-258  
  System Sound Services C-style interface, 255-256  
  generating alerts, 245  
  multi-option alerts, 248-255  
  simple alerts, 246-248  
  movie playback, 537
- user preferences. See preferences**  
**userOutput outlet**, 242

**V**

- valueForProperty: method**, 552  
**variables**  
  alertView, 246-247  
  declaration, 67-69  
  definition, 58  
  GNU Debugger, 608-609  
  sender, 180  
  soundID, 257  
**velocity property**, 500  
**versions, testing with iPhone Simulator**, 51  
**view controller logic**  
  action sheets, 319  
  Date Pickers, 295-299  
  FlowerWeb application, 228-232  
  implementation, 203-206  
  multiview applications, 360  
  text entry, 182-183  
  using popovers with toolbars, 266  
**view controllers**  
  card, 444-445

## view controllers

ContentViewController, 333  
 logic implementation, 151-152  
 modalContent, 337  
 multiview applications, 343-347  
 MVC structure, 136-138  
 outlets and actions, 144-145  
 popovers, 270-276  
 UIViewController class, 94  
 universal applications, 585-596  
**view icon (XIB files), 110**  
**view switching, multiview applications, 350-351**  
**View-Based Application template, 139**  
 creating views, 145-150  
 implementation, 139-140  
 object release, 152  
 project setup, 140-144  
 testing application, 152-153  
 view controllers  
   logic, 151-152  
   outlets and actions, 144-145  
**view-rotation logic, 483-484**  
**view-swapping logic, 483-485**  
**viewDidLoad method, 176, 195, 268, 305, 423, 605**  
**views, 158**  
 connection to outlets, 171-172  
 Debugger (GNU Debugger), 613-615  
 definition, 111  
 editing attributes, 168-169  
 image views, 188  
 modal. See modal views  
 multiview applications. See multiview applications  
 MVC structure, 135-136  
 pickers. See picker views  
 popovers, 261-262, 268  
 scrolling, 170-171, 232-238  
 Split View-based Application template. See Split View-based Application template  
 swapping, rotatable/resizable interfaces, 465, 479-485

table views. See table views  
 UIView class, 92  
**View-Based Application template, 145-152**  
 web views. See web views  
**viewWillDisappear method, 422**  
**virtual keys, 160**  
**void return type (methods), 63**  
**volume calculation logic, 370-374**  
**volume view (multiview applications)**  
 creating the view, 368-370  
 outlets and actions, 367-370  
 volume calculation logic, 370-371

**windows, 93**  
**writing application preferences, 418-423**

## X-Y-Z

**Xcode, 29**  
 build configurations, 604  
 building applications, 42-45  
 debugging  
   GNU Debugger, 603-615  
   Instruments tool, 615-621  
   NSLog function, 602-603  
   Shark profiler, 622-629  
 documentation system  
   Cocoa Touch, 83-85,  
     91-99  
   exploration of frameworks,  
     100-103  
   editing, 36-42  
   editor, 38-39  
   gutter, 605  
   launching IB from, 122  
   modifying project properties,  
     45-47  
   navigating, 36-42  
 project management  
   adding existing  
     resources, 35  
   adding new code files, 34  
   creating a new project,  
     31-32  
   project groups, 32-34  
   removal of files and  
     resources, 35-36  
**Xcode 3 Unleashed, 79, 629**  
**Xcode Debugging Guide, 629**  
**Xcode menu commands, Preferences, 101**  
**XIB files (Interface Builder), 108**  
 Document icons, 111-112  
 Document window, 109-111  
 universal applications, 594-595  
**View-Based Application template, 142-144**

## W

**warnings, 44-45**  
**watchpoints, GNU Debugger, 612-613**  
**web views, 123, 213**  
 FlowerWeb application, 224-225  
 loading remote content, 214-215  
 supported content types, 214  
**websites**  
 Apple, 9  
 application promotion, 651-653  
**WiFi supplementation, 7**  
**WiFi technology, 560**  
**Window menu commands, Document, 192**  
**window objects, UIWindow class, 92**  
**Window-based templates (universal applications), 581**  
 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, 582-583  
 project preparation, 584