

LEARNING iPad
PROGRAMMING

A Hands-On Guide to Building iPad Apps with iOS 5



KIRBY TURNER
TOM HARRINGTON

Foreword by Mark Dalrymple

Praise for 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 Burlison
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

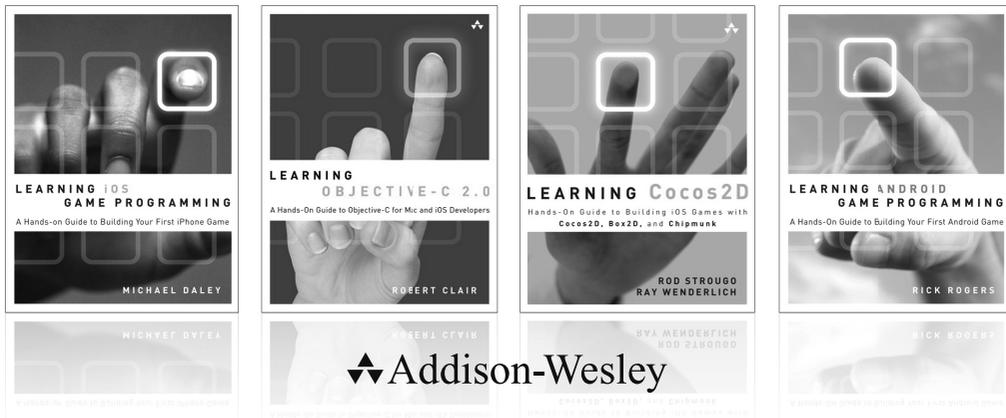
“The iPad is changing the way we think about and use technology. *Learning iPad Programming* is one of the most in-depth and well-executed guides to get both new and seasoned developers up to speed on Apple’s exciting new platform.”

—Justin Williams

Crew Chief, Second Gear

Learning iPad Programming

Addison-Wesley Learning Series



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.com

Safari
Books Online

Learning iPad Programming

A Hands-On Guide to Building
iPad Apps with iOS 5

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 with iOS 5 / Kirby Turner, Tom Harrington.

p. cm.

Includes index.

ISBN 978-0-321-75040-2 (pbk. : alk. paper)

1. iPad (Computer)—Programming. 2. Application software—Development. 3. Mobile computing. 4. Laptop computers. 5. Macintosh (Computer) 6. iOS (Electronic resource) I. Harrington, Tom. II. Title.

QA76.8.I863T87 2012

005.258—dc23

2011042203

Copyright © 2012 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: 978-0-321-75040-2

ISBN-10: 0-321-75040-3

Text printed in the United States on recycled paper at Edwards Brothers in Ann Arbor, Michigan.

First printing, December 2011

Editor-in-Chief

Mark Taub

Senior Acquisitions Editor

Chuck Toporek

Development Editor

Chuck Toporek

Managing Editor

John Fuller

Project Editor

Anna Popick

Copy Editor

Barbara Wood

Indexer

Ted Laux

Proofreader

Linda Begley

Technical

Reviewers

Patrick Burseson

Matt Martel

Erik Price

Mike Shields

Publishing

Coordinator

Olivia Basegio

Cover Designer

Chuti Prasertsith

Compositor

Rob Mauhar



*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, the person who made me who I am today.
— Kirby Turner*

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



Contents at a Glance

Foreword	xxv
Preface	xxix
Acknowledgments	xliii
About the Authors	xlv

I Getting Started 1

1	Your First App	3
2	Getting Started with Xcode	19
3	Getting Started with Interface Builder	43
4	Getting Started with Objective-C	65
5	Getting Started with Cocoa	89
6	Provisioning Your iPad	115
7	App Design	141

II Building PhotoWheel 165

8	Creating a Master-Detail App	167
9	Using Table Views	189
10	Working with Views	231
11	Using Touch Gestures	253
12	Adding Photos	269
13	Data Persistence	285
14	Storyboarding in Xcode	329
15	Doing More with View Controllers	351
16	Building the Main Screen	377
17	Creating a Photo Browser	457
18	Supporting Device Rotation	499

19	Printing with AirPrint	525
20	Sending Email	533
21	Web Services	547
22	Syncing with iCloud	583
23	Producing a Slideshow with AirPlay	609
24	Visual Effects with Core Image	631

III The Finishing Touches 659

25	Debugging	661
26	Distributing Your App	683
27	The Final Word	701
A	Installing the Developer Tools	703
	Index	711

Contents

Foreword	xxv
Preface	xxix
Acknowledgments	xlili
About the Authors	xlv

I Getting Started 1

1 Your First App 3

Creating the Hello World Project	3
Getting Text on the Screen	10
Say Hello	12
Summary	17

2 Getting Started with Xcode 19

The IDE	19
Workspace Window	20
Toolbar Area	20
Navigation Area	22
Editor Area	23
Utility Area	24
Debug Area	25
Preferences	26
Fonts and Colors	26
Text Editing	27
Key Bindings Preferences	31
Code Completion	33
Developer Documentation	34
Editors	35
Project Settings	36
Schemes	39
Organizer	40
Other Xcode Tools	41
Summary	41

3	Getting Started with Interface Builder	43
	Interface Builder	43
	How Does IB Work?	44
	Getting Hands-On with IB	45
	Selecting and Copying Objects	48
	Aligning Objects	49
	Layout Rectangle	52
	Changing State	52
	Connecting Your NIB to Your Code	57
	Defining an Outlet in Code	58
	Using the Assistant Editor	61
	Storyboards	63
	Summary	64
4	Getting Started with Objective-C	65
	What Is Objective-C?	65
	Hands-On with Objective-C	66
	Let's Write Some Code	69
	Object	70
	Class	71
	NSObject	73
	Interface	74
	Instance Variables	74
	Declared Properties	75
	Methods	78
	Implementation	78
	Synthesize	80
	init	80
	super	81
	flip	81
	Selector	82
	Dot Syntax	83
	Using the <code>CoinTosser</code> Class	84
	Memory Management	85
	Automatic Reference Counting	86
	Summary	87

5	Getting Started with Cocoa	89
	The Cocoa Stack	89
	Foundation	91
	Data Type	92
	Collection Classes	97
	Utility Classes and Functions	99
	UIKit	103
	UIApplication	103
	UIWindow	104
	UIScreen	104
	UIView	104
	UIViewController	104
	UIWebView	104
	UILabel	104
	UITextField	104
	UITextView	105
	UIButton	105
	UITableView and UITableViewCell	106
	UIScrollView	107
	UIPageControl	107
	UIPickerView	107
	UIDatePicker	107
	UISwitch	108
	UISlider	108
	UIMenuController and UIMenuItem	108
	UIImage	108
	UIImageView	108
	UINavigationController	109
	UIToolbar	110
	UITabBar	110
	UIBarButtonItem	111
	UISegmentedControl	111
	Common Design Patterns in Cocoa	112
	Model-View-Controller	112
	Target-Action	113
	Summary	113

6 Provisioning Your iPad	115
About the iOS Provisioning Portal	115
The Provisioning Process: A Brief Overview	117
What Is a Device ID?	117
What Is an App ID?	118
What Is a Development Provisioning Profile?	119
Setting Up Your Development Machine	121
Requesting a Development Certificate	121
Submit Your CSR for Approval	124
Download and Install Your Certificate	126
Setting Up Your Device	128
Use for Development	128
Using the iOS Provisioning Portal	131
Adding a Device ID	131
Adding an App ID	133
Creating a Development Provisioning Profile	135
Downloading a Development Provisioning Profile	137
Installing a Development Provisioning Profile	137
Summary	139
7 App Design	141
Defining Your App	141
App Name	142
App Summary	142
Feature List	143
Target Audience	144
Revisit Your Feature List	145
Competing Products	145
A Sample App Charter	146
UI Design Considerations	148
Read the HIG	148
Make Your App “Tapworthy”	148
Design for the Device	148
People Use iOS Devices Differently from the Web or Desktop	149
Wear Your Industrial Designer Hat	149

Metaphors	150
Sound Effects	151
Customize Existing Controls	152
Hire a Designer	153
Mockups	154
What Is a Mockup?	154
What to Mock Up	156
Tools to Use	156
Prototyping	160
What Is a Prototype?	161
How to Create a Prototype	162
Summary	163

II Building PhotoWheel 165

8 Creating a Master-Detail App	167
Building a Prototype App	167
What Is the Split View Controller?	168
Create a New Project	170
Using the Simulator	171
A Closer Look	173
Project Structure	173
App Delegate	174
Launch Options	179
Other UIApplicationDelegate Methods	180
A Tour of UISplitViewController	181
Assigning the Split View Controller Delegate	185
Detail View Controller	185
Master View Controller	186
Summary	187
Exercises	187
9 Using Table Views	189
First Things First	189
A Closer Look	193
UITableView	193
UITableViewCell	194

UITableViewDelegate	194
UITableViewDataSource	194
UITableViewController	194
Working with a Table View	194
A Simple Model	195
Display Data	197
Add Data	203
Edit Data	220
Delete Data	225
Reorder Data	226
Select Data	227
Summary	230
Exercises	230
10 Working with Views	231
Custom Views	231
View Controller Not	232
A Wheel View	233
A Carousel View	240
A Photo Wheel View Cell	248
Using PhotoWheelViewCell	250
Summary	252
Exercises	252
11 Using Touch Gestures	253
Touch Gestures Explained	253
Predefined Touch Gestures	254
Gesture Types	254
How to Use Gesture Recognizers	255
Custom Touch Gestures	258
Creating a Spin Gesture Recognizer	259
Using the Spin Gesture Recognizer	262
Summary	266
Exercises	267
12 Adding Photos	269
Two Approaches	269
Assets Library	269

Image Picker Controller	271
Using the Image Picker Controller	271
Using Action Sheets	274
Using UIImagePickerController	278
Saving to the Camera Roll	283
Summary	284
Exercises	284

13 Data Persistence 285

The Data Model	285
Photos	285
Photo Albums	286
Thinking Ahead	286
Building the Model with Property Lists	286
What Is a Property List?	286
Setting Up the Data Model	287
Reading and Saving Photo Albums	288
Adding New Photos to an Album	293
Displaying Photos in an Album	297
Building the Model with Core Data	298
What Is Core Data?	299
Managed Objects and Entity Descriptions	299
Managed Object Contexts	300
Persistent Stores and Persistent Store Coordinators	301
Adding Core Data to PhotoWheelPrototype	302
Adding the Core Data Framework	302
Setting Up the Core Data Stack	303
Using Core Data in PhotoWheel	307
The Core Data Model Editor	307
Adding the Entities	308
Creating NSManagedObject Subclasses	310
Adding Custom Code to Model Objects	315
Reading and Saving Photo Albums with Core Data	320
Adding New Photos to an Album with Core Data	323
Displaying Photos in an Album with Core Data	326

Using SQLite Directly	326
Summary	327
Exercises	327
14 Storyboarding in Xcode	329
What Is a Storyboard?	329
Using a Storyboard	330
Scenes	331
Segues	332
Storyboarding PhotoWheel	333
Workspace	333
Add the Main Storyboard	336
Set <code>UIMainStoryboardFile</code>	338
Update <code>AppDelegate</code>	339
Add Images	339
App Icon	341
Initial View Controller	341
Another Scene	344
Creating a Segue	346
Summary	349
Exercises	349
15 Doing More with View Controllers	351
Implementing a View Controller	351
Segue	355
Creating a Custom Segue	355
Setting the Scene	355
Implementing a Custom Segue	358
Before You Compile	362
Customizing the Pop Transitions	364
Container View Controller	367
Create a Container View Controller	369
Add the Child Scenes	369
Add Child View Controllers	371
Fix the Custom Push Segue	375
Summary	376
Exercises	376

16 Building the Main Screen	377
Reusing Prototype Code	378
Copy Files	378
Core Data Model	380
Changes to WheelView	385
Displaying Photo Albums	398
Implementing the Photo Albums View Controller	400
Setting the Managed Object Context	406
Adding Photo Albums	408
Managing Photo Albums	409
Selecting the Photo Album	410
Naming the Photo Album	414
Fixing the Toolbar Display	421
Removing the Photo Album	422
A Better Photo Album Thumbnail	425
Adding Photos	429
Displaying Photos	434
Using the GridView Class	446
Building the Image Grid View Cell	451
Summary	455
Exercises	455
17 Creating a Photo Browser	457
Using the Scroll View	457
Setting Up the Photo Browser UI	466
Launching the Photo Browser	467
Improving the Push and Pop	470
Adding Chrome Effects	477
Zooming	482
Deleting a Photo	489
Summary	498
Exercise	498
18 Supporting Device Rotation	499
How to Support Rotation	499
Supported Orientations	500
Using Autoresizing	501

Customized Rotation	502
Rotating the Photo Albums Scene	507
Rotating the Photo Album Scene	508
Tweaking the <code>WheelView</code> Class	509
Rotating the About View	511
Rotating the Photo Browser	511
Fixing the Trouble Spots	511
Fixing the Photo Browser	511
Fixing the Main Screen	518
Launch Images	520
Summary	523
Exercises	523
19 Printing with AirPrint	525
How Printing Works	525
Print Center	526
Requirements for Printing	526
Printing API	527
Adding Printing to PhotoWheel	527
The Printer Simulator	530
Summary	531
Exercises	532
20 Sending Email	533
How It Works	533
The <code>MFMailComposeViewController</code> Class	535
The <code>SendEmailController</code> Class	535
Introducing the <code>SendEmailController</code> Class	536
Using <code>SendEmailController</code>	540
Summary	546
Exercises	546
21 Web Services	547
The Basics	547
RESTful Web Services Using Cocoa	548
Flickr	549
Adding Flickr to PhotoWheel	551

Updating the Flickr View Controller Scene	553
Displaying the Flickr Scene	555
Wrapping the Flickr API	557
Downloading Photos Asynchronously	564
Implementing <code>FlickrViewController</code>	570
One More Thing	580
What's Missing	582
Summary	582
Exercises	582
22 Syncing with iCloud	583
Syncing Made Simple	583
iCloud Concepts	584
File Coordinators and Presenters	584
<i>UIDocument</i> and <i>UIManagedDocument</i>	585
Ubiquitous Persistent Stores	585
Device Provisioning, Revisited	586
Configuring the App ID	586
Provisioning for iCloud	588
Configuring iCloud Entitlements	589
iCloud Considerations for PhotoWheel	592
Don't Sync More Than You Need to Sync	592
Using Transient Core Data Attributes	592
Updating PhotoWheel for iCloud	593
Syncing Photos with iCloud	598
Making the Persistent Store Coordinator Ubiquitous	598
Receiving Changes from iCloud	602
Summary	607
Exercises	607
23 Producing a Slideshow with AirPlay	609
External Display Options	609
App Requirements for External Displays	609
External Display API	610
Adding a Slideshow to PhotoWheel	611
Updating the Storyboard	612
Adding the Slideshow Display	613

Managing External Displays	616
Advancing to the Next Photo	620
Adding Slideshow User Interface Controls	622
Updating the Photo Browser	624
A Note on Testing and Debugging	625
Adding AirPlay Support	626
Using AirPlay	628
Summary	629
Exercises	629
24 Visual Effects with Core Image	631
Core Image Concepts	631
Introducing <code>CIFilter</code>	633
Filter Types	634
Using <code>CIFilter</code>	634
Image Analysis	636
Automatic Enhancement	636
Face Detection	637
Adding Core Image Effects to PhotoWheel	638
New Delegate Methods	638
Instance Variables for Filter Management	640
User Interface Additions	640
Creating the <code>CIFilter</code> Effects	647
Applying the Filters	651
Implementing Auto-Enhance	652
Implementing Face Zoom	653
Other Necessary Methods	655
Summary	656
Exercises	656
III The Finishing Touches	659
25 Debugging	661
Understand the Problem	661
What Went Wrong?	661
Reproducing Bugs	661

Debugging Concepts	662
Breakpoints	662
Debugging in Xcode	663
Setting and Managing Breakpoints	663
Customizing Breakpoints	664
Hitting a Breakpoint	666
Checking on Variables	667
Debugging Example: External Display Code	670
When You Really Need NSLog	674
Profiling Code with Instruments	676
Profiling Example: Slideshow UI Control Updates	679
Summary	682
26 Distributing Your App	683
Distribution Methods	683
Building for Ad Hoc Distribution	684
Provisioning for Ad Hoc Distribution	684
Prepare the (Ad Hoc) Build!	684
Building for App Store Distribution	688
Provisioning for the App Store	688
Prepare the (App Store) Build!	689
Next Steps	691
The App Store Process	691
What If Apple Rejects the App?	692
App Information for the App Store	692
App Store Assets	694
Using iTunes Connect	695
User Roles	696
Managing Applications	696
Submitting the App	696
Going Further	698
Summary	699
27 The Final Word	701
What's Next	702

A	Installing the Developer Tools	703
	Membership Has Its Privileges	703
	Joining the iOS Developer Program	704
	Which Program Type Is Right for You?	704
	What You Need to Register	706
	Downloading Xcode	708
	Installing Xcode	708
	Index	711

Foreword

Aren't books great?

Anyone who's known me for any amount of time knows I'm a total bookworm. I love books. Well-written books are one of the cheapest and fastest tools for self-education. 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 Chericoff's early Mac programming books; Dave Mark's C programming books; Robert C. Martin's horribly titled (but wonderfully full of aha! moments) *Designing Object Oriented C++ Applications Using the Booch Method*; and of course the late W. Richard Stevens's UNIX and network programming books. I remember lessons learned from these tomes, even those I read over 25 years ago.

Unfortunately not all books are created equal. I've seen some real stinkers in my time. When I first made the transition from Mac programming to iPhone programming, some of the books I got were great. And some were terrible. Really terrible, almost as if someone had filed the serial numbers off of *Instant Visual Basic Programming Guide for Complete Dummies in 24 Hours* and pasted on pictures of iPhones. There was one early iPad book that literally had an error on every page I skimmed at the bookstore. 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 or she 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 into 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*, just by judging its heft, contains a lot of material. Well, assuming you've got the print version in hand. *War and Peace* weighs the same as *The Little Prince* in ebook form, so it's hard to tell. Just skim through the table of contents and you can see that it covers a lot of stuff. A metric freakload of stuff. And it's all relevant stuff. It covers the basics like installing the development tools. Model-View-Controller. Master-Detail. 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 storyboards, AirPrint, AirPlay,

iCloud, and Core Image. Kirby and Tom have suffered the arrows in their backs dealing with months of flaky prerelease software so that you don't have to.

Very good books are timely but not exploitative. I saw the 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 because nobody had had a device in hand for that long. It was pumped out as fast as possible to hit the market, and it showed. The core of *Learning iPad Programming* has been in development for well over a year as I write this. Good books take time to achieve high levels of awesomeness.

Great books transcend their subject matter. This book is called *Learning iPad Programming*. It would 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 up-front design. This book, though, takes you from design, to a throw-away 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 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's much more than how to single-step with the debugger. As I was reading preproduction versions of the chapters so that I could sound halfway intelligent in this foreword, I emitted an audible “SQUEE” when I hit Chapter 25. I love debugging, and 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 25.

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 are 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, and get out.

Kirby and Tom are different. They're known entities. They have blogs. Tom has his name on a Core Data book. They've shipped products. They have happy customers. They answer questions online. They organize and speak at conferences. They organize CocoaHeads chapters. 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. I have the feeling it's destined to become one of those influential books for some of you out there.

—Mark Dalrymple

Cofounder of CocoaHeads, the international Mac and iPhone programmer community

Author of *Advanced Mac OS X Programming: The Big Nerd Ranch Guide*

November 12, 2011

Preface

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

- Ninety-two percent of Fortune 500 companies are testing or deploying iPads.
- Over 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.

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 5, 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 A5 processor, and 3G and/or WiFi networking, the iPad is the benchmark in a post-PC world. More important, though, is how the iPad 2 fits into the Mac/iOS ecosystem. Mac OS X Lion and iOS 5 users can use FaceTime for video chat from desktop to device. What's more, iOS 5's iMessage enables users to text from their iPad with other iPad and iPhone users. The iPad is a unique marriage of hardware and technology, and it is the Gold Standard for tablets.

This book is written with iOS 5 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 the book will include brief discussions of iPhone programming where appropriate, the primary focus of the book is the iPad. The book highlights those areas of the iOS 5 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 5, such as container view controllers, iCloud, and Core Image, as well as some of the great new features in Xcode 4.2, 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 PhotoWheel from the App Store: itunes.apple.com/app/photowheel/id424927196&mt=8. 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 over email, and view them on your TV wirelessly using AirPlay. But more important than the app is what you will learn as you build the app.

You will learn how to take advantage of the latest features in iOS 5 and Xcode, including storyboarding, Automatic Reference Counting, iCloud, and Core Image. You will learn how to leverage other iOS features such as 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 make 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 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, like the iPhone and iPod touch. You use 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.

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. This means that 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 more weather-related information on a single screen than you can get on the iPhone



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.)

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. Say, for example, 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, however, you open a folder, you see a new list of files and folders. Continue tapping folders to navigate further down the hierarchy.

To move back up the hierarchy, tap the back button in the upper left corner. 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 arrowlike 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 and the left-side view disappears. This allows the user to focus his attention on the main content displayed on the right side.

Note

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

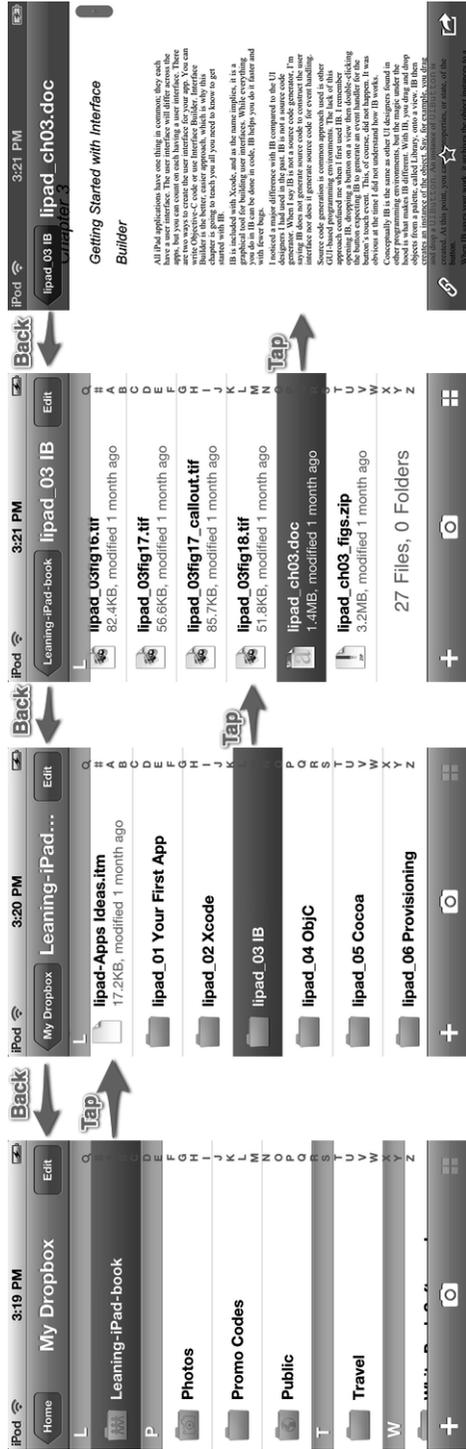


Figure P.2 Example of navigating the hierarchy of folders and files using the Dropbox app on the iPhone. You tap to move forward, or drill down, to more content, and you tap the back button to move backward.

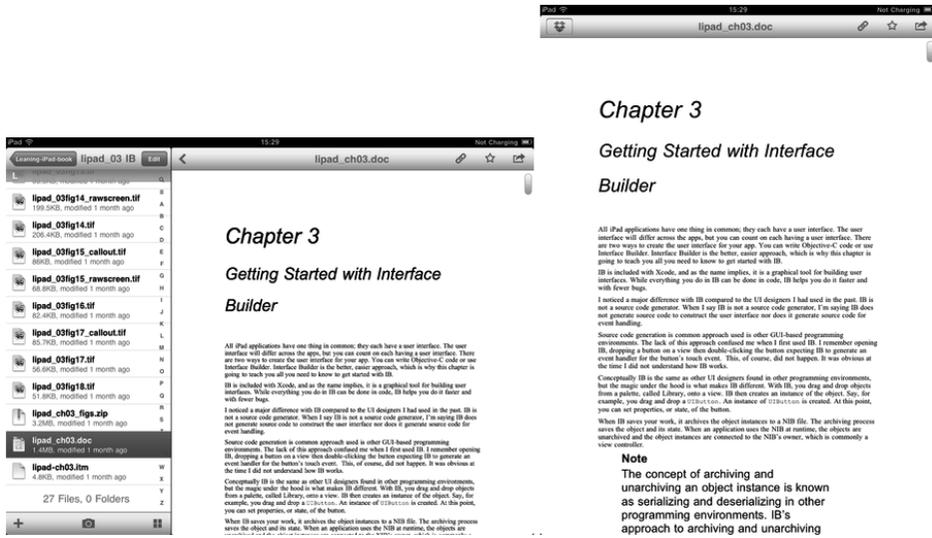


Figure P.3 Screen shots of Dropbox running on the iPad. Notice how the navigation is displayed in the left-side view when the device is held in a landscape orientation and is hidden when the iPad is rotated to portrait.

This view pattern is often called “master–detail,” where the master view is displayed on the left side and the detail view is displayed on the right side. 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. Like the iPad, the iPhone does support rotation and 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 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

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 connect the user interface and does not 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 in 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

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, 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 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 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 (mattgemmell.com/2010/05/09/ipad-multi-touch).

Another way to explore the iPad multi-touch interface is to play with Uzu for iPad, only \$1.99 in the App Store (bit.ly/learnipadprog-UzuApp). 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.)

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, 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

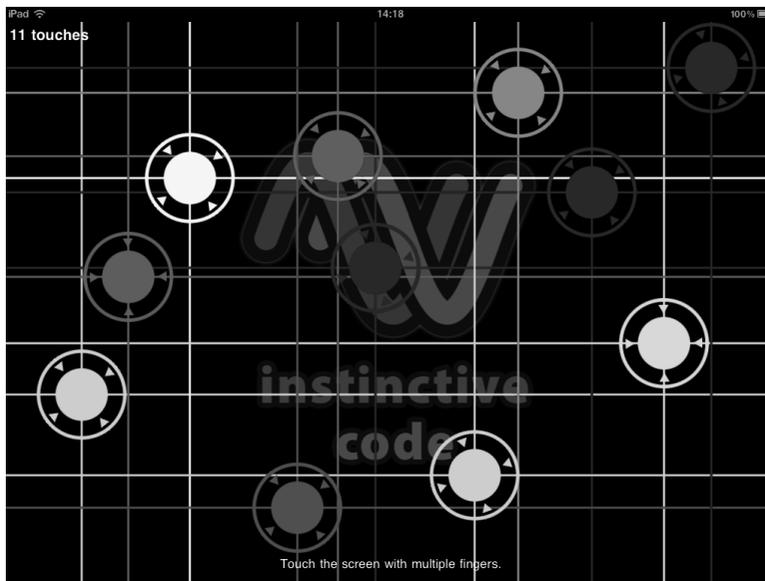


Figure P.4 Matt Legend Gemmell's multi-touch sample app for the iPad illustrating 11 touches

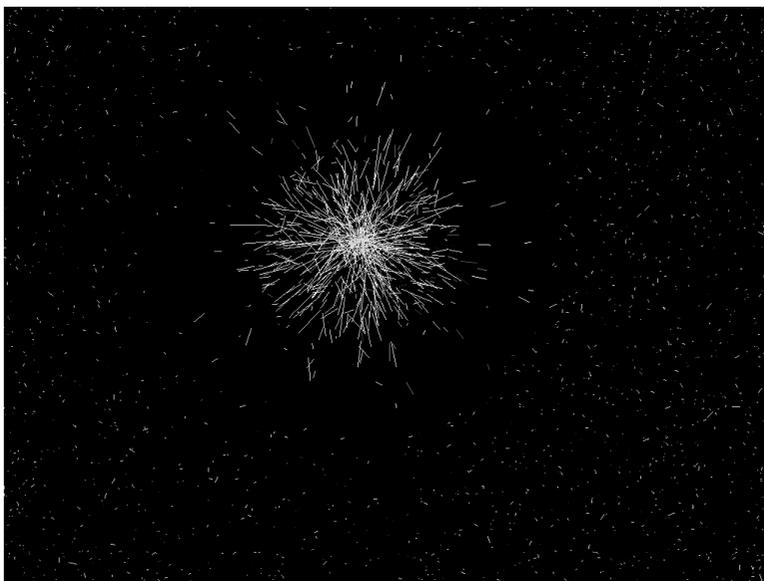


Figure P.5 Uzu, the particle visualizer for the iPad

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 27 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. The chapter 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 IDE (Integrated Development Environment) 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 user interface (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 to an `IBAction`.
- Chapter 4, “Getting Started with Objective-C”

This chapter introduces you to Objective-C with a brief overview of the programming language of choice for iPad programming. The goal for this chapter is not to be a comprehensive review of the programming language but instead to provide enough information to get you started 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 scary 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. This is where you get hands-on building a real iPad app. The app you build is no simple Hello World app. It is PhotoWheel, a full-featured photo app. In Part II, you learn everything from 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 get a chance to learn about the split-view 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, “Working with Views”

In this chapter, you dive into the world of views. Here you learn how to create a custom wheel view for displaying photos.
- Chapter 11, “Using Touch Gestures”

In this chapter, you learn 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 is about photos, so it is only natural that you need to learn how to add photos to the app. In this chapter, you learn how to retrieve photos from the Photos app library and how to take new photos using the built-in camera.
- Chapter 13, “Data Persistence”

PhotoWheel won’t be very useful if people can’t save their work. There are numerous methods for storing and retrieving app data. In this chapter, you explore two of them, and you learn to use Core Data.
- Chapter 14, “Storyboarding in Xcode”

A storyboard is an exciting new way for designing an app’s user interface. In this chapter, you get hands-on with storyboarding, and you learn how you can do more with less code using Interface Builder.
- Chapter 15, “Doing More with View Controllers”

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.
- 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. And it is your job to ensure that your app displays properly regardless. That is what you learn in this chapter: how to support device rotation.
- 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, “Sending Email”

Virtually everyone has an email account these days, and everyone loves looking at photos. So it only makes sense that PhotoWheel users want to share photos with family and friends using email. In this chapter, you learn how to send email from your app.
- 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 communicate between an iPad app and a Web server to 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 hard, but with iCloud it is a lot easier than it could be. 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. And you do it using AirPlay, 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 this to PhotoWheel in a convenient user interface that allows people to preview effects before committing to them.
- Part III, “The Finishing Touches”

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

- Chapter 25, “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 introduces you to the GDB and shows you how to turn on and off breakpoints, and how to use sounds to debug. The chapter also introduces you to more advanced debugging techniques such as using Instruments to track down memory leaks.

- Chapter 26, “Distributing Your App”

The application is written. It has been debugged and tested. The next step is getting 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 27, “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. This includes 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. However, the book assumes that you have some prior programming experience with other tools and programming languages. It 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 5. You are expected to have a Mac on which you can program using Xcode and Interface Builder, as well as an iOS developer account and an iPad. Some programming experience is helpful, particularly knowledge of C, although there is a chapter on object-oriented programming with Objective-C to give you a head start.

The book will also appeal to experienced iOS developers, people who have programmed and have shipped 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 (**learnipadprogramming.com/source-code/**). 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 (**github.com/kirbyt/PhotoWheel**).

You will also find more how-to articles and tips for improving PhotoWheel at the book's blog site (**learnipadprogramming.com/blog/**).

And 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](https://twitter.com/kirbyt) or [@atomicbird](https://twitter.com/atomicbird) on Twitter.

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 can be downloaded from the Apple iOS Dev Center.¹

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.

1. Apple's iOS Dev Center: developer.apple.com/ios.

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. I want to thank Tom for agreeing to be coauthor during the final stages of this book, which would have been delayed even more if not for his help. I want to give a huge THANKS to Chuck Toporek for giving me this opportunity to write a book. And, of course, I want to say thanks to the technical reviewers and the production team for all their hard work in a short amount of time.

I want to send an extra thanks to Daft Punk for the *TRON: Legacy* album. It was the soundtrack for most of this book.

I want to thank Steve Jobs and the amazing Apple engineers for bringing the fun back to programming for me. Last, 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 especially like to thank Chuck Toporek, our technical reviewers, and the rest of the production team for all their hard work making me look good in print. Kirby and I wrote this book while iOS 5 was still in beta testing, and we often found it necessary to make revisions before we were finished, just to keep up with the pace of change in iOS and the developer tools. 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 5 and the iPad. Without them we wouldn't have such a cool topic to write about.

Finally, as of this writing we've only just learned that Steve Jobs has passed away after a long and highly successful tenure as CEO of Apple. I got started writing software nearly 30 years ago on an Apple II, and it set the tone for my future career. Thanks for all you've done over the years, Steve.

About the Authors

This book is brought to you by...

Kirby Turner

Kirby Turner is an independent software developer and business owner focusing on Mac and iOS programming. Kirby has been programming since the early 1980s. He sells his own apps through his company, White Peak Software, and he does contract programming when time allows.

When Kirby is not sitting behind the keyboard, he can be found hanging out with his wife, Melanie, and son, Rowan, hiking the mountains of New England, kayaking the waters in and around Salem, Massachusetts, and snowboarding down mountains in search of perfect powder.

Tom Harrington

Tom Harrington switched from writing software for embedded systems and Linux to Mac OS X in 2002 when he started Atomic Bird, LLC. After six years of developing highly regarded Mac software, he moved to iPhone in 2008. He now develops iOS software on a contract basis for a variety of clients. In addition to this book, Tom is coauthor of *Core Data for iOS*. Tom also organizes iOS developer events in Colorado. When not writing software, he can often be found on his mountain bike. You can find Tom on Twitter as [@atomicbird](#).

Provisioning Your iPad

Before you ship the next killer app, you must test it on an iPad. Using the iPad Simulator to test and debug your application will only get you so far. That's why it's important to always test your app on the real thing. But before your app will run on a real device, you must set up your iPad as a development device. This is where provisioning comes in, and that's exactly what you will do in this chapter: provision your iPad for development purposes.

The steps involved are tedious. Fortunately you do not have to repeat them often, only a few times a year. And Apple is continuously improving the provisioning process. Provisioning a device today using Xcode is easy-peasy compared to what one had to do back in the day ... the dark days of 2008.

About the iOS Provisioning Portal

The iOS Provisioning Portal is the Web site, shown in Figure 6.1, used to request and download certificates, register device IDs, create App IDs, and create and download provisioning profiles. The Web site and all of its features are not available to everyone. You have access to the portal if you are one of the following:

- An individual registered as a paid member of the iOS Developer Program. You can be registered as an individual or a company.
- An individual registered as a team member for a company that is a paid member of the iOS Developer Program. Depending on your team member role, your access to the Provisioning Portal may be limited. There is no cost to the individual team member, and team members have access to all of the same developer resources that a paid member has, courtesy of the paid company membership.

If you are not a paid member or a team member of a company, you will not have access to the iOS Provisioning Portal. Without access, you will not be able to provision your device.

Note

You can read more about joining the iOS Developer Program in Appendix A, *Installing the Developer Tools*.

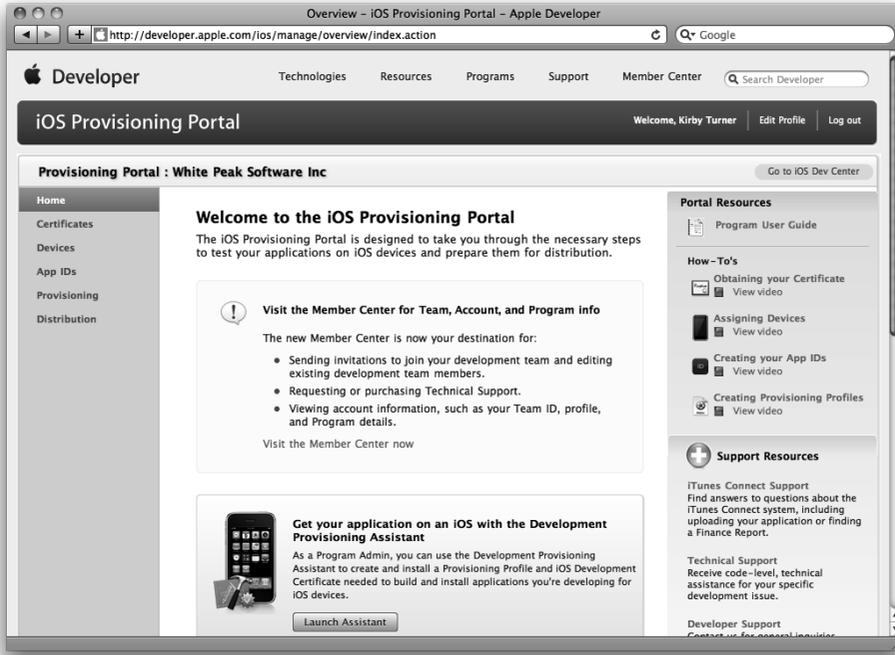


Figure 6.1 iOS Provisioning Portal home page

iOS Developer Program Team Roles

When you join the iOS Developer Program as a paid member, you have the option to join as an individual or a company. The main difference between the two is that a company can include team members and an individual cannot. As a company, you can invite other developers to join your team and assign roles to them. The roles are as follows:

- **Team Agent:** A Team Agent is the individual who originally enrolled in the iOS Developer Program. A Team Agent has full access to the iOS Provisioning Portal, privileges not available to others. A Team Agent can invite others to be Team Admins or Team Members. A Team Agent can also approve certificate requests, register devices, create App IDs, create Push Notification Service SSL certificates, enable In App Purchase, retrieve the distribution certificate, and create development and distribution provisioning profiles.
- **Team Admin:** A Team Admin can invite new Team Admins and Team Members to join the team. The Team Admin can also approve certificate requests, register devices, and create development provisioning profiles.
- **Team Member:** A Team Member can request and download a development certificate and download development provisioning profiles.

- **No Access:** No Access is a role given to developers to prevent them from accessing the iOS Provisioning Portal. This role is used when the company is enrolled in multiple Apple developer programs, that is, the iOS Developer Program and Mac Developer Program.

The Provisioning Process: A Brief Overview

Provisioning a new device is not difficult in and of itself, but there are a number of steps that must be performed first. You must be a paid member of the iOS Developer Program or be a team member of a company that is a paid member. You must request and install a development certificate on your development machine. You must register your device ID. You must create an App ID. You must create and install a development provisioning profile. All of this must be done before you can provision your iPad and run your application on it.

Note

It should go without saying, but you really should have an iPad if you plan to write iPad applications. You can learn iPad programming without an iPad, but you will be limited in what you can test. It's my opinion that having a physical device is a necessity for anyone serious about iPad programming.

Luckily you do not have to perform each of these steps often. Some of the steps, such as requesting and installing a development certificate, need to be done only once a year, and Xcode performs other steps for you as needed. Still, knowing about each step helps when a problem does come up, and yes, you will encounter a problem with the process at some point in your iOS career. It's just the nature of writing apps for mobile devices.

Note

A development certificate is valid for one year. You must request a new development certificate when the old one expires.

You've had a number of new terms thrown your way, such as device ID, App ID, and provisioning profile, but what exactly are these?

What Is a Device ID?

A device ID, also known as the Unique Device Identifier or UDID for short, is a 40-character string that uniquely identifies the device. The UDID is tied to a single device; no two devices will ever share the same device ID. The device ID is added to a provisioning profile. This restricts applications built with the provisioning profile to run on only the devices associated to the profile.

You register devices in the iOS Provisioning Portal. You can register up to 100 devices per year. The devices you register are for development and testing purposes only. You do not register the devices of your customers who download and install your application through the App Store.

Registering a device counts toward your yearly limit even if you remove the device. Say you register your iPad, then delete it from the list of registered devices. It still counts toward your annual limit of registered devices. If after deleting your iPad UDID from the list you decide to add it back, it will count again toward your yearly limit. Put another way, deleting a device does not reduce the number of registered devices for the year.

What Is an App ID?

The App ID is used during the development and provisioning processes to share Keychain data among a suite of applications, and it is used for document sharing, syncing, and configuration of iCloud. The App ID also allows the application to communicate with the Push Notification Service and external hardware accessories.

The App ID is the combination of a Bundle Seed ID and a Bundle Identifier. The Bundle Seed ID is a universally unique, ten-character, alphanumeric string. Its value is generated by Apple within the iOS Provisioning Portal. The Bundle Identifier is a string value you add to your application bundle. The Bundle Identifier is used by the operating system to identify your application for tasks such as applying application updates.

The common naming convention for the Bundle Identifier is the reverse domain name style. If your company name is Acme and your application name is Awesome App, you would define the Bundle Identifier as `com.acme.awesomeApp`.

The Bundle Identifier portion of the App ID can contain the wildcard character (*). The wildcard character, if used, must be the last character of the Bundle Identifier, for example, `com.acme.*`. Using the wildcard character allows the App ID, and its associated provisioning profiles, to be used with multiple applications.

Note

The Bundle Identifier portion of the App ID can be a wildcard only. In other words, the Bundle Identifier value can be a single asterisk character. As you will learn momentarily, the wildcard App ID created by Xcode's Organizer window is a single asterisk character.

A wildcard App ID is convenient because it can be used for multiple applications. Xcode's Organizer window creates a wildcard App ID that can be used by any application. The App ID looks like the last example in Table 6.1.

Note

When creating an App ID in the Provisioning Portal, you add a description of the purpose of the App ID. The App ID created by Organizer has the description "Xcode: Wildcard AppID."

Table 6.1 App ID Examples

App ID	Remarks
ABCDE12345.com.acme.awesomeApp	An explicit App ID for Acme's Awesome App
ABCDE12345.com.acme.*	A wildcard App ID for Acme applications
ABCDE12345.*	A wildcard App ID for all applications

You can also create an explicit App ID. An explicit App ID restricts the provisioning profile to a single application. Certain Apple services such as Push Notification and In App Purchase require an explicit App ID. If you plan to use an Apple service, you must use an explicit App ID.

Changing from a Wildcard to an Explicit App ID

You can change from a wildcard App ID to an explicit ID. You may find that you need to do this if, for example, you decide to add Game Center support to your application after it has been released. While you will be changing the App ID, you must not change the Bundle ID as defined in your application's *info.plist*. The Bundle ID defined in the app is used to identify your application for new releases, that is, updates to your application. Changing this breaks the app update process, and Apple will reject your app update submission if the Bundle ID changes.

Changing the Bundle ID suffix in the App ID is not the same as changing the Bundle ID in the application. This is why you can safely switch from a wildcard App ID to an explicit App ID. Changing the App ID's Bundle ID suffix does not change the application's real Bundle ID.

To change from a wildcard App ID, you must create a new App ID. You can use the same Bundle Seed ID or generate a new one. For the Bundle ID suffix, enter the bundle ID exactly as it is defined in your application's *info.plist*.

Because you now have a new App ID, you need to create a new provisioning profile that uses the new App ID.

Note

If your existing application uses the Keychain, select the same Bundle Seed ID for the new App ID. If you use a different Bundle Seed ID, your application will not be able to access any existing Keychain data.

What Is a Development Provisioning Profile?

A development provisioning profile ties developers, devices, and App IDs to a development team. A provisioning profile makes it possible for the developer to install and run the application on a device for the purpose of debugging and testing. To make this possible, the development provisioning profile must be installed on the device. A device, however, can contain more than one development provisioning profile.

Note

There is a second type of provisioning profile, the distribution provisioning profile. A distribution provisioning profile is used for Ad Hoc distribution—distribution to registered devices for the purpose of testing an application—and App Store distribution. More information on the distribution provisioning profile is included in Chapter 26, “Distributing Your App.”

Now that you understand the pieces, let’s get your development computer and device ready.

Do I Need a Dedicated Development Device?

A common question asked by new iOS developers is whether a dedicated development device is needed or not. The answer to this question depends entirely on you. Many iOS developers own only one device, but many other developers own multiple devices.

I have seven devices at the time of this writing: two iPads, three iPod touches (a first gen, a second gen, and a fourth generation), one iPhone 3GS, and one iPhone 4 CDMA. I use the older iPod touches to test my applications on older versions of iOS. I use the fourth-gen iPod touch to test on the latest version of iOS and to test retina display images. I usually keep one iPad up-to-date with the latest public release of iOS, and I use the other iPad, which is my primary iPad development device, to run beta versions of iOS. And I use my iPhone 3GS for testing on a phone, which can behave differently from an iPod touch, receiving incoming phone calls and SMS messages, for example.

I write a number of client apps, and my clients have different requirements, which is why I have multiple devices—and that number is growing every few months as I continue to buy new devices. It’s doubtful that the average iOS developer needs this many devices. You can produce your application using only one device. However, there are advantages to having a dedicated development device.

I find I often need to reset my device to factory-install state, or I want to test my app on the latest beta release of iOS. You can still do this while owning only one device, but it does mean your device is constantly changing. That game you bought last night and got halfway through gets deleted when you reset your device, or your favorite app crashes on the new iOS beta release. This can be frustrating if your development device is your only device and you use the device for personal reasons.

Because of this, it is helpful—and it saves you time—to own devices dedicated to development purposes. Owning multiple devices, however, can get expensive, especially for the iPad. That’s why I typically buy used devices for development. The Apple Refurbish Store is a good place to shop for used devices at an affordable price. Also, ask family and friends what they plan to do with their old devices if they are upgrading to newer hardware. This might be an opportunity to score a development device for free or close to free.

Setting Up Your Development Machine

The first thing you must do is set up your Mac development machine for code signing. Coding signing your application serves two purposes: It confirms the app author, and it guarantees that the app has not been altered since it was signed. iOS requires each application to be digitally signed before the app can run on a device. Code signing is never a joy, but it is a necessity, ensuring that the application comes from a trusted source.

To code sign your app, you must have a public and private key pair and a digital certificate. When your application is in development, you use a development certificate to code sign the application. This allows you to run and test your app on your own device. When you are ready to deploy your application to other devices, whether through the App Store or through Ad Hoc and enterprise distribution, you use a distribution certificate to code sign the app.

Note

Chapter 26, “Distributing Your App,” covers distribution of your application. Therefore, I do not cover the distribution certificate in this chapter. Steps to request and install the distribution certificate are covered in Chapter 26.

Requesting a Development Certificate

To prepare your development Mac for code signing, you must first request a development certificate. To request a development certificate you need to generate a Certificate Signing Request (CSR). You use the Mac desktop application Keychain Access to generate the CSR. As Keychain Access creates your CSR, it also generates a public and private key pair for you and stores the pair in the login Keychain. The key pair identifies you as an iOS developer and is associated to the development certificate.

The Keychain Access application is available in the **Applications > Utilities** folder. Alternatively, you can launch the app using Spotlight. Press **⌘-Space** and start typing “Keychain,” without the quotes, in the Spotlight box. Spotlight will find the Keychain Access application for you. All you need to do then is press the **Enter** key to launch the application.

The first thing you need to do in Keychain Access is select **Preferences** from the menu (or type **⌘-,**). Click **Certificates** and turn off Online Certificate Status Protocol (OCSP) and Certificate Revocation List (CRL) as shown in Figure 6.2.

Close the Preferences window, then select **Keychain Access > Certificate Assistant > Request a Certificate from a Certificate Authority** from the menu bar. At this point, you are ready to enter the certificate information, as seen in Figure 6.3. Enter your email address in the User Email Address field. This must be the same email address you submitted when registering as an iOS Developer.

Enter your name in the Common Name field. The name must match the name submitted when you registered as an iOS Developer. Leave the CA Email Address field blank. Mark the options “Saved to disk” and “Let me specify key pair information.”

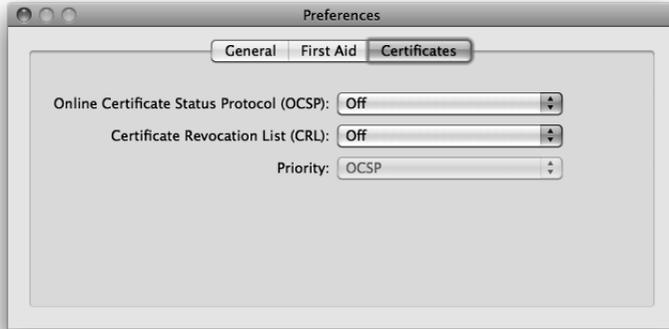


Figure 6.2 Turn off the OCSP and CRL settings in the **Preferences > Certificates** screen.



Figure 6.3 Certificate Assistant window in Keychain Access

When you have finished, click the **Continue** button. The Assistant will ask where to save the CSR. Your desktop is as good a place as any, so save the CSR to your desktop.

Make sure you selected the “Let me specify key pair information” option in the Certificate Information window (Figure 6.3). This tells the Certificate Assistant to display the Key Pair Information window, shown in Figure 6.4. Here you set the options for generating the key pair. Select 2048 bits for the Key Size and RSA for the Algorithm. Click **Continue**.

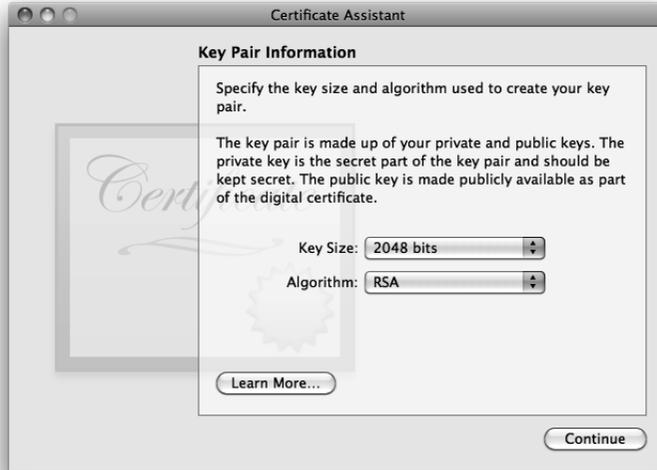


Figure 6.4 Key Pair Information window

Note

Your certificate request will be rejected if you do not specify the Key Size as 2048 bits and the Algorithm as RSA.

The Certificate Assistant will generate the CSR and save it to your desktop. A public and private key pair is also generated for you and stored in the login Keychain. The key pair can be viewed in the Keychain Access application under the Keys category as seen in Figure 6.5.

Note

If you are doing development from more than one Mac, you must copy the private key to each development machine. You will not be able to sign your application and test on your iPad without your private key. Use Keychain Access to export your private key and import it to your other development machines.

For a step-by-step guide on exporting your private key, visit developer.apple.com/ios/manage/certificates/team/howto.action, scroll to the bottom of the Web page, and read the information under the “Saving your Private Key and Transferring to other Systems” section.

Click **Done** to close the Certificate Assistant. The generated CSR file is on your desktop. Your next step is to submit the CSR for approval.



Figure 6.5 Public and private key pairs stored in the login Keychain

Getting More Help

The iOS Provisioning Portal requires you to perform a number of steps before you can test your application on your iPad and prepare your applications for distribution. A set of helpful resources is available in the Portal Resources.

The Portal Resources include a detailed user guide and how-to videos for requesting and installing your development certificate, assigning devices, creating App IDs, and creating provisioning profiles. If you need additional help with the iOS Provisioning Portal or you prefer seeing the steps performed, you should check out the Portal Resources.

The Portal Resources are available on the iOS Provisioning Portal home page under the section Portal Resources, found on the upper right side of the page, as seen in Figure 6.6.

Submit Your CSR for Approval

The next step is submitting your CSR for approval, and it is less involved than the previous step. A Team Agent or Admin approves or rejects your CSR. You will receive an email notifying you of your certificate status. If your request is approved, you download your digital certificate from the Provisioning Portal and install it on your development machine.

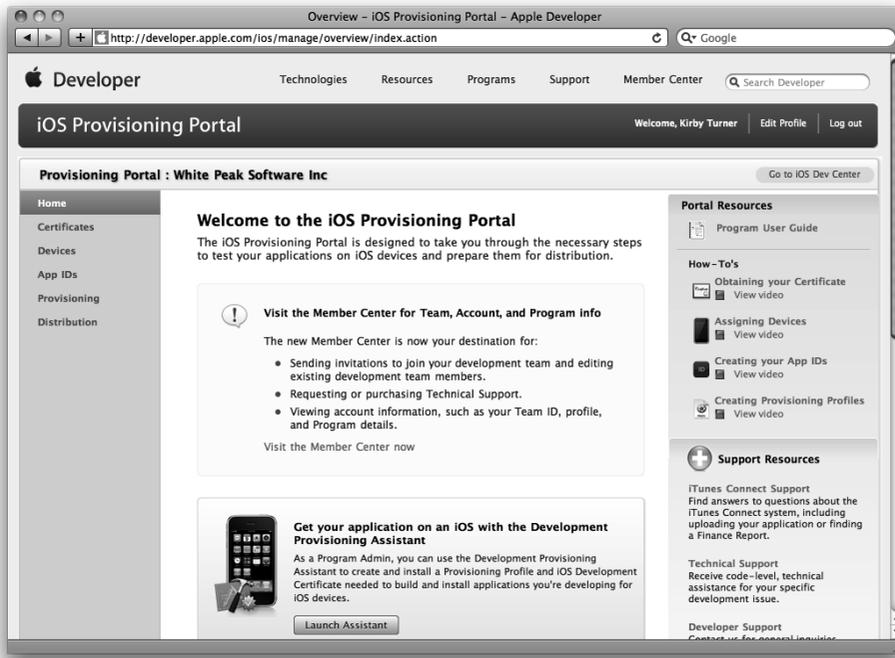


Figure 6.6 Portal Resources containing a user guide and how-to videos

Note

All certificate requests must be approved through the iOS Provisioning Portal. If you are the Team Agent or Team Admin, you still must approve your own certificate request.

To submit your CSR, sign in to the iOS Provisioning Portal. If you have trouble remembering the URL of the iOS Provisioning Portal, sign in to the iOS Dev Center (developer.apple.com/ios). Toward the upper right side of the iOS Dev Center home page there is a section titled iOS Developer Program, shown in Figure 6.7. This section includes links for the iOS Provisioning Portal, iTunes Connect, Apple Developer Forums, and the Developer Support Center. Click the **iOS Provisioning Portal** link to be transported to the portal Web site.

From the iOS Provisioning Portal home page, click the **Certificates** link found in the left-side menu bar. Next click the **Development** tab, then the **Add Certificate** button. Scroll down to find the **Choose file** button. Click the button and select the CSR file that you saved to the desktop. Click the **Submit** button to upload your CSR. If you are unable to submit your CSR through the Web site, email the CSR file to the Team Agent.

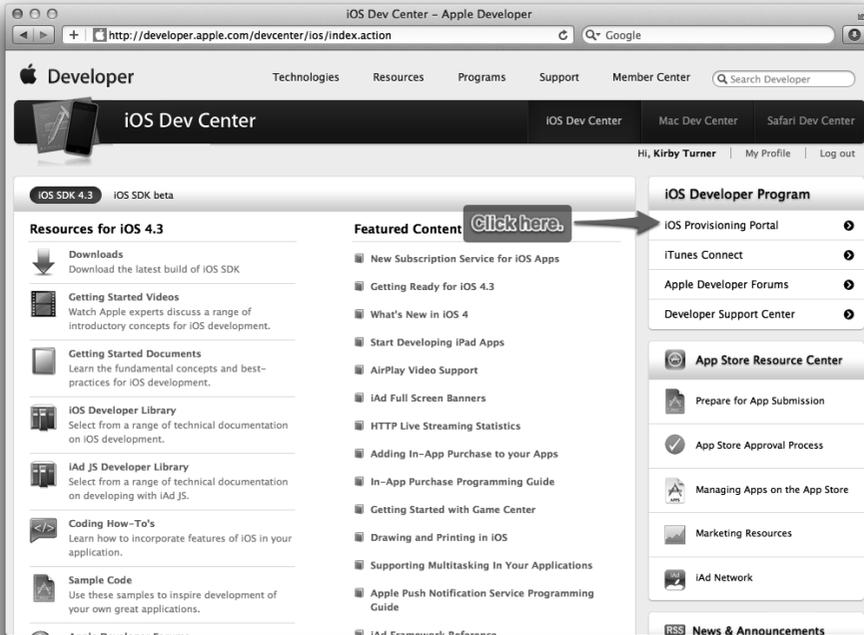


Figure 6.7 iOS Dev Center home page with a link to the iOS Provisioning Portal

Download and Install Your Certificate

The Team Admin will be notified by email after your submitted CSR has been received. Once the Admin approves or rejects your request, you will receive a notification email with your certificate status. When it has been approved, you sign in to the Provisioning Portal again, then click **Certificates > Development**. You'll see your approved certificate listed at the top. Click the **Download** button under the Action column to save the certificate to your development machine.

Note

If this is your first time setting up your development machine, you need to download and install the WWDR intermediate certificate. Click the download link and save the *AppleWWDRCA.cer* file to your development machine. Use Finder to navigate to the saved *AppleWWDRCA.cer* file, and double-click the file to launch Keychain Access; this installs the certificate on your machine.

On your development machine, use Finder to locate the saved *.cer* file. Double-click the *.cer* file to launch Keychain Access and to install your certificate. Save the

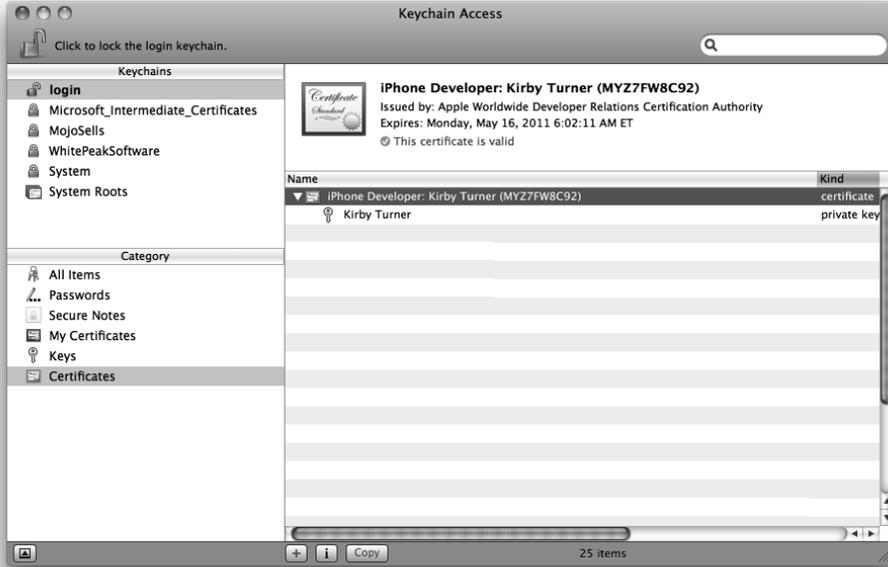


Figure 6.8 You can view your certificate in the Certificates category.

certificate to your login Keychain. Once it is installed, you can view the certificate by selecting the Certificates category for the login Keychain in Keychain Access (Figure 6.8). Your certificate name will be “iPhone Developer: Your Name.”

While still in Keychain Access, click the Keys category for the login Keychain. Here you will see your public and private keys generated by the Certificate Assistant. Expand the private key by clicking the disclosure triangle. You will see that the certificate has been associated to your private key. Apple never receives your private key when you submit the CSR. Your private key is available only to you. This is why it is important that you not lose it.

Note

Make sure you have a backup of your key pair. If you do not have a backup and you lose the private key, you must go through the certificate request process all over again. I use Time Machine to make hourly backups of my primary development machine. I also use SuperDuper! to make complete system backups weekly. This provides a suitable backup of my public-private key pair. You should do something similar.

Your development machine is now set up to code sign builds of your application, but you cannot run your app on your iPad yet. You still have a few more steps to follow. Next up is setting up your device.

Setting Up Your Device

Now that your development machine is set up, it's time to set up your iPad for development. Here's what needs to happen:

1. You need to register your device ID.
2. You need to create an App ID.
3. You need to create a development provisioning profile.
4. You need to download and install the development provisioning profile.

These steps can be performed in one of two ways: by using the Xcode Organizer window or the iOS Provisioning Portal. Using Organizer is by far the easier way to set up your iPad for development. It performs the steps automatically for you. This approach is, however, not without its limits.

Organizer creates a wildcard App ID, and as you may recall, a wildcard App ID cannot be used if you plan to use Apple services like Game Center, In App Purchase, and Push Notification. That said, you should still let Xcode do its magic. While you may not use a wildcard App ID for your next awesome iPad app, it can be used to build and run sample apps and to test proofs of concept and prototype applications on your iPad.

Use for Development

The steps to set up your iPad for development are quite easy using Xcode. Tether your iPad to your development computer. Launch Xcode and open the Organizer window (**Windows > Organizer** or **Shift-⌘-2**). Organizer shows the list of registered and attached iOS devices. Attached devices have a status icon displayed to the right of the device name. A white status icon means the device is not ready for development. A green status icon means the device is ready for development. A yellow status icon means the device is busy.

Click the name of your iPad in the Devices list. You should see a screen similar to the one shown in Figure 6.9. Click the **Use for Development** button. Xcode will prompt you for your iOS Provisioning Portal credentials. Enter your Apple ID and password used for your iOS Developer account.

Xcode automatically sets up your device for development. It registers your device with the iOS Provisioning Portal, creates a wildcard App ID and development provisioning profile, if needed, and last downloads and installs the provisioning profile.

Note

Figure 6.9 shows a new iPod touch. The setup instructions are the same regardless of the device type. I chose to use an iPod touch to capture the screen shot because my development iPad was in use at the time.

The process can take a few minutes. While the process is running, the status icon is yellow. Do not disconnect your iPad from your computer while the process is



Figure 6.9 Organizer window with a new Apple device attached to the computer

running. Once the process is complete, the status icon changes to green and you will see a screen similar to the one in Figure 6.10.

You are now ready to build and run iOS applications on your iPad. To test that everything has been set up correctly, create a new project in Xcode. You can select any iOS application template; it does not matter which one. Make sure you select your iPad as the device for the active scheme (seen in Figure 6.11). Build and run (**⌘-R**) the project. Assuming your development machine and iPad are set up correctly, you will see the sample app running on your iPad.

Note

Sign in to the iOS Provisioning Portal to see the App ID and development provisioning profile created by Xcode. The App ID has the description “Xcode: Wildcard AppID,” and the development provisioning profile has the name “Team Provisioning Profile: *.” The status for the profile will also show “Managed by Xcode.”

As you just learned, using Xcode is the easiest way to set up a new device. But what happens if there is a problem? And what if you plan to use Apple services like iCloud, Game Center, or In App Purchase, which require an explicit App ID? You need to use the iOS Provisioning Portal to manually perform the steps.

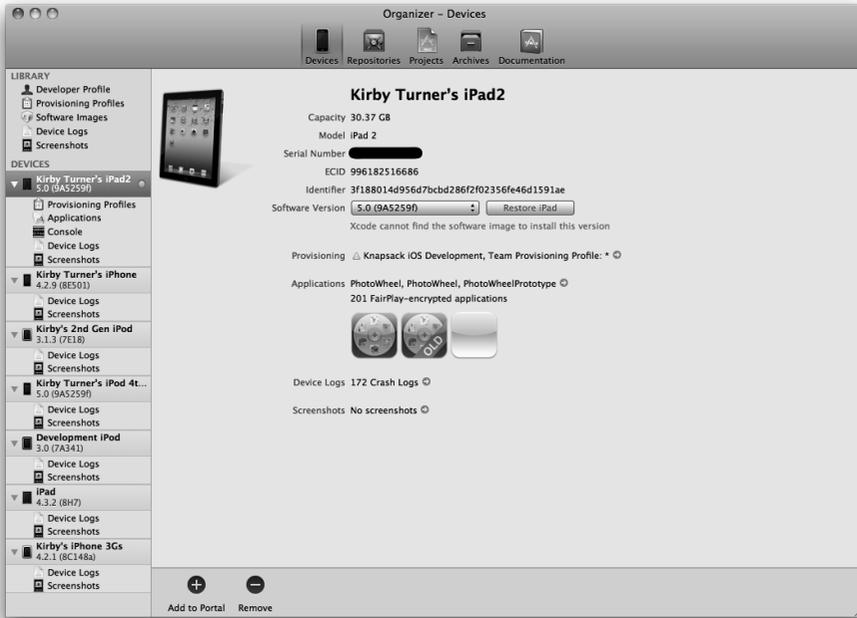


Figure 6.10 Organizer window with an attached Apple device ready for development



Figure 6.11 Use the scheme popup menu to set the run destination.

Using the iOS Provisioning Portal

As you already know, the iOS Provisioning Portal is used to request and download developer certificates, to register devices, to create App IDs, and to create and download provisioning profiles. You have already gone through the steps for requesting and downloading the developer certificate; the rest of this chapter will focus on the other areas of the Provisioning Portal.

Adding a Device ID

You can add a device ID individually, or you can upload a batch of device IDs in a *.deviceids* file generated by the iOS Configuration Utility. The iOS Configuration Utility is available to enterprise members only, so it is not covered here. Instead, the following walks you through adding an individual device ID.

Start by signing into the iOS Provisioning Portal Web site (developer.apple.com/ios/manage/overview/index.action), then click Devices in the left-side menu bar. Click the **Add Devices** button found on the right side of the Devices Web page. Enter a device name followed by the 40-character device ID, as seen in Figure 6.12. (I tend to be descriptive with the device name, adding the device owner's name and device type, for example, "Kirby Turner's iPad2.") Click the plus sign (+) button to

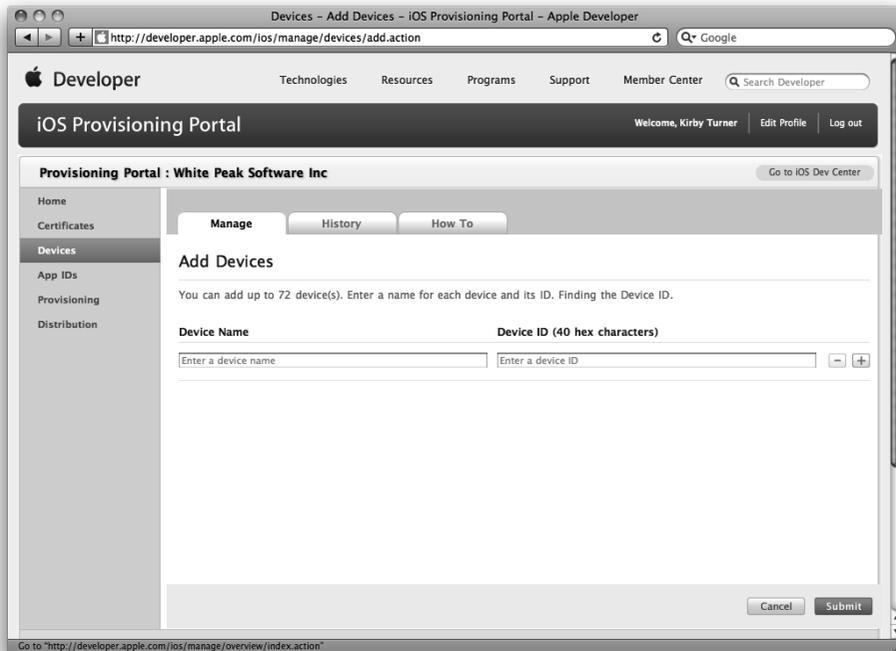


Figure 6.12 Add Devices page in the iOS Provisioning Portal

enter another device. When you have entered all the devices, click the **Submit** button. That's it. The device IDs are now registered.

Note

Only the Team Agent and Team Admin can register a device ID. A Team Member must send his device ID to the agent or admin to be registered.

You can edit the name of a registered device, but you cannot change the device ID. If a device ID is no longer valid—for example, you no longer own the device—you can remove the device from the list by marking the check box next to the device name, then clicking the **Remove Selected** button at the bottom of the registered device list.

How to Find the UDID for a Device

Finding the UDID for a device is not hard if you know where to look. There are different ways to find the ID of a device. The easiest for developers is to use Xcode's Organizer window. Open the Organizer window (**Shift-⌘-2**) and select the device. The device ID is displayed in the Identifier field as shown in Figure 6.13.

Another way to find the ID, albeit more obscure, is to use iTunes. Connect your device to your computer, then launch iTunes. In iTunes, select the device to see the device information screen, shown in Figure 6.14. Click the serial number. This will display the



Figure 6.13 Device information displayed in the Organizer window



Figure 6.14 Device information displayed in iTunes

UDID in place of the serial number. Once the UDID is showing, type **⌘-C** to copy it to the clipboard. Click the field again to return to the serial number.

Note

You can also click the software version to see the serial number.

Another common way to retrieve the UDID is to use one of the many free UDID apps available in the App Store. These apps not only retrieve the UDID from the device but also provide options to copy the ID to the clipboard and to email the device ID to the recipient of your choosing. Using a UDID app is the easiest way for nondevelopers to send you their device IDs.

To download a UDID app, go to the App Store and search on “UDID.” Select the app that appeals to you the most.

Adding an App ID

Adding an App ID is almost as easy as registering a device ID. Once again, sign in to the iOS Provisioning Portal. Click the App IDs menu item in the left-side menu bar, then click the **New App ID** button found on the right side of the page, shown in Figure 6.15. This takes you to the Create App ID page.

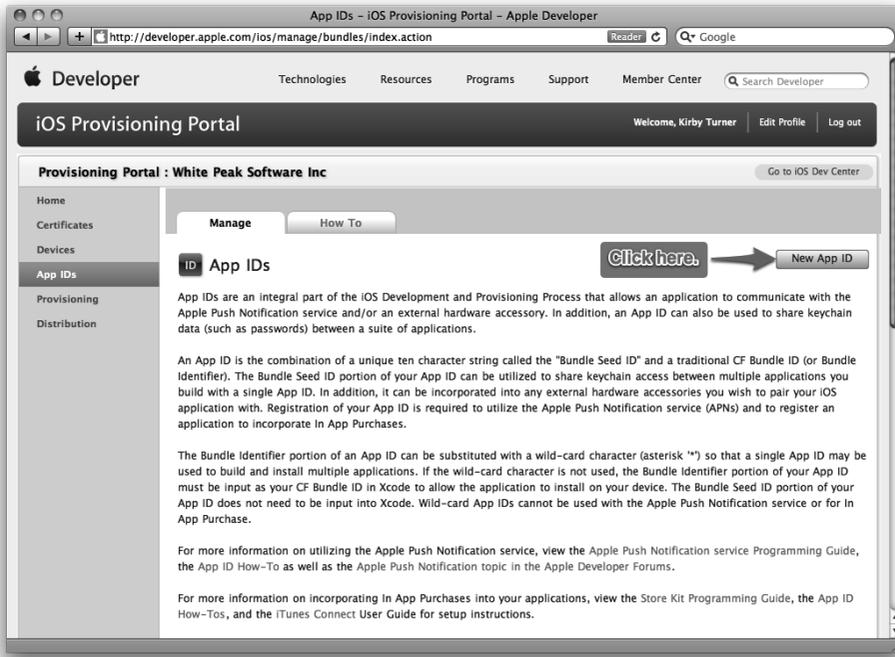


Figure 6.15 Click the **New App** button found on the App IDs page in the iOS Provisioning Portal.

Note

The Team Agent is the only team member who can add a new App ID.

On the Create App ID page, seen in Figure 6.16, enter the description for the App ID. Use a name or description that makes sense to you. For instance, you might use the name of your application when creating an explicit App ID. The description you enter is used throughout the portal, so enter a name or description that will help you identify the App ID.

Next, set the Bundle Seed ID, aka the App ID prefix. You can have the portal generate a new seed for you, or you can select from a seed that you previously generated. If you plan to share Keychain data access among multiple applications, you need to use the same Bundle Seed ID for each application's App ID.

Last, enter the Bundle Identifier, aka the App ID suffix. Remember to use the reverse domain name naming convention. Include an asterisk (*) as the last character if you wish to create a wildcard App ID.

Click the **Submit** button to save the App ID. Clicking the button returns you to the App IDs page. Scroll down the page to see the newly created App ID in the list of App IDs.

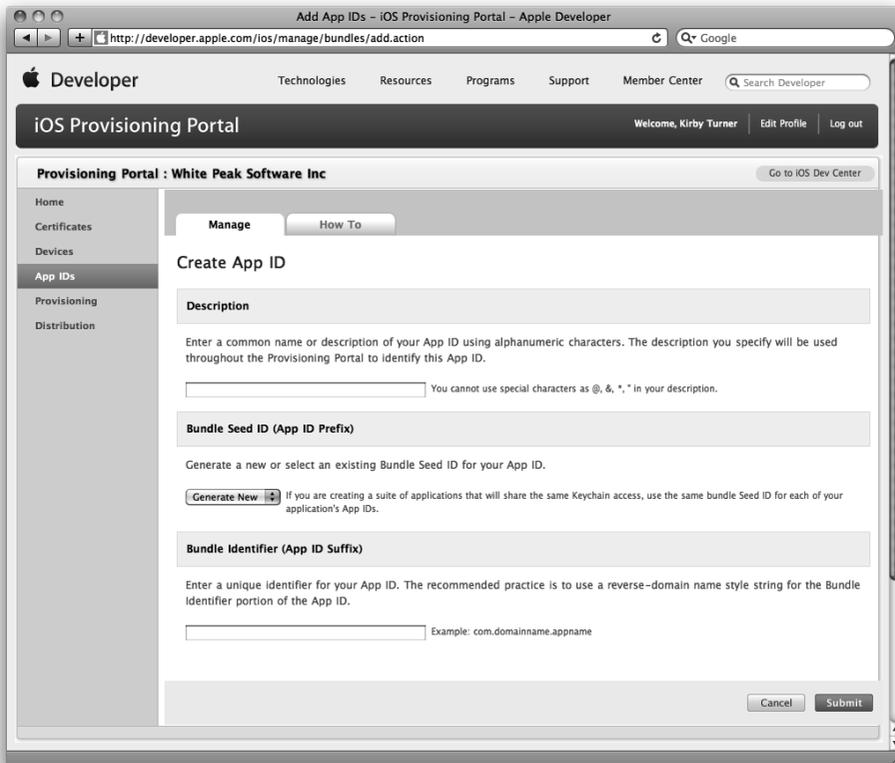


Figure 6.16 The Create App ID page

In the list of App IDs, you can choose between two actions: Details and Configure. The Details action is available only for wildcard App IDs. Click the action to see the details of the wildcard App ID.

The Configure action is available only for explicit App IDs. Click the action to configure the App ID for Push Notification. You do not need to configure the explicit App ID for In App Purchase and Game Center services as these services are enabled by default for explicit App IDs.

Note

You cannot delete an App ID once it has been created.

Creating a Development Provisioning Profile

The development provisioning profile brings you (the developer), your device, and your App ID together so that you can run and test your application on your iPad. To create a new development provisioning profile, log in to the iOS Provisioning Portal

and click the Provisioning link found in the left-side menu bar. Click the **Development** tab at the top of the Provisioning screen, then click the **New Profile** button. This will take you to the Create iOS Development Provisioning Profile page, seen in Figure 6.17.

Note

Only Team Agent and Team Admin members can create development provisioning profiles.

Enter a name for the provisioning profile. (I like to use descriptive names, for example, “Hey Peanut Dev Profile.”)

Select the development certificates that will be associated to the provisioning profile. This identifies the development certificates used to code sign the application for development. You should select the developer certificate for each team member who will be using the development provisioning profile.

Select the App ID for the profile. Each profile can have only one App ID. You can use a wildcard App ID if you wish to use the same development provisioning profile for more than one application.

Finally, select the devices that the developers can use for running and testing the application on a device. Then click the **Submit** button to generate the development provisioning profile.

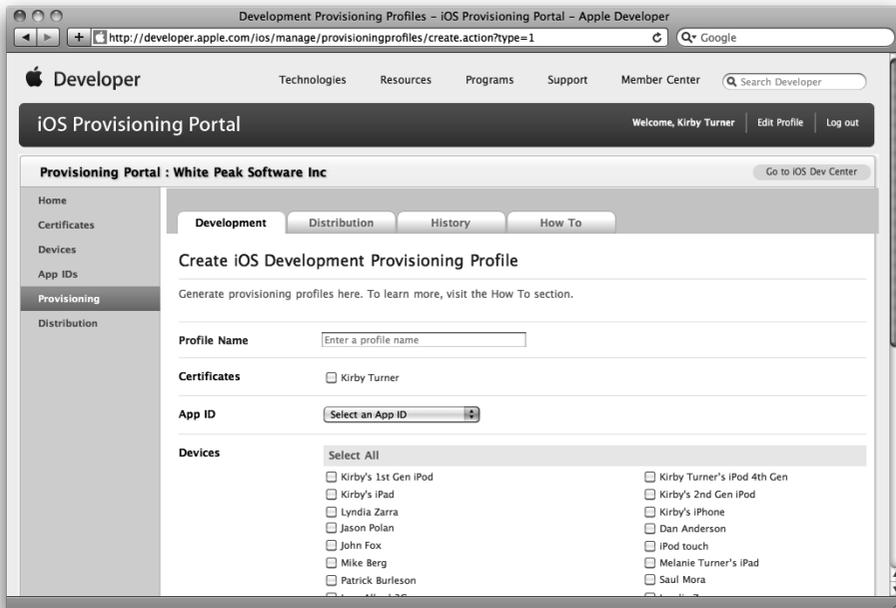


Figure 6.17 Create iOS Development Provisioning Profile page

Downloading a Development Provisioning Profile

Once a development provisioning profile has been generated, team members can download and install the profile. To download the development provisioning profile, log in to the iOS Provisioning Profile, click the **Provisioning** menu item, then click the **Development** tab on the Provisioning page. You will see a list of development provisioning profiles, as shown in Figure 6.18. Click the **Download** button for the profile you wish to install.

Now you are ready to install the provisioning profile.

Installing a Development Provisioning Profile

There are a few ways to install a development provisioning profile. You can copy the profile file to `~/Library/MobileDevice/Provisioning Profile`, but the most common way is to use the Organizer window in Xcode. This will ensure that the profile is stored in the proper directory, and it will create the directory if it does not already exist.

Note

The development provisioning profile you download has the file extension `.mobileprovision`.

To install using Organizer, launch Xcode and open the Organizer window (**Shift-⌘-2**). Select a device on which to install the provisioning profile, then click the

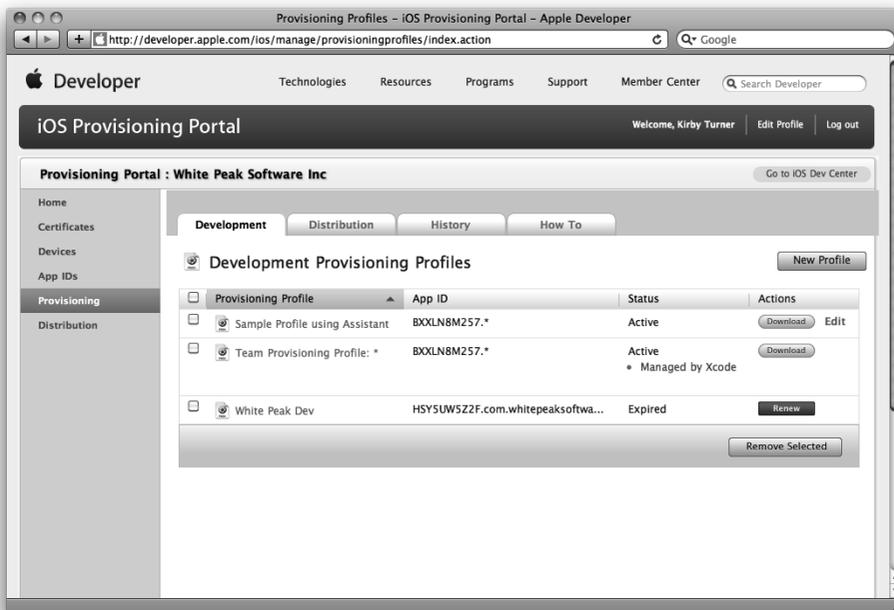


Figure 6.18 List of development provisioning profiles



Figure 6.19 Click the + button or drag and drop the .mobileprovision file to add the development provisioning profile.

+ button to select the *.mobileprovision* file you downloaded from the iOS Provisioning Portal. Alternatively, you can drag and drop the *.mobileprovision* file onto the Provisioning list in the Organizer window; see Figure 6.19.

Another way to install a development provisioning profile is to drag and drop the file onto the iTunes icon in the dock. Note, however, that this will fail if the provisioning profile directory does not exist.

You can also copy the *.mobileprovision* file directly to the *~/Library/MobileDevice/Provisioning Profiles* directory. To make it easier, add a shortcut for the directory to Finder. When you need to install a new provisioning profile, simply drag and drop the downloaded file onto this shortcut; see Figure 6.20.

A third option is to drag and drop the provisioning file onto the Xcode icon in the Dock. This will copy the provisioning profile to the appropriate directory and to your iPad device if connected.

That’s it for installing a development provisioning profile.

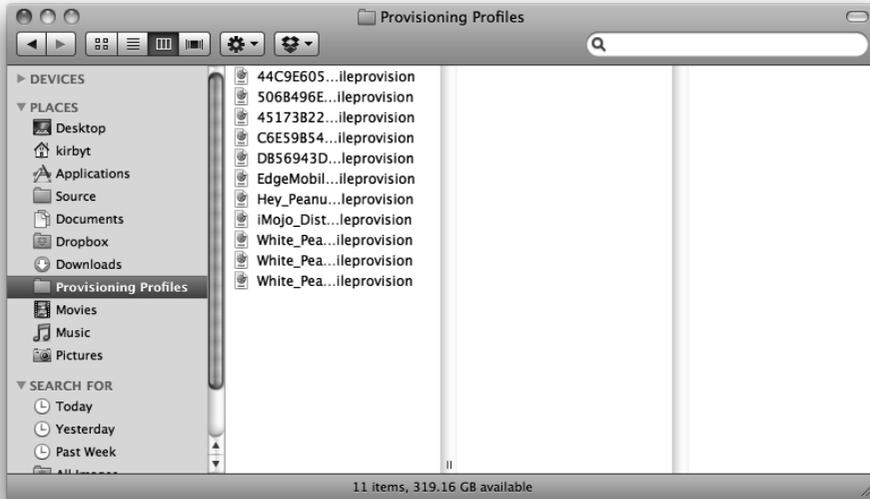


Figure 6.20 Finder window with a shortcut folder to the provisioning profile directory

Summary

At this point, your development machine should be set up for code signing iOS applications, and your iPad should be ready for development. You're now ready to write iPad applications that you can run on your own device. But first, let's talk a bit about app design as it relates to the iPad.

Index

A

- About screen
 - creating, 344–346
 - rotating, 511
 - segues, 346–347
- AboutSceneSegue identifier, 347, 355
- AboutViewController class, 352–354, 511
- Above the fold display, 149
- Accessorizer tool, 33
- accessoryButtonTappedForRowWithIndexPath method, 221–222, 225, 292, 322
- Action sheets, 274–278
- actionButton property, 495
- Actions
 - IBAction. *See* IBAction macro
 - slideshow controls, 623
- Ad Hoc distribution, 683
 - preparing, 684–688
 - provisioning, 684
- Add Devices page, 131
- add method, 205, 218
- addButton method, 205
- addButtonsToNavigationBar method
 - deleting photos, 490–493, 495
 - filter containers, 645
 - slideshows, 623
- addChildViewController method, 367–368, 373
- Adding
 - Core Data entities, 308–310
 - images, 340
 - photo albums, 400–401, 404, 408–409
 - photos to albums, 293–297, 323–325, 429–434
 - slideshows, 612
 - table view data, 203–220
- addPhoto method
 - adding photos to albums, 431, 434
 - Flickr, 555
 - naming photo albums, 416, 420
 - removing photo albums, 423
- addphoto.png file, 579
- addPhotoAlbum method, 400–401, 404, 408–409
- addPhotosObject method, 313
- Admin role in iTunes Connect, 696
- Adobe Photoshop, 157
- advanceSlide method, 621
- Advancing photos, 620–621
- Age calculations, 94
- AirPlay. *See* Slideshows
- AirPrint. *See* Printing
- ALAssetsLibrary class, 269
- Albums. *See* Photo albums
- alertView method, 424
- Aligning objects, 49–52
- alloc method, 80, 85
- Allocations tool, 677–678
- allowMultipleSelection property, 445–446
- Allows External Storage option, 308
- allowsMultipleSelection property, 436–437
- alwaysBounceVertical property, 444
- Angle brackets (<>)
 - classes, 175
 - header files, 73
- Angle of spin gesture rotation, 262, 266
- angleOffset property, 510
- API
 - external display, 610–611
 - Flickr, 557–564
 - GCD (Grand Central Dispatch), 580–582
 - Printing, 527

- API keys in Flickr, 549–551
- App Charters, 141–142, 146–147
- App icon, 341
- App IDs
 - iCloud service, 586–587
 - iOS Provisioning Portal, 133–135
 - overview, 118–119
- App Store distribution, 683
 - assets, 694–695
 - information for, 692–694
 - preparing, 689–690
 - process, 691–692
 - provisioning, 688
 - rejected apps, 692
- AppDelegate class
 - Core Data stack, 304–305, 307
 - photo albums, 408
 - PhotoWheel app, 173–177
 - storyboards, 339
 - table view data, 229
- AppKit framework, 91
- Apple IDs, 708
- Application bundles, 100
- applicationDidBecomeActive method, 181
- applicationDidEnterBackground method, 181
- applicationFrame property, 618
- applicationWillEnterForeground method, 181
- applicationWillResignActive method, 180
- applicationWillTerminate method, 181
- applyFilter method, 652
- Applying filters, 651–652
- applySpecifiedFilter method, 651–653
- Apps, 141
 - competing products, 145–146
 - defining, 141–142
 - delegates, 174–179
 - distributing. *See* Distribution
 - feature lists, 143–145
 - icons, 154
 - managing, 696–697
 - mockup. *See* Mockup apps
 - names, 142, 693
 - prototyping, 160–163
 - quitting, 172
 - summaries, 142
 - target audience, 144–146
 - templates, 6
 - types, 704
 - UI design, 148–154
 - UIApplication, 103–104
- ARC (Automatic Reference Counting), 86–87
- arc4random function, 70
- arc4random_uniform function, 647
- Arithmetic, date, 93
- Arrays
 - creating, 97
 - property lists, 286–287
 - strings, 195
 - table views, 195
- Ash, Mike, 581
- assetForURL method, 269
- Assets Library framework, 269–271
- assign attribute, 77
- Assistant editor, 35–36, 61–63
- Asterisk character (*)
 - App IDs, 119, 586
 - Bundle IDs, 118
 - pointers, 74
- Asynchronous photo downloading, 564–570
- At signs (@) for string literals, 81
- Atomic properties, 77
- Attributes. *See* Properties
- Attributes inspector
 - opening, 11
 - scenes, 345–346
 - titles, 15
 - working with, 54
- Authentication, 582
- autoAdjustmentFiltersWithOptions method, 637
- autoEnhancedVersionOfImage method, 636
- Autohiding chrome, 477–482
- Automatic image enhancement, 636–637, 652–653
- Automatic Reference Counting (ARC), 86–87
- automaticallyForwardAppearanceAndRotationMethodsToChildViewControllers method, 368, 508
- Autosizing
 - objects, 54
 - rotation, 501–502
 - scenes, 344–345
 - text fields, 215–216

Availability by country information for apps, 693
 Availability date information for apps, 693
 availableModes property, 611
 awakeFromInsert method, 318–319

B

Background apps, 172
 background-landscape-right-grooved.png file, 506
 background-portrait-grooved.png file, 342
 backOnePhoto method, 623–624
 Backups of key pairs, 127
 Balsamiq Mockups app, 160
 Bar buttons, 111
 Base settings, 38
 Base URLs in Flickr, 562
 Began state, 258, 261
 beginAnimation method, 242
 __block directive, 578
 .bmp files, 109
 Books, recommended, 702
 Booleans, 77–78
 Bounce effect, 243
 Bounds of frames, 672
 bounds property, 611
 Brackets ([]) in code, 81–82
 Brainstorming technique, 143
 Breakpoint navigator, 22
 Breakpoints, 662–663
 customizing, 664–666
 debugging example, 671
 hitting, 666–667
 setting, 663–664
 __bridge syntax, 249–250
 Bugs. *See* Debugging
 Build Settings, 36
 buildFlickrURLWithParameters method, 560–561, 564
 Bumgarner, Bill, 265
 Bundle Display Name setting, 330
 Bundle IDs, 118, 693
 Bundle Seed IDs, 118, 134
 Bundles, 99–100
Business of iPhone App Development, 699
 buttonIndex property, 277

ButtonMaker application, 106
 Buttons, 13
 bar, 111
 classes for, 105–106
 identifiers, 345
 table views, 205
 buttonTapped method, 59–60

C

C programming language, 65
 Caching
 cells, 200
 images, 570
 photo albums, 404
 CALayer class, 248–249
 call command, 673
 Camera roll, saving photos to, 283–284
 Cameras
 checking for, 273–274
 full-screen, 280–281
 cancel method
 filters, 655–656
 Flickr, 553, 574
 table view data, 212, 214
 cancelChromeDisplayTimer method, 480, 482
 Cancelled state, 258
 canEditRowAtIndexPath method, 220, 225–226
 canMoveRowAtIndexPath method, 227
 canSendMail method, 537, 539–540
 Carousel view, 240–247
 cellAtIndex method
 DetailViewController, 240
 Flickr, 576–577
 GridView, 439–440, 445, 450–451
 photo album thumbnails, 427–428
 photo albums, 403, 406
 photo display, 436
 WheelView, 387, 396, 398
 cellDoubleTapped method, 258
 cellForRowAtIndexPath method
 albums, 321
 photos, 324
 property lists, 292, 294
 table views, 198–200, 222, 226
 cellIndexesToDisplay method, 391–392, 398

- cellIndexForIndex method, 391, 398
- Cells
 - nonvisible rows, 200
 - table views, 194
- cellTapped method
 - image picker controller, 272–274
 - property lists, 294
 - touch gestures, 256
- .cer files, 126–127
- Certificate Assistant, 121–123, 127
- Certificate Revocation Lists (CRLs), 121
- Certificate Signing Requests (CSRs), 121, 124–126
- Certificates
 - CSRs, 124–126
 - development, 121–124
 - downloading and installing, 126–127
 - Provisioning Portal, 684
- CFNetwork framework, 565
- CGAffineTransform setting, 651
- CGGeometry class, 361
- CGImage format, 635
- CGImageRef structure, 249, 651
- CGRectContainsPoint function, 446
- CGRectInset setting, 654
- CGRectNull setting, 654
- CGRectOffset function, 361–362, 364
- Changed state, 258, 261
- Charters, 141–142, 146–147
- Child scenes, 369–371
- Child view controllers, 371–375
- childViewControllers property, 368
- Chrome effects, 477–482
- chromeHidden variable, 481, 617
- chromeHideTimer variable, 418, 617
- CIAffineTransform filter, 649
- CIColor class, 647
- CIColorInvert filters, 633, 649
- CIColorMonochrome filters, 649
- CIContext class, 632, 635
- ciContext variable, 640
- CICrop filters, 633, 654
- CIDetector class, 637–638
- CIDetectorTypeFace class, 638
- CIFaceFeature class, 637–638
- CIFalseColor filter, 649
- CIFilter class, 631–632
 - effects, 647–651
 - image enhancement, 637
 - overview, 633–634
 - working with, 634–636
- CIHueAdjust filters, 633, 649
- CIImage class, 631, 635–636, 653
- CISepiaTone filters, 633–636
- CIVector class, 654
- Clair, Robert, 65
- Clark, Josh, 148
- @class directive, 228–229
- Class Extensions Explained*, 265
- Classes
 - defining, 74
 - extensions, 265–266
 - implementing, 78–82
 - methods, 78
 - overview, 71–73
- ClearToolbar class, 421–422, 495
- clickedButtonAtIndex method
 - action sheets, 276–277
 - adding photos, 431–432, 434
 - deleting photos, 494, 496
 - email, 542, 545
 - Flickr, 555–556
 - printing, 528–529
 - removing photo albums, 424
- clipsToBounds property, 444–445
- Clock Radio app, 152–153
- Cocoa framework, 61, 89
 - design patterns, 112–113
 - Foundation library. *See* Foundation library
 - RESTful Web services, 548–549
 - stack, 89
 - UIKit, 103–112
- Cocoa Samurai, 32
- Cocoa Touch layer, 89–90
- Code completion feature, 33
- Code folding, 28
- Code names, 142
- Code Pilot tool, 33
- Code Signing settings, 690
- Code Snippet library, 24
- Coding styles, 30

- CoinToss project and CoinTosser class, 70–73
 - algorithm, 69–70
 - creating, 66–69
 - declared properties, 75–78
 - dot syntax, 83–84
 - implementation, 78–82
 - instance variables, 74–75
 - interfaces, 74
 - methods, 78
 - selectors, 82–83
 - working with, 84–85
- Collection classes
 - NSArray and NSMutableArray, 97
 - NSDictionary and NSMutableDictionary, 97–98
 - NSSet, NSMutableSet, and NSCountedSet, 98–99
- Colon characters (:):
 - class names, 74
 - methods, 205
 - parameters, 78
- Color settings, 26–27
- colorTintFilter method, 648
- Combining images, 634
- Command Line Tool template, 66–67
- Commercial keys in Flickr, 550
- commitAnimations method, 242
- commonInit method
 - GridView, 437–438, 444–446
 - spin gesture recognizers, 262, 266
 - WheelView, 389, 397
- commonInitWithSize method, 452
- Competing products, 145–146
- Compilers for Objective-C, 65
- Concurrent programming, 580–582
- Conditional breakpoints, 663–664
- configureExternalScreen method, 617–620, 671
- confirmDeletePhotoAlbum method, 423–424
- Conflicts
 - Core Data entities, 311
 - iCloud service. *See* iCloud service names, 80
- connection method, 103
- connectionDidFinishLoading method
 - images, 568, 570
 - SimpleDownloader, 103
- Connections
 - classes, 102–103
 - NIB files to code, 44, 57–63
 - objects to outlets, 16
- Connections inspector, 55–56
- Console
 - apps, 66
 - debugger messages, 667
- Contact email address information for apps, 694
- Container IDs, 600
- Container view controllers
 - child scenes, 369–371
 - child view controllers, 371–375
 - creating, 369
 - overview, 367–368
 - push segues, 375
- Containers, iCloud, 590, 600
- contentSizeForViewInPopover property, 190–191
- Contexts, managed objects. *See* Managed object contexts
- Continuous gestures, 254
- Continuous recognizers, 258
- Control-Click
 - connections, 16, 60–63, 353, 400
 - Finder, 378
 - projects, 334, 338
 - segues, 346–347, 356
 - views, 330, 399
- controllerDidChangeContent method, 450
- Controls
 - slideshows, 622–624
 - UI design, 152–153
- Converting
 - radians to degrees, 266
 - strings to dates, 100
 - UIImage to PNG format, 296
- Coordinate systems for grids, 472
- copy attribute, 77
- Copying
 - files, 378–380
 - methods, 184
 - objects, 48
- Copyrights, 694
- Core Animation*, 243
- Core Animation for MacOS X and the iPhone*, 243

- Core Animation framework, 242–243
 - Core Data: Apple's API for Persisting Data on Mac OS X*, 413
 - Core Data for iOS*, 298
 - Core Data framework, 298
 - adding, 302–303
 - changing models, 593
 - iCloud, 585
 - managed object contexts, 300–301
 - managed objects and entity descriptions, 299–300
 - model editor, 307–308
 - NSManagedObject subclasses, 310–314
 - overview, 299
 - persistent stores and persistent store coordinators, 301, 305–306
 - photo albums, adding, 408
 - photo albums, displaying, 326
 - photo albums, reading and saving, 320–323
 - photos, adding, 323–325
 - photos, entities, 308–310
 - PhotoWheel, 307–315
 - PhotoWheelPrototype, 302
 - prototype code, 380–385
 - stack setup, 303–307
 - transient attributes, 592–593
 - Core Foundation, 91
 - Core Image effects
 - applying filters, 651–652
 - auto-enhance, 652–653
 - CIFilter, 633–635, 647–651
 - concepts, 631–632
 - delegate methods, 638–640
 - face zoom, 653–654
 - image analysis, 636–638
 - instance variables, 640
 - interface additions for, 640–647
 - utility methods, 655–656
 - Core OS layer, 90
 - Core Services layer, 90
 - Cox, Brad, 79
 - CPUs for images, 632
 - Crashing apps, 692
 - Create a new Xcode project option, 4
 - Create App ID page, 134–135
 - Create iOS Development Provisioning Profile page, 136
 - createCGImage method, 635
 - createScaledImagesForImage method, 594, 598
 - CRLs (Certificate Revocation Lists), 121
 - Cropping images, 654
 - CSRs (Certificate Signing Requests), 121, 124–126
 - .cur files, 109
 - Curly braces ({} for local variables, 74
 - Current line with breakpoints, 666
 - currentAlbumIndex property, 292–293, 320
 - currentAngle property, 265–266
 - currentCalendar method, 93
 - currentIndex property
 - external displays, 620
 - PhotoBrowserViewController, 463, 466
 - slideshows, 613–614, 616, 621, 624–625
 - currentPhotoView property, 613, 615
 - Custom breakpoints, 664–666
 - Custom queues, 580
 - Custom touch gestures, 258–266
 - Custom views, 231–232
 - carousel, 240–247
 - photo wheel view cell, 248–252
 - wheel, 233–240
 - CustomNavigationController class
 - photo browser, 470, 474–476
 - pop transitions, 365–366
 - CustomPushSegue class
 - container view controllers, 375
 - implementing, 358–362
 - photo browser, 467–468, 470, 472–474
- ## D
- Data models
 - data persistence, 285–286
 - property lists, 287–288
 - Data persistence, 285
 - Core Data. *See* Core Data framework
 - custom code to model objects, 315–320
 - data model, 285–286
 - property lists. *See* Property lists
 - SQLite, 326–327

- Data stores, 299
- Data types, 74, 78
 - NSCalendar, 93
 - NSData and NSMutableData, 92–93
 - NSDate, 93
 - NSDateComponents, 93–94
 - NSDecimalNumber, 94–95
 - NSInteger and NSUInteger, 95
 - NSNull, 96
 - NSNumber, 95–96
 - NSObject, 96
 - NSString and NSMutableString, 96–97
- dataSource property
 - GridView, 436, 554
 - photo albums, 399
 - table view, 195
 - UITableView, 197
 - UITableViewDataSource, 194
 - wheel view, 234, 237–238
- dateAdded property, 309
- Dates
 - arithmetic, 93
 - formatting, 100
 - pick lists, 107
- DDEBUG compiler option, 675–676
- dealloc method, 602
- Debug area, 25–26
- Debug build configurations, 685
- DEBUG compiler option, 675
- Debug navigator, 22
- Debugging, 625–626, 661
 - breakpoints, 662–667
 - concepts, 661–662
 - external display code example, 670–674
 - NSLog, 674–676
 - problem description, 661–662
 - profiling codes, 676–682
 - tools, 662
 - variable inspection, 667–670
 - Xcode, 663–670
- Declared properties, 75–78
- Dedicated development devices, 120
- Default.png file, 521
- Default-landscape.png file, 521
- defaultNameText property, 224–225
- defaultPhoto.png file, 250, 406, 426
- Degrees, converting, 266
- delaysContentTouches property, 444–445
- delegate property, 255
- Delegates
 - apps, 174–179
 - Core Image, 638–640
 - split view controllers, 185
 - table view data, 210
- deleteImageAtIndex method, 490, 495–497, 639
- deletePhoto method, 494, 496
- deletePhotoConfirmed method, 493–494, 496
- Deleting
 - objects, 48
 - photos, 489–497
 - table view data, 225–226
- dequeueReusableCell method
 - displaying photos, 436
 - GridView, 439, 445
 - WheelView, 387, 395, 398
- dequeueReusableCellWithIdentifier method, 200
- description method, 670
- Descriptions for apps, 693
- Deserialization, 44
- Design patterns, 112–113
- Designers, hiring, 153–154
- Destination controllers, 361, 468
- Destination frames, 474
- Destination image view, 362
- Detail view controllers, 169
- detailNavigationController, 179
- DetailView.xib field, 237
- DetailViewController class, 173
 - action sheets, 274–278
 - Carousel view, 245–247
 - image picker controllers, 272–273, 279–280
 - master detail apps, 179, 185–186
 - photos, 296–297, 323–326
 - PhotoWheelViewCell, 250
 - property lists, 293–294
 - table view data, 229–230
 - titles, 192
 - touch gesture recognizers, 255–257
 - wheel view, 238–240
- Detection, face, 637–638

- Dev Center, 708
- Developer documentation, 34
- Development machines, 121
 - certificates, 121–124, 126–127
 - CSRs, 124–126
- Development provisioning profiles, 119–120, 135–138
- Development setup, 128–130
- Device family types, 7
- Device IDs
 - iOS Provisioning Portal, 131–133
 - overview, 117–118
- .deviceids file, 131
- Devices
 - managing, 40
 - mirroring, 628–629
 - provisioning. *See* Provisioning
 - rotating. *See* Rotation
 - schemes, 39
 - setup, 128–130
- DIB (Windows Bitmap Format) format, 109
- Dictionaries
 - classes, 97–98
 - face detection, 638
 - Flickr, 563–564
 - iCloud, 606
 - property lists, 286–287
 - variables, 667–669
 - WheelView, 397
- didAccessValueForKey method, 598
- didChangeObject method, 402, 405
- didChangeValueForKey method, 596
- didDeselectCellAtIndex method, 436, 577
- didDismissWithButtonIndex method, 276–277
- didFailWithError method, 568, 570
- didFinishLaunchingWithOptions method
 - AppDelegate, 176, 179–180
 - Core Data stack, 307
 - NIB-based projects, 331
 - rotation, 501
 - storyboards, 339
 - table view data, 229–230
- didFinishPickingMediaWithInfo method, 281, 283–284, 295, 324–325
- didFinishWithResult method, 540
- didMoveToParentViewController method
 - child view controllers, 371, 373–375
 - container view controllers, 368
 - photo albums, 401, 404, 416, 418
 - selecting photo albums, 414
- didReceiveData method, 567, 569
- didReceiveResponse method, 567, 569
- didRotateFromInterfaceOrientation method, 499, 517
- didSelectCellAtIndex method
 - Flickr, 577
 - photo albums, 403, 406, 410–412
 - photo browser, 467
 - WheelView, 387
- didSelectRowAtIndexPath method
 - photos, 324
 - property lists, 292–293
 - table view data, 227, 229
- didTap method, 442–443, 446
- disablesAutomaticKeyboardDismissal method, 572, 578
- Discrete gestures, 254
- dismissModalViewControllerAnimated method, 353
- dispatch_apply function, 651
- dispatch_async function, 582, 600–601
- dispatch_get_global_queue function, 582
- Dispatch queues, 580–581
- Display area, 104
- Display buttons in table views, 205
- displayHelloName method, 14–15
- Displaying
 - external. *See* External display
 - Flickr, 555–557
 - photo albums, 398–408
 - photos, 297–298, 326, 434–446
 - slideshows, 613–616
 - table view data, 197–203
- distantFuture method, 623–624
- Distribution
 - Ad Hoc, 684–688
 - App Store. *See* App Store distribution
 - extra steps, 698–699
 - iTunes Connect, 695–698
 - methods, 683–684
 - provisioning profiles, 120
 - submitting apps, 696–698
- DLog macro, 674–675

- Dock
 - IB, 46
 - Xcode in, 3
 - Documentation, developer, 34
 - documentsDirectory method, 296
 - done method
 - table view data, 213–214
 - view controllers, 353–354
 - Don't Repeat Yourself (DRY) principle, 231
 - Dot syntax, 83–84
 - Double quotation marks (") for strings, 96
 - Double tap gesture, 257–258, 487
 - doubleTapped method, 486
 - downloaders property, 577, 579
 - downloadImageAtURL method, 567–569
 - Downloading
 - certificates, 126–127
 - development provisioning profiles, 137
 - images, 340
 - photos, 564–570
 - Xcode, 708
 - Downloading Images for a Table without Threads*
 - blog post, 565
 - downloadWithURL method, 102–103
 - Dragging, 254
 - drawRect method, 421
 - DRY (Don't Repeat Yourself) principle, 231
 - Dudney, Bill, 243
 - Dynamic arrays, 97
 - @dynamic directive, 314
 - Dynamic sets, 98
- E**
- editButtonItem property, 221
 - Editing
 - breakpoints, 664
 - table view data, 220–225
 - Editor area, 23
 - Editors, 35–37
 - Editor's gutter, 664
 - Email
 - MFMailComposeViewController, 535
 - operation, 533–535
 - SendEmailController, 535–546
 - emailCurrentPhoto method, 542–543
 - emailPhotos method, 546
 - Empty Application template, 6, 334, 336
 - Enable Entitlements option, 590
 - Encapsulation, 75
 - @end directive, 74, 79, 175
 - End state, 258
 - End User License Agreement (EULA), 694
 - enhancedCIImage property, 637
 - enhanceImage method, 652
 - Enhancement of images, 636–637, 652–653
 - Enterprise program type, 704
 - Entitlements in iCloud, 589–591
 - Entity descriptions, 299–300
 - enumerateGroupsWithTypes method, 269–270
 - EULA (End User License Agreement), 694
 - Events, touch, 253–254
 - Exception breakpoints, 663
 - Exceptions, 61
 - Exponents, 94
 - Extensions, classes, 265–266
 - External display
 - API, 610–611
 - debugging example, 670–674
 - managing, 616–620
 - options, 609
 - requirements, 609–610
 - testing and debugging, 625–626
 - externalDisplaySlideShowController variable, 617, 620
 - externalScreenWindow variable, 617, 620
- F**
- Face down orientation, 500
 - Face up orientation, 500
 - Faces
 - detecting, 637–638
 - zooming, 653–654
 - faceZoomRect property, 654
 - Failed state, 258
 - fakeSave notification, 606
 - falseColorFilter method, 648
 - Fault objects, 300
 - Feature lists for apps, 143–145
 - featuresInImage method, 638

- fetchResultsController method
 - GridView, 449
 - iCloud, 601
 - photo albums, 401–402, 404–405
 - fetchFlickrPhotoWithSearchString method, 575, 579, 581
 - fetchRequestWithEntityName method, 404
 - fetchResponseWithURL method, 560, 563–564
 - Fielding, Roy, 548
 - File coordinators in iCloud, 584
 - File Inspector, 38
 - File Template library, 24
 - Files
 - copying, 378–380
 - header, 72, 174, 177, 183–184
 - owners, 56
 - fileURLForAttributeNamed method, 595, 598
 - filteredLargeImage variable, 640
 - filteredThumbnailImage variable, 640, 645, 651
 - filteredThumbnailPreviewImages variable, 640, 642, 652
 - Filters, 100
 - applying, 651–652
 - CIFilter. *See* CIFilter class
 - delegate methods, 638–640
 - face zoom, 653–655
 - image analysis, 636–638, 652–653
 - instance variables, 640
 - interface additions for, 640–647
 - types, 632–634
 - utility methods, 655–656
 - filterViewController property, 641
 - filterWithAffineTransform method, 648, 651
 - Finance role in iTunes Connect, 696
 - Fixed space bar buttons, 111
 - Flashlight app, 9
 - Flexible space bar buttons, 111
 - Flickr, 549–551
 - displaying, 555–557
 - downloading photos, 564–570
 - FlickrViewController class, 570–580
 - PhotoWheel, 551–553
 - view controller scene, 553–555
 - wrapping API, 557–564
 - flickrJSONsWithParameters method, 561–564
 - flickrPhotos property, 577, 579
 - FlickrViewController class, 551–554, 557
 - arrays, 566
 - implementing, 570–580
 - flip method, 78–79, 81–82, 85
 - Flipping images, 651
 - Floating-point number format specifiers, 99
 - FMDB project, 327
 - Fonts
 - labels, 346
 - settings, 26–27
 - Foreground apps, 172
 - Format specifiers, 99
 - forRowAtIndexPath method, 220, 225–226
 - Forwarding messages, 368
 - forwardOnePhoto method, 623–624
 - Foundation.h file, 73
 - Foundation library, 91–92
 - collection classes, 97–99
 - data types, 92–97
 - utility classes and functions, 99–103
 - frameForPageAtIndex method, 462, 466
 - frameForPagingScrollView method, 461–462, 465
 - Frames, 672
 - Framework & Library target, 6
 - Framework bundles, 100
 - Full-screen cameras, 280–281

G

 - Garbage collection, 86
 - GCD (Grand Central Dispatch) API, 580–582
 - Gestures. *See* Touch gestures
 - getExternalScreen method, 617, 619
 - getter attribute, 77
 - Getter methods, 59, 75–77
 - GIF (Graphic Interchange Format) format, 109
 - Git source code repositories, 36, 66, 170
 - Global queues, 580
 - GlobalPhotoKeys class, 287–288, 290, 294
 - Glyphish icon set, 158
 - Google Objective-C Style Guide*, 30

- GPUs for images, 632
- Gradient buttons, 106
- Grand Central Dispatch (GCD) API, 580–582
- Graphic Interchange Format (GIF) format, 109
- Grid view
 - Flickr, 554
 - photos, 434–446
- Grids
 - coordinate systems, 472
 - IB, 46
- GridView class
 - photos, 435–446
 - working with, 446–451
- GridViewCell class, 436
- gridViewCellSize method, 451, 577
- GridViewDataSource protocol, 436, 447, 570
- gridViewNumberOfCells method, 450, 576
- Grouped tables, 193
- GUI PSD template, 157
- Guides for object alignment, 49–51

H

- Harrington, Tom, 298
- Hashes for URI strings, 595
- Header (.h) files, 72, 174, 177, 183–184
- Heads-Up Display (HUD), 16
- Hello World project
 - creating, 3–10
 - functionality, 12–17
 - text, 10–12
- Help
 - online, 236
 - provisioning, 124
 - Quick Help, 34
- Hide System Libraries option, 679–680
- hideChrome method, 480, 482
- hideFilters method, 644–645
- hideOverlay method, 574–575, 579
- HIG, 148
- Hillegass, Aaron, 65
- Hiring designers, 153–154
- Home button, 500–501
- Horizontal guides, 49–50
- HUD (Heads-Up Display), 16
- hueAdjustFilter method, 648

I

- IBAction macro
 - connections, 44
 - description, 14
 - NIB code, 57–61
 - PhotoBrowserViewController, 641
 - table view data, 215
- iBooks, 150–151
- IBOutlet macro
 - connections, 44
 - description, 14
 - NIB code, 57–61
 - PhotoBrowserViewController, 641–643, 646
 - table view data, 213, 215
- IBPlaygroundViewController class, 58–63
- IBPlaygroundViewController.xib file, 46–47
- iCloud service
 - changes from, 602–607
 - concepts, 584
 - device provisioning, 586–591
 - entitlements, 589–591
 - file coordinators and presenters, 584
 - limitations, 592
 - overview, 583
 - PhotoWheel, 592–598
 - ubiquitous persistent stores, 585
 - UIDocument and UIManagedDocument, 585
- .ico files, 109
- Icon*.png files, 341
- Icons
 - apps, 154
 - sets, 158
- IDE (Integrated Development Environment), 19–20
- Identity inspector, 52–54, 353
- Image picker controllers
 - action sheets, 274–278
 - saving photos to camera roll, 283–284
 - working with, 271–274, 278–282
- imageAtIndex method, 460, 469
- imageDataForAttributeNamed method, 597
- ImageDownloader class, 566–570
- imageFilters variable, 640
- ImageGridViewCell class, 451–455

- imageGridViewCellWithSize method, 453
- imagePickerController method, 281, 431, 433–434
- Images
 - caching, 570
 - child scenes, 370
 - classes for, 108
 - Core Image. *See* Core Image effects
 - downloading, 340
 - enhancement, 636–637, 652–653
 - face detection, 637–638
 - flipping, 651
 - ImageGridViewCell, 451–455
 - Mockup apps, 158–160
 - photo album thumbnails, 425–429
 - rotating, 520–522
 - scaling, 594, 596
 - segues, 360–362
 - storyboards, 339–340
- imageTapped method, 481–482
- imageView property, 453–454
- Immutable classes, 92
- iMockups app, 159–160
- @implementation directive, 79, 177
- Implementation of classes, 78–82
- #import statements, 73, 84
- Indentation preferences, 28–29
- Index cards, 143
- Index paths, 199, 223
- index property, 486
- indexesForSelectedCells method, 436, 444, 446
- indexForSelectedCell method, 436, 444, 446
- indexForSelectedGridCell method, 471–472
- indexInWheelView property, 396
- indexPath property, 223
- Indistinct objects, 98
- Industrial design, 149–150
- info.plist file
 - Bundle IDs, 119
 - launch images, 521–522
 - PhotoWheel app, 174
 - storyboards, 330
- Info settings, 36
- Information hiding, 75
- Inheritance, 175
- init method
 - CoinTosser, 79–81, 85
 - GridView, 438
 - ImageGridViewCell, 452–454
 - prototype code, 389, 397
 - SimpleDownloader, 102
 - spin gesture recognizers, 263, 266
- Initial view controllers, 341–344
- initWithPhotoViewCache method, 460–461, 464
- initWithBarButtonSystemItem method, 204
- initWithCalendarIdentifier method, 93
- initWithCoder method
 - GridView, 438, 445
 - prototype code, 389
 - spin gesture recognizers, 263, 266
- initWithDefaultNib method, 212–213, 219
- initWithFrame method
 - GridView, 438
 - prototype code, 389
 - spin gesture recognizers, 263, 266
 - zooming, 484, 487
- initWithNibName method, 189–190, 246, 278–280
- initWithSize method, 453–454
- initWithViewController method, 537–539
- Inspectors area, 24
- Installing
 - certificates, 126–127
 - development provisioning profiles, 137–138
 - Xcode, 708–709
- Instance methods, 78
- Instance variables (ivars), 45
 - Objective-C, 74–75
 - renaming, 80
- Instruments tool, 41, 676–682
- int data type, 74
- Integrated Development Environment (IDE), 19–20
- Interface Builder (IB), 11, 43–44
 - aligning objects, 49–52
 - layout rectangles, 52–53
 - NIB connections to code, 57–63
 - operation, 44–45
 - selecting and copying objects, 48
 - states, 52–57

- storyboards, 63–64, 333
 - working with, 45–48
- @interface directive, 74, 175
- Interfaces
 - Objective-C, 74–75
 - user. *See* User interface (UI)
- Intro to Grand Central Dispatch*, 581
- invalidatingBarButtonItem method, 183
- invertColorFilter method, 648
- iOS
 - device family types, 7
 - targets, 6
 - touch gestures, 253–254
- iOS Configuration Utility, 131
- iOS Dev Center, 125–126
- iOS Developer Program, 115, 703
 - joining, 704–706
 - membership privileges, 703–704
 - registration requirements, 706–708
 - team roles, 116–117
- iOS Developer Program Agreement, 692
- iOS Human Interface Guidelines*, 148
- iOS Provisioning Portal, 124, 131
 - App IDs, 133–135
 - certificates, 684
 - development provisioning profile, 135–138
 - device IDs, 131–133
 - iCloud, 588
 - overview, 115–117
- iOS Simulator
 - external display, 626, 670
 - Printer Simulator, 530
- .ipa files, 688
- iPad, universal apps for, 8
- iPad device family, 7
- iPad Simulator, 9–10
 - iCloud, 589
 - limitations, 41
 - working with, 171–172
- iPhone device family, 7, 426
- iPhone emulator, 8
- iPod touch, 128–129
- isCameraDeviceAvailable method, 273
- isIndexVisible method, 390–391, 397
- isSelectedItemForAngle method, 390, 397
- isSourceTypeAvailable method, 273, 294
- Issue navigator, 22

- Isted, Tim, 298
- isZoomed method, 484–485, 487
- iTunes Connect, 695–698
- ivars (instance variables), 45
 - Objective-C, 74–75
 - renaming, 80

J

- Joint Photographic Experts Group (JPEG)
 - format, 109, 296, 298
- JSON Framework, 549, 562–563
- Jump bar, 23

K

- kCICategoryStillImage category, 633
- kCIContextUseSoftwareRenderer setting, 635
- kCIImageAutoAdjustRed Eye setting, 637
- Key bindings, 31–32
- Key Pair Information window, 122–123
- Key pairs for certificates, 122–123, 127
- Key-value coding (KVC), 300
- Key-value pairs for dictionaries, 97–98
- Key windows, 180
- Keyboards, virtual, 152
- Keychain Access application, 121–124, 127
- Keychain data, 119
- Keynote Wireframe Toolkit, 158
- Keys
 - dictionary, 286–287
 - Flickr, 549–551
 - photos, 298
- Keywords for apps, 694
- KissXML parser, 101
- Kochan, Stephen G., 65
- KVC (key-value coding), 300

L

- Label class, 11, 15
- Labels
 - copying, 48–49
 - creating, 11–12
 - scenes, 346
 - text property, 201
- Labor Mate app icon, 156

- LaMarche, Jeff, 106, 262, 565
 - Landscape orientation, 504
 - helper methods, 509
 - landscape left and landscape right, 500
 - launch images, 521
 - photo browser, 474
 - split view controllers, 169
 - Large app icon, 694
 - largeImage attribute, 595
 - largeImageData attribute, 309, 592–593, 597
 - Launch images, 520–522
 - Launch options, 179–180
 - Launchpad, 3–4
 - Layout rectangles, 52–53
 - layoutForLandscape method, 505–506, 509
 - layoutForPortrait method, 506–507, 509
 - layoutScrollViewSubviews method, 516–517
 - layoutSubviews method
 - GridView, 440–442, 445–446
 - spin gesture recognizers, 264, 266
 - WheelView, 236, 395, 398
 - Leaks tool, 677
 - Learning iPad Programming* blog, 582
 - Learning Objective-C 2.0*, 65
 - Lee, Mike, 520
 - Legal role in iTunes Connect, 696
 - Libraries, 24
 - Foundation. *See* Foundation library
 - Library area, 24–25
 - Object, 11
 - Library target type, 6
 - libxml2 parser, 101
 - Line wrapping, 29
 - Literals, string, 81
 - loadPage method
 - chrome effects, 478–479
 - PhotoBrowserViewController, 462, 466
 - zooming, 488–489
 - loadSubviewsWithFrame method, 484, 487
 - Local variables, 45
 - Objective-C, 74–75
 - renaming, 80
 - localUserInfo dictionary, 606
 - Location Services for photos, 270–271
 - Log Message actions, 665
 - Log navigator, 22
 - Logical conditions in searching and filtering
 - data, 100
 - Long, Matt, 243
 - Long presses, 254
- ## M
- .m files, 72, 78–79
 - Magic Piano app, 149
 - Mail app, 169, 533
 - Mail composition view, 534
 - mailComposeController method, 539
 - main.m file, 69
 - CoinTosser, 84–85
 - PhotoWheel, 174
 - Main queue, 580
 - Main screen, 377–378
 - copying files, 378–380
 - Core Data model, 380–385
 - GridView, 446–451
 - ImageGridViewCell, 451–455
 - photo albums, adding, 408–409
 - photo albums, displaying, 398–408
 - photo albums, managing, 409–410
 - photo albums, naming, 414–421
 - photo albums, removing, 422–424
 - photo albums, selecting, 410–414
 - photo albums, thumbnails, 425–429
 - photos, adding, 429–434
 - photos, displaying, 434–446
 - rotating, 518–520
 - toolbar display, 421–422
 - WheelView, 385–398
 - Main storyboards, 336–337
 - Main View Controller Scene, 371
 - MainSlideShowViewController class
 - advancing photos, 621
 - external displays, 620
 - slideshows, 611–613, 616–620
 - MainStoryboard.storyboard file
 - container view controllers, 369
 - Flickr, 553–554
 - GridView, 446
 - naming photo albums, 415
 - navigation, 356
 - PhotoBrowserViewController, 641
 - pop transitions, 366

- scenes, 356
- scroll view, 466
- toolbars, 422
- View Controller setting, 353–354
- MainViewController class
 - child view controllers, 372
 - container view controllers, 369
 - iCloud, 601–602, 606
 - photo albums, 407, 410–411
 - rotation, 502–507, 518–520
- makeKeyAndVisible method, 179–180, 618
- Making Apps That Don't Suck*, 520
- Manage schemes window, 39
- Managed object contexts
 - creating, 306–307
 - iCloud, 603
 - overview, 300–301
 - photo albums, 318–320, 406–408
- Managed objects, 299–300
- managedObjectContext property
 - iCloud, 603
 - photo albums, 318, 400, 406–408, 413
- managedObjectModel method, 664–667
- Mantissas, 94
- Master–Detail Application template, 6
- Master–Detail apps
 - app delegates, 174–179
 - creating, 170–171
 - detail view controller, 185–186
 - launch options, 179–180
 - master view controller, 186–187
 - project structure, 173–174
 - prototype, 167–173
 - split view controller delegates, 185
 - split view controllers, 168–169, 181–184
- Master view controllers, 169, 186–187
- masterNavigation Controller, 179
- MasterViewController class, 173
 - Core Data stack, 307
 - displaying data, 197–198
 - managed objects, 306
 - master detail apps, 179, 186–187
 - photo albums, 320–325
 - property lists, 288–289, 292
 - table view data, 203, 217–223, 225–226, 228–229
 - table views, 194–197
 - titles, 189–192
- maximumContentOffset method, 513
- Media layer, 90
- Media library, 24–25
- MediaPlayer framework, 626
- Member Center, 589–590
- Memory management, 77, 85–87
 - Allocations tool, 677–678
 - leaks, 83, 85, 434
 - photos, 296
- Menu items, classes for, 108
- mergeChangesFrom_iCloud method, 604
- mergeChangesFromContextDidSaveNotification method, 606
- mergeiCloudChanges method, 605–606
- Merging iCloud changes, 605–606
- Message UI Framework, 533–536
- Messages
 - debugger, 667
 - forwarding, 368
 - to nil objects, 214
 - sending, 99
 - SMS, 533
- Metaphors in UI design, 150–151
- Methods
 - copying, 184
 - Objective-C, 78
- MFMailComposeViewController class, 534–536, 540
- MFMailComposeViewControllerDelegate protocol, 535
- Mini toolbar, 23, 25
- minimumContentOffset method, 513
- Mirror image filters, 651
- .mobileprovision files, 138
- MockApp template, 158
- Mockup apps, 154
 - necessity, 156
 - overview, 154–155
 - PhotoWheel, 167–168
 - tools, 156–160
 - wireframes, 158–160
- Model editor, 307–308
- Model objects, custom code for, 315–320

- Model-View-Controller (MVC) design
 - pattern, 15, 112–113
- mogenerator tool, 314–315
- .momd files, 304–305
- Motion events, 253
- motionBegan method, 253
- motionCancelled method, 253
- motionEnded method, 253
- Mouse clicks in design, 148–149
- moveRowAtIndexPath method, 226–227
- Moving guides, 49
- MPVolumeView class, 626–628
- Multiple Xcode versions, 709
- Multitasking, 172
- Multithreaded applications, 77
- Mutable classes, 92
- MVC (Model-View-Controller) design
 - pattern, 15, 112–113
- myBalsamiq app, 160

N

- Name editor for albums, 322–323
- NameEditorView class, 217
- NameEditorViewController class
 - albums, 322
 - property lists, 292
 - table view data, 207–212, 215, 217–225
- NameEditorViewControllerDelegate
 - protocol, 210
- nameEditorViewControllerDidCancel
 - method, 210, 214, 218–219
- nameEditorViewControllerDidFinish
 - method
 - albums, 322–323
 - property lists, 292–293
 - table view data, 210, 214, 218–222
- Names
 - albums, 322–323
 - apps, 142, 693
 - ivars, 80
 - organization, 38–39
 - parameters, 211
 - photo albums, 414–421
 - registered devices, 132
- Navigation area, 22–23
- Navigation bar
 - classes for, 109–110
 - scenes, 356–358
 - segues, 362
- navigationItem property, 205
- Navigator
 - debug, 666–667
 - descriptions, 22
- New project window, 4–5
- newPhotoAlbumWithName method,
 - 289–290, 292–293
- NIB files, 11
 - connections to code, 57–63
 - overview, 44
 - vs. storyboards, 330–331
- nibWithNibName method, 426
- nil objects, messages to, 214
- nil value for properties, 86–87
- No Access role, 117
- nonatomic attribute, 77
- Nonvisible rows, 200
- Notifications in iCloud, 601–602
- NSArray class
 - description, 97
 - property lists, 286–287
 - table views, 195
- NSBundle class, 99–100
- NSCalendar class, 93
- NSConfinementConcurrencyType setting,
 - 604
- NSCountedSet class, 98–99
- NSData class, 92–93, 286–287, 290
- NSDate class, 93, 286
- NSDateComponents class, 93–94
- NSDateFormatter class, 100
- NSDecimalNumber class, 94–95
- NSDefaultRunLoopMode mode, 569
- NSDictionary class
 - description, 97–98
 - Flickr, 563
 - iCloud, 606
 - image picker controllers, 281
 - property lists, 286–287, 290
- NSEntityDescription class, 299
- NSError class, 563
- NSFetchedResultsController class, 606

- NSFetchedResultsControllerDelegate protocol, 400, 404, 447
 - NSFileCoordinator class, 584
 - NSFileManager class, 100
 - NSFilePresenter protocol, 584–585
 - NSIndexPath type, 194, 294
 - NSInteger class, 95
 - NSJSONSerialization class, 549, 562–563
 - NSLocale keys, 93
 - NSLocalizedString function, 190
 - NSLog function
 - breakpoints, 665
 - CoinTosser, 85
 - debugging, 674–676
 - description, 59, 99
 - output, 70
 - for testing, 625
 - NSMainNibFile setting, 330
 - NSMainQueueConcurrencyType setting, 603
 - NSManagedObject class, 299–300, 310–315
 - NSManagedObjectContext class, 300–301, 413, 606
 - NSManagedObjectContextDidSaveNotification notification, 606
 - NSMergeByPropertyObjectTrumpMergePolicy setting, 604
 - NSMutableArray class, 97, 288
 - NSMutableData class, 92–93
 - NSMutableDictionary class, 97–98, 290
 - NSMutableOrderedSet class, 201, 288
 - NSMutableSet class, 98–99
 - NSMutableString class, 96–97
 - NSNotification observer, 291
 - NSNotificationCenter, 602
 - NSNull class, 96
 - NSNumber class, 95–96, 286–287
 - NSNumberFormatter class, 100
 - NSObject class, 73, 96
 - NSOrderedSet class, 195
 - NSPersistentStoreCoordinator class, 301
 - NSPersistentStoreDidImportUbiquitousContentChangesNotification notification, 603–605
 - NSPersistentStoreUbiquitousContentNameKey setting, 598
 - NSPersistentStoreUbiquitousContentURLKey setting, 598, 600
 - NSPredicate class, 100
 - NSPropertyListMutableContainers class, 290
 - NSPropertyListSerialization class, 290
 - NSRegularExpression class, 101
 - NSRunLoopCommonModes mode, 569
 - NSSet class, 98–99
 - NSString class, 74, 96–97, 286–287
 - NSStringFromClass function, 404
 - NSTimer class, 101, 477, 565, 621
 - NSUInteger class, 95
 - NSURL class, 527, 548–549
 - NSURLConnection class
 - description, 102, 549
 - Flickr, 563
 - photos, 565–570
 - NSURLConnectionDelegate protocol, 549, 570
 - NSURLRequest class, 102, 548–549
 - NSURLResponse class, 563
 - NSXMLParser class, 101, 549
 - NSXMLParserDelegate protocol, 101
 - Null-terminated char array format specifiers, 99
 - NULL value, 96
 - numberOfCells method, 389–390, 397
 - numberOfPhotos method, 460
 - numberOfRowsInSection method, 198–199
 - numberOfRowsInTableView method, 198–199
 - numberOfTapsRequired property, 255
 - numberOfTouchesRequired property, 255
 - numberOfVisibleCells method, 390, 397
- ## O
- Object library, 11, 24–25
 - objectAtIndex method, 412
 - objectID property, 413, 595
 - Objective-C, 13, 19–20, 65
 - classes, 71–73
 - declared properties, 75–78
 - dot syntax, 83–84
 - implementation, 78–82
 - instance variables, 74–75
 - interfaces, 74–75
 - memory management, 85–87
 - methods, 78

- Objective-C, (*continued*)
 - objects, 70–71
 - overview, 65–66
 - selectors, 82–83
 - working with, 66–70
 - Objective-C Programming*, 65
 - ObjectiveFlickr framework, 557
 - Objects
 - aligning, 49–52
 - managed, 299–300. *See also* Managed
 - object contexts
 - model, 315–320
 - overview, 70–71
 - selecting and copying, 48
 - size, 54–55
 - OCSP (Online Certificate Status Protocol), 121
 - OmniGraffle app, 160
 - On/Off button, classes for, 108
 - Online Certificate Status Protocol (OCSP), 121
 - Online help, 236
 - OpenGL Game template, 6
 - Opening header files, 184
 - Optimization, 676–682
 - Option key for objects, 51–52
 - Option-Click
 - copying objects, 48
 - documentation popover, 183
 - names, 184
 - Quick Help popup, 34
 - @optional directive, 214
 - Organization name, 38–39
 - Organization Profile, 590
 - Organizer window
 - app submissions, 696–697
 - description, 40
 - device setup, 128–130
 - UUIDs, 132
 - Orientation. *See also* Rotation
 - launch images, 521
 - photo browser, 474
 - split view controllers, 169
 - supported, 499–501
 - originalImageData property, 308
 - Outlets, 57
 - connecting objects to, 16
 - defining, 58–61
 - IBOutlet. *See* IBOutlet macro
 - overlayView property, 553
 - overlayViewTapped method, 575, 579
 - Owners of files, 56
- ## P
- PADDING macro, 465
 - Page-Based Application template, 6
 - Page control, 107
 - Panning, 254
 - Paper and pencil for Mockup apps, 157
 - Parameters
 - Flickr, 562–563
 - methods, 78
 - names, 211
 - Parsers, 101
 - Paths
 - attributes, 595
 - index, 199, 223
 - pause method, 623–624, 679–681
 - Penultimate app, 157
 - perform method for segues
 - custom, 355
 - CustomPushSegue, 472–474
 - implementing, 358–362
 - push, 375
 - Performance, 676–682
 - performBlock method, 603–604
 - performBlockAndWait method, 603
 - performSegueWithIdentifier method, 346
 - performSelector method, 82, 434
 - Persistence. *See* Data persistence
 - Persistent stores and persistent store coordinators, 301
 - creating, 305–306
 - iCloud, 585
 - ubiquitous, 598–602
 - persistentStoreCoordinator method, 598–600, 664
 - Person interface, 76, 177–178
 - Photo albums, 286
 - adding, 400–401, 404, 408–409
 - adding photos to, 293–297, 323–325, 429–434
 - displaying, 398–408

- displaying photos in, 297–298, 326
- managed object contexts, 318–320
- managing, 409–410
- naming, 414–421
- reading and saving, 288–293, 320–323
- removing, 422–424
- scene rotation, 507
- selecting, 410–414
- thumbnails, 425–429
- toolbars, 421–422
- Photo Albums View Controller Scene, 399
- Photo browser
 - chrome effects, 477–482
 - deleting photos, 489–497
 - launching, 467–469
 - push and pop, 470–477
 - rotating, 511–517
 - scroll view, 457–467
 - slideshows, 624–625
 - user interface, 466–467
 - zooming, 482–489
- Photo class, 315
 - email, 539–540
 - iCloud, 594–596
 - prototype code, 381–385
- Photo entity, 380
- PhotoAlbum class, 313–314, 318–320
- PhotoAlbum entity, 309–313, 380
- photoAlbumPath method, 289–290
- photoAlbumSaveNeeded method, 290–291
- PhotoAlbumsViewController class
 - child scenes, 370
 - child view controllers, 373
 - container view controllers, 369
 - implementing, 400–406
 - managed object context, 406–408
 - photo albums, adding, 404, 408–409
 - photo albums, displaying, 399
 - photo albums, selecting, 410–413
 - scene rotation, 507–511
- PhotoAlbumViewController class
 - child scenes, 370–371
 - child view controllers, 373–375
 - container view controllers, 369
 - email, 535, 540, 544–546
 - Flickr, 555–557
 - grids, 446–447
 - GridView, 446
 - iCloud, 602, 606
 - photo albums, naming, 415–421
 - photo albums, removing, 422–424
 - photo albums, selecting, 410–413
 - photo browser, 467–470, 474
 - photos, adding, 429–434
 - photos, deleting, 496–497
- PhotoBrowserPhotoView class, 483–487, 511–514
- PhotoBrowserViewController class, 457–467, 469
 - chrome effects, 477–482
 - Core Image effects, 638
 - deleting photos, 490–497
 - email, 535, 540–544
 - printing, 528–530
 - rotation, 514–517
 - slideshows, 612, 616, 624–625
 - user interface additions, 640–647
 - zooming, 482, 487–489
- PhotoBrowserViewControllerDelegate proto-
col, 458–459
 - Core Image effects, 638–639
 - deleting photos, 495
 - slideshows, 613
- photoBrowserViewControllerNumberOf-
Photos method, 469
- PhotoBrowserWheelController class, 647
- Photos, 269
 - adding to albums, 293–297, 323–325, 429–434
 - advancing, 620–621
 - Assets Library framework, 269–271
 - custom code for, 315–320
 - data model, 285
 - deleting, 489–497
 - displaying, 297–298, 326, 434–446
 - downloading, 564–570
 - iCloud, 598–606
 - image picker controller. *See* Image picker
controllers
 - saving to camera roll, 283–284
 - photos property, 536
 - photoSetListWithUserId method, 560
 - Photoshop, 157
 - photosWithPhotoSetId method, 560

- photosWithSearchString method, 559, 562
- photoTapped method, 620
- photoViewCache property, 463
- PhotoWheel app, 167
 - app delegates, 174–181
 - charter, 146–147
 - Core Image effects. *See* Core Image effects
 - custom views. *See* Custom views
 - data persistence. *See* Data persistence
 - debugging. *See* Debugging
 - detail view controller, 185–186
 - device rotation. *See* Rotation
 - distributing. *See* Distribution
 - email, 535–546
 - iCloud. *See* iCloud service
 - launch options, 179–180
 - main screen. *See* Main screen
 - master view controller, 186–187
 - photo browser. *See* Photo browser
 - photos. *See* Photo albums; Photos
 - printing, 527–531
 - project structure, 173–174
 - prototype, 167–173
 - slideshows. *See* Slideshows
 - split view controller, 181–185
 - storyboarding. *See* Storyboarding
 - table views. *See* Table views
 - target audience, 146
 - touch gestures. *See* Touch gestures
 - utility methods, 655–656
 - view controllers. *See* View controllers
 - Web services. *See* Web services
- PhotoWheel-Info.plist file, 338, 341
- PhotoWheel-Prefix.pch file, 601
- PhotoWheel.xcdatamodeld file, 380
- PhotoWheelAppDelegate protocol, 603, 664
- PhotoWheelPrototype app, 171–172, 302
- PhotoWheelPrototype-Info.plist file, 174
- PhotoWheelPrototype-Prefix.pch file, 174, 303
- PhotoWheelPrototype.xcdatamodeld file, 304, 307, 380
- PhotoWheelPrototypeAppDelegate.h file, 304
- PhotoWheelViewController class, 248
 - header files, 248
 - implementation, 249–250
 - photo album thumbnails, 425–429
 - touch gestures, 255–257
 - working with, 250–252
- photoWheelViewController method, 425–426
- Pick lists, 107
- Pinch gesture, 254
- placeholder property, 105, 415
- Placeholders
 - code completion, 33
 - fault objects, 300
 - File's Owner, 56
 - text, 105
- Plain tables, 193
- play method, 679–681
- PLDatabase project, 327
- plist files, 330
- Plug-ins, 100
- PNG (Portable Network Graphic) format, 109, 296
- Pointer address format specifiers, 99
- Pointers, 74
- pointToCenterAfterRotation method, 513
- Pop segues
 - customizing, 364–367
 - improving, 470–477
- Popover segues, 355
- popToRootViewControllerAnimated method, 356
- popToViewController method, 356
- popViewControllerAnimated method, 356, 365–366, 475–476
- Portable Network Graphic (PNG) format, 109, 296
- Portal Resources, 124–125
- Portrait orientation, 500
 - helper methods, 509
 - launch images, 521
 - portrait upside down, 500
 - split view controllers, 169
- Position
 - child content view, 374–375
 - guides, 49–50
- Possible memory leaks, 434
- Possible state, 258, 261
- Post-It Notes, 143
- #pragma mark statements, 184
- Pragmatic Programmer: From Journeyman to Master*, 231

- Predefined touch gestures, 254
- Preferences, 26
 - code completion, 33
 - coding style, 30
 - development certificates, 121–122
 - fonts and colors, 26–27
 - key bindings, 31–32
 - text, 27–30
- preferredMode property, 611
- Premature optimization, 676
- prepareForSegue event
 - Flickr, 556–557
 - photo browser, 468
 - scenes, 333
 - slideshows, 624
- Presentation property, 355
- presentCamera method, 276, 278, 280, 432
- Presenters in iCloud, 584
- presentFlickr method, 556–557
- presentPhotoLibrary method
 - adding photos, 432
 - camera checking, 273–274
 - image picker controllers, 278, 280
- presentPhotoPickerMenu method
 - action sheets, 276–277
 - adding photos, 430, 432–433
 - camera checking, 273–274
 - Flickr, 555–556
- __PRETTY_FUNCTION__ macro, 205–206, 255
- Price of apps, 693
- Primary app categories, 693
- Primitive data types, 74
- primitiveValueForKey method, 598
- Print Center, 526
- Print jobs, 526
- printCurrentPhoto method, 528–530
- Printer Options view, 525
- Printer Simulator, 530–531
- printFormatter property, 527
- printInfo property, 530
- Printing
 - API, 527
 - operation, 525–526
 - PhotoWheel, 527–531
 - requirements, 526
- printingItem property, 530
- Private key pairs, 123–124
- Private keys, 127
- Profiling code with Instruments, 676–682
- Programming in Objective-C 2.0*, 65
- Project navigator, 22
- Project options screen, 7
- Project template, 5–6
- Projects
 - creating, 3–10, 170–171
 - settings, 36–39
- Properties
 - declared, 75–78
 - dot syntax, 83–84
 - filters, 633–634
 - objects, 70–71
 - transient, 592–593
 - values, 54
- @property directive, 58–59, 75–77, 210
- Property lists
 - adding photos to albums, 293–297
 - data models, 287–288
 - displaying photos in albums, 297–298
 - overview, 286–287
 - reading and saving photo albums, 288–293
- Property synthesis, 14
- @protocol directive, 613
- Prototype apps and code, 160–163, 167–168
 - copying files, 378–380
 - Core Data model, 380–385
 - project creation for, 170–171
 - reusing, 378
 - simulators, 171–172
 - split view controllers, 168–169
 - WheelView, 385–398
- Provisioning, 115
 - Ad Hoc distribution, 684
 - App IDs, 118–119
 - App Store distribution, 688
 - development machine setup, 121–128
 - development provisioning profiles, 119–120
 - device IDs, 117–118
 - device setup, 128–130
 - iCloud service, 586–591
 - iOS Provisioning Portal. *See* iOS Provisioning Portal
 - overview, 117
- Public key pairs, 123–124
- Public keys, 127

Push segues, 355
 container view controllers, 375
 improving, 470–477
 pushViewController method, 356
 PW_Default.png file, 522
 pw_imageSnapshot method, 363
 PWDefault-landscape.png file, 522

Q

QuartzCore.h file, 249
 queueNonVisibleCells method, 391, 398
 queueReusableCells method
 GridView, 439, 445
 WheelView, 395, 398
 Queues, dispatch, 580–581
 Quick Help popup, 34
 Quitting apps, 172
 Quotation marks (") for strings, 96

R

Radians, converting, 266
 RAND_IN_RANGE macro, 647
 Random CIFilter effects, 647–651
 Random numbers, 70, 81
 randomCIColor method, 647
 randomCIColorAlpha method, 647
 randomizeFilters method, 645, 649–650
 Rating apps, 694
 Reachability, 582
 Reading photo albums, 288–293, 320–323
 readonly attribute, 77
 readSavedPhotoAlbums method, 289–291
 readwrite attribute, 77
 receivedData property, 568–570
 Receivers in Objective-C, 66
 Recipes in Core Image, 631–632
 Recognized state, 258
 Recognizers
 spin gesture, 259–266
 touch gesture, 254–258
 Recommended books, 702
 Red-eye correction, 637
 Reference counting, 7, 66, 86–87, 170
 Registering devices, 118
 Regular expressions, 101

Rejected apps, 692
 Relationships with Core Data entities,
 309–311
 Release build configurations, 685
 reload method
 GridView, 448–449
 naming photo albums, 417–420
 selecting photo albums, 413–414
 reloadData method
 GridView, 439, 445
 photo albums, 405
 photo display, 436
 table view data, 220
 WheelView, 387, 396, 398
 removeFromParentViewController method,
 367–368
 removePhotosObject method, 313
 Removing
 breakpoints, 664
 guides, 49
 photo albums, 422–424
 Renaming ivars, 80
 Rentzsch, Jonathan "Wolf", 315
 Reordering table view data, 226–227
 Requesting development certificates, 121–124
 requireGestureRecognizerToFail method,
 257–258
 resignFirstResponder method, 421
 Resizing
 labels, 12
 scenes, 370
 Resolving conflicts. *See* iCloud service
 respondsToSelector method, 214
 RESTful Web services
 Cocoa, 548–549
 description, 548
 Flickr, 557
 restoreCenterPoint method, 514
 resume method, 623–624
 retain attribute, 77
 reusableCells property, 397
 Reverse domain name style, 118
 revertToOriginal method, 641, 655–656
 Review notes, 694
 Roles
 iOS Developer Program, 116–117
 iTunes, 696

Root view controllers, 110
 rootViewController property, 179–180, 331
 Rotation
 About screen, 511
 customized, 502–507
 gesture type, 254
 launch images, 520–522
 main screen, 518–520
 MainViewController, 502–507
 photo browser, 474, 511–517
 scenes, 507
 spin gesture recognizers, 262, 266
 split view controllers, 169
 supporting, 499–502
 WheelView, 509–511
 Round Rect Button, 15
 Rounding calculations, 95
 row property, 194
 Run button, 9
 Runtime loops, 565, 569

S

Sales role in iTunes Connect, 696
 save method in Flickr, 553, 573–574, 578
 saveChanges method, 419–420
 saveContext method, 305
 saveContextAndExit method, 572
 saveImage method
 Core Image effects, 639
 filters, 655–656
 Photo class, 382, 384–385, 594
 PhotoBrowserViewController, 641
 photos, 315, 317, 325
 savePhotoAlbum method, 289–290, 293
 saveSelectedPhotos method, 572–573, 578
 Saving
 images, 296
 photo albums, 288–293, 320–323
 photos to camera roll, 283–284
 scaleAndCropToMaxSize method, 316–317, 383–384
 scaleAspectToMaxSize method, 315, 383
 scaleToRestoreAfterRotation method, 513
 Scaling images, 594, 596
 Scenes, 331–332
 child, 369–371
 creating, 344–346
 navigating, 356–358
 resizing, 370
 rotation, 507
 setting, 355–358
 Scheme manager window, 689–690
 Schemes, 39, 689–690
 Schneider, Michael, 699
 Scope depth of code, 28
 Screen
 About, 344–347, 511
 classes for, 104
 main. *See* Main screen
 Screen shots of apps, 694–695
 screens method, 610
 Scroll view, 457–467
 Scrollable views, 107
 Scrolling in design, 150
 scrollToIndex method, 461, 465
 scrollViewDidScroll method, 463, 466, 517
 scrollViewWillBeginDragging method, 479, 482
 Search navigator, 22
 searchBarCancelButtonClicked method, 576
 searchBarSearchButtonClicked method, 576, 579
 searchBarShouldBeginEditing method, 575
 searchBarTextDidEndEditing method, 576
 Searches
 for data, 100
 text-based, 553
 Secondary app categories, 693
 section property, 194
 Security, 582
 segmentedControlValueChanged method, 247
 Segments, 111
 Segues, 332–333
 creating, 346–348, 355
 description, 355
 implementing, 358–362
 improving, 470–477
 photo browser, 467–468
 push, 375
 scene setting, 355–358
 selected property, 437, 453
 selectedCellFrame method, 470–472, 474, 477
 selectedImage method, 470–472

- selectedIndex property, 387
- selectedIndicator property, 453–454
- selectedWheelViewCellIndex property, 294, 325
- Selecting
 - objects, 48
 - photo albums, 410–414
 - table view data, 227–230
- @selector directive, 82, 205
- Selectors, 82–83
- self, 80, 82–83
- Semicolon characters (;) for declared properties, 76
- sendEmail method, 537–539
- SendEmailController class
 - overview, 535–540
 - working with, 540–546
- sendEmailController property, 543
- SendEmailControllerDelegate protocol, 536–537
- sendEmailControllerDidFinish method, 537–538, 543
- sender method in table views, 204
- Senders in Objective-C, 66
- Sending messages, 99
- sepiImageFromImage method, 635
- Serial numbers, 132
- Serialization, 44
- Session 120 - Simplifying Touch Event Handling with Gesture Recognizers* video, 258
- setAngle method
 - Carousel view, 242–244
 - spin gesture recognizers, 263–264, 266
 - WheelView, 235–236, 392, 398, 510
- setAngleOffset method, 510
- setBounds method, 672
- setCurrentIndex method
 - external displays, 620, 671
 - PhotoBrowserViewController, 463, 466
 - slideshows, 614–616
- setDetailViewController method, 307
- setFilterButtons method, 646
- setFrame method, 673
- setHeadsCount method, 83
- setImage method
 - photo album thumbnails, 428
 - PhotoWheelViewCell, 248–249
 - zooming, 484, 486–487
- setImageData method, 596
- setLargeImageData method, 595–596
- setLastResult method, 82–83
- setMaxMinZoomScalesForCurrentBounds method, 512–513
- setPhotoAlbum method, 293, 297, 324, 326
- setPrimitiveValue method, 596, 598
- sets, 98–99
- setScrollViewContentSize method, 461, 464
- setSelected method, 453–454
- setSmallImageData method, 595–596
- setStyle method
 - Carousel view, 242
 - spin gesture recognizers, 264–265
 - WheelView, 395, 398
- setter attribute, 77
- Setter methods, 59, 75–77
- setText method, 429
- setThumbnailImageData method, 596
- setTitleWithCurrentIndex method, 461, 465
- sharedApplication method, 408
- Sharing schemes, 39
- Shortcut keys
 - key bindings, 31–32
 - navigator, 23
- shouldAutorotateToInterfaceOrientation method
 - About screen, 511
 - adding, 502–503
 - autosizing, 501
 - Flickr, 552, 572
 - MainViewController, 505–506
 - overriding, 499
 - Photo Browser, 511
 - PhotoBrowserViewController, 515
 - scenes, 507
 - slideshows, 616
 - table view data, 212, 214
- Show Obj-C Only option, 680
- showActionMenu method
 - deleting photos, 494
 - email, 541, 545
 - naming photo albums, 420
 - printing, 528–529
 - removing photo albums, 422–423
- showFilters method, 644–645, 649
- showFromBarButtonItem method, 277
- showFromRect method, 278

- showFromTabBar method, 277
- showFromToolBar method, 277
- showOverlay method, 574, 579
- showOverlayCount property, 578
- Shows Navigation Bar property, 357–358
- shrinkToPoint value, 476–477
- Signed integer format specifiers, 99
- SimpleFlickrAPI class, 558–564
- Simulators, 41
 - iOS Simulator, 530, 626, 670
 - iPad Simulator. *See* iPad Simulator
 - Printer Simulator, 530–531
 - schemes, 39
- Single inheritance, 175
- Single View Application template, 6–7
- Size
 - child content view, 374–375
 - labels, 12
 - objects, 54–55
 - rotation, 501–502
 - scenes, 344–346, 370
 - text fields, 215–216
- Size inspector
 - autoresizing, 501
 - description, 54–55
 - scenes, 344–346
- sizeThatFits method, 628
- skipRotation property, 520
- SKU numbers, 693
- slideAdvanceTimer variable, 616
- Sliders, 152
- slideshow method, 494
- Slideshowes
 - adding, 611–612
 - AirPlay support, 626–629
 - displaying, 613–616
 - external display. *See* External display
 - photo advancing, 620–621
 - photo browser, 624–625
 - profiling example, 679–682
 - storyboards, 612
 - user interface controls, 622–624, 679–682
- SlideShowViewController class, 611–616, 618, 671
- Slow motion animation, 362
- smallImage attribute, 382, 595
- smallImageAtIndex method, 639
- smallImageData attribute, 592, 597
- Smalltalk language, 66
- SMS messages, 533
- SOAP-based Web services, 547–548
- Sort descriptor for photo albums, 405
- Sorting filter buttons, 646–647
- Sound effects, 151–152
- Source code repositories, 36–37, 66, 170
- Source image view for segues, 361
- sourceViewController property, 470
- Spaces vs. tabs, 29
- Spin gesture recognizers
 - creating, 259–262
 - working with, 262–266
- spin method
 - spin gesture recognizers, 265–266
 - WheelView, 395, 398
- SpinGestureRecognizer class, 259–261, 265–266
- Split view controllers
 - delegates, 185
 - overview, 168–169
 - working with, 181–184
- SQLite, 299, 301, 305, 326–327
- stack-add.png file, 343
- stack-bg.png file, 343
- stack-overlay.png file, 427
- stack-viewer-bg-portrait.png file, 343–344
- stack-viewer-shadow.png file, 447
- Stacks
 - Cocoa, 89
 - Core Data, 303–307
- Standard Company program type, 704
- Standard editor, 35, 47
- Standard Individual program type, 704
- startAtIndex property, 457, 468
- startChromeDisplayTimer method, 480–482
- startImmediately property, 569
- startIndex property, 613, 616, 618
- States
 - gesture recognizers, 258–259
 - objects, 52–57
- Static sets, 98
- statusBarHeight property, 481
- Stencils, 160
- Step into button, 666
- Step out button, 666
- Step over button, 666
- Stopping apps, 10

- Storyboarding, 8, 63–64
 - app icon, 341
 - AppDelegate, 339
 - Flickr, 553–554
 - images, 339–340
 - main, 336–337
 - overview, 329–330
 - scenes, 331–332, 344–346
 - segues, 332–333, 346–348
 - slideshows, 612
 - UIMainStoryboardFile setting, 338
 - view controllers. *See* View controllers
 - working with, 330–331
 - workspace, 333–336
 - stringByRemovingFlickrJavaScript method, 561, 564
 - Strings
 - arrays, 195
 - classes, 96–97
 - converting to dates, 100
 - format specifiers, 99
 - literals, 81
 - stringWithData method, 561, 564
 - strong attribute, 77, 86
 - Style property
 - bar buttons, 111
 - Carousel view, 241–242
 - segues, 355
 - Styles, coding, 30
 - Subclass generation, 314–315
 - Submitting apps, 9–10, 696–698
 - Subversion source code repositories, 36, 66
 - Summaries for apps, 142
 - super keyword, 81
 - Swipe gesture, 254
 - Symbol navigator, 22
 - Syncing. *See* iCloud service
 - @synthesize directive, 80, 83
 - AppDelegate, 177–178
 - data instances, 196
 - description, 59
 - spin gesture recognizers, 265
- T**
- Tab bar classes, 110–111
 - Tab key, 29
 - Tabbed Application template, 6
 - Table views
 - adding data, 203–220
 - classes, 106, 193–194
 - deleting data, 225–226
 - displaying data, 197–203
 - editing data, 220–225
 - reordering data, 226–227
 - selecting data, 227–230
 - simple models, 195–197
 - working with, 194–195
 - Tagged Image File Format (TIFF) format, 109
 - Tap gestures, 254
 - Flickr, 578
 - PhotoWheelViewCell, 255–257
 - zooming, 487
 - tapped method, 486
 - Tapworthy apps, 148
 - Tapworthy: Designing Great iPhone Apps*, 148
 - Target-Action pattern, 113
 - Target audience for apps, 144–145, 146
 - Targets
 - settings, 38
 - types, 5–6
 - Team Admins, 116, 126, 132
 - Team Agents, 116, 132, 134
 - Team IDs, 590
 - Team Members, 116, 132
 - Team roles in iOS Developer Program, 116–117
 - Technical role in iTunes Connect, 696
 - Templates, 5–6
 - creating, 208–209
 - types, 6
 - Testing, 625–626
 - Text
 - classes for, 104–105
 - labels. *See* Labels
 - preferences, 27–30
 - on screen, 10–12
 - Text-based searches, 553
 - textFieldDidEndEditing method, 419–420
 - textFieldShouldBeginEditing method, 419–420
 - textFieldShouldReturn method, 419–421
 - Third-party apps for photos, 269
 - Threads
 - atomic properties, 77
 - GCD (Grand Central Dispatch), 580–581
 - NSManagedObjectContext class, 413

- thumbnailImage property, 595
 - thumbnailImageData property, 309, 592, 597
 - Thumbnails for photo albums, 425–429
 - TIFF (Tagged Image File Format) format, 109
 - Time
 - classes, 93
 - formatting, 100
 - pick lists, 107
 - Time Machine, 127
 - Time Profiler tool, 677–678
 - Timers
 - chrome effects, 477
 - classes, 101
 - slideshow photos, 621
 - Titles, 189–192
 - titleLabel property, 246
 - toggleChrome method, 479–480, 482
 - toggleChromeDisplay method
 - chrome effects, 479, 482
 - PhotoBrowserViewController, 641
 - zooming, 487–489
 - Toll-free bridging, 92
 - Toolbars
 - classes for, 110–111
 - photo albums, 421–422
 - PhotoWheel, 641, 645–646
 - scenes, 344–345
 - slideshows, 623
 - workspace window, 20–22
 - Touch gestures
 - custom, 258–266
 - in design, 148–149
 - events, 253
 - overview, 253–254
 - predefined, 254
 - recognizers, 255–258
 - types, 254
 - zooming, 487
 - Touch Up Inside events, 60
 - touchesBegan method, 253–254, 258–260
 - touchesCancelled method, 258–260
 - touchesEnd method, 259–261
 - touchesEnded method, 253–254, 258, 260
 - touchesMoved method, 253, 258–261
 - TouchJSON library, 549
 - TouchXML parser, 101
 - Transient Core Data attributes, 592–593
 - Transition property for segues, 355
 - transitionFromViewController method, 368
 - Transitions
 - pop, 364–367, 470–477
 - scenes, 356–357
 - segues, 332–333
 - turnOffZoom method, 485–487
 - TV Out options, 626
- ## U
- Ubiquitous content. *See* iCloud service
 - Ubiquitous persistent store coordinators, 598–602
 - Ubiquitous persistent stores, 585
 - UDIDs (Unique Device Identifiers), 117–118, 132–133
 - UI. *See* User interface (UI)
 - UIActionSheet class, 274, 277
 - UIActionSheetDelegate protocol, 277, 422
 - UIActivityIndicatorView class, 553
 - UIAlertView view, 423
 - UIApplication class, 103–104, 408
 - UIApplicationDelegate class, 179–181
 - UIApplicationMain function, 174
 - UIBarButtonItem class, 415
 - UIBarButtonItem class, 111, 182
 - profiling example, 681–682
 - slideshows, 623
 - table view data, 220
 - UIBarButtonItemSystemItemAdd button, 204
 - UIButton class, 25, 105
 - UIDatePicker class, 107
 - UIDocument class, 585
 - UIGestureRecognizer class, 254–255, 258–259
 - UIGestureRecognizerDelegate protocol, 255
 - UIGestureRecognizerStateBegan state, 258
 - UIGestureRecognizerStateCancelled state, 258
 - UIGestureRecognizerStateChanged state, 258
 - UIGestureRecognizerStateEnd state, 258
 - UIGestureRecognizerStateFailed state, 258
 - UIGestureRecognizerStatePossible state, 258
 - UIGestureRecognizerStateRecognized state, 258

- UIGestureRecognizerSubclass.h file, 258–259
- UIImage class
 - conversions with, 296, 318
 - email, 539–540
 - with filters, 635–636
 - format support, 108–109
 - model objects, 315
 - slideshows, 615
- UIImageJPEGRepresentation method, 296
- UIImagePickerController class
 - adding photos, 324, 430, 434
 - displaying photos, 297
 - working with, 271–274, 278–282
- UIImagePickerControllerDelegate protocol
 - adding photos, 272, 429–430
 - image picker controllers, 279, 281
- UIImagePickerControllerSourceTypePhotoLibrary source type, 281
- UIImageView class
 - child scenes, 370
 - description, 108
 - photo album thumbnails, 426
 - PhotoWheelViewCell, 248
 - storyboards, 341–343
 - wheel view, 233
 - zooming, 482, 489
- UIImageWriteToSavedPhotosAlbum function, 283
- UIInterfaceOrientationIsLandscape macro, 507
- UIInterfaceOrientationIsPortrait macro, 507
- UIPagePickerController class, 294
- UIKit classes, 103–112
 - importing, 234
 - Printing API, 527
- UIKit Framework document, 34
- UILabel class, 11, 14, 104
- UILongPressGestureRecognizer gesture, 254
- UIMainStoryboardFile setting, 338
- UIManagedDocument class, 585
- UIMarkupTextPrintFormatter class, 527
- UIMenuController class, 108
- UIMenuItem class, 108
- UINavigationController class, 109
- UINavigationController class
 - description, 110
 - master detail apps, 179, 186
 - pop transitions, 364–366
 - rotation, 518
 - view controllers, 356, 367
- UINavigationControllerDelegate protocol, 279, 429–430
- UIPageControl class, 107
- UIPanGestureRecognizer gesture, 254
- UIPickerView class, 107
- UIPinchGestureRecognizer gesture, 254
- UIPopoverController class, 182, 191
- UIPrintFormatter class, 527
- UIPrintInfo class, 527, 530
- UIPrintInfoOutputPhoto setting, 530
- UIPrintInteractionController class, 527, 530
- UIPrintInteractionControllerDelegate protocol, 527
- UIPrintPageRenderer class, 527
- UIPrintPaper class, 527
- UIResponder class, 253
- UIRotationGestureRecognizer gesture, 254, 259
- UIScreen class, 104, 611
- UIScreenDidConnectNotification notification, 611, 620, 628
- UIScreenDidDisconnectNotification notification, 611, 620
- UIScrollView class, 193
 - description, 107
 - displaying photos, 435
 - photo browser, 457
 - zooming, 482
- UIScrollViewDelegate protocol, 457
- UISearchBar class, 553
- UISegmentedControl class, 111
- UISimpleTextPrintFormatter class, 527
- UISlider class, 108, 152–153
- UISplitViewController class
 - container view controllers, 367
 - iPad Simulator, 172
 - master-detail apps, 168–169, 179
 - methods implementation, 185
 - overview, 181–183
- UISplitViewControllerDelegate protocol, 182
- UIStoryboardSegue class, 355, 358
- UISwipeGestureRecognizer gesture, 254
- UISwitch class, 108

- UITabBar class, 110–111
- UITabBarController class, 367
- UITableView class
 - description, 106
 - displaying data, 197
 - editing data, 220
 - overview, 193
 - reordering data, 226
- UITableViewCell class
 - description, 106, 194
 - styles, 199–200
- UITableViewCellStyleDefault style, 199, 201
- UITableViewCellStyleSubtitle style, 199
- UITableViewCellStyleValue1 style, 199
- UITableViewCellStyleValue2 style, 199
- UITableViewController class, 187, 194, 220–221
- UITableViewDataSource class, 194–195, 198, 226–227
- UITableViewDelegate class, 194–195
- UITableViewRowAnimationFade class, 225
- UITapGestureRecognizer gesture, 254–255
- UITextField class
 - description, 104–105
 - photo albums, 415
 - table view data, 209, 215–216
- UITextFieldDelegate protocol, 420
- UITextView class, 105
- UIToolbar class
 - description, 110–111
 - photo albums, 415
 - slideshows, 623
- UIView class
 - description, 104
 - events, 253
 - Flickr, 553–554
 - photo albums, 399
 - slideshows, 615
 - wheel view, 233, 237
- UIViewAutoresizing settings, 501
- UIViewController class
 - container view controllers, 367–369
 - description, 104
 - storyboards, 351–353
- UIViewPrintFormatter class, 527
- UIView+PWCategory class, 362–363
- UIWebView class, 104
- UIWindow class
 - debugging example, 672
 - description, 104
 - external displays, 618
 - master–detail apps, 179
- Underscores (_) for ivars, 179
- Unicode characters, 96
- Unique Device Identifiers (UDIDs), 117–118, 132–133
- Unique value propositions, 142
- Universal apps, 8
- Universal device family, 7
- universally unique IDs (UUIDs), 296
- University program type, 704
- unloadPage method, 462, 466, 488–489
- Unsigned integer format specifiers, 99
- updateNavBarButtonsForPlayingState method, 622–623, 627–628, 679–680
- updateToNewImage method, 639
- URI string hashes, 595
- URLForUbiquityContainerIdentifier method, 600
- URLs
 - apps, 694
 - Flickr, 562–564
- Use Automatic Reference Counting option, 7, 66, 170
- Use for Development option, 128
- Use Storyboard option, 331
- User input, 13
- User interface (UI)
 - controls, 152–153
 - designers for, 153–154
 - device design, 149–150
 - HIG, 148
 - industrial design, 149–150
 - metaphors, 150–151
 - Photo browser, 466–467
 - PhotoWheel, 640–647
 - rotation, 502–504, 507
 - slideshow controls, 622–624
 - sound effects, 151–152
 - tapworthy apps, 148
- User roles in iTunes Connect, 696
- userIdForUsername method, 559
- userInfo property, 604
- userInteractionEnabled flag, 481

Utilities area, 24–25
 Utility Application template, 6
 Utility classes and functions, 99–103
 UUIDs (universally unique IDs), 296
 uuidString method, 296

V

Variables, 45
 inspecting, 663, 667–670
 Objective-C, 74–75
 renaming, 80
 Version editor, 36–37
 Versions
 apps, 693
 Xcode, 709
 Vertical guides, 49–50
 View controllers
 container. *See* Container view controllers
 detail, 169. *See also* DetailViewController class
 Flickr, 553–555
 iCloud, 601, 606
 implementing, 351–354
 initial, 341–344
 master, 169, 186–187. *See also* Master-View-Controller class
 pop transitions, 364–367
 segues, 355–364
 split, 168–169, 181–185
 View-Master device, 232
 ViewController class, 11, 13–15
 viewController property, 536
 viewDidAppear event, 368, 625
 viewDidDisappear event, 368
 viewDidLoad method
 Carousel view, 246–247
 child view controllers, 372
 chrome effects, 478, 481
 data instances, 196–197
 deleting photos, 490–492, 495
 external displays, 619
 Flickr, 571
 iCloud, 601
 photo albums, 320–321, 323, 407, 410–411, 420

 PhotoBrowserViewController, 459, 464–465
 PhotoWheelViewCell, 250–251
 property lists, 291–292
 rotation, 505–506, 519
 table view data, 201, 203–204, 220–221, 224
 titles, 189–190
 touch gestures, 256–257
 wheel view, 239, 386
 viewDidUnload method
 Flickr, 571–572
 GridView, 446
 iCloud, 602
 photo albums, 401, 404, 417–418, 420
 PhotoBrowserViewController, 459, 464
 rotation, 505–506
 table view data, 212–214
 viewForZoomingInScrollView method, 486–487
 Views, 231
 carousel, 240–247
 custom, 231–232
 photo wheel view cell, 248–252
 table. *See* Table views
 wheel. *See* WheelView class
 viewWillAppear event
 chrome effects, 478, 481
 container view controllers, 368
 navigation bar, 358
 PhotoBrowserViewController, 460, 464, 625
 PhotoWheel, 644
 rotation, 518–520
 slideshows, 616, 621
 viewWillDisappear event, 368, 478, 481, 621
 Virtual keyboards, 152
 visibleCellIndexes property, 397
 Visual effects. *See* Core Image effects

W

wantsFullScreenLayout flag, 464
 Watchpoints, 663
 weak attribute, 77, 86
 Web services, 547
 basics, 547–548

- concurrent programming, 580–582
- Flickr. *See* Flickr
- RESTful, 548–549
- WebKit Coding Style Guidelines*, 30
- Welcome to Xcode screen, 4–5, 66–67
- Wenderlich, Ray, 101
- Wheeler, Colin, 32
- WheelView class, 233
 - Carousel view, 240–247
 - defining, 235
 - header file, 233–235
 - implementation, 235–240
 - photo albums, 399
 - prototype code, 385–398
 - rotation, 509–511
 - spin gesture recognizers, 262
- WheelViewCell class, 234, 248
 - defining, 235
 - prototype code, 386, 396
- WheelViewDataSource protocol, 234–235, 238–239, 387
- WheelViewDelegate protocol, 387
- wheelViewNumberOfCells method, 239, 403, 405
- wheelViewNumberOfVisibleCells method, 387, 403, 405
- Wildcard App IDs, 128, 134
- Wildcard characters (*)
 - App IDs, 119, 586
 - Bundle Identifiers, 118
- willAccessValueForKey method, 598
- willAnimateFirstHalfOfRotationToInterfaceOrientation method, 500
- willAnimateRotationToInterfaceOrientation method
 - MainViewController, 506–507
 - overriding, 500, 504
 - PhotoBrowserViewController, 516–518
 - scenes, 508–509
 - WheelView, 510–511
- willAnimateSecondHalfOfRotationFromInterfaceOrientation method, 500
- willChangeValueForKey method, 596
- willHideViewController method, 182
- willMoveToParentViewController method, 368, 373
- willPresentViewController method, 182
- willRotateToInterfaceOrientation method

- action sheets, 275, 277
- overriding, 499
- PhotoBrowserViewController, 515–517
- Windows, classes for, 104
- Windows Bitmap Format (DIB) format, 109
- Windows Cursor format, 109
- Windows Icon Format, 109
- Wireframe mockups, 154–155, 158–160
- Wooldridge, Dave, 699
- Workspace window, 20, 48
 - Debug area, 25–26
 - Editor area, 23
 - Navigation area, 22–23
 - Toolbar area, 20–22
 - Utilities area, 24–25
- Workspaces, creating, 333–336
- Wrapping feature, 386
- Wrapping Flickr API, 557–564
- WWDR intermediate certificates, 126

X

- .xbm files, 109
- .xcdatamodeld extension, 304–305
- Xcode, 19
 - debugging, 663–670
 - developer documentation, 34
 - downloading, 708
 - editors, 35–37
 - IDE, 19–20
 - installing, 708–709
 - in Launchpad, 3–4
 - organizer, 40
 - Preferences, 26–33
 - project settings, 36–39
 - schemes, 39
 - tools, 41
 - Workspace window, 20–26
- Xcode 4 User Guide*, 4
- .xib files, 11, 44
- XML with Flickr, 562
- Xmo'd plug-in, 315
- XWindow bitmap format, 109

Z

- Zarra, Marcus, 243, 413
- Zarra Studios Coding Style Guide*, 30

Zooming

faces, 653–654
photos, 482–489

zoomRectForScale method, 485, 487

zoomToFaces method, 653–654

zoomToLocation method, 485, 487