LEARNING iPad PROGRAMMING

A Hands-On Guide to Building iPad Apps

SECOND EDITION

KIRBY TURNER
TOM HARRINGTON

Foreword by Mark Dalrymple

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Praise for the First Edition of Learning iPad Programming

"This amazing, thorough book takes an interesting approach by working through the design and development of a simple, yet realistic, iPad app from start to finish. It is refreshing to see a technical book that explains how and why without inundating you with endless toy examples or throwing you into a sea of mind-numbing details. Particularly amazing is that it does this without assuming a large amount of experience at first. Yet it covers advanced topics at sufficient depth and in a logical order for all developers to get plenty of valuable information and insight. Kirby and Tom know this material and have done a great job of introducing the various frameworks and the reasoning behind how, why, and when you would use them. I highly recommend *Learning iPad Programming* to anyone interested in developing for this amazing platform."

—Julio Barros
E-String.com

"This is a great introduction to iPad programming with a well-done sample project built throughout. It's great for beginners as well as those familiar with iPhone development looking to learn the differences in developing for the larger screen."

—Patrick Burleson
Owner, BitBQ LLC (http://bitbq.com)

"Kirby Turner and Tom Harrington's *Learning iPad Programming* provides a comprehensive introduction to one of today's hottest topics. It's a great read for the aspiring iPad programmer."

—Robert Clair
Author, *Learning Objective-C 2.0*

"*Learning iPad Programming* is now my go-to reference when developing apps for the iPad. This book is an absolute treasure trove of useful information and tips for developing on the iPad. While it's easy to think of the iPad as just a bigger iPhone, there are specific topics that need to be treated differently on the iPad, such as making best use of the larger display. *Learning iPad Programming* provides an incredible amount of depth on all areas of iPad programming and takes you from design to fully functioning application—which for me is a killer feature of the book. This should be in everyone's reference library."

—Mike Daley
Author, *Learning iOS Game Programming*
Cofounder, 71Squared.com

"A truly well-rounded book with something for every iOS developer, be they aspirant or veteran. If you are new to iOS, there is a solid foundation provided in Part I that will walk you through Objective-C, the core Apple frameworks, provisioning profiles, and making the best of Xcode. If you've been around the block but want solid insight into iPad programming, Part II has you covered: Rather than just providing canned example code, Kirby and Tom give you real code that incrementally builds and improves a real app. And if you've been working with iOS for a while, but would benefit from a walk-through of the plethora of new features that have come our way with iOS 5 and Xcode 4, dive into the chapters on Storyboards, iCloud, and Core Image. Best of all, the book is well-written and conversational, making it a joy to read. This book is stellar."

—Alexis Goldstein
Coauthor, *HTML5 & CSS3 for the Real World*

"*Learning iPad Programming* is one of the most comprehensive resources on the planet for those developing for Apple's iPad platform. In addition to coverage of the language, frameworks, and tools, it dives into features new in iOS 5, like Automatic Reference Counting, Storyboarding, and connecting your applications with iCloud. But where this book really shines is in the tutorials and the application you will build as you read through this book. Rather than being a toy that employs only off-the-shelf iOS user interface components from Interface Builder, the PhotoWheel app demonstrates custom view programming and view controller containment, nonstandard gesture/user input handling, and provides insight into how a complex iOS project comprised of multiple subsystems is assembled into a shipping application. In other words, *Learning iPad Programming* shows how to deal with the challenges you'll face in real iPad development."

—Erik Price
Senior Software Engineer, Brightcove

"A thoroughly crafted guide for learning and writing iOS applications, from the humble beginnings in Xcode and Interface Builder to creating a full-featured iPad application. There are many books that try to cover the gamut of knowledge required to take a reader from zero to app; Kirby and Tom have actually done it in this book. It is a fun and comprehensive guide to the world of developing apps for Apple's magical device."

—Rod Strougo
Founder, Prop Group

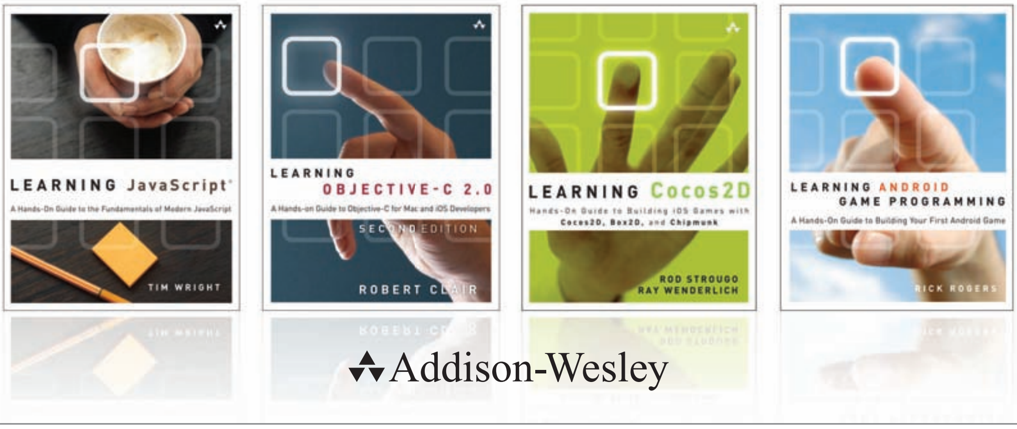
"The iPad is changing the way we think about and use technology. *Learning iPad Programming* is one of the most in-depth and well-executed guides to get both new and seasoned developers up to speed on Apple's exciting new platform."

—Justin Williams
Crew Chief, Second Gear

Learning iPad Programming

Second Edition

Addison-Wesley Learning Series


LEARNING JavaScript
A Hands-On Guide to the Fundamentals of Modern JavaScript
TIM WRIGHT
LEARNING OBJECTIVE-C 2.0
A Hands-on Guide to Objective-C for Mac and iOS Developers
SECOND EDITION
ROBERT CLAIR
LEARNING Cocos2D
Hands-On Guide to Building iOS Games with Cocos2D, Box2D, and Chipmunk
ROD STROUGO
RAY WENDERLICH
LEARNING ANDROID GAME PROGRAMMING
A Hands-On Guide to Building Your First Android Game
RICK ROGERS
Addison-Wesley

Visit informit.com/learningseries for a complete list of available publications.

The Addison-Wesley Learning Series is a collection of hands-on programming guides that help you quickly learn a new technology or language so you can apply what you've learned right away.

Each title comes with sample code for the application or applications built in the text. This code is fully annotated and can be reused in your own projects with no strings attached. Many chapters end with a series of exercises to encourage you to reexamine what you have just learned, and to tweak or adjust the code as a way of learning.

Titles in this series take a simple approach: they get you going right away and leave you with the ability to walk off and build your own application and apply the language or technology to whatever you are working on.

Addison-Wesley informIT the trusted technology learning source | Safari Books Online

ALWAYS LEARNING PEARSON

Learning iPad Programming

A Hands-On Guide to Building iPad Apps

Second Edition

Kirby Turner

Tom Harrington

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

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

For sales outside the United States, please contact:

International Sales
international@pearson.com

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Turner, Kirby, 1966–
Learning iPad programming : a hands-on guide to building iPad apps / Kirby Turner, Tom Harrington. — Second edition.
pages cm
Includes index.
ISBN 978-0-321-88571-5 (pbk. : alk. paper)
1. iPad (Computer)—Programming. 2. Application software—Development. 3. Mobile computing. 4. iOS (Electronic resource) I. Harrington, Tom. II. Title.
QA76.8.I863T87 2013
005.258—dc23
2013004508

Copyright © 2013 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

ISBN-13: 978-0-321-88571-5
ISBN-10: 0-321-88571-6
Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing, May 2013

Editor-in-Chief
Mark L. Taub

Senior Acquisitions Editor
Trina MacDonald

Senior Development Editor
Chris Zahn

Managing Editor
John Fuller

Full-Service Production Manager
Julie B. Nahil

Production Project Manager
Vicki Rowland

Copy Editor
Jill E. Hobbs

Indexer
Ted Laux

Proofreader
Janice R. Norris

Technical Reviewers
Patrick Burleson
Michael Haberman
Andrew Moore

Editorial Assistant
Olivia Basegio

Compositor
Vicki Rowland

❖

To Steve Jobs, who saw further than most.
— Kirby Turner and Tom Harrington

To Melanie and Rowan, for their continuous love and support.
And to my mom, my personal hero.
—Kirby Turner

To Carey, who gave me the courage to pursue my dreams.
—Tom Harrington

❖

This page intentionally left blank

Contents at a Glance

Foreword **xxv**

Preface **xxix**

Acknowledgments **xlv**

About the Authors **xlvii**

I Getting Started 1

1 Your First App **3**

2 Getting Started with Xcode **21**

3 Getting Started with Interface Builder **47**

4 Getting Started with Objective-C **71**

5 Getting Started with Cocoa **97**

6 Provisioning Your iPad **125**

7 App Design **151**

II Building PhotoWheel 175

8 Creating a Master-Detail App **177**

9 Using Table Views **197**

10 Using Collection and Custom Views **235**

11 Using Touch Gestures **269**

12 Adding Photos **285**

13 Data Persistence **301**

14 Storyboarding in Xcode **337**

15 View Controllers and Segues **359**

16 Building the Main Screen **387**

17 Creating a Photo Browser **447**

18 Supporting Device Rotation **489**

19 Printing with AirPrint **517**

20 Sharing with Others **525**

21 Web Services **541**

22 Syncing with iCloud **579**

23 Producing a Slideshow with AirPlay **605**

24 Visual Effects with Core Image **625**

25 Going Universal **653**

III The Finishing Touches 663

26 Debugging **665**

27 Distributing Your App **689**

28 The Final Word **707**

A Installing the Developer Tools **709**

Index **717**

Contents

Foreword xxv

Preface xxix

Acknowledgments xlv

About the Authors xlvii

I Getting Started 1

1 Your First App 3

Creating the Hello World Project 3

Getting Text on the Screen 11

Say Hello 13

Summary 19

2 Getting Started with Xcode 21

The IDE 21

Workspace Window 22

- Toolbar Area 22
- Navigation Area 24
- Editor Area 25
- Utility Area 26
- Debug Area 28
- The Design of the Workspace Window 28

Preferences 29

- Fonts and Colors 29
- Text Editing 30
- Line Wrapping 32
- Key Bindings Preferences 34
- Code Completion 36

Developer Documentation 37

Editors 38

- Standard Editor 38
- Assistant Editor 38
- Version Editor 40

Project Settings 40

Schemes 42
Organizer 43
Other Xcode Tools 44
Summary 45

3 Getting Started with Interface Builder 47
Interface Builder 47
How Does IB Work? 48
Getting Hands-On Practice with IB 49
Selecting and Copying Objects 53
Aligning Objects 53
Layout Rectangle 56
Inspectors 57
Connecting Your NIB to Your Code 61
Defining an Outlet in Code 62
Using the Assistant Editor 68
Storyboards 70
Summary 70

4 Getting Started with Objective-C 71
What Is Objective-C? 71
Hands-On Practice with Objective-C 72
Let's Write Some Code 75
Memory Management 93
Automatic Reference Counting 94
Summary 95

5 Getting Started with Cocoa 97
The Cocoa Stack 97
Foundation 99
Data Type 100
Collection Classes 106
Utility Classes and Functions 108
UIKit 112
UIApplication 113
UIWindow 113
UIScreen 113

UIView 113
UIViewController 113
UIWebView 113
UILabel 113
UITextField 114
UITextView 114
UIButton 114
UITableView and UITableViewCell 115
UIScrollView 115
UIPageControl 116
UIPickerView 116
UIDatePicker 117
UISwitch 117
UISlider 117
UIMenuController and UIMenuItem 117
UIImage 117
UIImageView 118
UINavigationBar 119
UINavigationController 119
UIToolbar 119
UITabBar 120
UIBarButtonItem 120
UISegmentedControl 120
Common Design Patterns in Cocoa 121
Model-View-Controller 121
Target-Action 122
Summary 123

6 Provisioning Your iPad 125
About the iOS Provisioning Portal 125
The Provisioning Process: A Brief Overview 127
What Is a Device ID? 127
What Is an App ID? 128
What Is a Development Provisioning Profile? 129
Setting Up Your Development Machine 130
Requesting a Development Certificate 131
Submit Your CSR for Approval 134

Download and Install Your Certificate **135**
Setting Up Your Device **138**
Use for Development **139**
Using the iOS Provisioning Portal **141**
Adding a Device ID **141**
Adding an App ID **143**
Creating a Development Provisioning Profile **145**
Downloading a Development Provisioning Profile **146**
Installing a Development Provisioning Profile **147**
Summary **149**

7 App Design 151
Defining Your App **151**
App Name **152**
App Summary **152**
Feature List **153**
Target Audience **154**
Revisit Your Feature List **155**
Competing Products **155**
A Sample App Charter **156**
UI Design Considerations **158**
Read the HIG **158**
Make Your App “Tapworthy” **158**
Design for the Device **159**
People Use iOS Devices Differently from the Web or Desktop **159**
Wear Your Industrial Designer Hat **160**
Metaphors **160**
Sound Effects **162**
Customize Existing Controls **162**
Hire a Designer **164**
Mockups **164**
What Is a Mockup? **164**
What to Mock up **166**
Tools to Use **166**
Prototyping **171**
What Is a Prototype? **171**

How to Create a Prototype 172
Summary 173

II Building PhotoWheel 175

8 Creating a Master-Detail App 177
Building a Prototype App 177
What Is the Split View Controller? 178
Create a New Project 180
Using the Simulator 181
A Closer Look 183
Project Structure 183
App Delegate 184
Launch Options 188
Other UIApplicationDelegate Methods 188
A Tour of UISplitViewController 189
Assigning the Split View Controller Delegate 192
Detail View Controller 194
Navigation Controller 194
Master View Controller 195
Summary 195
Exercises 196

9 Using Table Views 197
First Things First 197
A Closer Look 201
UITableView 201
UITableViewCell 202
UITableViewDelegate 202
UITableViewDataSource 202
UITableViewController 202
Working with a Table View 203
A Simple Model 203
Display Data 206
Add Data 210
Edit Data 224
Delete Data 228

Reorder Data 229
Select Data 230
Summary 233
Exercises 234

10 Using Collection and Custom Views 235
Collection Views 235
The Collection View Family of Objects 236
Flow Layout 236
Custom Cells 242
Custom Views 246
View Controller... Not 247
A Wheel View 248
A Carousel View 256
A Photo Wheel View Cell 263
Using PhotoWheelViewCell 265
Summary 268
Exercises 268

11 Using Touch Gestures 269
Touch Gestures Explained 269
Predefined Touch Gestures 270
Gesture Types 270
How to Use Gesture Recognizers 271
Custom Touch Gestures 275
Creating a Spin Gesture Recognizer 276
Using the Spin Gesture Recognizer 278
Summary 283
Exercises 283

12 Adding Photos 285
Two Approaches 285
Assets Library 285
Image Picker Controller 286
Using the Image Picker Controller 286
Using Action Sheets 289
Using UIImagePickerController 293
Saving to the Camera Roll 297

Summary **299**

Exercises **299**

13 Data Persistence 301

The Data Model **301**

Photos **301**

Photo Albums **302**

Thinking Ahead **302**

Building the Model with Core Data **302**

What Is Core Data? **302**

Managed Objects and Entity Descriptions **303**

Managed Object Contexts **305**

Persistent Stores and Persistent Store Coordinators **306**

Adding Core Data to PhotoWheelPrototype **306**

Adding the Core Data Framework **306**

Setting up the Core Data Stack **308**

Using Core Data in PhotoWheel **312**

The Core Data Model Editor **312**

Adding the Entities **313**

Creating NSManagedObject Subclasses **317**

Adding Custom Code to Model Objects **321**

Reading and Saving Photo Albums with Core Data **327**

Adding New Photos to an Album with Core Data **331**

Displaying Photos in an Album with Core Data **334**

Using SQLite Directly **335**

Summary **336**

Exercises **336**

14 Storyboarding in Xcode 337

What Is a Storyboard? **337**

Using a Storyboard **338**

Scenes **339**

Segues **340**

Storyboarding PhotoWheel **341**

Workspace **341**

Add the Main Storyboard **344**

Set UIMainStoryboardFile **346**

Update AppDelegate **347**

Add Images **348**

App Icon **349**

Initial View Controller **350**

Another Scene **352**

Creating a Segue **355**

Summary **357**

Exercises **357**

15 View Controllers and Segues 359

Implementing a View Controller **359**

Container View Controllers **364**

Create a Container View Controller **365**

Add the Child Scenes **366**

Segue **369**

Creating a Custom Segue **370**

Setting the Scene **370**

Implementing a Custom Segue **375**

Before You Compile **380**

Customizing the Pop Transitions **381**

Summary **385**

Exercises **385**

16 Building the Main Screen 387

Reusing Prototype Code **388**

Copy Files **388**

Core Data Model **390**

Changes to WheelView **394**

Displaying Photo Albums **406**

Implementing the Photo Albums View Controller **408**

Setting the Managed Object Context **414**

Adding Photo Albums **416**

Managing Photo Albums **417**

Selecting the Photo Album **417**

Naming the Photo Album **421**

Fixing the Toolbar Display **426**

Removing the Photo Album 428

A Better Photo Album Thumbnail 430

Adding Photos 434

Displaying Photos 439

Using the Collection View 442

Summary 446

Exercises 446

17 Creating a Photo Browser 447

Using the Scroll View 447

Setting Up the Photo Browser UI 455

Launching the Photo Browser 456

Improving the Push and Pop 460

Adding Chrome Effects 468

Zooming 474

Deleting a Photo 480

Summary 488

Exercise 488

18 Supporting Device Rotation 489

How to Support Rotation 489

Supported Orientations 491

Using Cocoa Auto Layout 492

Customized Rotation 495

Rotating the Photos Scene 502

Rotating the Albums Scene 508

Rotating the About View 510

Rotating the Photo Browser 510

Launch Images 513

Summary 515

Exercises 516

19 Printing with AirPrint 517

How Printing Works 517

Print Center 518

Requirements for Printing 519

Printing API 519

Adding Printing to PhotoWheel 519

The Printer Simulator 522
Summary 523
Exercises 524

20 Sharing with Others 525
Sending Email 525
How It Works 525
The MFMailComposeViewController Class 527
The SendEmailController Class 527
Activity View Controller 537
Using the Activity View Controller 537
Summary 540
Exercises 540

21 Web Services 541
The Basics 541
RESTful Web Services Using Cocoa 542
Flickr 543
Adding Flickr to PhotoWheel 545
Updating the Flickr View Controller Scene 547
Displaying the Flickr Scene 548
Wrapping the Flickr API 551
Downloading Photos Asynchronously 559
Implementing FlickrViewController 565
One More Thing 575
What's Missing 576
Summary 577
Exercises 577

22 Syncing with iCloud 579
Syncing Made Simple 579
iCloud Concepts 580
File Coordinators and Presenters 580
UIDocument and UIManagedDocument 581
Ubiquitous Persistent Stores 581
Device Provisioning, Revisited 582
Configuring the App ID 582
Provisioning for iCloud 584

    Configuring iCloud Entitlements 585
  iCloud Considerations for PhotoWheel 586
    Don’t Sync More Than You Need to Sync 586
    Using Transient Core Data Attributes 586
  Updating PhotoWheel for iCloud 588
  Syncing Photos with iCloud 592
    Making the Persistent Store Coordinator Ubiquitous 592
    Monitoring iCloud Data 597
    Receiving Changes from iCloud 598
  Going Further with iCloud 600
    Preexisting Data Stores 600
    Duplicate Detection 601
    Repeated Calls to awakeFromInsert 601
    Deleted Documents and Data 602
  Summary 603
  Exercises 603

23 Producing a Slideshow with AirPlay 605
  External Display Options 605
  App Requirements for External Displays 606
  External Display API 606
  Attaching an External Display 607
  Adding a Slideshow to PhotoWheel 609
    Updating the Storyboard 610
    Adding the Slideshow Display 610
  Managing External Displays 613
  Advancing to the Next Photo 617
  Adding Slideshow User Interface Controls 620
  Updating the Photo Browser 622
  Finishing Up 623
  Summary 624
  Exercises 624

24 Visual Effects with Core Image 625
  Core Image Concepts 625
  Introducing CIFilter 627

Filter Types **628**
Using CIFilter **628**
Image Analysis **630**
Automatic Enhancement **630**
Face Detection **631**
Adding Core Image Effects to PhotoWheel **633**
Instance Variables for Filter Management **633**
User Interface Additions **634**
Creating the CIFilter Effects **640**
Applying the Filters **645**
Implementing Auto-Enhance **647**
Implementing Face Zoom **647**
Other Necessary Methods **649**
Summary **651**
Exercises **651**

25 Going Universal **653**
Why Go Universal? **653**
Reason Not to Go Universal **654**
Making a Universal App **655**
Two Storyboards **655**
Separating Code by Device Type **656**
The Tilde **657**
Pitfalls **659**
Avoid Hard-Coding **659**
Be Defensive **660**
Summary **661**
Exercises **662**

III The Finishing Touches **663**

26 Debugging **665**
Understand the Problem **665**
What Went Wrong? **665**
Reproducing Bugs **665**
Debugging Concepts **666**

Breakpoints **666**
Debugging in Xcode **667**
Setting and Managing Breakpoints **667**
Customizing Breakpoints **668**
Hitting a Breakpoint **670**
Checking on Variables **672**
Debugging Example: External Display Code **675**
When You Really Need NSLog **679**
Profiling Code with Instruments **681**
Profiling Example: Slideshow UI Control Updates **683**
Summary **686**

27 Distributing Your App 689
Distribution Methods **689**
Building for Ad Hoc Distribution **690**
Provisioning for Ad Hoc Distribution **690**
Prepare the (Ad Hoc) Build **691**
Building for App Store Distribution **694**
Provisioning for the App Store **695**
Prepare the (App Store) Build **695**
Next Steps **698**
The App Store Process **698**
What if Apple Rejects the App? **699**
App Information for the App Store **700**
App Store Assets **702**
Using iTunes Connect **703**
User Roles **703**
Managing Applications **704**
Submitting the App **705**
Going Further **706**
Summary **706**

28 The Final Word 707
What's Next **708**

A Installing the Developer Tools 709
Membership Has Its Privileges **709**

Joining the iOS Developer Program **710**
Which Program Type Is Right for You? **711**
What You Need to Register **712**
Installing Xcode **714**
  First Launch Experience **715**

Index **717**

Foreword

I love books. I really love books. Anyone who's known me for any amount of time knows I'm a total bookworm. Well-written books are one of the cheapest and fastest self-education tools. I can remember a number of books that were hugely significant in my personal and professional development—books like *Object Oriented Software Construction* by Bertrand Meyer, Scott Knaster and Stephen Chernicoff's early Mac programming books, Dave Mark's C programming books, Robert C. Martin's horribly titled (but full of wonderful a-ha! moments) *Designing Object Oriented C++ Applications Using the Booch Method*, and, of course, the late W. Richard Stevens' UNIX and Network programming books. I remember lessons learned from these tomes, even those I read many years ago.

Unfortunately, not all books are created equal. I've seen some real stinkers in my time. When I was making the transition from Mac programming to iOS programming, I got some really great books, but also some books that were terrible. Really, really terrible. In some of these volumes, it was almost as if someone had filed the serial numbers off of *Instant Visual Basic Programming Guide for Complete Dummies in 24 Hours*, sprinkled around some square brackets, and pasted in pictures of iPhones. I thumbed through one early iPad programming book that literally had an error on every page. Some were just typos. Some were subtle errors, understandable if you haven't already lived in the Cocoa universe for a couple of years. Some of it was downright bad advice, obviously from someone who did not know what he was doing. There is a certain expectation of trust when you drop your hard-earned currency on a book, and violating that trust is unforgivable.

So, this book—*Learning iPad Programming*. Is it worth the price? Does it fall in my first category of books (awesome), or the second (unequivocally lame)? Good question. Glad you asked!

First, a good book needs to cover its topic, and cover it well. *Learning iPad Programming, Second Edition*, judging just by its heft, contains a lot of material. Well, that's true, assuming you've got the printed version in hand; *War and Peace* weighs the same as *The Little Prince* in ebook form, so it's hard to tell them apart. Just skim through the table of contents—you can see the text covers a lot of stuff. A metric freakload of stuff. And this stuff is all relevant. It covers such basics as installing the development tools. Model-View-Controller. Master-Detail. Storyboards. Segues. Table views. `UIViewController`. Navigation views. Handling device rotation. There are also more advanced topics such as consuming Web services, the media library, touch gestures, data persistence, and the raw unpleasantness that is Apple's device provisioning. And there's some cutting-edge stuff, such as AirPrint, AirPlay, iCloud, and Core Image. Kirby and Tom have suffered the arrows in their backs dealing with months of flaky underdocumented prerelease software so you don't have to.

Very good books are timely, but not exploitative. I saw my first iPad programming book about three months after the device was announced. There was no way this book could convey the iPad gestalt to the reader, simply because the device hadn't been available to any author for a long enough period of time. The book was pumped out as fast as possible to hit the market, and it showed. *Learning iPad Programming* is a mature project, with years of work having gone into it. Good books take time to achieve high levels of awesomeness.

Great books transcend their subject matter. This book is called *Learning iPad Programming*. It'd be easy to assume that it just covers introductory iPad programming in a simplistic manner. "Views are cool!" "Yay! Tapping a button!" But it's more. Not many books have a single project that lives and evolves through the entire narrative. The reason not many books do this is because it is difficult to do well. Important toolkit features get shoehorned into weird places because the author didn't do enough upfront design time. This book, though, takes you from design to prototype to the Real Deal.

And then it goes further. Not many books talk about the inner game of design. This one does. Even fewer books talk about the inner game of debugging. Debugging is a fundamental part (if not *the* fundamental part) of the day-to-day life of a programmer, and few books devote more than a paragraph or two to it. *Learning iPad Programming* has an entire chapter on the topic, and it deals with much more than how to single-step with the debugger. As I was reading a preproduction version of this book, I emitted an audible "SQUEE" when I hit Chapter 26. I love debugging, and I love seeing such an important topic covered in detail in what is ostensibly a beginner's book. And as you can tell, I love learning stuff. I learned some stuff from Chapter 26, even after 23 years of programming professionally.

Finally, those who **create** the great books transcend the ordinary. The Mac and iPhone community is pretty small and well connected. You tend to learn quickly who the trusted players are. Many of the lame books I alluded to earlier were written by individuals I had never heard of before, and never heard from again. No blogs, no appearances at conferences, no footprint on the community. Get in, crank out something, exploit the community, and get out.

Kirby and Tom are different. They're known entities. They have blogs. Tom has his name on other books. They've shipped products. They've shipped the first edition of this book. They have happy customers. They answer questions online. They organize and speak at conferences. They're involved with CocoaHeads. They have invested a great deal of their time into the betterment of the community. It is why I am honored and humbled that they asked me to write this foreword.

As you can probably tell, I'm pretty excited about this book. There are many excellent introductory iOS programming books. I recommend reading all of them (at least the good ones) because iOS is such a huge topic that even Kirby and Tom can't cover everything you need to know in one volume. But if you're specifically targeting the iPad, this one is the one to get.

—Mark Dalrymple
Cofounder of CocoaHeads, the international Mac and iPhone programmer community
Author of *Advanced Mac OS X Programming: The Big Nerd Ranch Guide*
February 14, 2013

This page intentionally left blank

Preface

In October 2011, Apple CEO Tim Cook shared some interesting facts about the iPad:

- Ninety-two percent of Fortune 500 companies are testing or deploying iPads.
- More than 80 percent of U.S.-based hospitals are testing or piloting the iPad.
- Every state in the United States has some type of iPad deployment program in place or in pilot.

Just a year later, in October 2012, Tim Cook announced that 100 million iPads had been sold. Think about that for a moment: 100 million iPads sold in just two and a half years! That is an amazing feat.

And the news about the iPad doesn't stop there. The FAA has approved the use of the iPad instead of paper charts for on-duty airline pilots. Without a doubt, the iPad is changing the way people think about (and use) computers today. And it continues to get better with the release of iOS 6, the latest operating system for iPad and iPhone devices.

Make no mistake, the iPad packs a punch. With its patented multi-touch interface, an onboard graphics chip, the powerful A6X processor, and 4G and Wi-Fi networking, the iPad is the benchmark in a post-PC world. More important, though, is how the iPad fits into the Mac/iOS ecosystem. Mac OS X and iOS users can use FaceTime for video chat from desktop to device. What's more, iOS iMessage enables users to text from their iPad to reach other iPad, iPhone, and Mac users. The iPad represents a unique marriage of hardware and technology, and it is the Gold Standard for tablets.

This book was written with iOS 6 in mind and is aimed at new developers who want to build apps for the iPad. The book will also appeal to iPhone developers who want to learn more about how to make their apps sing on the iPad. While some people look at the iPad as just a bigger iPhone, it really isn't. There is a lot more that you as a developer can do with the iPad from a user interface perspective that you just can't do on the iPhone.

While this book includes brief discussions of iPhone programming where appropriate, its primary focus is the iPad. *Learning iPad Programming* highlights those areas of the iOS SDK that are unique to the iPad, and it isn't a rehash of similar books targeting the iPhone. Additionally, the book covers new features in iOS, such as embedded segues, container view controllers, iCloud, and Core Image, as well as some of the great new features in Xcode 4, such as storyboarding. Apple has gone to great lengths to make it easier for you to develop for iOS and OS X, and the plan for this book is to make it even easier for you to get there.

What Will I Learn?

This book will teach you how to build apps specifically for the iPad, taking you step by step through the process of making a real app that is freely available in the App Store right now! The app you'll build in this book is called PhotoWheel.

> **Download the App!**
>
> You can download the PhotoWheel app from the App Store.[1] The app is freely available, so go ahead—download PhotoWheel, and start playing around with it.

PhotoWheel is a spin on the Photos app that comes on every iPad (pun intended). With PhotoWheel, you can organize your favorite photos into albums, share photos with family and friends via email, and view photos on your TV wirelessly using AirPlay.

Even more important than the app itself is what you will learn as you build it.

You will learn how to take advantage of the latest features in iOS and Xcode, including storyboarding, Automatic Reference Counting, iCloud, and Core Image. You will learn how to leverage other iOS features such as the Activity View Controller, AirPrint, AirPlay, and Grand Central Dispatch (GCD). And you will learn how to extend the boundaries of your app by communicating with Web services hosted on the Internet.

Think of this book as an epic-length tutorial, showing you how you can create a real iPad app from start to finish. You'll be coding along with the book, and we'll explain things step by step. By the time you have finished reading and working through this book, you'll have a fully functional version of PhotoWheel that you can proudly show off to friends and family (you can even share it with them, too). Best of all, you'll have confidence and the knowledge of what it takes to design, program, and distribute iPad apps of your own.

What Makes the iPad So Different?

While the iPad runs the same version of iOS that runs on the iPhone, iPod touch, and Apple TV, the iPad is significantly different from those other iOS-based devices. Each device is used differently, and iOS brings certain things to the table for each of them. For example, the version of iOS that runs in your Apple TV doesn't yet offer the same touch interface; in fact, the interface is totally different. Apple TV's user interface (UI) runs as a layer on top of iOS, providing a completely different user experience.

But the iPad is so different. It is not something you can hold in the palm of your hand—unless you have an iPad mini. The iPad is something you use with both hands. You swipe. You touch. You interact with it more than with most iPhone apps. It's easy to dismiss the iPad as "just a large iPhone," but it really isn't.

1. PhotoWheel: https://itunes.apple.com/app/photowheel/id424927196&mt=8

While the physical size is the obvious difference between the iPad and iPhone, the real difference—the difference that sets the iPad apart from the iPhone—is conceptual. The conceptual differences stem from how an iPad application is designed and how the user interacts with the application. And the conceptual differences start with the bigger display.

Bigger Display

The iPad's bigger screen provides more than double the screen real estate found on the iPhone. In turn, your application can display more information, giving you more space to work with for your user interface. A good example of this is WeatherBug.

WeatherBug HD has been designed to take full advantage of the iPad's larger screen. As you can see in Figure P.1, the iPad version of WeatherBug displays much

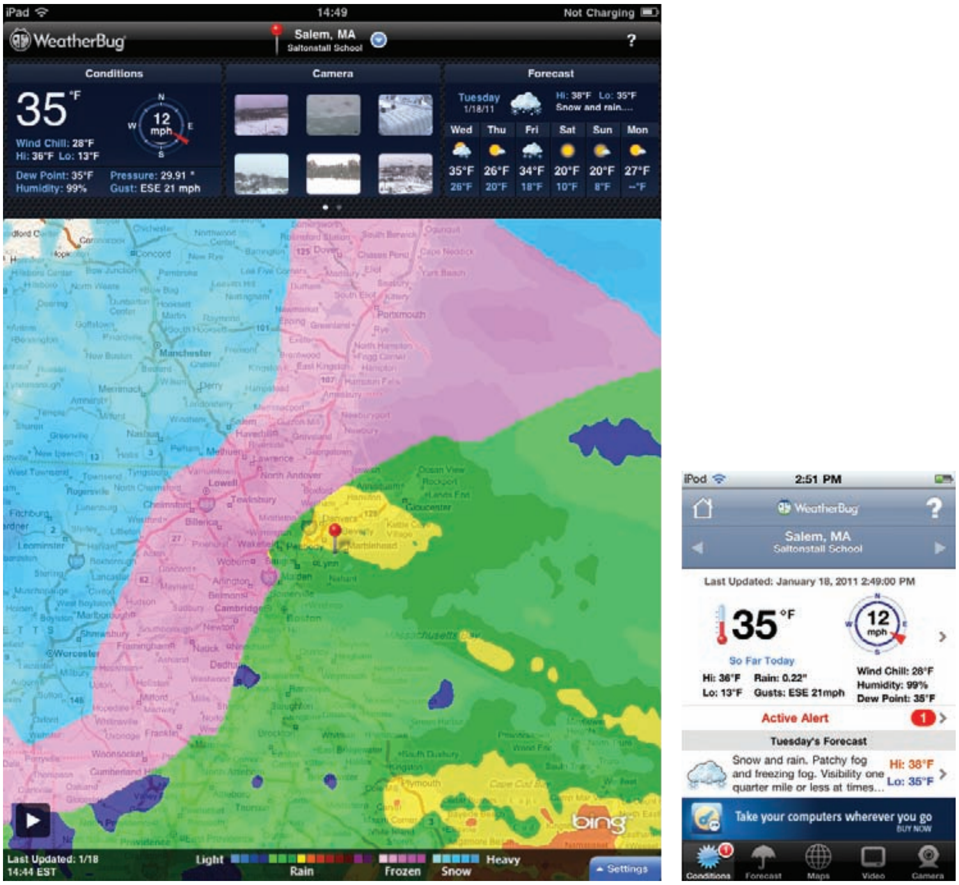


Figure P.1 On the left is the WeatherBug app displayed on the iPad. The screen shot on the right is the same WeatherBug app running on the iPhone. (Used with permission of Earth Networks.)

more weather-related information on a single screen than you can get on the iPhone version. Instead of your having to touch and swipe (and sometimes pray) to find additional weather information, WeatherBug HD on the iPad gives you everything you need to know right on the main screen—no additional touching or swiping needed. Of course, additional detail is still available at a touch.

Less Hierarchical

Because of the smaller screen, many iPhone applications tend to sport a hierarchical navigation system. You see this throughout many iPhone apps. The user taps an item and a new screen slides into view. Tap another item and another view slides in. To navigate back, you tap a back button, usually found in the upper-left corner of the screen.

The Dropbox app illustrates the hierarchical navigation system quite well. Dropbox, for those who may not know, is an online service that allows you to store your data files, documents, and images in the cloud. Stored files are then synced across all of your computers and devices that run the Dropbox client software. Suppose you are working on a text document from your laptop. You save the text document to your Dropbox folder. Later you need to review the text document, so you open the same text document on your iPhone. Dropbox makes this possible.

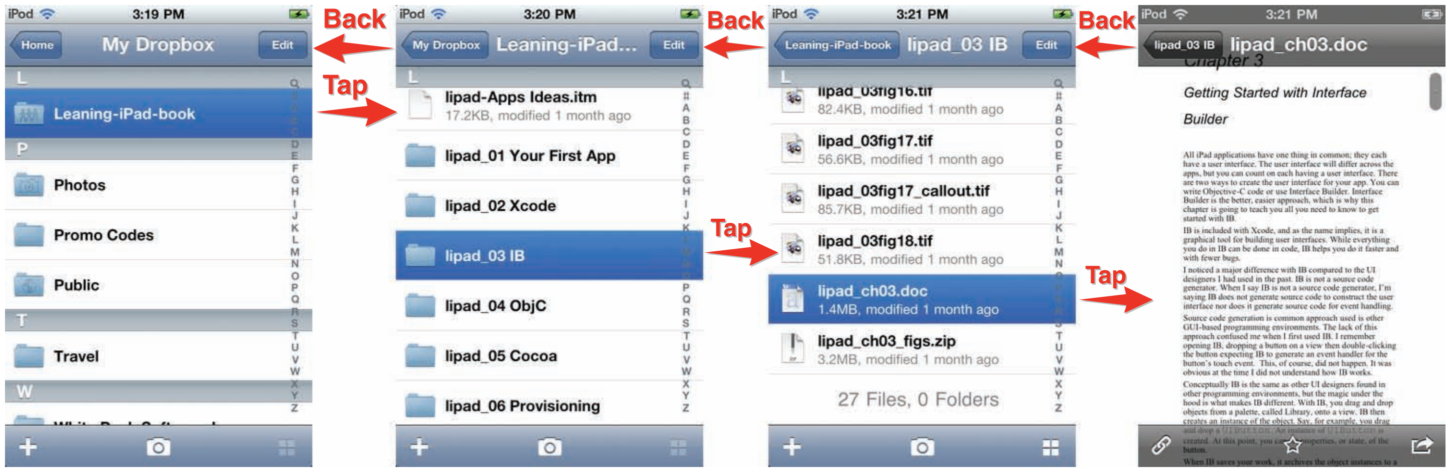
When you use the Dropbox app on your iPhone, you see a list of files and folders sorted alphabetically. Tapping a file or folder will open it, causing the new screen to slide into view. If you open a file, you see the contents of the file. If you open a folder, you see a new list of files and folders. Continue tapping folders to navigate farther down the hierarchy.

To move back up the hierarchy, tap the back button in the upper-left corner of the screen. The text label for this button can vary. Usually it displays the name of the previous item on the stack, but sometimes it displays the word *Back*. While the text label may vary, the style of the back button does not. The back button has a pointy left side. This almost arrow-like style conveys a sense of moving backward through the screens.

The forward and backward navigation through the hierarchy is illustrated in Figure P.2.

Dropbox is also available for the iPad. So how did the developers redesign an app that obviously requires hierarchical navigation to make it feel flatter, less hierarchical? They took advantage of an iOS object available only to the iPad called `UISplitViewController`, shown in Figure P.3.

The split view controller is a nonvisual object that controls the display of two side-by-side views. When you hold your iPad in landscape mode, the two views are displayed side by side. Rotate your iPad to portrait orientation, and the left-side view disappears. This allows the user to focus his attention on the main content displayed on the right side.



Chapter 3

Getting Started with Interface Builder

All iPad applications have one thing in common; they each have a user interface. The user interface will differ across the apps, but you can count on each having a user interface. There are two ways to create the user interface for your app. You can write Objective-C code or use Interface Builder. Interface Builder is the better, easier approach, which is why this chapter is going to teach you all you need to know to get started with IB.

IB is included with Xcode, and as the name implies, it is a graphical tool for building user interfaces. While everything you do in IB can be done in code, IB helps you do it faster and with fewer bugs.

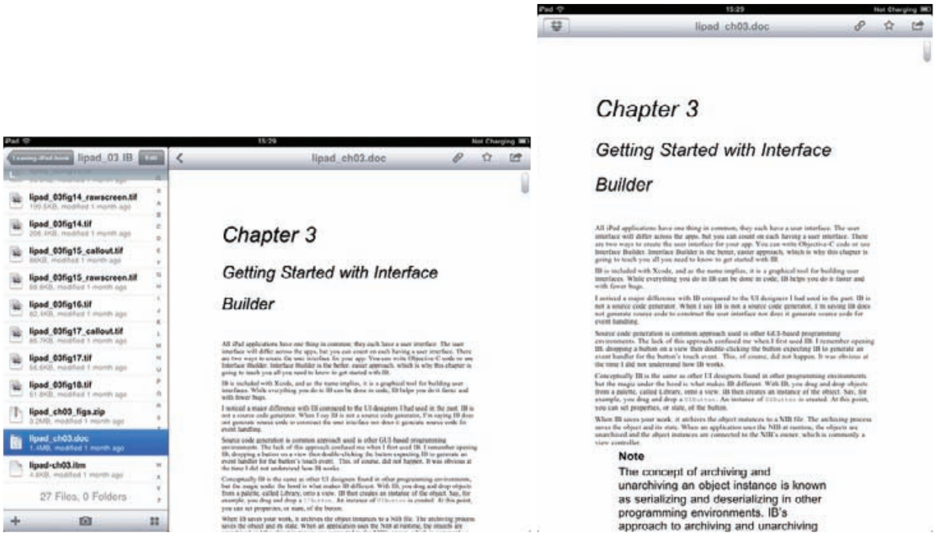
I noticed a major difference with IB compared to the UI designers I had used in the past. IB is not a source code generator. When I say IB is not a source code generator, I'm saying IB does not generate source code to construct the user interface nor does it generate source code for event handling.

Source code generation is common approach used is other GUI-based programming environments. The lack of this approach confused me when I first used IB. I remember opening IB, dropping a button on a view then double-clicking the button expecting IB to generate an event handler for the button's touch event. This, of course, did not happen. It was obvious at the time I did not understand how IB works.

Conceptually IB is the same as other UI designers found in other programming environments, but the magic under the hood is what makes IB different. With IB, you drag and drop objects from a palette, called Library, onto a view. IB then creates an instance of the object. Say, for example, you drag and drop a UIButton. An instance of UIButton is created. At this point, you can [illegible] properties, or state, of the button.

When IB saves your work, it archives the object instances to a

Figure P.2 Example of navigating the hierarchy of folders and files using the Dropbox app on the iPhone. You tap to move forward, or drill down, to more content, and you tap the back button to move backward.



Chapter 3

Getting Started with Interface Builder

All iPad applications have one thing in common, they each have a user interface. The user interface will differ across the apps, but you can count on each having a user interface. There are two ways to create the user interface for your app. You can write Objective-C code or use Interface Builder. Interface Builder is the better, easier approach, which is why this chapter is going to teach you all you need to know to get started with IB.

IB is included with Xcode, and as the name implies, it is a graphical tool for building user interfaces. While everything you do in IB can be done in code, IB helps you do it faster and with fewer bugs.

I noticed a major difference with IB compared to the UI designers I had used in the past. IB is not a source code generator. When I say IB is not a source code generator, I'm saying IB does not generate source code to construct the user interface nor does it generate source code for event handling.

Source code generation is common approach used in other GUI-based programming environments. The lack of this approach confused me when I first used IB. I remember opening IB, dropping a button on a view then double-clicking the button expecting IB to generate an event handler for the button's touch event. This, of course, did not happen. It was obvious at the time I did not understand how IB works.

Conceptually IB is the same as other UI designers found in other programming environments but the magic under the hood is what makes IB different. With IB, you drag and drop objects from a palette, called Library, onto a view. IB then creates an instance of the object. Say, for example, you drag and drop a UIButton. An instance of UIButton is created. At this point, you can set properties, or state, of the button.

When IB saves your work, it archives the object instances to a NIB file. The archiving process saves the object and its state. When an application uses the NIB at runtime, the objects are unarchived and the object instances are connected to the NIB's owner, which is commonly a view controller.

Note
The concept of archiving and unarchiving an object instance is known as serializing and deserializing in other programming environments. IB's approach to archiving and unarchiving

Figure P.3 Screen shots of Dropbox running on the iPad. Notice how the navigation is displayed in the left-side view when the device is held in a landscape orientation, but is hidden when the iPad is rotated to portrait.

Note

You get hands-on experience writing a split-view-based application in Chapter 8, "Creating a Master-Detail App."

This view pattern, in which the master view is displayed on the left side and the detail view is displayed on the right side, is often called "master-detail." The master view is used to navigate the hierarchy of data, or in the case of Dropbox, the master view is used to navigate the list of files and folders. When you find the file you want to view, tap it in the master view and the file contents are displayed on the right in the detail view. Rotate your iPad to a portrait position to focus your attention on the file's content, hiding the master view.

Orientation Matters

Most iPhone applications support only a single orientation. Many iPhone games are played in landscape mode, while many other iPhone apps are displayed in portrait mode. Like the iPad, the iPhone does support rotation and changes in orientation, but the small size of the device makes supporting different orientations unnecessary. Most users hold their iPhones in portrait mode with the Home button at the bottom when using applications, rotating to landscape orientation only to play a game.

The iPad is different. With the iPad, users grab the device and turn it on without regard to a certain orientation. This is even truer when the iPad is not in a case. Try this little experiment . . .

Place your iPhone, or iPod touch, on your desk or table with the Home button pointing at 10 o'clock. Walk away or turn around. Come back to the device and pick it up. Take a look at the device as you hold it in your hand. There's a good chance that as you picked up the device, you rotated it so that the Home button is at the bottom. You did this rotation even before turning on the device. It is an almost natural instinct to hold your iPhone with the Home button at the bottom.

Now try the same experiment, but this time use your iPad. Place it on your desk or table. Make sure the Home button is positioned away from you—say, at 10 o'clock—and then walk away. Come back and pick up your iPad. Chances are good you did not rotate the device. Instead, you are likely holding your iPad in the same orientation it was in before you picked it up.

Multi-Touch Amped Up

Did you know that the iPad and the iPhone support the same multi-touch interface? They do. As a matter of fact, the iOS multi-touch interface supports up to 11 simultaneous touches. This means that you can use all your fingers—and maybe one or two more if you have a friend nearby—to interact with an application.

The iPad, with its larger screen, makes multi-touch use more feasible. While two-handed gestures have limited use on the iPhone, they can become a natural part of interacting with an iPad application. Take, for example, Apple's own Keynote app for the iPad. It takes advantage of the multi-touch interface to provide features once reserved for the point-and-click world of the desktop. Selecting multiple slides and moving them is just one example of how Keynote on the iPad maximizes the user experience with multi-touch.

So you already know that the multi-touch interface supports up to 11 simultaneous touches, but how can you confirm this? Write an iPad app that counts the number of simultaneous touches. That is exactly what Matt Legend Gemmell did. He wrote a really neat iPad app, shown in Figure P.4, that shows the number of simultaneous touches. But Matt went beyond just showing the touch count. He made the app sci-fi-looking, which also makes it fun to play with.

You can read more about Matt's iPad multi-touch sample and download the source code from his blog posting.[2]

Another way to explore the iPad multi-touch interface is to play with Uzu for iPad, only $1.99 in the App Store.[3] Uzu is a "kinetic multi-touch particle visualizer" and it's highly addictive. (Figure P.5 doesn't do the app justice; you should really download and play around with Uzu if you want to see some clever use of multi-touch.)

2. Multi-touch sample: http://mattgemmell.com/2010/05/09/ipad-multi-touch/
3. Uzu: https://itunes.apple.com/app/uzu/id376551723?mt=8

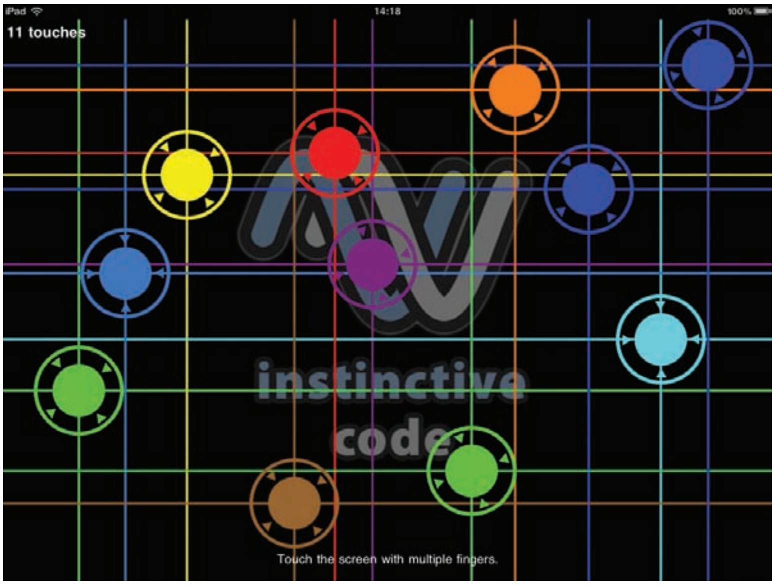


Figure P.4 Matt Legend Gemmell's multi-touch sample app for the iPad illustrating 11 touches

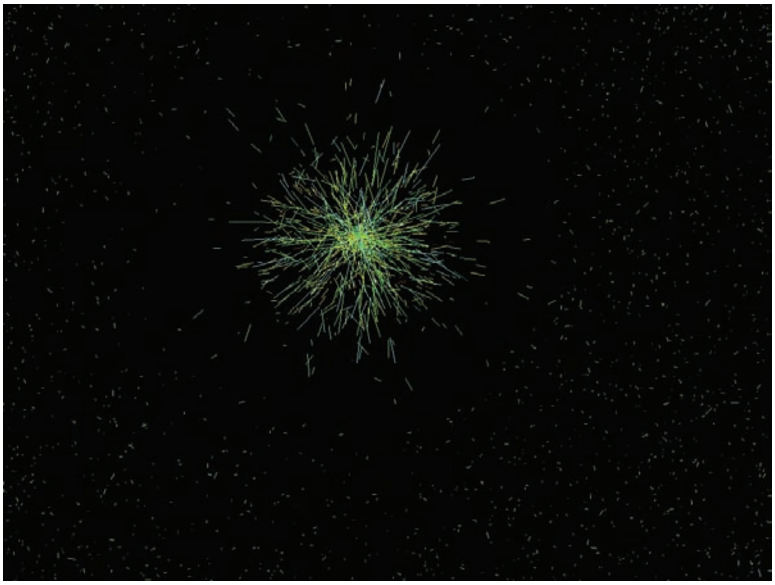


Figure P.5 Uzu, the particle visualizer for the iPad

The iPad Bridges the Gap between the Phone and the Computer

So, everyone agrees that the iPad is not an oversize iPhone. Great, glad to have you on the same page here. Now on to the larger question: Is the iPad a replacement for a laptop or desktop? No, not yet, but it's pretty darn close.

For many users, the iPad represents a mobile device bridging the gap between the smartphone and a full-fledged computer, whether a laptop or desktop computer. While many individuals use the iPad for content consumption, the iPad is also used to perform a good number of tasks previously left to the desktop or laptop computer. This causes iOS developers to rethink how to implement software concepts that have been around for eons. Word-processing software is one such concept that is seeing new life on the iPad.

The iPad opens the door to a wide range of applications not feasible on the small form factor of the iPhone. Word processing, again, is one such application that comes to mind.

While the iPhone is great for capturing quick notes, it is not ideal for writing lengthy documents. And while it is technically possible to implement a full-featured word processor on the iPhone, why would you? The screen is too small, and even in landscape mode, typing two-thumbed on a tiny screen would be less than productive. The iPhone is ideal for performing simple, quick tasks—writing a note, scheduling an event, marking a to-do item as complete—but it is less than ideal for lengthier tasks such as writing a book.

Enter the iPad

The iPad provides an experience similar to a small laptop. And when combined with a wireless keyboard, your iPad becomes a nice setup for writing long documents. I'm speaking from experience. A lot of the text in this book was originally written on an iPad. I can't imagine what writing a book on an iPhone would be like, but I know what it is like on the iPad, and it is a joy. Best of all, the iPad allows you to concentrate on a single task. This eliminates distractions and gives you better focus on the task at hand.

Organization of This Book

This book provides you with a hands-on guide for, as the book's title states, learning iPad programming. It walks you through every stage of the process—from downloading and installing the iOS SDK to submitting the first application to Apple for review.

There are 28 chapters and one appendix in the book, as follows:

- Part I, "Getting Started"

  Part I introduces you to the tools of the trade. Here you learn about developer tools such as Xcode and Interface Builder. You learn how to write code using

Objective-C and the Cocoa framework. And you learn what it takes to provision your iPad as a development device.

- Chapter 1, "Your First App"

  This chapter immediately immerses you in creating your first application. It provides a step-by-step guide to creating a simple, but functional, iPad application that runs in the iPad Simulator. You'll use Xcode to create the application, which means there is also some light coding to be done, but knowledge of Objective-C is not required at this point in the book. The goal of this chapter is for you to immediately get your hands on the tools and the code you'll use to create iPad apps.
- Chapter 2, "Getting Started with Xcode"

  Xcode is the developer's integrated development environment (IDE) used to write Objective-C code for iPad applications. This chapter highlights key features of Xcode, including recommended preference settings, commonly used shortcut keys, and descriptions of the various windows you will see when using Xcode.
- Chapter 3, "Getting Started with Interface Builder"

  In this chapter, you explore Interface Builder (IB). Interface Builder is the tool used to create an application UI with no programming required. This chapter explains how to use IB and many of its useful features. In addition, the chapter warns you about common mistakes made when using IB, such as forgetting to associate an event with an `IBAction`.
- Chapter 4, "Getting Started with Objective-C"

  This chapter introduces Objective-C by providing a brief overview of the programming language of choice for iPad programming. It is not intended to be a comprehensive review of the programming language, but instead to provide enough information to get you started in writing your first real iPad app.
- Chapter 5, "Getting Started with Cocoa"

  A programming language is only as powerful as the frameworks that support it, and Cocoa provides an impressive stack of frameworks and a library that make it possible for you to build your iPad app in less time.
- Chapter 6, "Provisioning Your iPad"

  Walking down the yellow brick road to the wonderful world of iPad development can have its own set of scary moments. One of the scariest is dealing with provisioning profiles, certificates, and registering a device for testing. Xcode 4 provides improvements in this area, but it is still far from perfect. This chapter guides you through the ominous forest of provisioning profiles, certificates, and device registration.

- Chapter 7, "App Design"

  You can't build an app if you don't know what you're building. This chapter shares tips on designing an application before the first line of code is ever written.

- Part II, "Building PhotoWheel"

  Part II is the heart of the book—where you get hands-on practice with building a real iPad app. The app you build is no simple "Hello, World" app. Rather, it is PhotoWheel, a full-featured photo app. In Part II, you learn about everything from creating custom animations for view transitions to iCloud syncing to viewing your photos on TV.

  - Chapter 8, "Creating a Master-Detail App"

    You start building PhotoWheel by first building a prototype of it. While building the prototype, you have a chance to learn about the splitview controller used in master-detail apps.

  - Chapter 9, "Using Table Views"

    In this chapter, you learn the basics of displaying data using table views. You also learn how to reorder, delete, and even edit data displayed in a table view.

  - Chapter 10, "Using Collection and Custom Views"

    In this chapter, you dive into the world of views. Here you learn how to use the collection view introduced in iOS 6, and create a custom wheel view for displaying photos.

  - Chapter 11, "Using Touch Gestures"

    This chapter teaches you how to take advantage of the iPad's multi-touch screen. You learn to use touch gestures so that users can interact with your app.

  - Chapter 12, "Adding Photos"

    PhotoWheel deals with photos, so it is only natural that you need to learn how to add photos to the app. In this chapter, you discover how to retrieve photos from the Photos app library and how to take new photos using the device's built-in camera.

  - Chapter 13, "Data Persistence"

    PhotoWheel won't be very useful if people can't save their work. In this chapter, you explore Core Data, and you learn how to use it to persist data in your application.

  - Chapter 14, "Storyboarding in Xcode"

    A storyboard is an exciting new way to design an app's user interface. In this chapter, you get hands-on practice with storyboarding, and you learn how you can do more with less code by using Interface Builder.

- Chapter 15, "View Controllers and Segues"

  A storyboard can take you only so far. At some point in time, you must write code to make your app really shine. In this chapter, you learn how to take advantage of view controllers to do more, and you learn how to create segues that transition between view controllers.
- Chapter 16, "Building the Main Screen"

  In this chapter, you dive into PhotoWheel. Prototyping is over and you have the basic UI in place with a storyboard. Now it's time to build the main screen, and that's exactly what you do in this chapter. You also learn how to use container view controllers, and you build a custom grid view that can be used in other projects.
- Chapter 17, "Creating a Photo Browser"

  In this chapter, you learn how to use a scroll view to create a full-screen photo browser. You also learn how to use a pinch gesture to zoom in and out on a photo.
- Chapter 18, "Supporting Device Rotation"

  Users expect iPad apps to display properly regardless of how the device is being held. A user may hold his iPad with the Home button on the left or right, or maybe on the top or bottom. As a developer, it is your job to ensure that your app displays properly regardless of the device's orientation. That is what you learn in this chapter: how to support device rotation. You also learn how to leverage Cocoa Auto Layout for supporting rotation of your user interface.
- Chapter 19, "Printing with AirPrint"

  This chapter gets straight to the point and teaches you how to print from your app using AirPrint.
- Chapter 20, "Sharing with Others"

  Virtually everyone has an email account these days, and everyone loves looking at photos. So it only makes sense that PhotoWheel users will want to share photos with family and friends using email. In this chapter, you learn how to send email from your app. But the chapter doesn't stop there—you also learn how to use the Activity View Controller for sharing photos through social networks such as Facebook and Twitter.
- Chapter 21, "Web Services"

  Adding photos already found on your iPad to PhotoWheel is a nice exercise, but many people keep their photos stored elsewhere. In this chapter, you learn how to make an iPad app communicate with a Web server so that you can search for and download photos from Flickr.

- Chapter 22, "Syncing with iCloud"

  Many people have multiple iOS devices, and it would be great if they could use PhotoWheel with the same data on all of them. Syncing can be a challenge, but with iCloud it becomes a lot easier. In this chapter, we add online syncing of photos and albums.
- Chapter 23, "Producing a Slideshow with AirPlay"

  The iPad has a great screen, but you might want to show photos to a group, and it's awkward to gather everyone around a hand-held device. In this chapter, you see how to make use of external wireless displays—a large TV set, maybe—from an iPad app. You'll use AirPlay for this purpose, so you don't need to run cables across the room.
- Chapter 24, "Visual Effects with Core Image"

  Core Image is an amazingly cool framework for analyzing and changing images. As if color effects and automatic photo enhancement weren't enough, you can also use Core Data Image to locate the faces of any people in the picture. You add all of these capabilities to PhotoWheel in a convenient user interface that allows people to preview effects before committing to them.
- Chapter 25, "Going Universal"

  Part II wraps up with a discussion of turning your iPad app into a universal app. A universal app takes full advantage of the device it's running on, and it extends the target audience for your iPad app to include iPhone users.

- Part III, "The Finishing Touches"

  In the final part of the book, you learn tips on debugging your app. Even more important, you learn how to distribute your app to others.

  - Chapter 26, "Debugging"

    At this point you know how to create an iPad application, but what happens when a problem occurs? This chapter is devoted to application debugging. It explores the LLDB. and shows you how to turn breakpoints on and off, and how to use sounds to debug your app. The chapter also introduces you to more advanced debugging techniques such as using Instruments to track down memory leaks.
  - Chapter 27, "Distributing Your App"

    At this point, the application has been written, debugged, and tested. The next step is to get the application into the hands of users. This chapter explores the options for distributing iPad applications, focusing on the two most commonly used distribution methods: Ad Hoc and App Store.
  - Chapter 28, "The Final Word"

    The book ends with some final words of encouragement for the new iPad programmer.

- Appendix A, "Installing the Developer Tools"

  This appendix walks you through the steps needed to start programming for the iPad. These measures include setting up an iOS developer account, downloading the iOS SDK, and installing the developer tools on your Mac.

Learning iPad Programming takes you from app design to the App Store. Along the way, you learn about the developer tools, the programming language, and the frameworks. But more important, you learn how to build a full-featured iPad app that you can show off to family and friends.

Audience for This Book

This book is intended for programmers who are new to the iOS platform and want to learn how to write applications that target the iPad. The book assumes that you are new to iPad programming and have little to no experience with Xcode and the Objective-C programming language. At the same time, it assumes that you have some prior programming experience with other tools and programming languages. *Learning iPad Programming* is not intended for individuals with absolutely no prior programming experience.

This book is targeted to programmers who want to learn how to develop sophisticated applications for the iPad using iOS 6. You are expected to have a Mac on which you can create programs using Xcode and Interface Builder, as well as an iOS developer account and an iPad. Some programming experience is helpful, particularly knowledge of the C programming language, although there is a chapter on object-oriented programming with Objective-C to give you a head start in this area.

Learning iPad Programming will also appeal to experienced iOS developers—people who have programmed and submitted apps to the App Store for the iPhone and iPod touch. If you are an experienced reader, you can skip over the basics, if you so choose, and quickly get to work on the example projects used throughout the book.

Getting the Source Code for PhotoWheel

The source code from each chapter as well as the source code for PhotoWheel as presented in this book is available from the book's Web site.[4] Work on PhotoWheel doesn't stop at the end of this book, either. There is so much more to do with the app and so much more to learn. The most up-to-date source code is available on github.[5]

4. PhotoWheel source code: http://www.learningipadprogramming.com/source-code/
5. PhotoWheel on github: https://github.com/kirbyt/PhotoWheel

You will also find more how-to articles and tips for improving PhotoWheel at the book's blog site.[6]

Should you have additional questions, or want to report a bug or contribute a new feature to PhotoWheel, feel free to send email to **kirby@whitepeaksoftware.com** or **tph@atomicbird.com**, or send a message to **@kirbyt** or **@atomicbird** on Twitter and App.net.

There is plenty of code to review throughout the book, along with exercises for you to try, so it is assumed that you have access to the Apple developer tools such as Xcode and the iOS SDK. Both of these toolkits can be downloaded from the Mac App Store as part of the Xcode download.[7]

Artwork Provided by

Matt McCray is the swell guy who provided the artwork in PhotoWheel. Reach out to Matt if you're looking for a designer for your next app. He can be reached at matt@elucidata.net and his Web site is at **www.elucidata.net**.

6. Book's blog site: http://www.learningipadprogramming.com/blog/
7. Xcode download: https://itunes.apple.com/us/app/xcode/id497799835?mt=12

This page intentionally left blank

Acknowledgments

As for any book that gets written, there's an entire cast and crew who remain hidden from the limelight; please take a moment to hear us out as we thank the supporting cast. . . .

Acknowledgments from Kirby Turner

I want to first thank my wife, Melanie, and my son, Rowan, for their support and patience while I focused on completing this book, and their understanding when I said I want to write a second edition. I want to thank Tom for agreeing to co-author this book. I want to give a huge THANKS to Chuck Toporek for convincing me to write, and Trina MacDonald for being an outstanding and patient editor. I also want to thank the production team for their hard work making this book look good. And, of course, I want to say thanks to the technical reviewers, Andrew, Michael, and Patrick. Your feedback is invaluable.

Lastly, I want to thank the amazing team of engineers at Apple for bringing the fun back to programming for me. And I want to thank the Mac and iOS developer community. None of this would be possible if not for the passion and spirit of this unique community.

Acknowledgments from Tom Harrington

I'd like to thank Kirby for inviting me to be part of this book. I'd also like to thank our technical reviewers and the rest of the production team for all their hard work making me look good in print. Apple continues to advance its software and tools at a breakneck pace, which makes it a challenge to write a book and get it into print while it's still current. Everyone involved has done a great job dealing with the challenges of writing a book on a topic that's constantly in flux.

On a closely related note, thanks to everyone at Apple for their hard work on iOS and the iPad. Without them we wouldn't have such a cool topic to write about.

This page intentionally left blank

About the Authors

This book is brought to you by. . . .

Kirby Turner is an independent software developer and Chief Code Monkey at White Peak Software Inc., where he focuses on iOS and Mac programming. When Kirby is not sitting behind the keyboard, he can be found hanging out with his wife and son, hiking the mountains of New England, kayaking the waters in and around Salem, Massachusetts, and snowboarding down mountains in search of magic powder. Follow Kirby on Twitter and App.net: @kirbyt.

Tom Harrington is an independent iOS and Mac software developer and is available for contract work, technical conferences, and parties. He also organizes iOS developer events in Colorado. Follow Tom on Twitter and App.net: @atomicbird.

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 use 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 we cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail we receive, we 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.

Email:	trina.macdonald@pearson.com
Mail:	Reader Feedback Addison-Wesley Learning Series 800 East 96th Street Indianapolis, IN 46240 USA

Reader Services

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

This page intentionally left blank

1
Your First App

There is no better way to learn than by actually doing something, so let's dive in by writing a really simple iPad app. The first application you will write is a Hello World app. Yes, the Hello World sample application is overdone, but don't worry—you will be building more sophisticated applications later in this book. For now, it's important to get your hands dirty with some code and the tools.

The goal for this chapter is to give you a sneak peek at the tools you will be using to build your iPad applications. If you are already familiar with Xcode, you may wish to skip ahead to Chapter 4, "Getting Started with Objective-C," or Chapter 6, "Provisioning Your iPad." If you are new to Xcode, please continue reading.

The rest of this chapter will guide you through the steps needed to create your first iPad application. The chapter does not go into detail about Xcode. Instead, those details are covered in the following chapters: Chapter 2, "Getting Started with Xcode," and Chapter 3, "Getting Started with Interface Builder."

> **Note**
>
> Before you begin, you must have Xcode and the iOS SDK installed on your Mac computer. If you do not have these installed, jump to Appendix A, "Installing the Developer Tools," for instructions on how to set up your Mac for iPad programming. This book assumes you are using Xcode 4.5 or newer. And yes, a Mac computer is required.

Creating the Hello World Project

Let's begin by launching Xcode. If you are running Mountain Lion (Mac OS X 10.8) and you downloaded Xcode from the Mac App Store, it is available in Launchpad, shown in Figure 1.1; otherwise, you can find it in your Applications folder. Click the Xcode icon to launch it.

> **Note**
>
> You may find having Xcode on the Dock more convenient than using Launchpad. Adding Xcode to the Dock is simple. First, launch Xcode from Launchpad. While Xcode is running, right-click (or **Control-click**) on the Xcode icon that appears in the Dock and select **Options > Keep in Dock.** This will keep the Xcode icon in the Dock even when the program is not running, making it easier to launch Xcode the next time you need it.

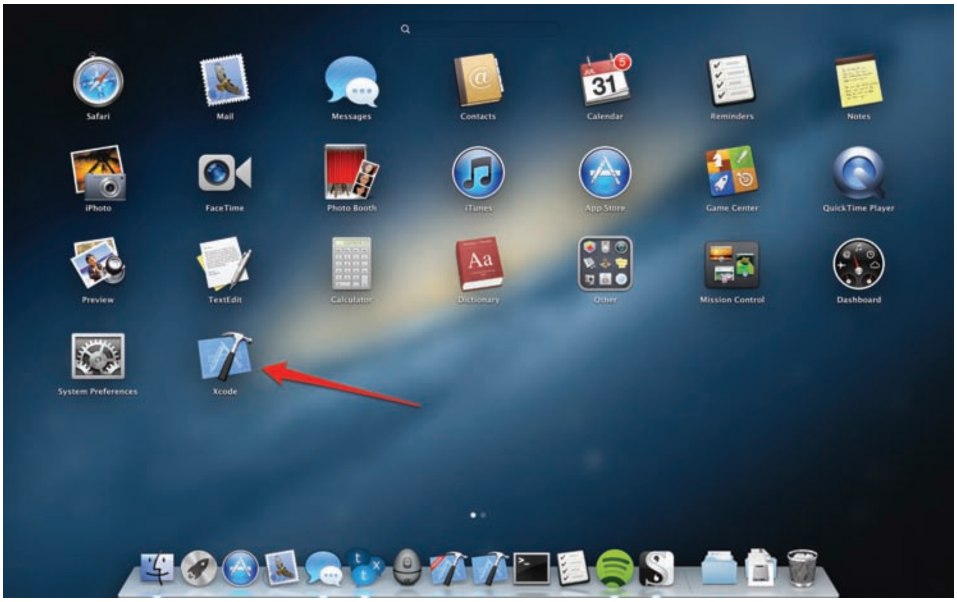


Figure 1.1 The Xcode icon as seen in Launchpad

The first window you see after launching Xcode is the Welcome to Xcode screen, shown in Figure 1.2. You can do a number of things from this window, including creating a new project, connecting to a source code repository, going to the *Xcode 4 User Guide* (a tutorial on using Xcode), or visiting Apple's Developer site.[1] If you have created or opened Xcode projects in the past, you will also see a list of recent projects on the right side of this screen. You can open a recent project by selecting it from the list and clicking **Open**.

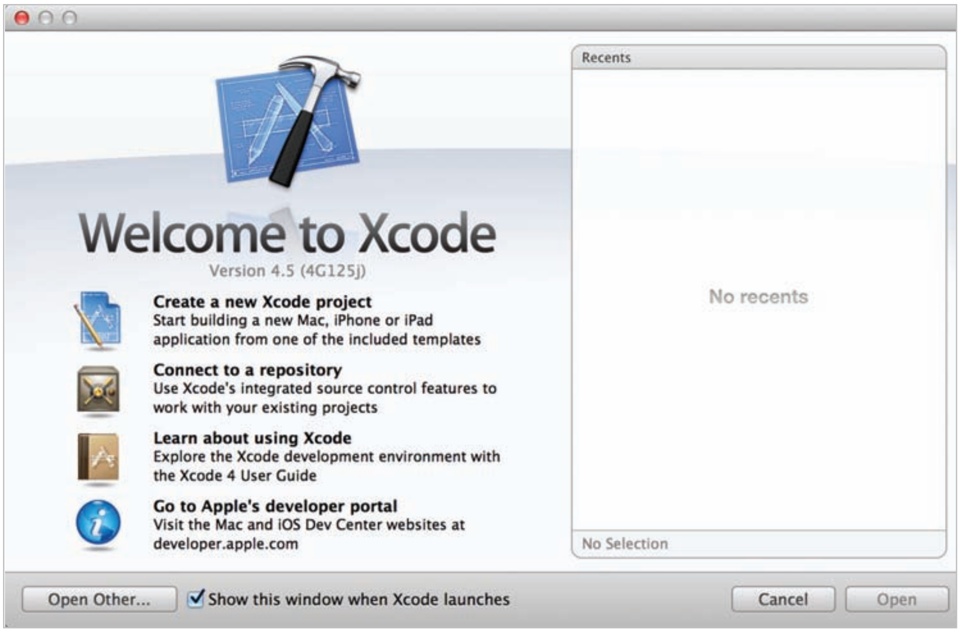
Tucked away in the lower-left corner is the **Open Other...** button. You can click this button to open an existing Xcode project found on the file system. Next to this button is a check box indicating whether the Welcome to Xcode window is displayed when Xcode launches.

Note

If you are new to Xcode, you should take the time to read through the *Xcode 4 User Guide*, which provides complete coverage of the entire Xcode tool set. You will learn about Xcode in this book; however, reading the official guides from Apple is always a good thing.

You want to create a new iPad application, so click the **Create a new Xcode project** button. This opens the new project window, as shown in Figure 1.3. Let's

1. Apple Developer site: http://developer.apple.com



Create a new Xcode project
Start building a new Mac, iPhone or iPad application from one of the included templates

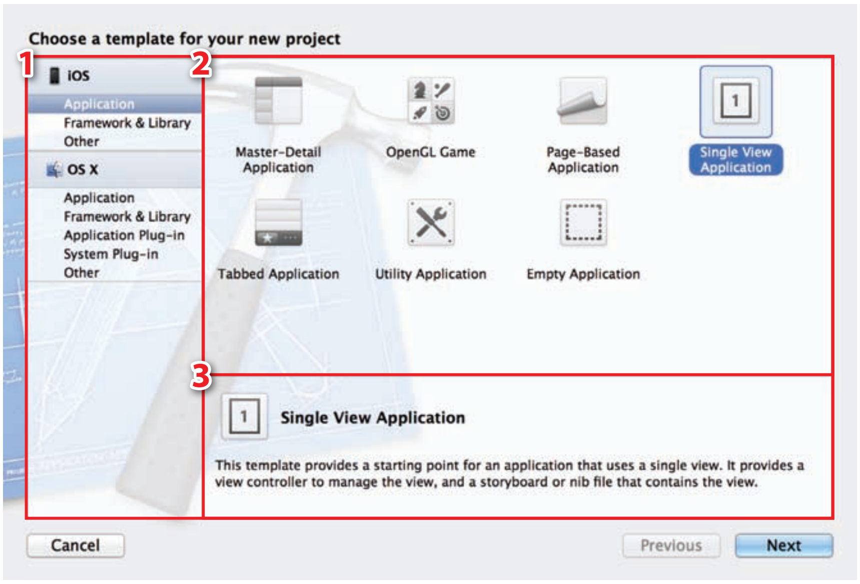
Connect to a repository
Use Xcode's integrated source control features to work with your existing projects

Learn about using Xcode
Explore the Xcode development environment with the Xcode 4 User Guide

Go to Apple's developer portal
Visit the Mac and iOS Dev Center websites at developer.apple.com

[✓] Show this window when Xcode launches

Figure 1.2 Welcome to Xcode window



Single View Application

This template provides a starting point for an application that uses a single view. It provides a view controller to manage the view, and a storyboard or nib file that contains the view.

Figure 1.3 The new project window in Xcode, with callouts on sections of Xcode's user interface (1: target type; 2: project template; 3: template detail)

explore this window for a moment before continuing. As you can see in the figure, the new project has three main sections:

1. Target type
2. Project template
3. Template detail

In section 1, you select the target type: iOS or Mac OS X. iPad applications run on iOS, so you can ignore the Mac OS X target type for now. Under iOS you can build two types of targets: Application and Framework & Library. The Application type is exactly what the name implies; you use it to build iPhone and iPad applications. The Library target type is for building reusable static libraries, which you can also ignore for now.

The Hello World application you are building is just that, an application. Thus, in section 1 under iOS, you select Application. When you do so, you'll notice that the content in section 2 changes. Section 2 now displays the list of available templates for the selected target type. A template is used to generate the initial files needed for an Xcode project.

If you have spent time playing with your iPad, you may have noticed that there are some common application types, or styles. The templates listed in section 2 help speed the process of creating an application of a particular style. For example, if you wanted to create an application that looks similar to the Mail app on the iPad, you would select Master–Detail Application.

Application Templates

The application templates you'll encounter in Xcode after selecting iOS as your target include the following types:

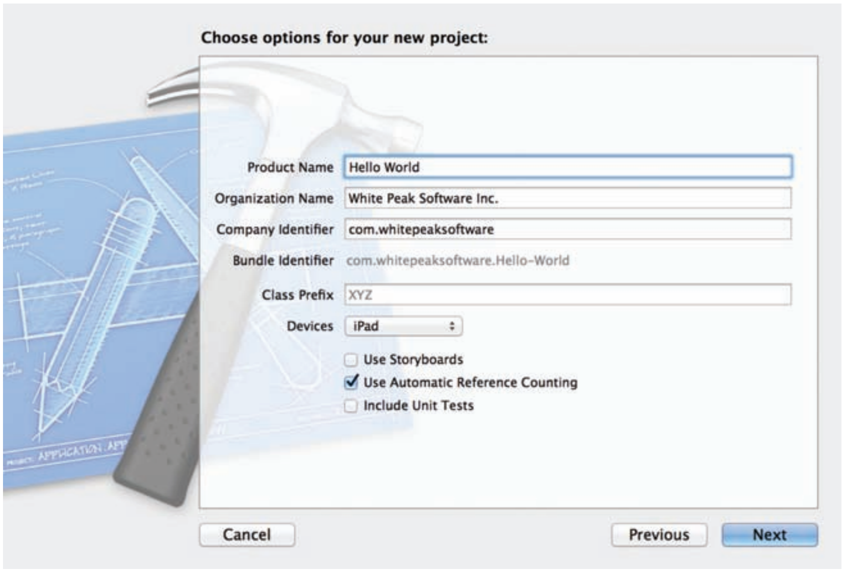
- **Master-Detail Application**: Select this template when you have a master-detail style of application and wish to leverage the split view controller for display.
- **OpenGL Game**: Select this template when you want to create a game using OpenGL ES. This template provides a view with an OpenGL scene and timer to animate the view.
- **Page-Based Application**: Select this template to create a book- or magazine-style app that uses the page view controller.
- **Single View Application**: Select this template for applications that use a single view.
- **Tabbed Application**: Select this template for applications that have separate areas defined by tabs. This template provides a tab bar controller and view controllers for two tabs.
- **Utility Application**: Select this template for applications that have a main view and an alternate view.
- **Empty Application**: This template provides a starting point for any type of application. Select this template when you want to start with a bare-bones project shell.

The Hello World application will consist of a single view, so select Single View Application from the list of templates. When you do so, notice that the content of the template detail section changes. This section shows a brief description of the template selected in the project template section.

Clicking the **Next** button takes you to the project options screen, shown in Figure 1.4. Project options vary slightly based on the template. Each template has options for the Product Name, Organization Name, Company Identifier, Bundler Identifier (which is completed for you based on the Company Identifier and Product Name), Class Prefix, and Device Family. Additional options that may be found on an application template include Use Storyboard, Use Automatic Reference Counting, Use Core Data, and Include Unit Tests. The application template you select determines which additional options are made available.

For the Hello World app you are building, enter "Hello World" for the Product Name. The Organization Name can be any value you like, as it is used only in the copyright message included in the comment section that is added to the top of each *.h* and *.m* file created by Xcode for the project.

For the Company Identifier, enter your name or company name using the reverse domain name format. (For example, com.kirbyturner is my individual name and com.whitepeaksoftware is my company name.) Chapter 6, "Provisioning Your iPad," explains the relationship between the company and bundle identifiers and describes how they are used to form the App ID.


Choose options for your new project:

Product Name: Hello World
Organization Name: White Peak Software Inc.
Company Identifier: com.whitepeaksoftware
Bundle Identifier: com.whitepeaksoftware.Hello-World
Class Prefix: XYZ
Devices: iPad

[ ] Use Storyboards
[✓] Use Automatic Reference Counting
[ ] Include Unit Tests

Figure 1.4 Project options for the Single View Application template

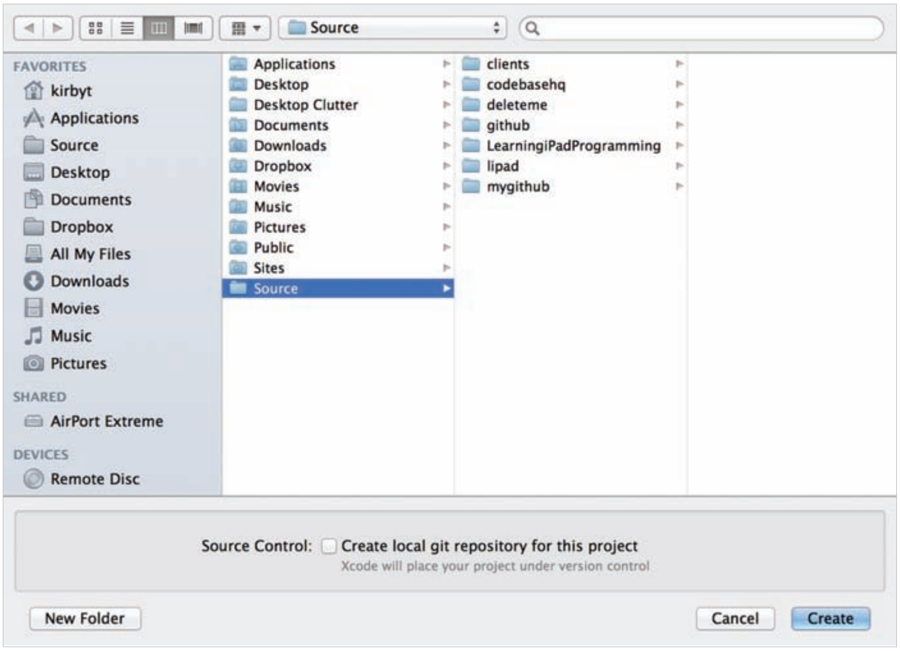
The Class Prefix can be used to append a string value to the beginning of each class generated by the application template. For the purpose of simplicity, you can leave the Class Prefix blank for this app.

Next, select iPad as the Device Family. There are three device family types in iOS: iPad, iPhone, and Universal. The device family iPad indicates that the app is designed for and runs on the iPad only. The iPhone device family indicates that the app is designed for the iPhone, and Universal says that the app is designed for and runs on both the iPad and the iPhone.

You do not need storyboard and unit tests in this Hello World app, so leave those options unselected. (Storyboarding is covered in Chapter 14, "Storyboarding in Xcode.") But do select the Use Automatic Reference Counting option—it determines how memory of an object is managed, and is explained in the Memory Management section of Chapter 4, "Getting Started with Objective-C." Click the **Next** button, choose a storage location for the Xcode project, and then click the **Create** button (shown in Figure 1.5).

Note

I like to keep all my source code together in a single location, so I created a *Source* directory within my home directory. I place all my Xcode projects under *Source* so I can easily locate them in the future.



Source Control: Create local git repository for this project
Xcode will place your project under version control

Figure 1.5 Choose the location where your Xcode project is stored.

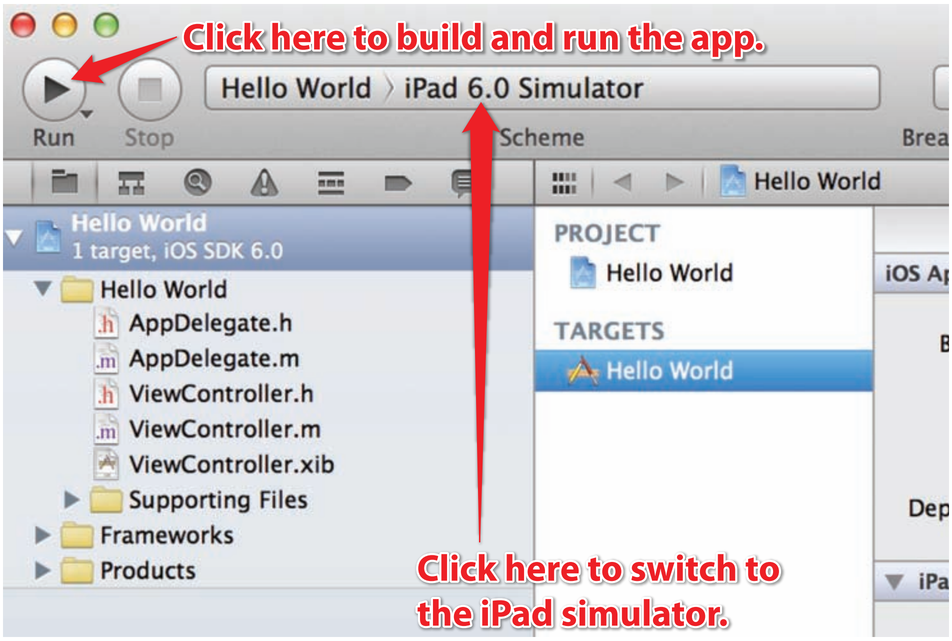


Figure 1.6 Xcode project window for the Hello World app

Congratulations! You just created your first iPad application. You don't believe it? Click the **Run** button (shown in Figure 1.6), or press **Command-R**. Be sure the active schema is set to the iPad Simulator. If it is not, click it and change it to the simulator.

Universal App

iPhone apps can run on the iPad, but they run in an iPhone emulator. Because they do not take advantage of the iPad's full screen, this behavior leads to a less than ideal user experience. A universal app, in contrast, is designed to take full advantage of the screen real estate provided by both the iPhone and the iPad. When a universal app is run on an iPhone, it looks as if it was designed for the iPhone. Conversely, when a universal app is run on an iPad, it looks like an iPad app, not an iPhone app.

A universal app gives the user the best of both worlds—a single app that looks great on both devices. However, this comes at a cost to you, the developer. Developing a universal app, in many ways, is like developing two separate apps, one for the iPad and one for the iPhone, and packaging them into a single app binary.

Universal apps are designed to target both the iPad and the iPhone. The focus of this book, however, is on writing iPad applications. To keep you focused, and to avoid the additional complexities of writing universal apps as you start your journey toward becoming an iOS developer, universal apps are not covered until the end of Part II in this book.

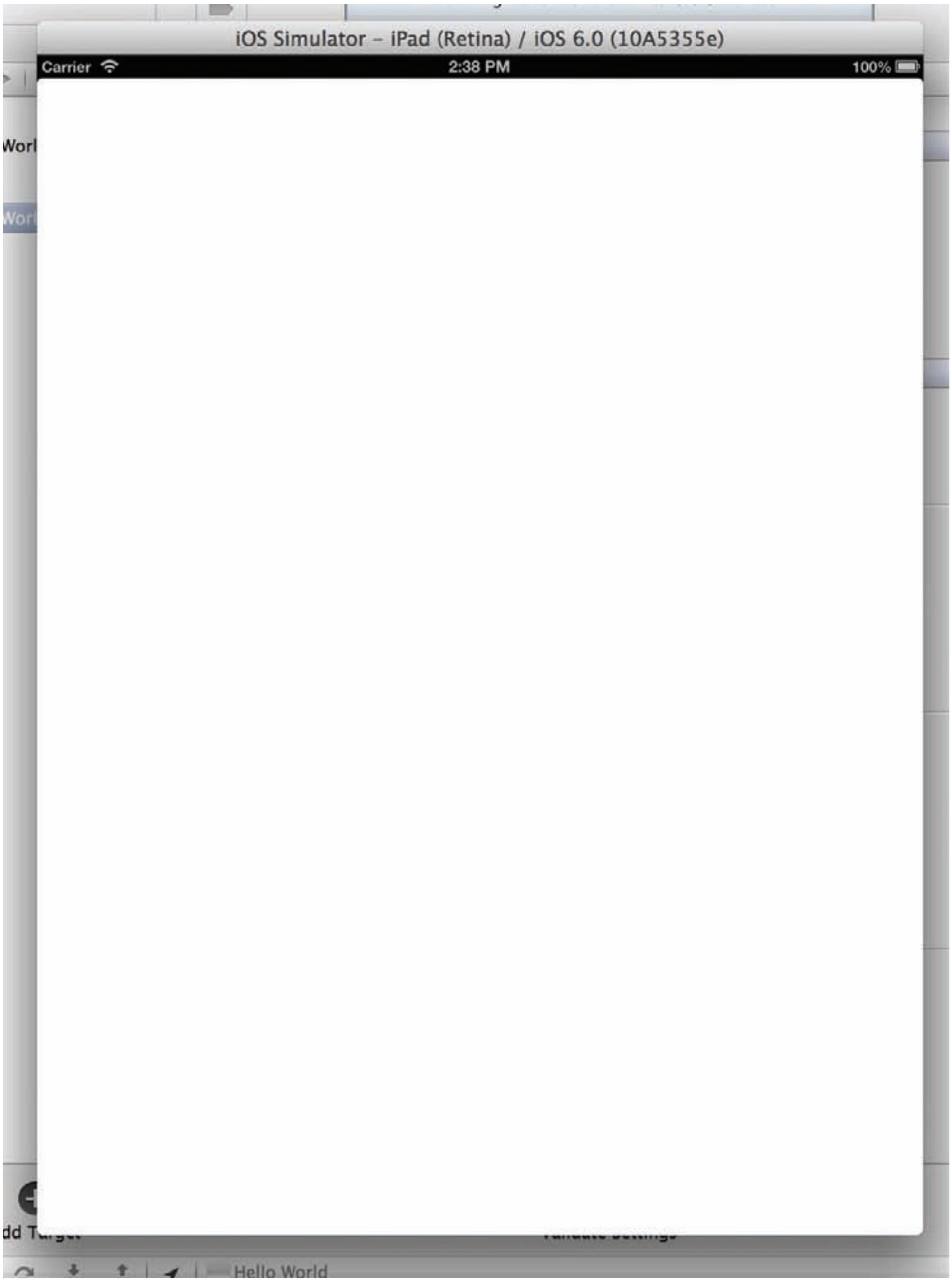


Figure 1.7 A “blank” single view app running in the iPad Simulator

When you click **Run**, Xcode compiles the project, builds an application package, installs the application on the iPad Simulator, and finally launches the application inside the simulator. As you can see in Figure 1.7, the application is nothing more than a white screen. Guess what? You just built your first flashlight app for the iPad!

> **Note**
>
> Sometimes you will notice a delay between the time the simulator is launched and the time your app launches within the simulator. When this delay happens, you see nothing but a black screen within the simulator. This is normal, and it usually happens only the first time you launch your app in the simulator.

You can take your newly created flashlight application and submit it to Apple for review. However, there is a high level of certainty that Apple will reject your masterpiece because of its lack of functionality. Besides, you are not done with this app. You want to build a Hello World application, and, as you can see, "Hello World" does not appear when this application is run. So let's continue working on it.

First, stop the app, which is running in the simulator. You can do so by clicking the **Stop** button at the upper-left side of Xcode or by pressing **Command-.** in Xcode (not in the simulator). Now you're ready to start modifying the app.

> **Note**
>
> When you use a project template, Xcode gives you a valid, runnable iPad application without your having to write a single line of code. Perhaps it is because I still have memories of being a teenager building apps 30 years ago, but I always get a little warm, fuzzy feeling when I see a new application run for the first time. As a matter of fact, the first thing I do when I create a new Xcode project is to build and run it. Seeing the application run for the first time gives me a little jolt of excitement.

Getting Text on the Screen

This is a Hello World app, so it should display "Hello World" somewhere on the screen. This can be accomplished by writing some code, but the easiest approach is to use Interface Builder. Interface Builder, or IB as it is often called, is the visual user interface designer built into Xcode. You'll learn more about IB in Chapter 3, "Getting Started with Interface Builder," but for now steps are provided to guide you through turning this blank application into a not-so-useful Hello World app.

To add "Hello World" to the display, you'll edit the file *ViewController.xib*. A *.xib* file, pronounced "zib," is an XML representation of a NIB file. A NIB file, or *.nib*, is the binary predecessor of the *.xib* file. Being text based, a *.xib* file has the benefit of working better with version-control systems when compared to the earlier binary *.nib* version. That said, *.xib* files are still compiled down to *.nib* files when you build the application.

What is a NIB file? It is a file created by Interface Builder to archive interface objects and their relationships. Put another way, a NIB represents the objects that make up the visual display of a screen. You create and edit NIB files using IB, and your application uses the NIB files at run time to display the user interface of the app.

Note

iOS developers often refer to a *.xib* file as a NIB file because it is, after all, just a text-based representation of a NIB file.

History

The *N* in NIB is a carryover from the NeXTSTEP days when it was used to indicate the NeXT-style property list file. The *IB* indicates that the file is an Interface Builder file.

Begin by opening the file *ViewController.xib*, available in the Project navigator. This changes the contents of the Editor area. It displays the NIB file using the IB designer, as shown in Figure 1.8.

Note

Chapter 3, "Getting Started with Interface Builder," covers all the utilities available with IB.

IB has a set of available utilities for working with a NIB file. Press **Control-Option-Command-3** to display the Object library. The Object library contains a list of visual and nonvisual components that are used to construct the user interface. In the filter bar at the bottom, type "Label" without the quotes. This will filter the object list, displaying only label-type objects.

Drag and drop the label object onto the view's canvas area. This creates a new `UILabel` instance, which is the type of object representing a label. Next, open the Attributes inspector (**Option-Command-4**). At the top of the Attributes inspector is a property named Text. Change the default value "Label" to "Hello World." Xcode should now look similar to Figure 1.8.

You may need to resize the label to view the entire "Hello World" content. To resize it, move the mouse cursor to the right edge of the label object. The cursor will change to the resize indicator. Click and drag the mouse to the right to increase the width of the label.

Build and run the app in the iPad Simulator. Congratulations! You have written your first Hello World app for the iPad.

Note

Don't worry if none of this is making sense yet. Remember—the goal of this chapter is to give you a sneak peek into iPad programming by way of a step-by-step guide. This discussion is intended to give you a sense of what it is like to program for the iPad. Later chapters will explain all you need to know in detail, and before you know it, the steps for creating iPad applications will be second nature to you.

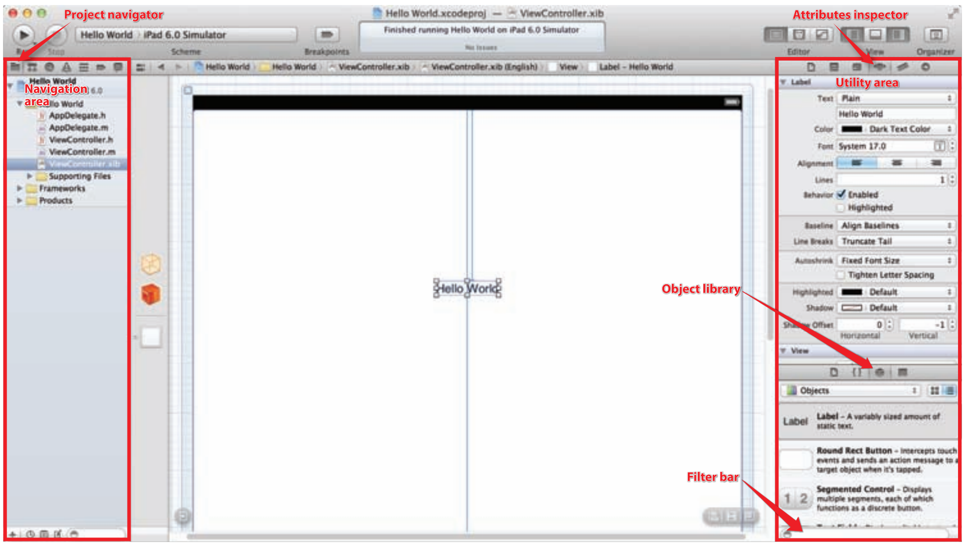


Figure 1.8 Adding “Hello World” to the main view of the app

Say Hello

Now that the excitement of creating your first application for the iPad has worn off, let’s extend the application by adding some functionality to it. Instead of having it always display “Hello World,” let’s change the app to first ask for a name, then display a “Hello” message in response to the name entered. This exercise is more involved and requires you to write some Objective-C code. Do not worry if you have never seen Objective-C code before. You will be told exactly what to type, and you will explore Objective-C in more detail in Chapter 4, “Getting Started with Objective-C.”

In life there is often more than one way to accomplish a task. The beauty of iPad programming is that there are many different ways to do something. It is this flexibility in the development tools that makes many programmers prefer Xcode to other development tools. But it does take time to learn all the ins and outs, which can be frustrating for programmers new to Xcode.

One of the goals of this book is to show you the different ways a task can be accomplished. Armed with this knowledge, you can decide which approaches work best for you. For example, it is possible to use IB to generate Objective-C code that declares objects and actions defined in a *.xib* file. However, this discussion is saved for a later chapter. Instead, you’re going to write the Objective-C code yourself to extend functionality in the Hello World app.

Two screen elements are needed: one that accepts user input for the name and the other to display “Hello.” A third element, a button, is also needed to tell the app when to display the “Hello” message. The NIB file defines the objects that make up the user interface, but there is no automatic connection between the objects and the source code. Instead, you must make the connection.

Start by opening the file *ViewController.h*. You can find this file in the Project navigator. When you click it, the Editor area will display the contents of the file. Modify the file's contents so that the source code looks exactly as it does in Listing 1.1.

Listing 1.1 **Modified Version of *ViewController.h***

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

@interface ViewController : UIViewController

@property (nonatomic, weak) IBOutlet UILabel *helloLabel;
@property (nonatomic, weak) IBOutlet UITextField *nameField;

- (IBAction)displayHelloName:(id)sender;

@end
```

Next, open the file *ViewController.m*. Replace the generated source code found in the file with the source code in Listing 1.2.

Listing 1.2 **Modified Version of *ViewController.m***

```
#import "ViewController.h"

@implementation ViewController

@synthesize helloLabel;
@synthesize nameField;

- (IBAction)displayHelloName:(id)sender
{
   NSString *hello = [NSString stringWithFormat:@"Hello %@", [nameField text]];
   [helloLabel setText:hello];
}

@end
```

The code in Listing 1.1 does a number of things. First, two properties are added to the class `ViewController`. These properties are marked with `IBOutlet`, which is a hint to IB that the class contains a reference to an object. Next, the method `-displayHelloName:` is declared. It is marked with `IBAction`, another hint to IB, this time telling IB that an action exists in the class definition. At this point, the interface for the class `ViewController` has been defined.

What Are `IBOutlet` and `IBAction`?

`IBOutlet` and `IBAction` are special indicators for Interface Builder—hence the IB prefix. Interface Builder uses these indicators to connect objects and actions to elements in the user interface.

An `IBOutlet` is used to connect an object reference defined in Objective-C code to the object instance used in Interface Builder. For example, earlier in this chapter you placed a label on the view. That label is actually a `UILabel`. (`UILabel` is the class name for the label.) To access the label in code, you must have a reference to the instance of the `UILabel`. You will see later in this chapter how you connect the reference declared in code to the instance displayed in IB.

An `IBAction` is used to connect an event sent by an object to a method defined in code. For example, a button has an event that is fired when a user lifts her finger. This action can be connected to the `IBAction` defined in the Objective-C class.

The code in Listing 1.2 represents the implementation for the class `ViewController`. This implementation begins by synthesizing the two properties declared in the class interface, `helloLabel` and `nameField`. Property synthesis is an Objective-C compiler feature that generates the accessor methods for these properties at compile time. You'll learn more about this feature in Chapter 4, "Getting Started with Objective-C."

The property synthesis is followed by the implementation for the method `-displayHelloName:`. This method is the action that is called when the user interacts with the app—specifically, when the user taps a button—which you will provide momentarily. The implementation of this method creates a local string variable containing the name entered by the user with the prefix "Hello." This string is then displayed on the screen as the text value for the `helloLabel`.

If you were to run the app at this point, you would see no difference from the earlier version. While the code has been updated to do what you want it to do, the user interface has not been updated and the connections for the outlets and actions have not been made.

Note

This decoupling of the source code—in this particular case, the controller—and the user interface (also known as the view) is representative of the Model-View-Controller design pattern, which is discussed in Chapter 5, "Getting Started with Cocoa."

To complete the app, you need to update the user interface and connect the UI objects to the properties defined in the controller class. Once again, open the file *ViewController.xib*. Double-click the "Hello World" label and change its text value to "What is your name?" Resize the label as needed to display the entire text.

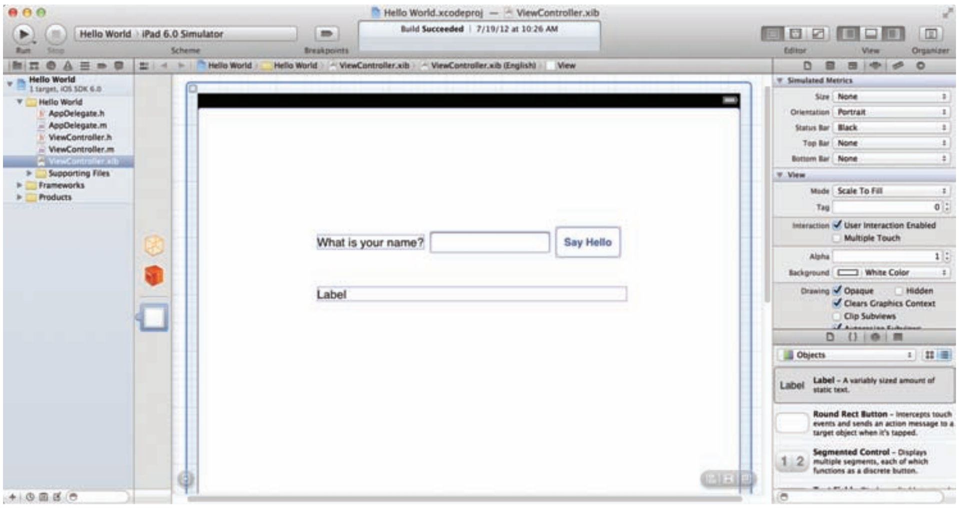
Search through the Object library in the Utilities area for the Text Field object. Alternatively, you can filter the object list by typing "text field" in the filter bar. Drag and drop a text field to the right of the "What is your name?" label.

Now search through the Object library for the Round Rect Button. Drag and drop an instance of this button to the right of the text field. In the Attributes inspector, change the Title property to "Say Hello."

Finally, search the Object library for Label, and drag and drop a new label onto the canvas, placing it under the other objects. Be sure to increase the width of the label to accommodate the string value created in the method `-displayHelloName:`. The view should look similar to Figure 1.9.

Now it's time to connect the objects and events defined in the NIB with the outlets and actions defined in the view controller source code. One way to connect objects to outlets and actions is to **Control-click** an object, and then drag the mouse cursor to another object. When the mouse button is released, IB will display a Heads-Up Display (HUD) of the connection options. For example, when you **Control-click** the *File's Owner* object (the translucent cube displayed in the left sidebar in the Editor area) and drag it to the text field (shown in Figure 1.10), a HUD is displayed, allowing you to connect the text field to the properties `nameField` and `view`. Select `nameField` to connect the text field to the property defined in *ViewController.h*.

Do the same thing to connect the label to the property `helloLabel`. **Control-click** the *File's Owner* cube and drag to the label where the output of the `-displayHelloName:` will be displayed.



Size: None
Orientation: Portrait
Status Bar: Black
Top Bar: None
Bottom Bar: None
Mode: Scale To Fill
Tag: 0
Interaction: ✓ User Interaction Enabled; Multiple Touch
Alpha: 1
Background: White Color
Drawing: ✓ Opaque; Hidden; ✓ Clears Graphics Context; Clip Subviews

Label – A variably sized amount of static text.
Round Rect Button – Intercepts touch events and sends an action message to a target object when it's tapped.
Segmented Control – Displays multiple segments, each of which functions as a discrete button.

Figure 1.9 The modified user interface file *ViewController.xib*

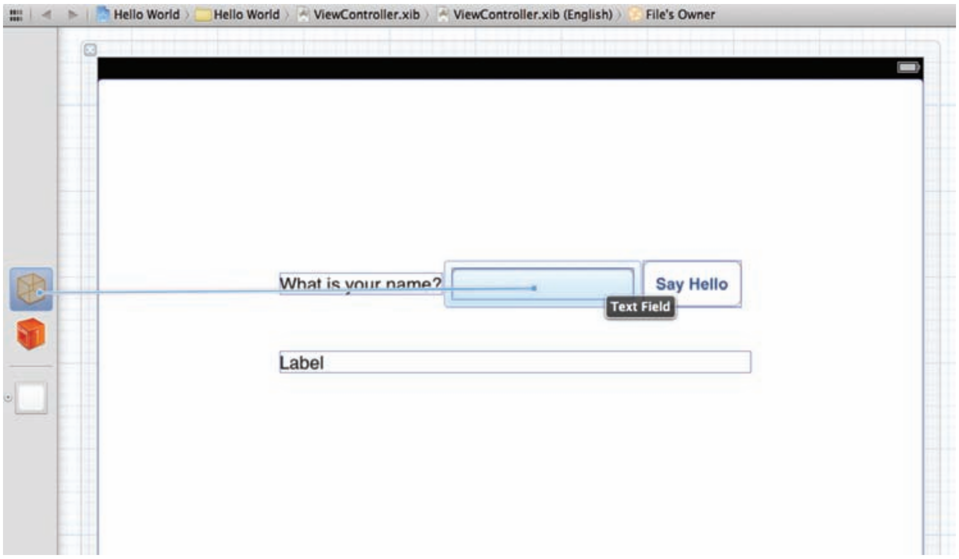


Figure 1.10 Connect the `nameField` property to the text field defined in the NIB file.

To connect the action to the **Say Hello** button, you **Control-click** the button and drag to the *File's Owner* cube. This will assign the action `-displayHelloName:` to the button event *Touch Up Inside*.

With the connections in place, the Hello World app is now functional. Build and run the app in the simulator. Tap the name field in the simulator to enter a value, and then tap the **Say Hello** button to dissplay the "Hello" message. The final app should look similar to Figure 1.11.

You might be wondering how IB is able to identify the correct Objective-C header file. It's simple: The file's owner is defined as being of type `ViewController`. This tells IB which source file to look at for outlets and actions. You can see this by clicking the *File's Owner* cube, and then typing **Option-Command-3**. The class name is set to `ViewController`. This is how an object defined in IB knows its type.

> **Note**
>
> A common mistake made in Interface Builder is forgetting to associate your outlets and actions. If you run the application and notice that the display does not update after the **Say Hello** button is touched, chances are good that the *Touch Up Inside* event for the `UIButton` is not associated with the `-displayHelloName:` action.

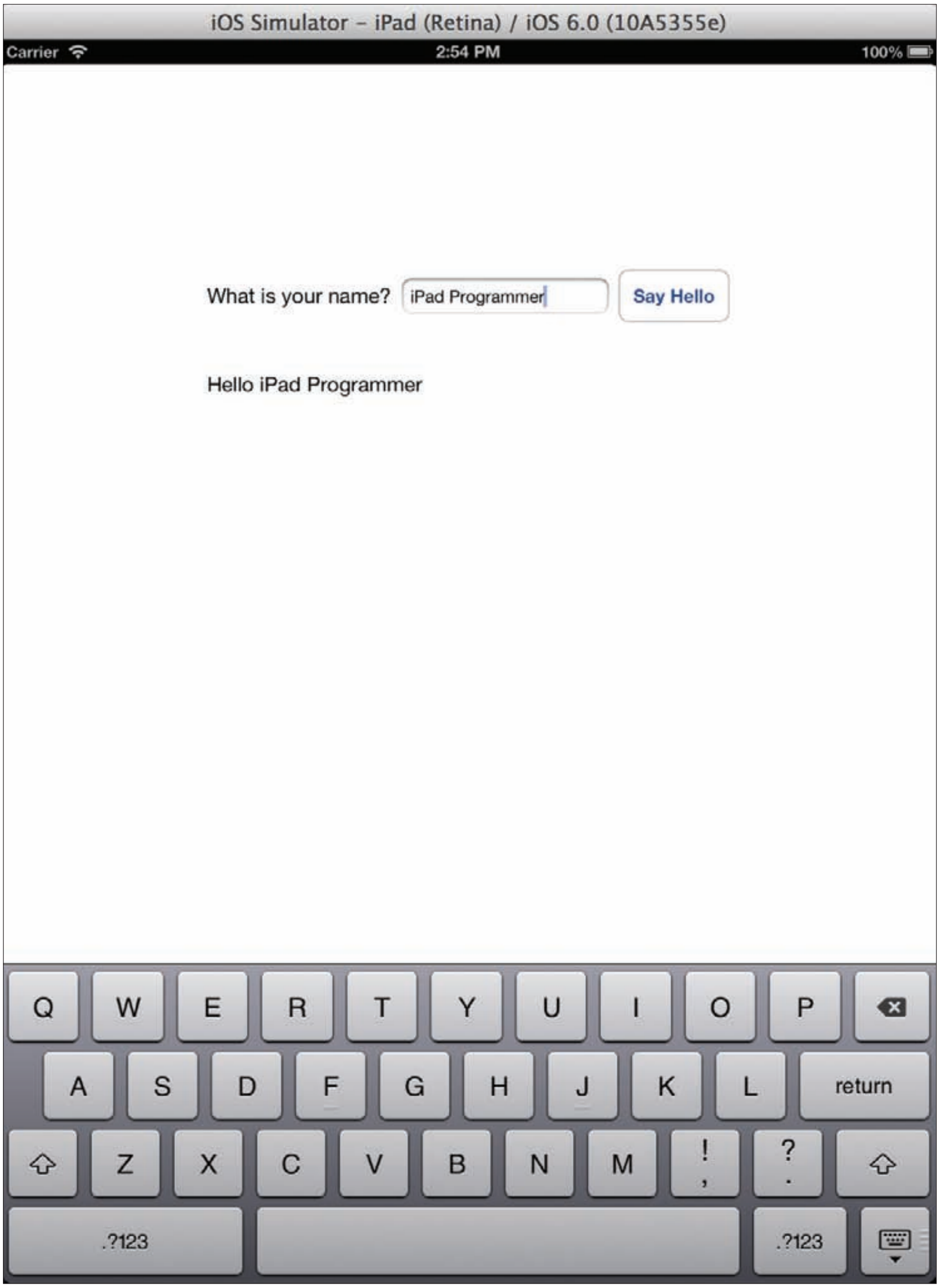


Figure 1.11 The new and improved Hello World app

Summary

Congratulations! You have completed your first iPad application—and you just got a sneak peek into iPad programming. This chapter should leave you itching to learn more. Before you dive into the meat of iPad programming, however, you need to learn more about the tools and programming language you will use. Let's begin by taking a closer look at Xcode in the next chapter.

This page intentionally left blank

Index

A

About scene
  creating, 352–355
  rotating, 510
  segues, 355–357
  view controllers, 359–363
AboutSceneSegue identifier, 355
Above the fold display, 159
Accessorizer tool, 36
accessoryButtonTappedForRowWithIndex
    Path method, 225, 228, 330
Action sheets, 289–293
actionButton property, 485
Actions
  IBAction. *See* IBAction macro
  slideshow controls, 621
Activity View Controller, 537–540
Ad Hoc distribution, 689
  preparing, 691–694
  provisioning, 690–691
Add Devices page, 141
addButton method, 211
addButtonsToNavigationBar method
  deleting photos, 481–483, 485
  filter containers, 638–639
  slideshows, 620
addChildViewController method, 364–365
Adding
  Core Data entities, 313–316
  images, 348–349
  photo albums, 408–409, 411, 416
  photos to albums, 331–334, 434–439
  slideshows, 609–610
  table view data, 210–224
addPhoto method
  adding photos to albums, 438
  Flickr, 548
  naming photo albums, 423
addphoto.png file, 548, 574
addPhotoAlbum method, 408–409, 411, 416
addPhotosObject method, 319
Admin role in iTunes Connect, 703
Adobe Photoshop, 167–168
*Advanced Collection Views and Building Custom
    Layouts* video, 248
advanceSlide method, 619
Advancing photos, 617–619
affineTileFilter method, 641
Age calculations, 102
AirPlay. *See* Slideshows
AirPrint. *See* Printing
AirServer app, 609
ALAssetsLibrary class, 285
Albums. *See* Photo albums
Albums View Controller Scene, 407
albumsView property, 500
AlbumsViewController class
  container view controllers, 366
  iCloud, 595–597, 600
  implementing, 408–413
  managed object context, 414–415
  photo albums, adding, 412, 416
  photo albums, displaying, 408
  photo albums, selecting, 417–419
  scene rotation, 509–510
alertView method, 429
Aligning objects, 53–56
alignScrollViewSubviews method, 512–513
alloc method, 88, 93
Allocations tool, 681–682
Allows External Storage option, 313–314

Angle brackets (<>)
    classes, 185
    header files, 79
Angle of spin gesture rotation, 278, 282
angleOffset property, 508
API
    external display, 606–607
    Flickr, 551–558
    GCD, 576–577
    Printing, 519
API keys in Flickr, 543–545
App Charters, 151–152, 156–158
App icon, 349–350
App IDs
    iCloud service, 582–584
    iOS Provisioning Portal, 143–145
    overview, 128–129
App Store distribution, 689
    assets, 702
    information for, 700–702
    preparing, 695–697
    process, 698–699
    provisioning, 695
    rejected apps, 699
AppDelegate class
    breakpoint example, 667–668
    Core Data stack, 309–310, 312
    photo albums, 415
    PhotoWheel app, 184–188
    storyboards, 347–348
    table view data, 230–231
AppKit framework, 99
Apple IDs, 712
Application bundles, 108–109
applicationDidBecomeActive method, 189
applicationDidEnterBackground method, 189
applicationFrame property, 616
applicationWillEnterForeground method, 189
applicationWillResignActive method,
        188–189
applicationWillTerminate method, 189
applyFilter method, 636, 646
Applying filters, 645–647
applySpecifiedFilter method, 645–647, 649
Apps, 151
    competing products, 155–156
    defining, 151–152
    delegates, 184–189
    distributing. *See* Distribution
    feature lists, 153–154
    icons, 164
    managing, 704
    mockup. *See* Mockup apps
    names, 152, 700
    prototyping, 171–173
    quitting, 182–183
    summaries, 152–153
    target audience, 154–158
    templates, 6
    types, 711–712
    UI design, 158–164
    UIApplication, 113
    universal. *See* Universal apps
ARC (Automatic Reference Counting),
        94–95
arc4random function, 76
arc4random_uniform function, 640
Arithmetic, date, 102
Arrays
    creating, 106
    literals, 89
    strings, 203
    table views, 203
Ash, Mike, 575
assetForURL method, 285
Assets Library framework, 285–286
assign attribute, 83–84
Assistant editor, 38–39, 68–70
Asterisk character (*)
    App IDs, 129, 145, 582
    Bundle IDs, 128
    pointers, 80
Asynchronous photo downloading, 559–565
At signs (@) for literals, 88–89
Atomic properties, 83
Attaching, external display, 607–609
Attributes. *See* Properties
Attributes inspector
    collection views, 240
    opening, 12
    scenes, 355
    titles, 16
    working with, 58
Authentication, 576–577

Auto Layout system
  overview, 58–59
  scene rotation, 496, 502–503, 507
  working with, 492–494
autoAdjustmentFiltersWithOptions method, 631
autoEnhancedVersionOfImage method, 630–631
Autohiding chrome, 468–473
Automatic image enhancement, 630–631, 647
Automatic Reference Counting (ARC), 94–95
Availability by country information for apps, 700
Availability date information for apps, 700
availableModes property, 607
awakeFromInsert method, 325–326, 601–602

B

Background apps, 182
background-landscape-right-grooved.png file, 501
background-portrait-grooved.png file, 351, 501
backOnePhoto method, 621–622
Backups of key pairs, 136
Balsamiq Mockups app, 170–171
Bar buttons, 120
Base settings, 40
Base URLs in Flickr, 556
Began state, 275, 278
__block directive, 572–573
.bmp files, 118
Books, recommended, 707
Booleans, 84
Bottom Space to Superview option, 504
Bounce effect, 258
Bounds of frames, 677–678
bounds property, 607
Brackets ([]) in code, 89
Brainstorming technique, 153
Breakpoint navigator, 24
Breakpoints, 666–667
  customizing, 668–670
  debugging example, 675
  hitting, 670–672
  setting, 667–668
__bridge syntax, 265
Broadcasting events, 417–418
Buck, Erik M., 121
Bugs. *See* Debugging
Build Settings, 40
buildFlickrURLWithParameters method, 555, 558
Bumgarner, Bill, 282
bumpDistortionFilter method, 641
Bundle Display Name setting, 338
Bundle IDs, 128, 700
Bundle Seed IDs, 128, 145
Bundles, 108–109
Business of iPhone App Development, 706
buttonIndex property, 291
ButtonMaker application, 115
Buttons, 13
  bar, 120
  classes for, 114–115
  identifiers, 353
  table views, 210
buttonTapped method, 64–65

C

C programming language, 71
Caching
  cells, 208
  images, 565
CALayer class, 263–264
Camera roll, saving photos to, 297–299
Cameras
  checking for, 287–288
  full-screen, 295
cancel method
  filters, 650
  Flickr, 546, 568
  table view data, 216, 219
cancelChromeDisplayTimer method, 472–473
Cancelled state, 275
canEditRowAtIndexPath method, 224, 228–229
canMoveRowAtIndexPath method, 230
canSendMail method, 529, 531
Carousel view, 256–262
cellAtIndex method
  photo album thumbnails, 433–434
  photo albums, 410, 413
  WheelView, 396, 404, 406

cellDoubleTapped method, 274
cellForItemAtIndexPath method
 collection views, 245
 displaying photos, 444
 Flickr, 570–571
cellForRowAtIndexPath method
 albums, 328
 collection views, 240
 photos, 332
 table views, 207–209, 225, 229–230
cellIndexesToDisplay method, 399–400, 406
cellIndexForIndex method, 399, 406
Cells
 collection views, 242–246, 439–440
 nonvisible rows, 208
 table views, 202
cellTapped method
 image picker controller, 287–288
 touch gestures, 273
.cer files, 136
Certificate Assistant, 132–133, 136
Certificate Revocation Lists (CRLs), 131
Certificate Signing Requests (CSRs), 131, 134–135
Certificates
 CSRs, 134–135
 development, 131–134
 downloading and installing, 135–137
 Provisioning Portal, 690
CFNetwork framework, 559–560
CGGeometry class, 378
CGImage format, 629
CGImageRef structure, 264, 644
CGRectInset setting, 649
CGRectNull setting, 648
CGRectOffset function, 378–379
Changed state, 275, 278
Charters, 151–152, 156–158
Child scenes, 366–369
childViewControllers property, 364
Chrome effects, 468–473
chromeHidden variable, 472, 614
chromeHideTimer property, 472, 614
CIAffineTile filters, 642, 646
CIAffineTransform filters, 642
CIBumpDistortion filters, 642, 644
CICircleSplashDistortion filters, 642, 644
CIColorInvert filters, 627, 642
CIColorPosterize filters, 642
CIContext class, 626, 628
ciContext variable, 634
CICrop filters, 627, 649
CIDetector class, 631–633
CIDetectorTypeFace class, 632
CIFaceFeature class, 631–633
CIFilter class, 625–626
 effects, 641–644
 image enhancement, 631
 overview, 627–628
 working with, 628–630
CIHueAdjust filters, 627, 642
CIImage class, 625, 628–631, 647
circleSplashDistortionFilter method, 641
CISepiaTone filters, 627–629
CITwirlDistortion filters, 642, 644
Clair, Robert, 71
Clark, Josh, 158
@class directive, 232
Class Extensions Explained, 282
Class Prefix setting, 8
Classes
 collection views, 236
 defining, 80
 extensions, 281–282
 implementing, 85–90
 methods, 84–85
 overview, 77–80
clickedButtonAtIndex method
 action sheets, 291
 adding photos, 436, 438
 deleting photos, 484–485
 email, 533–534, 536
 Flickr, 549–550
 printing, 520–521
 removing photo albums, 429
Clock Radio app, 163
Cocoa Auto Layout
 overview, 58–59
 scene rotation, 496, 502–503, 507
 working with, 492–494
Cocoa Design Patterns, 121
Cocoa framework, 65, 97
 design patterns, 121–122
 Foundation library. *See* Foundation library
 RESTful Web services, 542–543

stack, 97-99
UIKit, 112–121
Cocoa Samurai, 35
Cocoa Touch layer, 97–98
Code completion feature, 36
Code folding, 30
Code names, 152
Code Pilot tool, 36
Code separation, 656–657
Code signing
assets exporting and importing, 137–138
settings, 697
Code Snippet library, 27
Coding styles, 33
CoinToss project and CoinTosser class, 76–80
algorithm, 75–76
creating, 72–75
declared properties, 82–84
dot syntax, 91–92
implementation, 85–90
instance variables, 81–82
interfaces, 80
methods, 84–85
selectors, 90–91
working with, 92–93
Collection classes
NSArray and NSMutableArray, 106
NSDictionary and NSMutableDictionary, 106–107
NSSet, NSMutableSet, and NSCountedSet, 107–108
Collection views, 235
classes, 236
custom cells, 242–246
displaying photos, 442–446
flow layout, 236–242
scene rotation, 504–505
Colon characters (:)
class names, 80
methods, 210
parameters, 85
Color settings, 29–30
Combining images, 628
Command Line Tool template, 72–73
Commercial keys in Flickr, 544
commonInit method
spin gesture recognizers, 279, 282
WheelView, 397, 405
Company Identifier setting, 7
Competing products, 155–156
Compilers for Objective-C, 71
Concurrent programming, 576–577
Conditional breakpoints, 667–668
configureExternalScreen method, 615–616, 677
confirmDeletePhotoAlbum method, 428–429
Conflicts
Core Data entities, 317
iCloud service. *See* iCloud service
names, 87
connection method, 112
connectionDidFinishLoading method
images, 562, 564
SimpleDownloader, 112
Connections
classes, 111–112
NIB files to code, 48, 61–70
objects to outlets, 16–17
Connections inspector, 60–61
Console
apps, 73
debugger messages, 672
constraints in Cocoa Auto Layout, 492–494
constraintsWithVisualFormat method, 493–494
constraintWithItem method, 494, 500
Contact information for apps, 701
Container IDs, 594
Container view controllers
child scenes, 366–369
creating, 365–366
overview, 364–365
Containers, iCloud, 585, 594
contentSizeForViewInPopover property, 198–199
Contexts, managed objects. *See* Managed object contexts
Continuous gestures, 270
Continuous recognizers, 275
Control-Click
connections, 16–17, 64–65, 408
Finder, 388
projects, 342, 346
segues, 355
views, 338, 407
controllerDidChangeContent method, 444

Controls
  slideshows, 620–622
  UI design, 162–163
Converting
  radians to degrees, 282–283
  strings to dates, 109
Coordinate systems for photos, 462
copy attribute, 83–84
Copying
  files, 388–390
  methods, 191
  objects, 53
Copyrights, 701
Core Animation, 258
Core Animation for MacOS X and the iPhone, 258
Core Animation framework, 258
Core Data for iOS, 302
Core Data framework, 302
  adding, 306–307
  changing models, 587
  iCloud, 581
  managed object contexts, 305–306
  managed objects and entity descriptions, 303–305
  model editor, 312–313
  NSManagedObject subclasses, 317–321
  overview, 302–303
  persistent stores and persistent store coordinators, 306, 311–312
  photo albums, adding, 415
  photo albums, displaying, 334–335
  photo albums, reading and saving, 327–331
  photos, adding, 331–334
  photos, entities, 313–316
  PhotoWheel, 312–321
  PhotoWheelPrototype, 306
  prototype code, 390–393
  stack setup, 308–312
  transient attributes, 586–587
Core Data Model Versioning and Data Migration, 587
Core Foundation, 100
Core Image effects
  applying filters, 645–647
  auto-enhance, 647
  CIFilter, 627–630, 641–644
  concepts, 625–626
  face zoom, 647–649
  image analysis, 630–633
  instance variables, 633–634
  interface additions for, 634–640
  utility methods, 649–651
Core OS layer, 98
Core Services layer, 98
Cox, Brad, 85
CPUs for images, 626
Crashing apps, 699
Create a new Xcode project option, 4
Create App ID page, 143–145
Create iOS Development Provisioning Profile page, 145–146
createCGImage method, 630
createScaledImagesForImage method, 323–324, 392–393, 591
CRLs (Certificate Revocation Lists), 131
Cropping images, 649
CSRs (Certificate Signing Requests), 131, 134–135
.cur files, 118
Curly braces ({}) for local variables, 80
Current line with breakpoints, 670
currentAlbumIndex property, 327
currentAngle property, 281–282
currentCalendar method, 101
currentIndex property
  external displays, 618
  PhotoBrowserViewController, 452, 454
  slideshows, 612–613, 619, 622–623
currentPhotoView property, 611–613
Custom breakpoints, 668–670
Custom cells, 242–246
Custom layouts, 242
Custom queues, 575
Custom touch gestures, 275–283
Custom views, 246–248
  carousel, 256–262
  photo wheel view cell, 263–267
  wheel, 248–256
CustomNavigationController class
  photo browser, 460, 466–468
  pop transitions, 382–383
  slideshows, 623–624

CustomPushSegue class
  implementing, 375–381
  photo browser, 457–458, 460, 464–466

D

Dalrymple, Mark, 89
Data persistence, 301
  Core Data. *See* Core Data framework
  custom code to model objects, 321–327
  data model, 301–302
  SQLite, 335
Data stores, 303
Data types, 80, 85, 100–101
  NSCalendar, 101
  NSData and NSMutableData, 101
  NSDate, 102
  NSDateComponents, 102
  NSDecimalNumber, 102–103
  NSInteger and NSUInteger, 103–104
  NSNull, 104–105
  NSNumber, 104
  NSObject, 105
  NSString and NSMutableString, 105
dataSource property
  GridView, 547
  photo albums, 407
  table view, 203
  UITableView, 206
  UITableViewDataSource, 202
  wheel view, 249, 253
dateAdded property, 314
Dates
  arithmetic, 102
  formatting, 109
  pick lists, 117
DDEBUG compiler option, 680
dealloc method, 420–421, 595
Debug area, 28
Debug build configurations, 692
DEBUG compiler option, 679–680
Debug navigator, 24, 670–672
Debugging
  breakpoints, 666–667
  concepts, 666–667
  external display code example, 675–679
  NSLog, 679–680
  overview, 665
  problem reproduction, 665–666
  profiling codes, 681–686
  tools, 666
  variable inspection, 672–674
  Xcode, 667–674
Declared properties, 82–84
Dedicated development devices, 130
Default.png file, 515
Default-landscape.png file, 515
defaultNameText property, 227
defaultPhoto.png file, 265, 413, 431
Defensive programming, 660–661
Degrees, converting, 282–283
delegate property, 271
Delegates
  apps, 184–189
  split view controllers, 192–194
  table view data, 214
Deleted iCloud documents and data, 602–603
deletePhoto method, 484–485
deletePhotoConfirmed method, 483–485
Deleting
  objects, 53
  photos, 480–487
  table view data, 228–229
Demo information for apps, 701
dequeueReusableCell method, 395, 403, 406
dequeueReusableCellWithIdentifier method, 209
description method, 673
Descriptions for apps, 701
Deserialization, 48
Design patterns, 121–122
Designers, hiring, 164
Destination controllers, 379, 457
Destination frames, 466
Destination image view, 379
Detail view controllers, 179
detailNavigationController, 187
DetailView.xib field, 252
DetailViewController class, 184
  action sheets, 289–293
  Carousel view, 260–262
  collections, 237–241, 245–246
  image picker controllers, 287–288, 293–295

DetailViewController class (*cont'd*)
- master detail apps, 187, 194
- photos, 331–334
- PhotoWheelViewCell, 265–266
- table view data, 230–232
- titles, 200
- touch gesture recognizers, 271–273
- wheel view, 252–255

Detection, face, 631–633
Dev Center, 135, 714
Developer documentation, 37
Development machines, 130–131
- certificates, 131–137
- CSRs, 134–135

Development provisioning profiles, 129–130, 146–149
Development setup, 139–141
Device family types, 8
Device IDs
- iOS Provisioning Portal, 141–143
- overview, 127–128

.deviceids file, 141
Devices
- managing, 43–44
- provisioning. *See* Provisioning
- rotating. *See* Rotation
- schemes, 42
- setup, 138–141

DIB (Windows Bitmap Format) format, 118
Dictionaries
- classes, 106–107
- face detection, 632
- Flickr, 557–558
- literals, 89
- variables, 673–674
- views, 499, 507
- WheelView, 405

didAccessValueForKey method, 591
didChangeObject method, 410
didChangeValueForKey method, 590
didDeselectItemAtIndex method, 571
didDismissWithButtonIndex method, 291–292
didFailWithError method, 562, 564
didFinishLaunchingWithOptions method
- AppDelegate, 186–188
- Core Data stack, 312
- NIB-based projects, 339
- rotation, 492
- storyboards, 347
- table view data, 231

didFinishPickingMediaWithInfo method, 296–299, 333
didFinishWithResult method, 532
didMoveToParentViewController method, 364–365
didReceiveData method, 562, 564
didReceiveMemoryWarning method, 218
didReceiveResponse method, 562, 564
didRotateFromInterfaceOrientation method, 490, 512–513
didSelectAlbum method, 420–421
didSelectCellAtIndex method
- photo albums, 411, 413, 417, 419
- WheelView, 395

didSelectItemAtIndexPath method
- collections, 238–240
- displaying photos, 444
- Flickr, 571
- photo browser, 457, 459–460, 462

didSelectRowAtIndexPath method
- photos, 332
- table view data, 230, 233

disablesAutomaticKeyboardDismissal method, 566, 572
Discounts for apps, 700
Discrete gestures, 270–271
dismissAbout method, 361–363
dispatch_async function, 576, 594–595
dispatch_get_global_queue function, 576
Dispatch queues, 575
Display area, 113
Display buttons in table views, 210
displayHelloName method, 14–15
Displaying
- external. *See* External display
- Flickr, 548–551
- photo albums, 406–416
- photos, 334–335, 439–446
- slideshows, 610–613
- table view data, 206–210

distantFuture method, 621
Distribution
- Ad Hoc, 691–694
- App Store. *See* App Store distribution

extra steps, 706
iTunes Connect, 703–706
methods, 689–690
provisioning profiles, 130
submitting apps, 705–706
DLog macro, 679–680
Dock
IB, 51
Xcode in, 3
Document Outline, 504
Documentation, developer, 37
Done bug, 384–385
done method
table view data, 216, 219–220
view controllers, 361–362
Don't Repeat Yourself (DRY) principle, 247
Dot syntax, 91–92
Double quotation marks (") for strings, 105
Double tap gesture, 273–274, 478
doubleTapped method, 477
downloaders property, 572–573
downloadImageAtURL method, 561–563
Downloading
certificates, 135–137
development provisioning profiles, 146–147
images, 348
photos, 559–565
Xcode, 714
Downloading Images for a Table without Threads blog post, 560
downloadWithURL method, 111–112
Dragging, 270
DRY (Don't Repeat Yourself) principle, 247
Dudney, Bill, 258
Duplicate detection in iCloud service, 601
Dynamic arrays, 106
@dynamic directive, 320
Dynamic sets, 107

E

editButtonItem property, 224
Editing
breakpoints, 669
table view data, 224–228
Editor area, 25
Editor gutters, 667
Editors, 38–40
Educational discounts for apps, 700
Email
MFMailComposeViewController, 526–527
operation, 525–527
SendEmailController, 527–537
Email addresses for apps, 701
emailCurrentPhoto method, 534–535
emailPhotos method, 536–537
Embedded seques, 368
Empty Application template, 6, 342, 344
Encapsulation, 82
@end directive, 80, 86, 185
End state, 275
End User License Agreement (EULA), 702
enhancedCIImage property, 631
enhanceImage method, 647
Enhancement of images, 630–631, 647
Enterprise program type, 710
Entitlements in iCloud, 585
Entity descriptions, 303–305
enumerateGroupsWithTypes method, 285–286
EULA (End User License Agreement), 702
Events
broadcasting, 417–418
touch, 269–270
Exception breakpoints, 667
Exceptions, 66
Exponents, 102
Exporting code signing assets, 137–138
Extensions, classes, 281–282
External display
API, 606–607
attaching, 607–609
debugging example, 675–679
managing, 613–617
options, 605
requirements, 606
externalDisplaySlideShowController variable, 614, 617
externalScreenWindow variable, 614, 617
ExternalSlideShowViewController class, 609–613
externalViewFrame, 678

F

Face down orientation, 491
Face up orientation, 491
Faces
  detecting, 631–633
  zooming, 647–649
faceZoomRect property, 648–649
Failed state, 275
Fault objects, 305
Feature lists for apps, 153–154
featuresInImage method, 632
fetchedResultsController method
  displaying photos, 443
  method, 411–412
  syncing photos, 595
fetchFlickrPhotoWithSearchString method, 569, 573, 575–576
fetchRequestWithEntityName method, 412
fetchResponseWithURL method, 554–555, 557–558
Fielding, Roy, 542
File coordinators in iCloud, 580–581
File Inspector, 41
File Template library, 27
Files
  copying, 388–390
  header, 79, 184, 187, 191–192
  owners, 61
fileURLForAttributeNamed method, 591
filteredCenterFactor variable, 634, 644
filteredLargeImage variable, 634
filteredRadiusFactor variable, 634, 644
filteredThumbnailImage variable, 634, 638, 644
filteredThumbnailPreviewImages variable, 634, 636, 647
Filters, 109
  applying, 645–647
  CIFilter. *See* CIFilter class
  face zoom, 647–649
  hiding, 650–651
  image analysis, 630–633, 646–647
  instance variables, 633–634
  interface additions for, 634–640
  types, 626–629
  utility methods, 649–651
filterViewContainer property, 635
Finance role in iTunes Connect, 704
Fixed space bar buttons, 120
Flashlight app, 11
Flexible space bar buttons, 120
Flickr, 543–545
  displaying, 548–551
  downloading photos, 559–565
  FlickrViewController class, 565–574
  PhotoWheel, 545–547
  view controller scene, 547–548
  wrapping API, 551–558
flickrJSONSWithParameters method, 555–558
flickrPhotos property, 572–573
FlickrViewController class, 545–548, 551
  arrays, 560
  implementing, 565–574
flip method, 85, 88–90, 93
Floating-point number format specifiers, 108
Flow layout for collection views, 236–242
FMDB project, 335
Fonts
  labels, 355
  scenes, 354–355
  settings, 29–30
forCellWithReuseIdentifier method, 239
Foreground apps, 182
Format specifiers, 108
forRowAtIndexPath method, 224, 228–229
Forwarding messages, 364–365
forwardOnePhoto method, 621–622
Foundation.h file, 79
Foundation library, 99–100
  collection classes, 106–108
  data types, 100–105
  utility classes and functions, 108–112
frameForPageAtIndex method, 451, 454
frameForPagingScrollView method, 451, 454
Frames, 677–678
Framework & Library target, 6
Framework bundles, 109
Freeform table views, 220
Full-screen cameras, 295
Future Proofing Your Applications, 661

G

Garbage collection, 93–94
GCD (Grand Central Dispatch) API, 576–577

GeoJSON files, 702
Gestures. *See* Touch gestures
getExternalScreen method, 614–615, 617
getter attribute, 83–84
Getter methods, 64, 82
GIF (Graphic Interchange Format) format, 118
Git source code repositories, 40, 74, 181
Github repository, 602
Global queues, 575
Glyphish icon set, 168
Google Objective-C Style Guide, 33
GPUs for images, 626
Gradient buttons, 115
Grand Central Dispatch (GCD) API, 576–577
Graphic Interchange Format (GIF) format, 118
Grids
  IB, 51
  photos, 439–446
Grouped tables, 202
GUI PSD template, 167
Guidelines for Cocoa Auto Layout, 492–493
Guides for object alignment, 53–54

H

handleCloudUpdate method, 595–596
Hard-coding universal apps, 659–660
Harrington, Tom, 302
Hashes for URI strings, 588
Header (.h) files, 79, 184, 187, 191–192
Heads-Up Display (HUD), 16
Hello World project
  creating, 3–11
  functionality, 13–18
  text, 11–13
Help
  online, 252
  provisioning, 134
  Quick Help, 37
Hide System Libraries option, 684–685
hideChrome method, 471, 473
hideFilters method, 638
hideOverlay method, 568–569, 573
Hiding filters, 650–651
HIG guideline, 158
Hillegass, Aaron, 71
Hiring designers, 164
Home button, 491
home.png file, 101
Horizontal guides, 53–54
HUD (Heads-Up Display), 16
hueAdjustFilter method, 640

I

IBAction macro
  connections, 48
  description, 14–15
  NIB code, 61–66
  PhotoBrowserViewController, 635
  table view data, 220
iBooks, 161–162
IBOutlet macro
  connections, 48
  description, 14–15
  NIB code, 61–66
  PhotoBrowserViewController, 635–637
  table view data, 219–220
IBOutletCollection macro, 639
IBPlaygroundViewController class, 62–67
IBPlaygroundViewController.xib file, 50–51
iCloud service
  awakeFromInsert calls, 601–602
  changes from, 598–600
  concepts, 580
  deleted documents and data, 602–603
  device provisioning, 582–585
  duplicate detection, 601
  entitlements, 585
  file coordinators and presenters, 580–581
  limitations, 586
  monitoring data, 597–598
  overview, 579–580
  PhotoWheel, 586–591
  preexisting data stores, 600–601
  ubiquitous persistent stores, 581–582
  UIDocument and UIManagedDocument, 581
.ico files, 118
Icon*.png files, 349
Icon72x72.png file, 355

Icons
  apps, 164
  sets, 168
IDE (Integrated Development Environment), 21–22
Identity inspector, 57, 362
Image picker controllers
  action sheets, 289–293
  saving photos to camera roll, 297–299
  working with, 286–289, 293–297
Image View class
  Document Outline, 504
  scene rotation, 504–505
imageAtIndex method, 449
imageDataForAttributeNamed method, 590–591
ImageDownloader class, 561–564
imageFilters variable, 634
imagePickerController method, 296, 435, 437–438
imagePickerPopoverController property, 295
Images
  caching, 565
  child scenes, 367–368
  classes for, 117–118
  collection view cells, 440–441
  Core Image. *See* Core Image effects
  downloading, 348–349
  enhancement, 630–631, 647
  face detection, 631–633
  Mockup apps, 168
  photo album thumbnails, 430–434
  rotating, 513–515
  scaling, 589–590
  segues, 377–379
  storyboards, 348–349
imageTapped method, 473
Immutable classes, 100
iMockups app, 169
@implementation directive, 86, 187
Implementation of classes, 85–90
#import statements, 79, 92
Importing code signing assets, 137–138
Indentation preferences, 30, 32
Index cards, 153
Index paths, 208, 226
index property, 477
indexInWheelView property, 404
indexPath property, 226
Indistinct objects, 107
Industrial design, 160
info.plist file
  Bundle IDs, 129
  launch images, 515
  PhotoWheel app, 184
  rotation support, 489–490
  storyboards, 338
  universal apps, 656
Info settings, 40
Information hiding, 82
Inheritance, 185
init method
  CoinTosser, 87–88, 93
  prototype code, 397, 405
  SimpleDownloader, 111
  spin gesture recognizers, 279, 282
Initial view controllers, 350–352
initPhotoViewCache method, 449–450, 453
initWithCalendarIdentifier method, 101
initWithCoder method
  prototype code, 397
  spin gesture recognizers, 279, 282
initWithDefaultNib method, 216–218, 222
initWithFrame method
  prototype code, 397
  spin gesture recognizers, 279, 282
  zooming, 475, 478
initWithNibName method, 197–198, 261, 293–294
initWithViewController method, 529–531
insertNewObject method, 210–211, 221
Inspectors area, 27
Inspectors overview, 57–61
Installing
  certificates, 135–137
  development provisioning profiles, 147–149
  Xcode, 714–715
Instance methods, 85
Instance variables (ivars), 49
  Objective-C, 81–82
  renaming, 86–87
Instruments tool, 44, 681–686
int data type, 80

Integrated Development Environment (IDE), 21–22
Interface Builder (IB), 11–12, 47–48
  aligning objects, 53–56
  collection views, 242–245
  hands-on practice, 49-53
  layout rectangles, 56
  NIB connections to code, 61–70
  operation, 48–49
  selecting and copying objects, 53
  states, 57–61
  storyboards, 70, 341
  working with, 49–50
@interface directive, 80, 185
Interfaces
  Objective-C, 80
  user. *See* User interface (UI)
Intro to Grand Central Dispatch, 575
invalidatingBarButtonItem method, 191
invertColorFilter method, 640–641
iOS
  device family types, 8
  targets, 6
  touch gestures, 269–270
iOS Configuration Utility, 141
iOS Dev Center, 135, 714
iOS Developer Program, 125, 709
  joining, 710
  membership privileges, 709–710
  registration requirements, 712–713
  team roles, 126
iOS Developer Program Agreement, 699
iOS Human Interface Guidelines, 158
iOS Provisioning Portal, 134, 141
  App IDs, 143–145
  certificates, 690
  development provisioning profile, 146–149
  device IDs, 141–143
  iCloud, 584–585
  overview, 125–127
iOS Simulator
  external display, 675
  Printer Simulator, 522–523
.ipa files, 693
iPad, universal apps for. *See* Universal apps
iPad device family, 8
iPad Simulator, 11
  iCloud, 584
  limitations, 44
  working with, 181–182
_ipad suffix, 657
iPhone, universal apps for. *See* Universal apps
iPhone device family, 8, 431
iPhone emulator, 9
_iphone suffix, 657
isCameraDeviceAvailable method, 288
isIndexVisible method, 398–399, 405
isSelectedItemForAngle method, 398, 405
isSourceTypeAvailable method, 288
Issue navigator, 24
Isted, Tim, 302
isZoomed method, 476, 478
iTunes Connect, 703–706
ivars (instance variables), 49
  Objective-C, 81–82
  renaming, 86–87

J

Joint Photographic Experts Group (JPEG) format, 118
JSON Framework, 543, 556–558
Jump bar, 25

K

kCICategoryStillImage category, 627
kCIContextUseSoftwareRenderer setting, 630
kCIImageAutoAdjustRed Eye setting, 631
Key bindings, 34–35
Key Pair Information window, 133
Key pairs for certificates, 133, 136
Key-value coding (KVC), 304, 629
Key-value pairs for dictionaries, 106–107
Key windows, 188
Keyboards, virtual, 162
Keychain Access application, 131–134, 136
Keychain data, 129
Keynote Kung-Fu toolkit, 168
Keys, Flickr, 543–545
Keywords for apps, 701
KissXML parser, 110

Kochan, Stephen G., 71
kPhotoWheelDidDeletePhotoAtIndex notification, 486
kPhotoWheelDidSelectAlbum notification, 419–421
kRefetchAllDataNotification notification, 595–596
KVC (key-value coding), 304, 629

L

Label class, 12, 16
Labels
  copying, 51–52
  creating, 12
  scenes, 355
  text property, 209
Labor Mate app icon, 166
LaMarche, Jeff, 115, 560
Landscape orientation
  landscape left and landscape right, 491
  launch images, 514
  photo browser, 466
  split view controllers, 179
Large app icon, 702
largeImage attribute, 589
largeImageData attribute, 314, 586–587, 591
Launch images, 513–515
Launch options, 188
Launchpad, 3–4
Layout rectangles, 56
Layouts, custom, 242
layoutSubviews method
  scene rotation, 509
  spin gesture recognizers, 281–282
  WheelView, 251–252, 403, 406
Leading Space to Superview option, 504
Leaks tool, 681
Learning Objective-C 2.0, 71
Lee, Mike, 513
Left-right design, 28–29
Legal role in iTunes Connect, 703
Libraries, 26
  Foundation. *See* Foundation library
  Library area, 27
  Object, 12
Library target type, 6
libxml2 parser, 110
Line wrapping, 32–33
Literals
  Objective-C, 89
  string, 88
loadPage method
  chrome effects, 470
  PhotoBrowserViewController, 451, 454
  zooming, 479–480
loadSubviewsWithFrame method, 475, 478
Local variables, 49
  Objective-C, 81–82
  renaming, 86–87
Location Services for photos, 285
Log Message actions, 669
Log navigator, 24
Logical conditions in searching and filtering data, 109
Long, Matt, 258
Long presses, 270

M

.m files, 79, 85
Magic Piano app, 159
Magical Record framework, 303
Mail app, 179, 525–526
Mail composition view, 526
mailComposeController method, 532
main.m file, 75
  CoinTosser, 92–93
  PhotoWheel, 184
Main queue, 575
Main screen, 387–388
  copying files, 388–390
  Core Data model, 390–393
  photo albums, adding, 416
  photo albums, displaying, 406–416
  photo albums, managing, 417
  photo albums, naming, 421–426
  photo albums, removing, 428–429
  photo albums, selecting, 417–421
  photo albums, thumbnails, 430–434
  photos, adding, 434–439
  photos, displaying, 439–446
  toolbar display, 426–427
  WheelView, 394–406

Main storyboards, 344–345
MainScreenSlideShowViewController class, 616–620, 622–624
MainSlideShowViewController class, 609–611, 613–617
MainStoryboard.ipad.storyboard file, 656
MainStoryboard.iphone.storyboard file, 656
MainStoryboard.storyboard file, 656
 container view controllers, 366
 displaying photos, 439
 done bug, 385
 Flickr, 547
 naming photo albums, 422
 navigation, 371
 PhotoBrowserViewController, 635, 651
 pop transitions, 383
 rotation, 496
 scene rotation, 503, 507
 scenes, 371
 scroll view, 455
 View Controller setting, 361–362
MainViewController class
 container view controllers, 365
 photo albums, 414–415, 417
 photo browser, 457–458, 461–463, 466
 rotation, 495–502
 storyboards, 360–363
makeKeyAndVisible method, 188, 616
Making Apps That Don't Suck, 513
Manage schemes window, 42
Managed object contexts
 creating, 311–312
 iCloud, 598–599
 overview, 305–306
 photo albums, 325–327, 414–415
Managed objects
 overview, 303–305
 path attributes, 588
managedObjectContext property
 iCloud, 598–599
 photo albums, 325, 408, 414–415
managedObjectModel method, 667, 669, 672
Mantissas, 102
Master-Detail Application template, 6
Master-Detail apps
 app delegates, 184–189
 creating, 180–181
 detail view controller, 194
 launch options, 188
 master view controller, 195
 navigation controller, 194–195
 project structure, 184–188
 prototype, 177–183
 split view controller delegates, 192–194
 split view controllers, 178–179, 189–194
Master view controllers, 179, 194–195
masterNavigation Controller, 187
MasterViewController class, 184
 Core Data stack, 312
 displaying data, 206–207
 managed objects, 312
 master detail apps, 187, 194–195
 photo albums, 327–332
 table view data, 210, 221–222, 228–233
 table views, 203–206
 titles, 197–200
Media layer, 98
Media library, 27
Memory management, 84, 93–95
 Allocations tool, 681
 leaks, 91, 93, 439
Menu items, classes for, 117
mergeChangesFromCloud method, 599–600
mergeChangesFromContextDidSave Notification method, 600
Merging iCloud changes, 598–599
Message UI Framework, 525–529
Messages
 debugger, 672
 forwarding, 364–365
 to nil objects, 219
 sending, 108
 SMS, 525
Metaphors in UI design, 160–162
Methods
 copying, 191
 Objective-C, 84–85
MFMailComposeViewController class, 526–527, 532
MFMailComposeViewControllerDelegate protocol, 527
migratePersistentStore method, 601

Mini toolbar, 25, 27
.mobileprovision files, 147–148
MockApp template, 168
Mockup apps, 164
  necessity, 166
  overview, 164–165
  PhotoWheel, 177–178
  tools, 166–167
  wireframes, 169–171
Model editor, 312–313
Model objects, custom code for, 321–327
Model-View-Controller (MVC) design pattern, 15, 121–122
mogenerator tool, 321
.momd files, 309–310
monitoring iCloud data, 597–598
Motion events, 269
motionBegan method, 269
motionCancelled method, 269
motionEnded method, 269
Mouse clicks in design, 159
moveRowAtIndexPath method, 229–230
Moving guides, 54
Multitasking, 182
Multithreaded applications, 83
Mutable classes, 100–101
MVC (Model-View-Controller) design pattern, 15, 121–122
myBalsamiq app, 171

N

Name editor for albums, 330–331
NameEditorViewController class
  albums, 330–331
  table view data, 212–217, 220–227
NameEditorViewControllerDelegate protocol, 214–215
nameEditorViewControllerDidCancel method, 215, 218–219, 222–223
nameEditorViewControllerDidFinish method
  albums, 331
  table view data, 215, 218–219, 221–225
Names
  albums, 330–331
  apps, 152, 700
  ivars, 86–87
  organization, 41–42
  parameters, 215
  photo albums, 421–426
  registered devices, 141
Navigation area, 24–25
Navigation bar
  classes for, 119
  scenes, 371, 374–375
  segues, 379
Navigation controller, 194–195
navigationItem property, 211
Navigator
  debug, 670–672
  descriptions, 24
New project window, 4–5
NIB files, 11–12
  connections to code, 61–70
  overview, 48
  vs. storyboards, 338–339
nibWithNibName method, 431
nil objects, messages to, 219
nil value for properties, 94–95
No Access role, 126
nonatomic attribute, 83
Nonvisible rows, 208
Notifications
  broadcasting events, 417–418
  iCloud, 595–596
NSArray class
  description, 106
  literals, 89
  table views, 203
NSBundle class, 108–109
NSCalendar class, 101
NSClassFromString method, 661
NSConferenceiPhoneCoreDataRecipes scheme, 602
NSConfinementConcurrencyType setting, 599
NSCountedSet class, 107–108
NSData class, 101
NSDate class, 102, 302
NSDateComponents class, 102
NSDateFormatter class, 109
NSDecimalNumber class, 102–103
NSDefaultRunLoopMode mode, 564
NSDictionary class
  description, 106–107

Flickr, 557
image picker controllers, 296
literals, 89
variables, 673–674
NSDictionaryOfVariableBindings function, 494, 499, 507
NSEntityDescription class, 304
NSError class, 557
NSFetchedResultsController protocol, 411, 600
NSFetchedResultsControllerDelegate protocol, 408, 445
NSFetchRequest class, 412
NSFileCoordinator class, 580
NSFileManager class, 109
NSFilePresenter protocol, 580–581
NSIndexPath class, 202, 226, 419
NSInteger class, 103–104
NSJSONSerialization class, 111, 543, 556–558
NSLocale keys, 101
NSLocalizedString function, 193–194
NSLog function
breakpoints, 669
CoinTosser, 93
debugging, 679–680
description, 64, 108
output, 76
NSMainNibFile setting, 338
NSMainQueueConcurrencyType setting, 599
NSManagedObject class, 303–305
photos, 322–324
subclasses, 317–321
NSManagedObjectContext class, 305–306, 601
NSMergeByPropertyObjectTrumpMerge Policy setting, 599
NSMutableArray class, 106
NSMutableData class, 101
NSMutableDictionary class, 106–107
NSMutableOrderedSet class, 209
NSMutableSet class, 107–108
NSMutableString class, 105
NSNotificationCenter class, 418, 595
NSNull class, 104–105
NSNumber class, 89, 104
NSNumberFormatter class, 109
NSObject class, 80, 105
NSOrderedSet class, 203
NSPersistentStoreCoordinator class, 306
NSPersistentStoreDidImportUbiquitous ContentChangesNotification notification, 598–600
NSPersistentStoreUbiquitousContentName Key setting, 592
NSPersistentStoreUbiquitousContentURL Key setting, 592, 594
NSPredicate class, 109
NSPrivateQueueConcurrencyType setting, 599
NSRegularExpression class, 110
NSRunLoopCommonModes mode, 564
NSSet class, 107–108
NSString class, 105
description, 80
literals, 89
notifications, 418
NSStringFromClass function, 411
NSTimer class, 110, 469, 559, 619
NSUInteger class, 103–104
NSURL class, 519, 543
NSURLConnection class
description, 111, 543
Flickr, 558
photos, 559–564
NSURLConnectionDelegate protocol, 543, 564
NSURLRequest class, 111, 543
NSURLResponse class, 557
NSXMLParser class, 110, 543
NSXMLParserDelegate protocol, 110
Null-terminated char array format specifiers, 108
NULL value, 104–105
Number literals, 89
numberOfCells method, 398, 405
numberOfItemsInSelection method, 570
numberOfPhotos method, 449
numberOfRowsInSection method, 207, 240
numberOfSectionsInTableView method, 207
numberOfTapsRequired property, 271
numberOfTouchesRequired property, 271
numberOfVisibleCells method, 398, 405

O

Object library, 12, 27
objectAtIndex method, 255

objectID property, 589
Objective-C, 13, 21–22, 71
  classes, 77–80
  declared properties, 82–84
  dot syntax, 91–92
  implementation, 85–90
  instance variables, 81–82
  interfaces, 80
  literals, 89
  memory management, 93–95
  methods, 85
  objects, 76–80
  overview, 71–72
  selectors, 90–91
  working with, 72–75
Objective-C Programming, 71
ObjectiveFlickr framework, 552
Objects
  aligning, 53–56
  managed, 303–305. *See also* Managed object contexts
  model, 321–327
  overview, 76–80
  selecting and copying, 53
  size, 58–59
OCSP (Online Certificate Status Protocol), 131
OmniGraffle app, 169–171
On/Off button, classes for, 117
Online Certificate Status Protocol (OCSP), 131
Online help, 252
OpenGL Game template, 6
Opening header files, 191
Optimization, 681–686
Option-Click
  copying objects, 53
  documentation popover, 191
  Quick Help popup, 37
@Option key for objects, 54–55
@optional directive, 219
Organization name, 41–42
Organizational Name setting, 7
Organizer window
  app submissions, 705
  code signing assets, 137–138
  description, 43–44
  device setup, 139–141
  UUIDs, 143
Orientation. *See also* Rotation
  launch images, 514
  photo browser, 466
  split view controllers, 179
  supported, 490–492
originalImageData property, 313
Outlets, 61
  checking, 66–67
  connecting objects to, 16–17
  defining, 62–66
  IBOutlet. *See* IBOutlet macro
overlayView property, 547
overlayViewTapped method, 569, 573
Owners of files, 61

P

PADDING macro, 454
Page-Based Application template, 6
Page control, 116
Panning, 270
Paper and pencil for Mockup apps, 167
Parameters
  Flickr, 556–558
  methods, 85
  names, 215
Parsers, 110
Passwords for code signing assets, 137–138
Paths
  attributes, 589
  index, 208, 226
pause method, 621, 683–686
Penultimate app, 167
perform method for segues
  custom, 370
  CustomPushSegue, 464–466
  implementing, 375–381
Performance, 681–686
performBlock method, 599
performBlockAndWait method, 599
performSegueWithIdentifier method, 456
performSelector method, 90, 438–439
Persistence. *See* Data persistence
Persistent stores and persistent store coordinators, 306

creating, 311–312
iCloud, 581–582
ubiquitous, 592–596
persistentStoreCoordinator method, 592–594, 667
Person interface, 82–83
Photo albums, 302
adding, 408–409, 416
adding photos to, 331–334, 434–439
displaying, 406–416
displaying photos in, 334–335
managed object contexts, 325–327
managing, 417
naming, 421–426
reading and saving, 327–331
removing, 428–429
scene rotation, 508–510
selecting, 417–421
thumbnails, 430–434
toolbars, 426–427
Photo browser
chrome effects, 468–473
deleting photos, 480–487
launching, 456–460
push and pop, 460–468
rotating, 510–513
scroll view, 447–456
slideshows, 622–623
user interface, 455–456
zooming, 474–480
Photo class, 321
email, 531
iCloud, 590–591
prototype code, 391–393
Photo entity, 390
PhotoAlbum class, 319–320, 325–327
PhotoAlbum entity, 314–317, 390
PhotoBrowserPhotoView class, 474–478, 510–513
PhotoBrowserViewController class, 447–458
chrome effects, 468–473
Core Image effects, 633, 637–638
deleting photos, 481–487
email, 527, 532–535
printing, 520–523
sharing content, 537–540
slideshows, 610, 614, 622–624
user interface additions, 634–640
zooming, 473, 478–480
Photos, 285
adding to albums, 331–334, 434–439
advancing, 617–619
Assets Library framework, 285–286
custom code for, 321–327
data model, 301
deleting, 480–487
displaying, 334–335, 439–446
downloading, 559–565
iCloud, 592–600
image picker controller. *See* Image picker controllers
saving to camera roll, 297–299
scene rotation, 502–508
photos method, 459–460
photos property, 528
photoSetListWithUserId method, 554
Photoshop, 167–168
PhotosViewCollector class, 442–446
PhotosViewController class
child scenes, 367
container view controllers, 366
email, 527, 532, 535–537
Flickr, 549–551
iCloud, 600
photo albums, naming, 423–426
photo albums, removing, 428–429
photo albums, selecting, 420–421
photo browser, 456–461
photos, adding, 435–439
photos, deleting, 486–487
scene rotation, 505–508
photosWithPhotoSetId method, 554
photosWithSearchString method, 553, 556
photoTapped method, 618
photoViewCache property, 452
PhotoWheel app, 177
app delegates, 184–189
charter, 156–158
collections. *See* Collection views
Core Image effects. *See* Core Image effects
custom views. *See* Custom views
data persistence. *See* Data persistence
debugging. *See* Debugging
detail view controller, 194

PhotoWheel app (*cont'd*)
  device rotation. *See* Rotation
  distributing. *See* Distribution
  email, 527–537
  iCloud. *See* iCloud service
  launch options, 188
  main screen. *See* Main screen
  master view controller, 195
  photo browser. *See* Photo browser
  photos. *See* Photo albums; Photos
  printing, 519–523
  project structure, 183–184
  prototype, 177–183
  sharing content, 537–540
  slideshows. *See* Slideshows
  split view controller, 189–195
  storyboarding. *See* Storyboarding
  table views. *See* Table views
  target audience, 156–158
  touch gestures. *See* Touch gestures
  universal apps. *See* Universal apps
  utility methods, 649–651
  view controllers. *See* View controllers
  Web services. *See* Web services
PhotoWheel-Info.plist file, 346, 349
PhotoWheel-Prefix.pch file, 418, 486, 495, 595
PhotoWheel.xcdatamodeld file, 390
PhotoWheelPrototype app, 181–182, 306
PhotoWheelPrototype-Info.plist file, 184
PhotoWheelPrototype-Prefix.pch file, 184, 308
PhotoWheelPrototype.xcdatamodeld file, 308, 312, 390
PhotoWheelViewCell class, 263
  header files, 263–264
  implementation, 264–265
  photo album thumbnails, 430–434
  touch gestures, 271–273
  working with, 265–267
photoWheelViewCell method, 430–431
Pick lists, 116–117
Pinch gesture, 270
Pipe character, 500
placeholder property, 114, 422
Placeholders
  code completion, 36
  fault objects, 305
  File's Owner, 61
  text, 114
Plain tables, 202
play method, 683–686
PLDatabase project, 335
plist files, 338
Plug-ins, 109
PNG (Portable Network Graphic) format, 118
Pointer address format specifiers, 108
Pointers, 80
Pop segues
  customizing, 381–385
  improving, 460–468
Popover segues, 370
popoverControllerDidDismissPopover method, 538
popToRootViewControllerAnimated method, 371
popToViewController method, 371
popViewControllerAnimated method, 371, 382–383, 466–467, 624
Portable Network Graphic (PNG) format, 118
Portal Resources, 134–135
Portrait orientation, 491
  launch images, 514
  portrait upside down, 491
  split view controllers, 179
Position guides, 53–54
Possible memory leaks, 439
Possible state, 275, 278
Post-It Notes, 153
posterizeFilter method, 641
#pragma mark statements, 192
Pragmatic Programmer: From Journeyman to Master, 247
Predefined touch gestures, 270
Preexisting iCloud data stores, 600–601
Preferences, 29
  code completion, 36
  coding style, 33
  development certificates, 131–132
  fonts and colors, 29–30
  key bindings, 34–35
  text, 30–32
preferredMode property, 607
Premature optimization, 681
prepareForSegue event, 414–415
  Flickr, 551

photo albums, 417
photo browser, 457–458, 463
scenes, 341
slideshows, 622
Presentation property, 370
presentCamera method, 290, 293, 295, 436
Presenters in iCloud, 580–581
presentFlickr method, 550–551
presentPhotoLibrary method
adding photos, 436–437
camera checking, 287–288
image picker controllers, 293, 295
presentPhotoPickerMenu method, 550–551
action sheets, 290–291
adding photos, 437
camera checking, 287–288
Flickr, 548
__PRETTY_FUNCTION__ macro, 222, 271
Price of apps, 700
Primary app categories, 701
Primitive data types, 80
primitiveValueForKey method, 591
Print Center, 518
Print jobs, 518
printCurrentPhoto method, 520, 522
Printer Options view, 517
Printer Simulator, 522–523
printFormatter property, 519
printInfo property, 522
Printing
API, 519
operation, 517–518
PhotoWheel, 519–523
requirements, 519
printingItem property, 519, 522
Private key pairs, 133
Private keys, 136
Product Name setting, 7
Profiling code with Instruments, 681–686
Programming in Objective-C 2.0, 71
Project navigator, 24
Project options screen, 7
Project Summary for universal apps, 656
Project template, 5–6
Projects
creating, 3–11, 180–181
settings, 40–42
Properties
declared, 82–84
dot syntax, 91–92
objects, 76
transient, 586–587
values, 58
@property directive, 63–64, 81–84, 185, 214
Property synthesis, 15
Prototype apps and code, 171–173, 177–178
copying files, 388–390
Core Data model, 390–393
project creation for, 180–181
reusing, 388
simulators, 181–182
split view controllers, 178–179
WheelView, 394–406
Provisioning, 125
Ad Hoc distribution, 690–691
App IDs, 128–129
App Store distribution, 695
development machine setup, 130–138
development provisioning profiles, 129–130
device IDs, 127–128
device setup, 139–141
iCloud service, 582–585
iOS Provisioning Portal. *See* iOS Provisioning Portal
overview, 127
Public key pairs, 133
Public keys, 136
Push segues
description, 370
improving, 460–468
PushPhotoBrowser segue, 456–457
pushViewController method, 371
PW-Default.jpg file, 515
pw_imageSnapshot method, 381
PWDefault-landscape.png file, 515

Q

QuartzCore.h file, 264
queueNonVisibleCells method, 399, 406
queueReusableCells method, 403, 406
Queues, dispatch, 575

Quick Help popup, 37
Quitting apps, 182–183
Quotation marks (") for strings, 105

R

Radians, converting, 282–283
RAND_IN_RANGE macro, 640
Random CIFilter effects, 641–644
Random numbers, 76, 88
randomizeFilters method, 638, 642–643
Rating apps, 701
Reachability, 576–577
Reading photo albums, 327–331
readonly attribute, 83–84
readwrite attribute, 83–84
receivedData property, 563–564
Receivers in Objective-C, 72
Recipes in Core Image, 625–626
Recognized state, 275
Recognizers
  spin gesture, 276–283
  touch gesture, 270–275
Recommended books, 707
Red-eye correction, 631
Reference counting, 8, 74, 94–95, 181
Reflector app, 609
Registering devices, 127
Regular expressions, 110
Rejected apps, 699
Relationships with Core Data entities, 315–316
Release build configurations, 691
reload method, 424–426
reloadData method, 420
  photo albums, 412
  table view data, 224
  WheelView, 395, 404, 406
removeConstraints method, 499
removeFromParentViewController method, 364–365
removePhotosObject method, 319
Removing
  breakpoints, 667
  guides, 54
  photo albums, 428–429
Renaming ivars, 86–87
Rentzsch, Jonathan "Wolf," 321
repositionNavigationBar method, 512–513
Requesting development certificates, 131–134
requireGestureRecognizerToFail method, 273–274
resignFirstResponder method, 426
Resizing
  Cocoa Auto Layout, 492
  labels, 12
  scenes, 367
Resolving conflicts. *See* iCloud service
respondsToSelector method, 219, 661
RESTful Web services
  Cocoa, 542–543
  description, 542
  Flickr, 551
restoreAfterRotation method, 511
resume method, 621
retain attribute, 83–84
Retina display, 346, 349
reusableCells property, 405
Reverse domain name style, 128
revertToOriginal method, 635, 649–650
Review notes, 701
Roles
  iOS Developer Program, 126
  iTunes, 703–704
Root view controllers, 119
rootViewController property, 187–188, 339
rotateToInterfaceOrientation method, 498–501, 506, 509–510
Rotation
  About screen, 510
  customized, 495–502
  gesture type, 270
  launch images, 513–515
  MainViewController, 495–502
  photo browser, 466, 510–513
  scenes, albums, 508–510
  scenes, photos, 502–508
  spin gesture recognizers, 278, 282–283
  split view controllers, 179
  supporting, 489–494
Round Rect Button, 16
Rounding calculations, 103
Routing app coverage files, 702

row property, 202
Run button, 9
Runtime loops, 559, 564

S

Sales role in iTunes Connect, 704
save method in Flickr, 545, 567–568, 573
saveChanges method, 425–426
saveContext method, 309
saveContextAndExit method, 566
saveImage method
  filters, 649–650
  Photo class, 391
  PhotoBrowserViewController, 635
  photos, 322, 324, 333
saveSelectedPhotos method, 566–567, 573
Saving
  photo albums, 327–331
  photos to camera roll, 297–299
scaleAndCropToMaxSize method, 322–323
scaleAspectToMaxSize method, 322
Scaling images, 589–590
Scenes, 339–340
  child, 366–369
  creating, 352–355
  navigating, 371, 374–375
  resizing, 367
  rotation, albums, 508–510
  rotation, photos, 502–508
  setting, 370–375
Scheme manager window, 696
Schemes, 42–43, 696–697
Schneider, Michael, 706
Scope depth of code, 30
Screen
  classes for, 113
  main. *See* Main screen
Screen shots of apps, 702
screenDidConnect method, 617
screenDidDisconnect method, 617
screens array, 606
Scroll view, 447–456
Scrollable views, 115
Scrolling in design, 160
scrollToIndex method, 450, 454
scrollViewDidScroll method, 452, 455
scrollViewWillBeginDragging method, 470–471, 473
Search navigator, 24
searchBarCancelButtonClicked method, 570
searchBarSearchButtonClicked method, 570, 574
searchBarShouldBeginEditing method, 569
searchBarTextDidEndEditing method, 570
Searches
  for data, 109
  text-based, 547
Secondary app categories, 701
section property, 202
Security, 576–577
segmentedControlValueChanged method, 262
Segments, 120–121
Segues, 340–341
  creating, 355–357, 370
  description, 369–370
  implementing, 375–381
  improving, 460–468
  photo browser, 456–457
  scene setting, 370–375
selectedCellFrame method, 466
selectedIndex property, 395
selectedPhotoFrame property, 461
selectedPhotoImage method, 461–462
selectedPhotoIndex property, 460
selectedPhotoWheelViewCell property, 287
selectedWheelViewCellIndex property, 334
Selecting
  objects, 53
  photo albums, 417–421
  table view data, 230–233
@selector directive, 90, 210
Selectors, 90–91
self variable, 88, 90–91
Semicolon characters (;) for declared properties, 83
sendAction method, 457
sendEmail method, 529–531
SendEmailController class
  overview, 527–532
  working with, 532–537
sendEmailController property, 534

SendEmailControllerDelegate protocol, 528–529, 532
sendEmailControllerDidFinish method, 529, 534–535, 537
sender method
  adding photos, 436
  table views, 213
Senders in Objective-C, 72
Sending messages, 108
sepiaImageFromImage method, 629
Seques
  embedded, 368
  unwind, 359–363, 384–385
Serial numbers, 143
Serialization, 48
Session 120 - Simplifying Touch Event Handling with Gesture Recognizers video, 275
setAngle method
  Carousel view, 258–260
  scene rotation, 509
  spin gesture recognizers, 279–280, 282–283
  WheelView, 251, 400, 406
setAngleOffset method, 509
setBounds method, 677
setCurrentIndex method
  external displays, 618
  filters, 650–651
  PhotoBrowserViewController, 452, 454–455
  slideshows, 611–612
setFilterButtons method, 640
setFrame method, 678
setHeadsCount method, 91
setImage method
  photo album thumbnails, 434
  PhotoWheelViewCell, 263–264
  zooming, 475, 477–478
setImageData method, 589
setLargeImageData method, 589
setLastResult method, 90–91
setManagedObjectContext method, 312
setPhotoAlbum method, 332, 334–335
setPrimitiveValue method, 590–591
sets, 107–108
setScrollViewContentSize method, 450, 453
setSmallImageData method, 589
setStyle method
  Carousel view, 258
  spin gesture recognizers, 281
  WheelView, 403, 406
setter attribute, 83–84
Setter methods, 64, 82
setText method, 434
setThumbnailImageData method, 589
setTitleWithCurrentIndex method, 450, 454
sharedApplication method, 415
Sharing schemes, 42
Shortcut keys
  key bindings, 34–35
  navigator, 25
shouldAutomaticallyForwardAppearance Methods method, 365
shouldAutomaticallyForwardRotation Methods method, 365
shouldAutorotate method, 613
shouldAutorotateToInterfaceOrientation method
  autosizing, 492
  overriding, 489
Show Obj-C Only option, 685
showActionMenu method, 423
  deleting photos, 484
  email, 532–533, 535–536
  printing, 520, 522
  removing photo albums, 428
  sharing content, 538–539
showFilters method, 637–638, 642
showFromBarButtonItem method, 292
showFromRect method, 292
showFromTabBar method, 292
showFromToolbar method, 292
showOverlay method, 568, 573
showOverlayCount property, 572
Shows Navigation Bar property, 375
Signed integer format specifiers, 108
SimpleFlickrAPI class, 552–558
Simulators, 44
  iOS Simulator, 522–523, 675
  iPad Simulator. *See* iPad Simulator
  Printer Simulator, 522–523
  schemes, 42
Single inheritance, 185
Single View Application template, 6–7
Size and Size Inspector
  Cocoa Auto Layout, 492
  collection view cells, 439–440

collection views, 242
description, 58–59
labels, 12
objects, 58–59
photo display, 439
scenes, 355, 367
text fields, 220
wheel view, 252–253
SKU numbers, 700
slideAdvanceTimer variable, 614
Sliders, 163
slideshow method, 484
Slideshows
adding, 609–610
displaying, 610–613
external display. *See* External display
finishing, 623–624
photo advancing, 617–619
photo browser, 622–623
profiling example, 683–686
storyboards, 610
user interface controls, 620–622, 683–686
SlideShowViewController class, 609–611, 675, 677
Slow motion animation, 379
smallImage attribute, 589
smallImage method, 392–393
smallImageData attribute, 391, 586, 591
Smalltalk language, 72
SMS messages, 525
Snapshot feature, 503
SOAP-based Web services, 541–542
Sort descriptor for photo albums, 412
Sorting filter buttons, 639–640
Sound effects, 162
Source code repositories, 8, 40, 74, 181
Source image view for segues, 379
sourceViewController property, 460
Spaces vs. tabs, 32
Spin gesture recognizers
creating, 276–278
working with, 278–283
spin method
spin gesture recognizers, 281–283
WheelView, 403, 406
SpinGestureRecognizer class, 276–278, 281–283
Split view controllers
delegates, 192–194
overview, 178–179
working with, 189–192
SQLite, 303, 306, 311, 335
stack-add.png file, 351, 368
stack-add-down.png file, 351
stack-bg.png file, 351, 368
stack-overlay.png file, 432
stack-viewer-bg-portrait.png file, 351–352, 366–367
stack-viewer-shadow.png image, 442
Stacks
Cocoa, 97
Core Data, 308–312
Standard Company program type, 710
Standard editor, 38, 51
Standard Individual program type, 710
startAtIndex property, 447, 458–459
startChromeDisplayTimer method, 471–473
startImmediately property, 563
startIndex property, 623
States
gesture recognizers, 275–276
objects, 57–61
Static sets, 107
statusBarHeight property, 472
Stencils, 171
Step into button, 672
Step out button, 672
Step over button, 672
Stopping apps, 11
Storyboarding, 8, 70
app icon, 349–350
AppDelegate, 347–348
Flickr, 547–549
images, 348–349
main, 344–345
overview, 337–338
scenes, 339–340, 352–355
segues, 340–341, 355–357
slideshows, 610
UIMainStoryboardFile setting, 346–347
universal apps, 655–656
view controllers. *See* View controllers
working with, 338–339
workspace, 341–344

stringByRemovingFlickrJavaScript method, 555, 558
Strings
 arrays, 203
 classes, 105
 converting to dates, 109
 format specifiers, 108
 literals, 88–89
stringWithData method, 555, 558
strong attribute, 83, 94
Style property
 bar buttons, 120
 Carousel view, 256–257
 segues, 370
Styles, coding, 33
Subclass generation, 321–322
Submitting apps, 11, 705–706
Subversion source code repositories, 40, 74
Summaries for apps, 152–153
super keyword, 88
supportedInterfaceOrientations method, 489
Swipe gesture, 270
Symbol navigator, 24
Syncing. *See* iCloud service
@synthesize directive, 86–87, 91
 AppDelegate, 187
 data instances, 204
 description, 64

T

Tab bar classes, 120
Tab key, 32
Tabbed Application template, 6
Table views, 197-200
 adding data, 210–224
 classes, 115–116, 201–202
 deleting data, 228–229
 displaying data, 206–210
 editing data, 224–228
 freeform, 220
 reordering data, 229–230
 selecting data, 230–233
 simple models, 203–206
 working with, 203
Tagged Image File Format (TIFF) format, 118
Tap gestures, 270
 Flickr, 573
 PhotoWheelViewCell, 271–273
 zooming, 478
tapped method, 477
Tapworthy apps, 158–159
Tapworthy: Designing Great iPhone Apps, 158
Target-Action pattern, 122
Target audience for apps, 154–158
Targeted Device Family setting, 655
Targets
 settings, 40–41
 types, 5–6
Team Admins, 126, 135, 141
Team Agents, 126, 141, 144–145
Team Members, 126, 141
Team roles in iOS Developer Program, 126
Technical role in iTunes Connect, 704
Templates, 5–6
Text
 classes for, 113–114
 labels. *See* Labels
 preferences, 30–32
 on screen, 11–13
Text-based searches, 547
textFieldDidEndEditing method, 425–426
textFieldShouldBeginEditing method, 425–426
textFieldShouldReturn method, 425–426
Third-party apps for photos, 285
Threads
 atomic properties, 83
 GCD, 575
ThumbnailCell class, 440–441
thumbnailImage property, 589
thumbnailImageData property, 314, 586, 591
Thumbnails for photo albums, 430–434
TIFF (Tagged Image File Format) format, 118
Tilde (~) naming convention, 657–658
Time
 classes, 101–102
 formatting, 109
 pick lists, 117
Time Machine, 136
Time Profiler tool, 681
Timers
 chrome effects, 469
 classes, 110

slideshow photos, 619
Titles, 197–200
titleView property, 261
toggleChrome method, 471, 473
toggleChromeDisplay method
chrome effects, 471, 473
zooming, 478–480
Toll-free bridging, 100
Toolbars
classes for, 119
photo albums, 426–427
PhotoWheel, 636, 639
scenes, 352
slideshows, 621
workspace window, 22–24
Top Space to Superview option, 504
Touch gestures
custom, 275–283
in design, 159
events, 269
overview, 269–270
predefined, 270
recognizers, 271–275
types, 270–271
zooming, 478
Touch Up Inside events, 64
touchesBegan method, 269–270, 275–277
touchesCancelled method, 275–277
touchesEnd method, 276–278
touchesEnded method, 269–270, 275, 277–278
touchesMoved method, 269, 275–278
TouchJSON library, 543
TouchXML parser, 110
Trailing Space to Superview option, 504
Transient Core Data attributes, 586–587
Transition property for segues, 370
transitionFromViewController method, 364
Transitions
pop, 381–385, 460–468
scenes, 371, 374–375
segues, 340–341
turnOffZoom method, 476–477
twirlFilter method, 641

U

Ubiquitous content. *See* iCloud service
Ubiquitous persistent store coordinators, 592–596
Ubiquitous persistent stores, 581–582
ubiquityIdentityToken method, 602
UDIDs (Unique Device Identifiers), 128–129, 143
UI. *See* User interface (UI)
UI_USER_INTERFACE_IDIOM macro, 656–657, 661
UIActionSheet class, 289, 292
UIActionSheetDelegate protocol, 291
UIActivityIndicatorView class, 547
UIActivityViewController class
email, 525
sharing content, 537–540
UIAlertView view, 428
UIApplication class, 113, 415
UIApplicationDelegate protocol, 185–188
UIApplicationMain function, 184
UIBarButtonItem class, 120, 190, 422
profiling example, 686
slideshows, 620–621
UIBarButtonSystemItemAdd button, 210
UIButton class, 27, 114–115
UICollectionDataSource protocol, 445
UICollectionReusableView class, 236
UICollectionView class, 236, 439
UICollectionViewCell class, 236, 239, 242–243
UICollectionViewController class, 236–237
UICollectionViewDataSource class, 236, 240–241
UICollectionViewDelegate protocol, 236–237, 445
UICollectionViewDelegateFlowLayout class, 236, 240–241
UICollectionViewFlowLayout class, 236
UICollectionViewLayout class, 236
UICollectionViewLayoutAttributes class, 236
UICollectionViewUpdateItem class, 236
UIDatePicker class, 117
UIDocument class, 581
UIGestureRecognizer class, 270–271, 275–276
UIGestureRecognizerDelegate protocol, 271
UIGestureRecognizerStateBegan state, 275
UIGestureRecognizerStateCancelled state, 275

UIGestureRecognizerStateChanged state, 275
UIGestureRecognizerStateEnd state, 275
UIGestureRecognizerStateFailed state, 275
UIGestureRecognizerStatePossible state, 275
UIGestureRecognizerStateRecognized state, 275
UIGestureRecognizerSubclass.h file, 275–276
UIImage class
  conversions with, 324–325
  email, 531
  with filters, 628–631
  format support, 117
  model objects, 321–322
  slideshows, 613
UIImagePickerController class
  adding photos, 438
  working with, 286–289, 293–297
UIImagePickerControllerDelegate protocol
  adding photos, 286, 333, 435
  image picker controllers, 293, 296
UIImagePickerControllerSourceTypePhoto Library source type, 295
UIImagePickerControllerSourceTypeSaved PhotoAlbum source type, 295
UIImageView class
  description, 118
  photo album thumbnails, 431–432
  PhotoWheelViewCell, 263
  storyboards, 350–352
  wheel view, 248
  zooming, 474, 480
UIImageWriteToSavedPhotosAlbum function, 297–298
UIInterfaceOrientationIsLandscape macro, 500, 502
UIInterfaceOrientationIsPortrait macro, 502
UIKit classes, 99, 112–121
  importing, 250
  Printing API, 519
UIKit Framework document, 37
UILabel class, 12, 15
  collection views, 243, 245
  description, 113
  photo album thumbnails, 432–433
UILongPressGestureRecognizer gesture, 270–271
UIMainStoryboardFile setting, 346–347
UIManagedDocument class, 581
UIMarkupTextPrintFormatter class, 519
UIMenuController class, 117
UIMenuItem class, 117
UINavigationBar class, 119
UINavigationController class
  description, 119
  master detail apps, 187, 194–195
  pop transitions, 381–383
  view controllers, 364, 371
UINavigationControllerDelegate protocol, 293, 435
UIPageControl class, 116
UIPanGestureRecognizer gesture, 270–271
UIPickerView class, 116
UIPinchGestureRecognizer gesture, 270
UIPopoverController class, 190–191, 199, 661
UIPrintFormatter class, 519
UIPrintInfo class, 519, 522
UIPrintInfoOutputPhoto setting, 522
UIPrintInteractionController class, 519, 522
UIPrintInteractionControllerDelegate protocol, 519
UIPrintPageRenderer class, 519
UIPrintPaper class, 519
UIResponder class, 269
UIRotationGestureRecognizer gesture, 270–271, 276, 283
UIScreen class, 113, 606–607
UIScreenDidConnectNotification notification, 607–608, 617
UIScreenDidDisconnectNotification notification, 607–608, 617
UIScrollView class, 201
  description, 115
  photo browser, 447
  zooming, 474
UIScrollViewDelegate protocol, 447
UISearchBar class, 547
UISegmentedControl class, 120–121
UISimpleTextPrintFormatter class, 519
UISlider class, 117, 163
UISplitViewController class
  container view controllers, 364
  iPad Simulator, 182

master-detail apps, 178–179, 187
methods implementation, 192
overview, 189–192
UISplitViewControllerDelegate protocol, 190
UIStoryboardSegue class, 370, 376
UISwipeGestureRecognizer gesture, 270
UISwitch class, 117
UITabBar class, 120
UITabBarController class, 364
UITableView class
description, 115–116
displaying data, 206
editing data, 224
overview, 201–202
reordering data, 229
UITableViewCell class
description, 115, 202
styles, 208–209
UITableViewCellStyleDefault style, 208–209
UITableViewCellStyleSubtitle style, 208
UITableViewCellStyleValue1 style, 208
UITableViewCellStyleValue2 style, 208
UITableViewController class, 195, 202, 224–225
UITableViewDataSource class, 203
description, 202
displaying data, 206–207
editing data, 224
moving rows, 230
UITableViewDelegate protocol, 202–203, 214
UITableViewRowAnimationFade class, 228
UITapGestureRecognizer gesture, 270–271
UITextField class
description, 114
photo albums, 422
table view data, 212–213, 220
UITextFieldDelegate protocol, 424–426
UITextView class, 114
UIToolbar class
description, 119
photo albums, 422
UIUserInterfaceIdiomPad value, 656
UIUserInterfaceIdiomPhone value, 656
UIView class
custom views, 246
description, 113
events, 269
Flickr, 548
photo albums, 407
slideshows, 612
wheel view, 248, 252
UIViewController class
container view controllers, 364–366
description, 113
storyboards, 359–360
UIViewPrintFormatter class, 519
UIView+PWCategory class, 380–381
UIWebView class, 113
UIWindow class, 607
debugging example, 677
description, 113
external displays, 616
master-detail apps, 187
"Unable to initiate item download" message, 582
Underscores (_) for ivars, 187
Unicode characters, 105
Unique Device Identifiers (UDIDs), 128–129, 143
Unique value propositions, 152–153
Universal apps, 9, 653
benefits, 653–654
code separation, 656–657
disadvantages, 654–655
pitfalls, 659–661
storyboards, 655–656
target setting, 655
tilde naming convention, 657–658
Universal device family, 8
University program type, 710
unloadPage method, 451–452, 454, 479–480
Unsigned integer format specifiers, 108
Unwind seques, 359–363, 384–385
updateNavBarButtonsForPlayingState method, 620–621, 684–685
updateviewConstraints method
rotation, 497, 499
scene rotation, 505–507
updateviewConstraintsForInterface Orientation method
rotation, 497–498, 501
scene rotation, 506
URI string hashes, 589

URLForUbiquityContainerIdentifier
method, 594
URLs
apps, 702
Flickr, 556–558
Use Automatic Reference Counting option,
8, 74, 181
Use for Development option, 139
Use Storyboard option, 339
User input, 13
User interface (UI)
controls, 162–163
designers for, 164
device design, 159–160
HIG, 158
industrial design, 160
metaphors, 160–162
Photo browser, 455–456
PhotoWheel, 634–640
rotation, 501–502
slideshow controls, 620–622
sound effects, 162
tapworthy apps, 158–159
User roles in iTunes Connect, 704
userIdForUsername method, 553–554
userInteractionEnabled flag, 473
Utility Application template, 6
Utility area, 26–27
Utility classes and functions, 108–112

V

Variables, 49
inspecting, 667, 672–674
Objective-C, 81–82
renaming, 86–87
Version editor, 40
Versions
apps, 701
Xcode, 714
Vertical guides, 53–54
View controllers
container. *See* Container view controllers
detail, 179. *See also* DetailViewController
class
Flickr, 547–548
iCloud, 595–596, 600
implementing, 359–363
initial, 350–352
master, 179, 195. *See also*
MasterViewController class
pop transitions, 381–385
segues, 369–381
split, 178–179, 189–195
ViewController class, 12, 14–15
viewController property, 528
viewDidAppear event, 365, 623
viewDidDisappear event, 365
viewDidLoad method
Carousel view, 261–262
chrome effects, 469, 472
collections, 238
data instances, 204–205
deleting photos, 481–483, 485–486
external displays, 616–617
Flickr, 566
iCloud, 595
photo albums, 327–328, 330, 332,
420–421, 427
PhotoBrowserViewController, 448–449,
453
PhotoWheelViewCell, 265–266
table view data, 216–218, 224–225, 227
titles, 197–198
touch gestures, 272–274
wheel view, 254, 394
viewForZoomingInScrollView method, 477
Views
carousel, 256–262
collection. *See* Collection views
custom, 246–248
dictionaries, 499, 507
photo wheel view cell, 263–267
table. *See* Table views
wheel. *See* WheelView class
viewWillAppear event
chrome effects, 469–470, 472
container view controllers, 365
customized rotation, 497, 499–500
navigation bar, 375
PhotoBrowserViewController, 449, 453,
623
PhotoWheel, 637
scene rotation, 505, 507, 509

slideshows, 618–619
viewWillDisappear event, 365, 470, 473, 619
viewWithTag method, 574
Virtual keyboards, 162
visibleCellIndexes property, 405
Visual effects. *See* Core Image effects
Visual Format Language, 493–494

W

wantsFullScreenLayout flag, 453
Watchpoints, 667
weak attribute, 83, 94
Web services, 541
basics, 541–542
concurrent programming, 576–577
Flickr. *See* Flickr
RESTful, 542–543
WebKit Coding Style Guidelines, 33
Welcome to Xcode screen, 4–5, 72–73
Wenderlich, Ray, 110
Wheeler, Colin, 35
WheelView class
Carousel view, 256–262
creating, 249
declaring, 250
defining, 250
header file, 249–250
implementation, 251–256
photo albums, 407
prototype code, 394–406
spin gesture recognizers, 278
WheelViewCell class, 250, 263
defining, 250
prototype code, 394, 404
WheelViewDataSource protocol, 250, 253–255, 395
WheelViewDelegate protocol, 395
wheelViewNumberOfCells method, 255, 413
wheelViewNumberOfVisibleCells method, 395, 410, 412
Wildcard characters (*)
App IDs, 129, 139, 145, 582
Bundle Identifiers, 128
willAccessValueForKey method, 591
willAnimateRotationToInterfaceOrientation method
overriding, 491
PhotoBrowserViewController, 511–513
willChangeValueForKey method, 590
willHideViewController method, 190
willMoveToParentViewController method, 364–365
willPresentViewController method, 191
willRotateToInterfaceOrientation method
action sheets, 290, 292
overriding, 490–491
PhotoBrowserViewController, 511–512
rotation, 498, 500
scene rotation, 506–507, 509–510
willShowViewController method, 191
Windows, classes for, 113
Windows Bitmap Format (DIB) format, 118
Windows Cursor format, 118
Windows Icon Format, 118
Wireframe mockups, 164–165, 169–171
Wooldridge, Dave, 706
Workspace window, 24, 52
Debug area, 28
design, 28–29
Editor area, 25
Navigation area, 24–25
Toolbar area, 22–24
Utility area, 26–27
Workspaces, creating, 341–344
Wrapping feature, 394
Wrapping Flickr API, 551–558
WWDR intermediate certificates, 136

X

.xbm files, 118
.xcdatamodeld extension, 308–310
Xcode, 21
debugging, 667–674
developer documentation, 37
editors, 38–40
IDE, 21–22
installing, 714–715
in Launchpad, 3–4
organizer, 43–44
Preferences, 29–36

Xcode (*cont'd*)
- project settings, 40–42
- schemes, 42
- tools, 44–45
- Workspace window, 22–29

Xcode 4 User Guide, 4
.xib files, 11–12, 48
XML with Flickr, 556
XWindow bitmap format, 118

Y

Yacktman, Donald A., 121

Z

Zarra, Marcus, 103, 258
Zarra Studios Coding Style Guide, 33
Zooming
- faces, 647–649
- photos, 474–480

zoomRectForScale method, 476, 478
zoomToFaces method, 647–649
zoomToLocation method, 476, 478