

Kyle Richter
Joe Keeley

Second Edition



Mastering iOS Frameworks

Beyond the Basics



FREE SAMPLE CHAPTER



SHARE WITH OTHERS

Mastering iOS Frameworks

Developer's Library Series



Visit developers-library.com for a complete list of available products

The **Developer's Library Series** from Addison-Wesley provides practicing programmers with unique, high-quality references and tutorials on the latest programming languages and technologies they use in their daily work. All books in the Developer's Library are written by expert technology practitioners who are exceptionally skilled at organizing and presenting information in a way that's useful for other programmers.

Developer's Library books cover a wide range of topics, from open-source programming languages and databases, Linux programming, Microsoft, and Java, to Web development, social networking platforms, Mac/iPhone programming, and Android programming.



Mastering iOS Frameworks

Beyond the Basics, Second Edition

Kyle Richter
Joe Keeley

◆Addison-Wesley

Hoboken, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Cape Town • 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.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact international@pearsoned.com.

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2015932706

Copyright © 2015 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, 200 Old Tappan Road, Old Tappan, New Jersey 07675, or you may fax your request to (201) 236-3290.

AirPlay, AirPort, AirPrint, AirTunes, App Store, Apple, the Apple logo, Apple TV, Aqua, Bonjour, the Bonjour logo, Cocoa, Cocoa Touch, Cover Flow, Dashcode, Finder, FireWire, iMac, Instruments, Interface Builder, iOS, iPad, iPhone, iPod, iPod touch, iTunes, the iTunes logo, Leopard, Mac, Mac logo, Macintosh, Multi-Touch, Objective-C, Quartz, QuickTime, QuickTime logo, Safari, Mountain Lion, Yosemite, Spotlight, and Xcode are trademarks of Apple, Inc., registered in the U.S. and other countries. OpenGL, or OpenGL Logo, is a registered trademark of Silicon Graphics, Inc.

ISBN-13: 978-0-134-05249-6

ISBN-10: 0-134-05249-8

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.

First printing: April 2015

Editor-in-Chief
Mark Taub

Senior Acquisitions
Editor
Trina MacDonald

Development Editor
Sheri Replin

Managing Editor
Kristy Hart

Project Editor
Elaine Wiley

Copy Editor
Cheri Clark

Indexer
Ken Johnson

Proofreader
Kathy Ruiz

Technical
Reviewers
Niklas Saers
Justin Williams

Editorial Assistant
Olivia Basegio

Cover Designer
Chuti Prasertsith

Senior Compositor
Gloria Schurick



I would like to dedicate this book to my co-workers who continually drive me to never accept the first solution.

—Kyle Richter

I dedicate this book to my wife, Irene, and two daughters, Audrey and Scarlett. Your boundless energy and love inspire me daily.

—Joe Keeley



Table of Contents

1 UIKit Dynamics 1

The Sample App 1

Introduction to UIKit Dynamics 2

Implementing UIKit Dynamics 3

Gravity 3

Collisions 4

Attachments 7

Springs 8

Snap 9

Push Forces 10

Item Properties 11

In-Depth `UIDynamicAnimator` and `UIDynamicAnimatorDelegate` 13

Summary 14

2 Core Location, MapKit, and Geofencing 15

The Sample App 15

Obtaining User Location 16

Requirements and Permissions 16

Checking for Services 19

Starting Location Request 19

Parsing and Understanding Location Data 22

Significant Change Notifications 23

Using GPX Files to Test Specific Locations 23

Displaying Maps 25

Understanding the Coordinate Systems 25

MKMapKit Configuration and Customization 25

Responding to User Interactions 27

Map Annotations and Overlays 28

Adding Annotations 28

Displaying Standard and Custom Annotation Views 31

Draggable Annotation Views 34

Working with Map Overlays 35

Geocoding and Reverse-Geocoding 36

Geocoding an Address 36

Reverse-Geocoding a Location 40

Geofencing	43
Checking for Regional Monitoring Capability	43
Defining Boundaries	44
Monitoring Changes	45
Getting Directions	47
Summary	52

3 Leaderboards 53

The Sample App	53
Spawning a Cactus	55
Cactus Interaction	58
Displaying Life and Score	60
Pausing and Resuming	62
Final Thoughts on Whack-a-Cac	63
iTunes Connect	63
Game Center Manager	66
Authenticating	68
Common Authentication Errors	69
iOS 6 and Newer Authentication	71
Submitting Scores	73
Adding Scores to Whack-a-Cac	76
Presenting Leaderboards	77
Score Challenges	79
Going Further with Leaderboards	81
Summary	83

4 Achievements 85

iTunes Connect	85
Displaying Achievement Progress	87
Game Center Manager and Authentication	88
The Achievement Cache	89
Reporting Achievements	90
Adding Achievement Hooks	92
Completion Banners	93
Achievement Challenges	94
Adding Achievements into Whack-a-Cac	97
Earned or Unearned Achievements	98
Partially Earned Achievements	99

Multiple Session Achievements	101
Piggybacked Achievements and Storing Achievement Precision	102
Timer-Based Achievements	103
Resetting Achievements	104
Going Further with Achievements	105
Summary	107
5 Getting Started with Address Book	109
Why Address Book Support Is Important	109
Limitations of Address Book Programming	110
The Sample App	110
Getting Address Book Up and Running	111
Reading Data from the Address Book	113
Reading Multivalued from the Address Book	114
Understanding Address Book Labels	115
Working with Addresses	116
Address Book Graphical User Interface	118
People Picker	118
Programmatically Creating Contacts	123
Summary	126
6 Working with Music Libraries	127
The Sample App	127
Building a Playback Engine	129
Registering for Playback Notifications	129
User Controls	131
Handling State Changes	132
Duration and Timers	137
Shuffle and Repeat	138
Media Picker	138
Programmatic Picker	141
Playing a Random Song	141
Predicate Song Matching	142
Summary	144
7 Implementing HealthKit	145
Introduction to HealthKit	145
Introduction to Health.app	146
The Sample App	147

Adding HealthKit to a Project	148
Requesting Permission for Health Data	149
Reading Characteristic HealthKit Data	152
Reading and Writing Basic HealthKit Data	152
Reading and Writing Complex HealthKit Data	155
Summary	160

8 Implementing HomeKit 161

The Sample App	161
Introduction to HomeKit	162
Setting Up HomeKit Components	162
Developer Account Setup	163
Enabling HomeKit Capability	163
Home Manager	164
Home	166
Rooms and Zones	168
Accessories	170
Services and Service Groups	176
Actions and Action Sets	178
Testing with the HomeKit Accessory Simulator	179
Scheduling Actions with Triggers	181
Summary	181

9 Working with and Parsing JSON 183

JSON	183
Benefits of Using JSON	183
JSON Resources	184
The Sample App	184
Accessing the Server	184
Getting JSON from the Server	185
Building the Request	185
Inspecting the Response	186
Parsing JSON	186
Displaying the Data	187
Posting a Message	189
Encoding JSON	189
Sending JSON to the Server	191
Summary	193

10 Notifications 195

- Differences Between Local and Push Notifications 195
- The Sample App 196
- App Setup 196
- Creating Development Push SSL Certificate 200
- Development Provisioning Profile 203
- Custom Sound Preparation 208
- Registering for Notifications 209
- Scheduling Local Notifications 211
- Receiving Notifications 212
- Push Notification Server 213
- Sending the Push Notifications 214
- Handling APNs Feedback 215
- Summary 216

11 Cloud Persistence with CloudKit 217

- CloudKit Basics 217
- The Sample App 218
- Setting Up a CloudKit Project 218
 - Account Setup 218
 - Enabling iCloud Capabilities 220
- CloudKit Concepts 220
 - Containers 220
 - Databases 221
 - Records 221
 - Record Zones 222
 - Record Identifiers 222
 - Assets 222
- CloudKit Basic Operations 222
 - Fetching Records 223
 - Create and Save a Record 224
 - Update and Save a Record 226
- Subscriptions and Push 227
 - Push Setup 227
 - Subscribing to Data Changes 227
- User Discovery and Management 229
- Managing Data in the Dashboard 233
- Summary 235

12 Extensions 237

- Types of Extensions 237
 - Today 237
 - Share 238
 - Action 238
 - Photo Editing 238
 - Document Provider 238
 - Custom Keyboard 238
- Understanding Extensions 238
- API Limitations 239
- Creating Extensions 240
- Today Extension 242
- Sharing Code and Information between Host App and Extension 243
- Apple Watch Extension 244
- Summary 247

13 Handoff 249

- The Sample App 249
- Handoff Basics 249
- Implementing Handoff 251
 - Creating the User Activity 252
 - Continuing an Activity 253
- Implementing Handoff in Document-Based Apps 255
- Summary 257

14 AirPrint 259

- AirPrint Printers 259
- Testing for AirPrint 261
- Printing Text 261
 - Print Info 262
 - Setting Page Range 263
 - `UISimpleTextPrintFormatter` 263
 - Error Handling 264
 - Starting the Print Job 264
 - Printer Simulator Feedback 265

- Print Center 266
 - UIPrintInteractionControllerDelegate 267
- Printing Rendered HTML 268
- Printing PDFs 269
- Summary 270

- 15 Getting Up and Running with Core Data 271**
 - Deciding on Core Data 272
 - Sample App 273
 - Starting a Core Data Project 274
 - Core Data Environment 275
 - Building Your Managed Object Model 278
 - Creating an Entity 280
 - Adding Attributes 280
 - Establishing Relationships 281
 - Custom Managed Object Subclasses 282
 - Setting Up Default Data 282
 - Inserting New Managed Objects 282
 - Other Default Data Setup Techniques 284
 - Displaying Your Managed Objects 285
 - Creating Your Fetch Request 285
 - Fetching by Object ID 287
 - Displaying Your Object Data 288
 - Using Predicates 290
 - Introducing the Fetched Results Controller 292
 - Preparing the Fetched Results Controller 292
 - Integrating Table View and Fetched Results Controller 294
 - Responding to Core Data Changes 296
 - Adding, Editing, and Removing Managed Objects 299
 - Inserting a New Managed Object 299
 - Removing a Managed Object 300
 - Editing an Existing Managed Object 301
 - Saving and Rolling Back Your Changes 301
 - Summary 303

- 16 Integrating Twitter and Facebook Using Social Framework 305**
 - The Sample App 305
 - Logging In 306

Using SLComposeViewController	308
Posting with a Custom Interface	311
Posting to Twitter	311
Posting to Facebook	315
Creating a Facebook App	315
Accessing User Timelines	322
Twitter	322
Facebook	327
Summary	331
17 Working with Background Tasks	333
The Sample App	334
Checking for Background Availability	334
Finishing a Task in the Background	335
Background Task Identifier	336
Expiration Handler	337
Completing the Background Task	337
Implementing Background Activities	339
Types of Background Activities	339
Playing Music in the Background	340
Summary	344
18 Grand Central Dispatch for Performance	345
The Sample App	345
Introduction to Queues	347
Running on the Main Thread	347
Running in the Background	349
Running in an Operation Queue	351
Concurrent Operations	351
Serial Operations	353
Canceling Operations	354
Custom Operations	355
Running in a Dispatch Queue	357
Concurrent Dispatch Queues	357
Serial Dispatch Queues	359
Summary	361

19 Using Keychain and Touch ID to Secure and Access Data 363

- The Sample App 364
- Setting Up and Using Keychain 364
 - Setting Up a New `KeychainItemWrapper` 365
 - Storing and Retrieving the PIN 366
 - Keychain Attribute Keys 367
 - Securing a Dictionary 368
 - Resetting a Keychain Item 370
 - Sharing a Keychain Between Apps 370
 - Keychain Error Codes 372
- Implementing Touch ID 372
- Summary 374

20 Working with Images and Filters 375

- The Sample App 375
- Basic Image Data and Display 376
 - Instantiating an Image 376
 - Displaying an Image 377
 - Using the Image Picker 379
 - Resizing an Image 382
- Core Image Filters 383
 - Filter Categories and Filters 383
 - Filter Attributes 386
 - Initializing an Image 388
 - Rendering a Filtered Image 389
 - Chaining Filters 390
- Feature Detection 391
 - Setting Up a Face Detector 391
 - Processing Face Features 392
- Summary 394

21 Collection Views 395

- The Sample App 395
- Introducing Collection Views 396
 - Setting Up a Collection View 397
 - Implementing the Collection View Data Source Methods 398
 - Implementing the Collection View Delegate Methods 401

Customizing Collection View and Flow Layout	403
Basic Customizations	403
Decoration Views	405
Creating Custom Layouts	408
Collection View Animations	413
Collection View Layout Changes	413
Collection View Layout Animations	414
Collection View Change Animations	416
Summary	417

22 Introduction to TextKit 419

The Sample App	420
Introducing <code>NSLayoutManager</code>	420
Detecting Links Dynamically	423
Detecting Hits	424
Exclusion Paths	425
Content Specific Highlighting	427
Changing Font Settings with Dynamic Type	432
Summary	433

23 Gesture Recognizers 435

Types of Gesture Recognizers	435
Basic Gesture Recognizer Usage	436
Introduction to the Sample App	437
Tap Recognizer in Action	438
Pinch Recognizer in Action	440
Multiple Recognizers for a View	441
Gesture Recognizers: Under the Hood	443
Multiple Recognizers for a View: Redux	444
Requiring Gesture Recognizer Failures	446
Custom <code>UIGestureRecognizer</code> Subclasses	448
Summary	448

24 Accessing the Photo Library 449

The Sample App	449
The Photos Framework	450

- Using Asset Collections and Assets 451
 - Permissions 451
 - Asset Collections 453
 - Assets 457
- Changes in the Photo Library 459
 - Asset Collection Changes 459
 - Asset Changes 462
- Dealing with Photo Stream 464
- Summary 465
- 25 Passbook and PassKit 467**
 - The Sample App 468
 - Designing the Pass 468
 - Pass Types 469
 - Pass Layout—Boarding Pass 469
 - Pass Layout—Coupon 470
 - Pass Layout—Event 471
 - Pass Layout—Generic 471
 - Pass Layout—Store Card 472
 - Pass Presentation 473
 - Building the Pass 474
 - Basic Pass Identification 476
 - Pass Relevance Information 476
 - Barcode Identification 477
 - Pass Visual Appearance Information 478
 - Pass Fields 478
 - Signing and Packaging the Pass 481
 - Creating the Pass Type ID 481
 - Creating the Pass Signing Certificate 483
 - Creating the Manifest 488
 - Signing and Packaging the Pass 489
 - Testing the Pass 489
 - Interacting with Passes in an App 491
 - Updating Passes Automatically 501
 - Summary 502

26	Debugging and Instruments	503
	Introduction to Debugging	503
	The First Computer Bug	504
	Debugging Basics with Xcode	504
	Breakpoints	506
	Customizing Breakpoints	507
	Symbolic and Exception Breakpoints	508
	Breakpoint Scope	508
	Working with the Debugger	509
	Instruments	510
	The Instruments Interface	512
	Exploring Instruments: The Time Profiler	514
	Exploring Instruments: Leaks	516
	Going Further with Instruments	519
	Summary	519
	Index	521

Foreword

I have been working with the iPhone SDK (now iOS SDK) since the first beta released in 2008. At the time, I was focused on writing desktop apps for the Mac and hadn't thought much about mobile app development.

If you chose to be an early adopter, you were on your own. In typical Apple fashion, the documentation was sparse, and since access to the SDK required an NDA—and, apparently, a secret decoder ring—you were on your own. You couldn't search Google or turn to StackOverflow for help, and there sure as hell weren't any books out yet on the SDK.

In the seven years (yes, it really has been only seven years) since Apple unleashed the original iPhone on the world, we've come a long way. The iPhone SDK is now the iOS SDK. There are dozens of books and blogs and podcasts and conferences on iOS development. And ever since 2009, WWDC has been practically impossible to get into, making it even harder for developers—old and new—to learn about the latest features coming to the platform. For iOS developers, there is so much more to learn.

One of the biggest challenges I have as an iOS developer is keeping on top of all the components and frameworks available in the kit. The iOS HIG should help us with that, but it doesn't go far enough—deep enough. Sure, now I can find some answers by searching Google or combing through StackOverflow; but, more often than not, those answers only explain the how and rarely the why, and they never provide the details you really need.

And this is what Kyle and Joe have done with this book—they're providing the detail needed so you can fully understand the key frameworks that make up the iOS SDK.

I've had the pleasure of knowing Kyle and Joe for a number of years. They are two of the brightest developers I have ever met. They have each written some amazing apps over the years, and they continuously contribute to the iOS development community by sharing their knowledge—speaking at conferences and writing other books on iOS development. If you have a question about how to do something in iOS, chances are good that Kyle and Joe have the answer for you.

But what makes these guys so awesome is not just their encyclopedic knowledge of iOS, but their willingness to share what they know with everyone they meet. Kyle and Joe don't have competitors, they have friends.

Kyle and Joe's in-depth knowledge of the iOS SDK comes through in this book. It's one of the things I like about this book. It dives into the details for each component covered at a level that you won't always find when searching online.

I also like the way the book is structured. This is not something that you'll read cover to cover. Instead, you'll pick up the book because you need to learn how to implement a collection view or sort out some aspect of running a task in a background thread that you can't quite wrangle. You'll pick up the book when you need it, find the solution, implement it in your own code,

and then toss the book back on the floor until you need it again. This is what makes *Mastering iOS Frameworks* an essential resource for any iOS developer—regardless of your experience level. You might think you’re a master with Core Location and MapKit, but I reckon you’ll find something here that you never knew before.

Kyle and Joe don’t come with egos. They don’t brag. And they sure don’t act like they are better than any other developer in the room. They instill the very spirit that has made the Mac and iOS developer community one of the friendliest, most helpful in our industry, and this book is another example of their eagerness to share their knowledge.

This book, just like the seminal works from Mark and LaMarche or Sadun, will always be within arm’s reach of my desk. This is the book I wish I had when I first started developing iOS apps in 2008. Lucky you, it’s here now.

—Kirby Turner

Chief Code Monkey at White Peak Software, author of *Learning iPad Programming: A Hands-On Guide to Building iPad Apps, Second Edition* (Addison-Wesley Professional), and Cocoa developer community organizer and conference junkie

Preface

Welcome to *Mastering iOS Frameworks: Beyond the Basics!*

There are hundreds of “getting started with iOS” books available to choose from, and there are dozens of advanced books in specific topics, such as Core Data or Security. There was, however, a disturbing lack of books that would bridge the gap between beginner and advanced niche topics.

This publication aims to provide development information on the intermediate-to-advanced topics that are otherwise not worthy of standalone books. It’s not that the topics are uninteresting or lackluster; it’s that they are not large enough topics. From topics such as working with JSON to accessing photo libraries, these are frameworks that professional iOS developers use every day but are not typically covered elsewhere.

Additionally, several advanced topics are covered to the level that many developers need in order to just get started. Picking up a 500-page Core Data book is intimidating, whereas Chapter 15 of this book provides a very quick and easy way to get started with Core Data. Additional introductory chapters are provided for debugging and instruments, TextKit, HomeKit, HealthKit, and CloudKit.

Topics such as Game Center leaderboards and achievements, AirPrint, music libraries, Address Book, and Passbook are covered in their entirety. Whether you just finished your first iOS project or you are an experienced developer, this book has something for you.

The chapters have all been updated to work with iOS 8. Please let us know if you encounter issues and we will release updates and corrections.

If you have suggestions, bug fixes, corrections, or anything else you’d like to contribute to a future edition, please contact us at mastering.ios.frameworks@gmail.com. We are always interested in hearing what would make this book better and are very excited to continue refining it.

—Kyle Richter and Joe Keeley

Prerequisites

Every effort has been made to keep the examples and explanations simple and easy to digest; however, this is to be considered an intermediate to advanced book. To be successful with it, you should have a basic understanding of iOS development, Objective-C, and C. Familiarity with the tools such as Xcode, Developer Portal, iTunes Connect, and Instruments is also assumed. Refer to *Programming in Objective-C*, by Stephen G. Kochan, and *Learning iOS Development*, by Maurice Sharp, Rod Strougo, and Erica Sadun, for basic Objective-C and iOS skills.

What You'll Need

Although you can develop iOS apps in the iOS simulator, it is recommended that you have at least one iOS device available for testing:

- **Apple iOS Developer Account:** The latest version of the iOS developer tools including Xcode and the iOS SDKs can be downloaded from Apple's Developer Portal (<http://developer.apple.com/ios>). To ship an app to the App Store or to install and test on a personal device, you will also need a paid developer account at \$99 per year.
- **Macintosh Computer:** To develop for iOS and run Xcode, you will need a modern Mac computer capable of running the latest release of OS X.
- **Internet Connection:** Many features of iOS development require a constant Internet connection for your Mac as well as for the device you are building against.

How This Book Is Organized

With few exceptions (Game Center and Core Data), each chapter stands on its own. The book can be read cover to cover but any topic can be skipped to when you find a need for that technology; we wrote it with the goal of being a quick reference for many common iOS development tasks.

Here is a brief overview of the chapters you will encounter:

- **Chapter 1, "UIKit Dynamics":** iOS 7 introduced UI Kit Dynamics to add physics-like animation and behaviors to UIViews. You will learn how to add dynamic animations, physical properties, and behaviors to standard objects. Seven types of behaviors are demonstrated in increasing difficulty from gravity to item properties.
- **Chapter 2, "Core Location, MapKit, and Geofencing":** iOS 6 introduced new, Apple-provided maps and map data. This chapter covers how to interact with Core Location to determine the device's location, how to display maps in an app, and how to customize the map display with annotations, overlays, and callouts. It also covers how to set up regional monitoring (or geofencing) to notify the app when the device has entered or exited a region.
- **Chapter 3, "Leaderboards":** Game Center leaderboards provide an easy way to add social aspects to your iOS game or app. This chapter introduces a fully featured iPad game called Whack-a-Cac, which walks the reader through adding leaderboard support. Users will learn all the required steps necessary for implementing Game Center leaderboards, as well as get a head start on implementing leaderboards with a custom interface.
- **Chapter 4, "Achievements":** This chapter continues on the Whack-a-Cac game introduced in Chapter 3. You will learn how to implement Game Center achievements in a fully featured iPad game. From working with iTunes Connect to displaying achievement progress, this chapter provides all the information you need to quickly get up and running with achievements.

- **Chapter 5, “Getting Started with Address Book”:** Integrating a user’s contact information is a critical step for many modern projects. Address Book framework is one of the oldest available on iOS; in this chapter you’ll learn how to interact with that framework. You will learn how to use the people picker, how to access the raw address book data, and how to modify and save that data.
- **Chapter 6, “Working with Music Libraries”:** This chapter covers how to access the user’s music collection from a custom app, including how to see informational data about the music in the collection, and how to select and play music from the collection.
- **Chapter 7, “Implementing HealthKit”:** HealthKit provides a centralized location for health information that can be shared among apps. This chapter explains how to get started with HealthKit, how to access information available in HealthKit, and how to read and write various types of health data.
- **Chapter 8, “Implementing HomeKit”:** This chapter explains how to get started using HomeKit, which enables iOS devices to communicate with home automation technology. It explains how to set up a home in HomeKit, and how to discover, set up, and interact with home automation devices such as lights, locks, and garage door openers.
- **Chapter 9, “Working with and Parsing JSON”:** JSON, or JavaScript Object Notation, is a lightweight way to pass data back and forth between different computing platforms and architectures. As such, it has become the preferred way for iOS client apps to communicate complex sets of data with servers. This chapter describes how to create JSON from existing objects, and how to parse JSON into iOS objects.
- **Chapter 10, “Notifications”:** Two types of notifications are supported by iOS: local notifications, which function on the device with no network required, and remote notifications, which require a server to send a push notification through Apple’s Push Notification Service to the device over the network. This chapter explains the differences between the two types of notifications, and demonstrates how to set them up and get notifications working in an app.
- **Chapter 11, “Cloud Persistence with CloudKit”:** CloudKit offers public and private remote data storage, with notifications for changes in data. This chapter explains the basic CloudKit concepts, and illustrates how to build an app that uses CloudKit for storing and syncing both private and public data remotely.
- **Chapter 12, “Extensions”:** Extensions provide a way to access an app’s functionality outside the app’s sandbox. This chapter explains the different types of extensions that are available, and illustrates how to create a Today extension and an Apple Watch extension.
- **Chapter 13, “Handoff”:** Handoff is one of the Continuity features introduced with iOS 8 and Yosemite. It enables the user to switch between devices and have an activity seamlessly move from one device to another. This chapter explains the basic Handoff mechanisms, and how to implement Handoff for developer-defined activities and document-based activities.

- **Chapter 14, “AirPrint”:** An often-underappreciated feature of the iOS, AirPrint enables the user to print documents and media to any wireless-enabled AirPrint-compatible printer. Learn how to quickly and effortlessly add AirPrint support to your apps. By the end of this chapter you will be fully equipped to enable users to print views, images, PDFs, and even rendered HTML.
- **Chapter 15, “Getting Up and Running with Core Data”:** This chapter demonstrates how to set up an app to use Core Data, how to set up a Core Data data model, and how to implement many of the most commonly used Core Data tools in an app. If you want to start using Core Data without digging through a 500-page book, this chapter is for you.
- **Chapter 16, “Integrating Twitter and Facebook Using Social Framework”:** Social integration is the future of computing, and it is accepted that all apps have social features built in. This chapter walks you through adding support for Facebook and Twitter to your app using the Social Framework. You will learn how to use the built-in composer to create new Twitter and Facebook posts. You will also learn how to pull down feed information from both services and how to parse and interact with that data. Finally, using the frameworks to send messages from custom user interfaces is covered. By the end of this chapter, you will have a strong background in Social Framework as well as working with Twitter and Facebook to add social aspects to your apps.
- **Chapter 17, “Working with Background Tasks”:** Being able to perform tasks when the app is not the foreground app was a big new feature introduced in iOS 4, and more capabilities have been added since. This chapter explains how to perform tasks in the background after an app has moved from the foreground, and how to perform specific background activities allowed by iOS.
- **Chapter 18, “Grand Central Dispatch for Performance”:** Performing resource-intensive activities on the main thread can make an app’s performance suffer with stutters and lags. This chapter explains several techniques provided by Grand Central Dispatch for doing the heavy lifting concurrently without affecting the performance of the main thread.
- **Chapter 19, “Using Keychain and TouchID to Secure and Access Data”:** Securing user data is important and an often-overlooked stage of app development. Even large public companies have been called out in the news over the past few years for storing user credit card info and passwords in plain text. This chapter provides an introduction to not only using the Keychain to secure user data but developmental security as a whole. By the end of the chapter, you will be able to use Keychain to secure any type of small data on users’ devices and provide them with peace of mind.
- **Chapter 20, “Working with Images and Filters”:** This chapter covers some basic image-handling techniques, and then dives into some advanced Core Image techniques to apply filters to images. The sample app provides a way to explore all the options that Core Image provides and build filter chains interactively in real time.

- **Chapter 21, “Collection Views”:** Collection views, a powerful new API introduced in iOS 6, give the developer flexible tools for laying out scrollable, cell-based content. In addition to new content layout options, collection views provide exciting new animation capabilities, both for animating content in and out of a collection view and for switching between collection view layouts. The sample app demonstrates setting up a basic collection view, a customized flow layout collection view, and a highly custom, nonlinear collection view layout.
- **Chapter 22, “Introduction to TextKit”:** iOS 7 introduced TextKit as an easier-to-use and greatly expanded update to Core Text. TextKit enables developers to provide rich and interactive text formatting to their apps. Although TextKit is a very large subject, this chapter provides the basic groundwork to accomplish several common tasks, from adding text wrapping around an image to inline custom font attributes. By the end of this chapter, you will have a strong background in TextKit and have the groundwork laid to explore it more in depth.
- **Chapter 23, “Gesture Recognizers”:** This chapter explains how to make use of gesture recognizers in an app. Rather than dealing with and interpreting touch data directly, gesture recognizers provide a simple and clean way to recognize common gestures and respond to them. In addition, custom gestures can be defined and recognized using gesture recognizers.
- **Chapter 24, “Accessing the Photo Library”:** The iPhone has actually become a very popular camera, as evidenced by the number of photos that people upload to sites such as Flickr. This chapter explains how to access the user’s photo library, and handle photos and videos in a custom app. The sample app demonstrates building some of the concepts from the iOS 8 version of Photos.app.
- **Chapter 25, “Passbook and PassKit”:** With iOS 6, Apple introduced Passbook, a standalone app that can store “passes,” or such things as plane tickets, coupons, loyalty cards, or concert tickets. This chapter explains how to set up passes, how to create and distribute them, and how to interact with them in an app.
- **Chapter 26, “Debugging and Instruments”:** One of the most important aspects of development is to be able to debug and profile your software. Rarely is this topic covered even in a cursory fashion. This chapter introduces you to debugging in Xcode and performance analysis using Instruments. Starting with a brief history of computer bugs, the chapter walks you through common debugging tips and tricks. Topics of breakpoints and debugger commands are briefly covered, and the chapter concludes with a look into profiling apps using the Time Profiler and memory analysis using Leaks. By the end of this chapter, you will have a clear foundation on how to troubleshoot and debug iOS apps on both the simulator and the device.

About the Sample Code

Each chapter of this book is designed to stand by itself; therefore, each chapter with the exception of Chapter 26, “Debugging and Instruments,” has its own sample project. Chapter 3, “Leaderboards,” and Chapter 4, “Achievements,” share a base sample project, but each expands

on that base project in unique ways. Each chapter provides a brief introduction to the sample project and walks the reader through any complex sections of the sample project not relating directly to the material in the chapter.

Every effort has been made to create simple-to-understand sample code, which often results in code that is otherwise not well optimized or not specifically the best way of approaching a problem. In these circumstances the chapter denotes where things are being done inappropriately for a real-world app. The sample projects are not designed to be standalone or finished apps; they are designed to demonstrate the functionality being discussed in the chapter. The sample projects are generic with intention; the reader should be able to focus on the material in the chapter and not the unrelated sample code materials. A considerable amount of work has been put into removing unnecessary components from the sample code and condensing subjects into as few lines as possible.

Many readers will be surprised to see that the sample code in the projects is built with Objective-C instead of Swift; this is by design as well. Since all the APIs illustrated are built with Objective-C, it is easier to interact with them using Objective-C, rather than add an additional layer of complexity by using Swift. The concepts illustrated are easily portable to Swift after the reader is comfortable with developing in Swift. The sample code is prefixed with “ICF” and most, but not all, sample projects are named after the chapter title.

When you’re working with the Game Center chapters, the bundle ID is linked to a real app, which is in our personal Apple account; this ensures that examples continue to work. It also has the small additional benefit of populating multiple users’ data as developers interact with the sample project. For chapters dealing with iCloud, Push Notifications, and Passbook, the setup required for the apps is thoroughly described in the chapter, and must be completed using a new App ID in the reader’s developer account in order to work.

Getting the Sample Code

You will be able to find the most up-to-date version of the source code at any moment at <https://github.com/dfsw/icf>, in the Mastering folder. The code is publicly available and open source. Each chapter is broken down into its own zip file containing an Xcode project; there are no chapters with multiple projects. We encourage readers to provide feedback on the source code and make recommendations so that we can continue to refine and improve it long after this book has gone to print.

Installing Git and Working with GitHub

Git is a version control system that has been growing in popularity for several years. To clone and work with the code on GitHub, you will want to first install Git on your Mac. A command-line version Git is included in the Xcode command-line tool installation, or a current installer for Git can be found at <http://git-scm.com/downloads>. Additionally, there are several GUI front ends for Git, even one written by GitHub, which might be more appealing to developers who avoid command-line interfaces. If you do not want to install Git, GitHub also allows for downloading the source files as a zip.

GitHub enables users to sign up for a free account at <https://github.com/signup/free>. After Git has been installed, from the terminal's command line `$git clone git@github.com:dfsw/icf.git` will download a copy of the source code into the current working directory. The sample code for this version of the book is in the Mastering folder. You are welcome to fork and open pull requests with the sample code projects.

Contacting the Authors

If you have any comments or questions about this book, please drop us an e-mail message at mastering.ios.frameworks@gmail.com, or on Twitter at [@kylerichter](#) and [@jwkeele](#).

Acknowledgments

This book could not have existed without a great deal of effort from far too many behind-the-scenes people; although there are only two authors on the cover, dozens of people were responsible for bringing this book to completion. We would like to thank Trina MacDonald first and foremost; without her leadership and her driving us to meet deadlines, we would never have been able to finish. The editors at Pearson have been exceptionally helpful; their continual efforts show on every page, from catching our typos to pointing out technical concerns. The dedicated work of Niklas Saers, Olivia Basegio, Justin Williams, Sheri Replin, Elaine Wiley, Cheri Clark, Chuti Prasertsith, and Gloria Shurick made the following pages possible.

We would also like to thank Jordan Langille of Langille Design (<http://jordanlangille.com>) for providing the designs for the Whack-a-Cac game featured in Chapters 3 and 4. His efforts have made the Game Center sample projects much more compelling.

The considerable amount of time spent working on this book was shouldered not only by us but also by our families and co-workers. We would like to thank everyone who surrounds us in our daily lives for taking a considerable amount of work off of our plates, as well as understanding the demands that a project like this brings.

Finally, we would like to thank the community at large. All too often we consulted developer forums, blog posts, and associates to ask questions or provide feedback. Without the hard efforts of everyone involved in the iOS community, this book would not be nearly as complete.

About the Authors

Kyle Richter is the Chief Executive Officer at MartianCraft, an award-winning Mobile Development Studio. Kyle began developing software in the early 1990s and has always been dedicated to the Apple ecosystem. He has authored and coauthored several books on iOS development, including *Beginning iOS Game Center Development*, *Beginning Social Game Development*, and *iOS Components and Frameworks*. Between running day-to-day operations at MartianCraft, Kyle travels the world speaking on development and entrepreneurship. He currently calls the Florida Keys home, where he spends his time with his border collie. He can be found on Twitter at @kylerichter.

Joe Keeley is a Partner and Lead Engineer at MartianCraft. Joe provides technical leadership on iOS projects for clients, and has led a number of successful client projects to completion. He has liked writing code since first keying on an Apple II, and has worked on a wide variety of technology and systems projects in his career. Joe has presented several technical topics at iOS and Mac conferences around the U.S. Joe lives in Denver, Colorado, with his wife and two daughters, and hopes to get back into competitive fencing again in his spare time. He can be reached on Twitter at @jwkeeley.

This page intentionally left blank

Working with and Parsing JSON

JSON is a great way to send data back and forth between servers, Web sites, and iOS apps. It is lighter and easier to handle than XML, and with iOS's built-in support for JSON, it is easy to integrate into an iOS project. Many popular Web sites, including Flickr, Twitter, and Google, offer APIs that provide results in JSON format, and many languages offer JSON support. This chapter demonstrates how to parse and present JSON from a sample message-board server in an app, and encode a new message entry in JSON to send to the server.

JSON

JavaScript Object Notation (JSON) is a lightweight format for sharing data. It is technically a part of the language JavaScript and provides a way to serialize JavaScript objects; however, practically, it is supported in a wide variety of programming languages, making it a great candidate for sharing data between different platforms. JSON also has the benefit of being human-readable.

JSON has a simple and intuitive syntax. At its most basic level, a JSON document can contain *objects*, which are essentially key-value dictionaries like what Objective-C programmers are familiar with, or arrays. JSON can contain arrays of objects and arrays of values, and can nest arrays and objects. Values stored in JSON, either in arrays or associated with a key, can be other JSON objects, strings, numbers, or arrays, or `true`, `false`, or `null`.

Benefits of Using JSON

There are many reasons to use JSON in an iOS app:

- **Server Support:** Communicating information to and from a remote server is a common use case for iOS apps. Since so many server languages have built-in support for JSON, it is a natural choice as a data format.

- **Lightweight:** JSON has little formatting overhead when compared to XML and can present a significant savings in the amount of bandwidth needed to transmit data between a server and a device.
- **iOS Support:** JSON is now fully supported as of iOS 5 with the addition of the `NSJSONSerialization` class. This class can conveniently provide an `NSDictionary` or `NSArray` (or even mutable varieties) from JSON data or can encode an `NSDictionary` or `NSArray` into JSON.
- **Presentation and Native Handling:** The simplest method to get data from a server to an iOS device is just to use a `UIWebView` and display a Web page; however, this approach has drawbacks in terms of performance and presentation. In many cases it is much better to just pull the data from the server, and present it on the device using native tools like `UITableView`. Performance can be much better, and presentation can be optimized to work on iOS screen sizes and take advantage of available retina displays.

JSON Resources

For more information on JSON, visit <http://json.org>. That site has a formal definition of JSON, with specific information on format and syntax.

The Sample App

The sample app for this chapter is Message Board, including a Ruby on Rails server and an iOS app.

The Ruby on Rails server consists of just one object: the message. It has been set up to support sending a list of messages in JSON, and to accept new messages in JSON format. The server also supports Web-based interactions.

The iOS app will pull messages from the server and display them in a standard table view and will be able to post new messages to the server in JSON format.

Accessing the Server

To view the Message Board Ruby on Rails server, visit <http://freezing-cloud-6077.herokuapp.com/>. The Messages home screen will be visible, as shown in Figure 9.1.

The messages server has been set up to handle creating and displaying messages on the Web and with JSON.

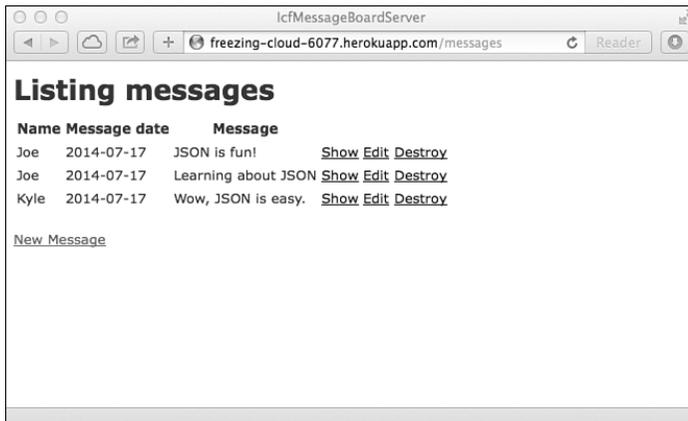


Figure 9.1 Messages home screen.

Getting JSON from the Server

To update the sample iOS app to handle JSON, the first thing to address is pulling the message list from the server and displaying it.

Building the Request

First, set up the URL so that the app can make calls to the right location:

```
NSString *const kMessageBoardURLString =
    @"http://freezing-cloud-6077.herokuapp.com/messages.json";
```

In the `ICFViewController.m` implementation, look at the `viewWillAppear:` method. This code will initiate the request to the server:

```
NSURL *msgURL = [NSURL URLWithString:kMessageBoardURLString];
NSURLSession *session = [NSURLSession sharedSession];

NSURLSessionTask *messageTask = [session dataTaskWithURL:msgURL
    completionHandler:^(NSData *data, NSURLResponse *response, NSError *error) {
    ...
}];
[messageTask resume];
```

This creates and initiates a network request to the `messages.json` resource at the server URL. The network request will run asynchronously, and when data comes back the completion handler block will be called. The important thing to note is that nothing special is required here for JSON; this is a standard network call. The only difference is that the `.json` extension

used in the URL tells the server that the response should be in JSON format. Other servers might use a `Content-Type` and/or `Accept` HTTP header that specifies `application/json` as the mime-type to indicate that a JSON response is desired.

Note

Using the `.json` extension is not required for servers to return JSON format data; that is just how the sample server was set up. It is a common approach but is not required.

Inspecting the Response

When the network request has returned, the completion handler will be called. In the sample app, the data is converted into a UTF-8 string so that it can be logged to the console. This should not be done for every request in a production app; it is done here to demonstrate how to see the response for debugging when a problem parsing JSON is encountered.

```
NSString *retString =
➔ [NSString stringWithUTF8String:[data bytes]];

NSLog(@"json returned: %@", retString);
```

The log message will display on the console the data received:

```
json returned: [{"message":{"created_at":"2012-04-29T21:59:28Z",
"id":3, "message":"JSON is fun!", "message_date":"2012-04-29",
"name":"Joe", "updated_at":"2012-04-29T21:59:28Z"}},
{"message":{"created_at":"2012-04-29T21:58:50Z", "id":2,
"message":"Learning about JSON", "message_date":"2012-04-
29", "name":"Joe", "updated_at":"2012-04-29T21:59:38Z"}},
{"message":{"created_at":"2012-04-29T22:00:00Z", "id":4,
"message":"Wow, JSON is easy.", "message_date":"2012-04-
29", "name":"Kyle", "updated_at":"2012-04-29T22:00:00Z"}},
{"message":{"created_at":"2012-04-29T22:46:18Z", "id":5,
"message":"Trying a new message.", "message_date":"2012-04-
29", "name":"Joe", "updated_at":"2012-04-29T22:46:18Z"}]}
```

Parsing JSON

Now that JSON has been received from the server, it is just a simple step to parse it. In the case of the sample app, an array of messages is expected, so parse the JSON into an `NSArray`:

```
NSError *parseError = nil;
NSArray *jsonArray =
➔ [NSJSONSerialization JSONObjectWithData:data
options:0
error:&parseError];
```

```

if (!parseError) {
    [self setMessageArray:jsonArray];
    NSLog(@"json array is %@", jsonArray);
} else {
    NSString *err = [parseError localizedDescription];
    NSLog(@"Encountered error parsing: %@", err);
}

```

NSJSONSerialization's method `JSONObjectWithData:options:error:` expects as parameters the data to be serialized, any desired options (for example, returning a mutable array instead of a regular array), and a reference to an `NSError` in case there are any parsing errors.

In this example, a local instance variable has been updated to the just-parsed array, the table view has been told to reload data now that there is data to display, and the activity view has been hidden. Note that the completion handler will most likely be called on a background queue, so if the user interface will be updated, it will be necessary to switch to the main queue.

```

dispatch_async(dispatch_get_main_queue(), ^{
    [self.messageTable reloadData];
    [self.activityView setHidden:YES];
    [self.activityIndicator stopAnimating];
});

```

Displaying the Data

Now that the JSON has been parsed into an `NSArray`, it can be displayed in a `UITableView`. The magic here is that there is no magic; the JSON received from the server is now just an array of `NSDictionary` instances. Each `NSDictionary` contains information for a message from the server, with attribute names and values. To display this in a table, just access the array and dictionaries as if they had been created locally.

```

- (UITableViewCell *)tableView:(UITableView *)tableView
  cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell =
    ➤ [tableView dequeueReusableCellWithIdentifier:@"MsgCell"];

    if (cell == nil) {
        cell = [[UITableViewCell alloc]
                ➤ initWithStyle:UITableViewCellStyleSubtitle
                ➤ reuseIdentifier:@"MsgCell"];

        cell.selectionStyle = UITableViewCellSelectionStyleNone;
    }
    NSDictionary *message =
    ➤ (NSDictionary *)[[self.messageArray
                      ➤ objectAtIndex:indexPath.row]
                      ➤ objectForKey:@"message"];

```

```

NSString *byLabel =
    ➤ [NSString stringWithFormat:@"by %@ on %@",
    ➤ [message objectForKey:@"name"],
    ➤ [message objectForKey:@"message_date"]];

cell.textLabel.text = [message objectForKey:@"message"];
cell.detailTextLabel.text = byLabel;
return cell;
}

- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section
{
    return [[self messageArray] count];
}

```

The parsed JSON data will be visible in a standard table view, as shown in Figure 9.2.

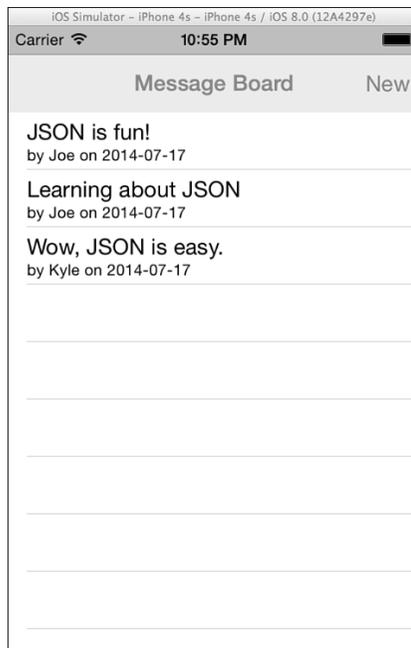


Figure 9.2 Sample app message table view.

Tip

When a `null` value is in the JSON source data, it will be parsed into an `[NSNull null]`. This can be a problem if `nil` is expected in a check or comparison, because `[NSNull null]` will return YES whereas `nil` will return NO. It is wise to specifically handle `[NSNull null]` when converting to a model object or presenting parsed JSON.

Posting a Message

The sample app includes `ICFNewMessageViewController` to post new messages to the server. There are two fields on that controller: one for a name and one for a message (see Figure 9.3). After the user enters that information and hits Save, it will be encoded in JSON and sent to the server.

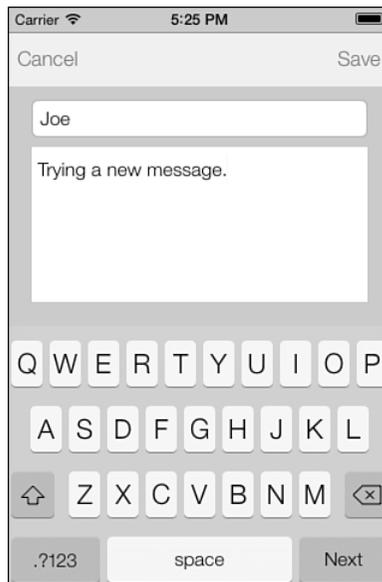


Figure 9.3 Sample app new message view.

Encoding JSON

An important detail for sending JSON to a Ruby on Rails server is to encode the data so that it mirrors what the Rails server provides. When a new message is sent to the server, it should have the same structure as an individual message received in the message list. To do this, a dictionary with the attribute names and values for the message is needed, and then a wrapper

dictionary with the key “message” pointing to the attribute dictionary. This will exactly mirror what the server sends for a message. In the `saveButtonTouched:` method, set up this dictionary, like so:

```
NSMutableDictionary *messageDictionary =
➤ [NSMutableDictionary dictionaryWithCapacity:1];

[messageDictionary setObject:[nameTextField text]
                        forKey:@"name"];

[messageDictionary setObject:[messageTextView text]
                        forKey:@"message"];

NSDate *today = [NSDate date];

NSDateFormatter *dateFormatter =
➤ [[NSDateFormatter alloc] init];

NSString *dateFmt = @"yyyy'-MM'-dd'T'HH':'mm':'ss'Z'";
[dateFormatter setDateFormat:dateFmt];
[messageDictionary setObject:[dateFormatter stringFromDate:today]
                        forKey:@"message_date"];

NSDictionary *postDictionary = @{@"message" : messageDictionary};
```

Note that `NSJSONSerialization` accepts only instances of `NSDictionary`, `NSArray`, `NSString`, `NSNumber`, or `NSNull`. For dates or other data types not directly supported by `NSJSONSerialization`, they will need to be converted to a supported format. For example, in this example the date was converted to a string in a format expected by the server. Now that there is a dictionary, it is a simple step to encode it in JSON:

```
NSError *jsonSerializationError = nil;
NSData *jsonData = [NSJSONSerialization
➤ dataWithJSONObject:postDictionary
➤ options:NSJSONWritingPrettyPrinted
➤ error:&jsonSerializationError];

if (!jsonSerializationError)
{
    NSString *serJSON =
        [[NSString alloc] initWithData:jsonData
                                encoding:NSUTF8StringEncoding];

    NSLog(@"serialized json: %@", serJSON);

    ...
} else
{
```

```

NSLog(@"JSON Encoding failed: %@",
      ↳ [jsonSerializationError localizedDescription]);
}

```

NSJSONSerialization expects three parameters:

1. An NSDictionary or NSArray with the data to be encoded.
2. Serialization options (in our case, we specified NSJSONWritingPrettyPrinted so that it's easy to read; otherwise, the JSON is produced with no whitespace for compactness).
3. A reference to an NSError.

If there are no errors encoding the JSON, it will look like this:

```

serialized json: {
  "message" : {
    "message" : "Six Test Messages",
    "name" : "Joe",
    "message_date" : "2012-04-01T14:31:11Z"
  }
}

```

Sending JSON to the Server

After the JSON is encoded, it is ready to be sent to the server. First, an instance of NSMutableURLRequest is needed. The request will be created with the URL for the server, and then will be customized with the HTTP method ("POST") and HTTP headers to indicate that the uploaded content data is in JSON format.

```

NSURL *messageBoardURL =
↳ [NSURL URLWithString:kMessageBoardURLString];

NSMutableURLRequest *request = [NSMutableURLRequest
                               requestWithURL:messageBoardURL
                               cachePolicy:NSURLRequestUseProtocolCachePolicy
                               timeoutInterval:30.0];

[request setHTTPMethod:@"POST"];

[request setValue:@"application/json"
 forHTTPHeaderField:@"Accept"];

[request setValue:@"application/json"
 forHTTPHeaderField:@"Content-Type"];

```

When the request is completed, an NSURLSessionUploadTask can be created. The task requires the request, the JSON data, and a completion handler. The completion handler will be called on a background thread, so any user interface updates must be dispatched to the main queue.

```

NSURLSession *session = [NSURLSession sharedSession];

NSURLSessionUploadTask *uploadTask =
[session uploadTaskWithRequest:uploadRequest fromData:jsonData
➤ completionHandler:^(NSData *data, NSURLResponse *response, NSError *error) {

    NSHTTPURLResponse *httpResponse = (NSHTTPURLResponse *)response;
    BOOL displayError = (error || httpResponse.statusCode != 200);

    dispatch_async(dispatch_get_main_queue(), ^{
        [self.activityView setHidden:YES];
        [self.activityIndicator stopAnimating];
        if (displayError) {
            NSString *errorMessage = error.localizedDescription;
            if (!errorMessage) {
                errorMessage =
                ➤ [NSString stringWithFormat:@"Error uploading - http status: %i",
                ➤ httpResponse.statusCode];
            }

            UIAlertController *postErrorAlertController =
            ➤ [UIAlertController alertControllerWithTitle:@"Post Error"
                message:errorMessage
                preferredStyle:UIAlertControllerStyleAlert];

            [postErrorAlertController addAction:
            ➤ [UIAlertAction actionWithTitle:@"Cancel"
                style:UIAlertActionStyleCancel
                handler:nil]];

            [self presentViewController:postErrorAlertController
                animated:YES
                completion:nil];
        } else {
            [self.presentingViewController dismissViewControllerAnimated:YES
                completion:nil];
        }
    });
});

[uploadTask resume];

```

When `resume` is called on the `uploadTask`, the request will be made to the server, and the completion handler will be called when it is complete. Both the `error` returned in the completion handler and the `response` should be checked for errors; an `error` will be returned if there is a problem connecting (for example, the device is in airplane mode), or an HTTP status code might indicate a different problem if there is an issue on the server (for example, if the URL is

not found, or if the server cannot process the data sent to it). If the request completes with no errors, the view controller will be dismissed and message board will be refreshed.

Summary

This chapter introduced JavaScript Object Notation (JSON). It explained how to request JSON data from a server in an iOS app, parse it, and display it in a table. The chapter also described how to encode an `NSDictionary` or `NSArray` into JSON, and send it over the network to a server.

This page intentionally left blank

Index

A

Accessories (HomeKit)

- Accessory Simulator tests, 179-180
- configuring, 171-175
- discovering, 162
- first time setups, 162

achievements (games), 85, 87, 107

- Achievement Challenges, 94-97
- achievement precision, storing, 102-103
- authenticating, 88
- caching, 89-90
- completion banners, 93
- creating, 85-86
- customizing, 105-107
- earned/unearned achievements, 98-99
- Hidden property, 87
- hooks, 92-93
- iTunes Connect, adding achievements to, 86
- localization information, 87
- multiple session achievements, 101-102
- partially earned achievements, 99-100
- piggybacked achievements, 102-103
- Point Value attribute, 87
- progress, displaying, 87-88
- reporting, 90-92
- resetting, 104-105
- timer-based achievements, 103-104
- Whack-a-Cac sample app, 97-98

Action Extensions, 238

action sets (HomeKit), 162, 178-179

actions (HomeKit), 178-179

- scheduling, 181
- triggers, 181

Address Book, 109, 111-113, 126

- GUI. *See* People Picker (Address Book)
- labels, 115-116
- limitations of, 110
- memory management, 113-114
- People Picker, 118-120
 - creating contacts, 122-125
 - customizing, 120
 - editing contacts, 120-121
 - viewing contacts, 120-121
- privacy and authorization, 110
- reading
 - data from Address Book, 113-114
 - multivalued from Address Book, 114-115
- sample app, 110
- street addresses, handling, 116-117
- support, importance of, 109

AirPrint, 259, 270

- error handling, 264
- PDF, printing, 269-270
- Print Center app, 266-267
- print jobs, starting, 264-265
- Print sample app, 260

- printer compatibility, 259
- Printer Simulator tool, 259, 265
- Printopia, 259
- rendered HTML, printing, 268-269
- testing, 259, 261
- text, printing, 261-262
 - configuring print info, 262-263
 - duplexing, 262-263
 - error handling, 264
 - page ranges, 263-264
- UIPrintInteractionControllerDelegate, 267

animations

- collection views, 395, 413
 - change animations, 416-417
 - layout animations, 414-416
 - layout changes, 413-414
- UIKit Dynamics, 1, 14
 - attachments, 7-8
 - classes of, 2
 - collisions, 3-6
 - dynamic behavior, 2
 - gravity, 3-4
 - introduction to, 2
 - item properties, 11-13
 - push forces, 10-11
 - sample app, 1
 - snaps, 9
 - springs, 8-9
 - UIAttachmentBehavior class, 2
 - UICollisionBehavior class, 2
 - UIDynamicAnimator, 2-3, 13
 - UIDynamicAnimatorDelegate, 13
 - UIDynamicItem protocol, 1, 12
 - UIDynamicItemBehavior class, 2
 - UIGravityBehavior class, 2

- UIPushBehavior class, 2

- UISnapBehavior class, 2

annotations in Map apps, 28

- adding, 28-31
- custom views, 31-33
- displaying, 31-33
- draggable views, 34
- standard views, 31-33

API (Application Programmer Interface)

- extension limitations, 239

APN (Apple Push Notifications), 195-196, 216

- Apple documentation, 214
- Development Push SSL Certificates, 200
- feedback, 215

App ID and push notifications, 196-199

Apple Maps, 15

Apple Watch Extensions, 244-247

asset collections (photo library), 453-457

assets

- asset collections (photo library), 459-461
- CloudKit, 222
- photo library, 457-458, 462-464

attachments (physics simulations), UIKit Dynamics, 7-8

attributes, adding to managed object models in Core Data, 280

authenticating

- achievements in Game Center, 88
- leaderboards in Game Center, 68-69
 - common errors, 69-71
- iOS 6 and newer authentication, 71-73

automation (home). See HomeKit

B

- background-task processing, 333, 339, 344**
 - background availability, checking for, 334-335
 - BackgroundTasks sample app, 334
 - expiration handlers, 337
 - GCD and performance, 349-351
 - identifiers, 336
 - LongRunningTasks sample app, 349-351
 - multitasking availability, checking for, 335
 - music, playing in a background, 340-342
 - tasks
 - completing, 337-339
 - executing, 335-336
 - types of background activities, 339-340
- boarding passes (Passbook), 469**
- body temperature data, reading/writing in HealthKit, 155-160**
- breakpoints (debugging), 506**
 - customizing, 507-508
 - exception breakpoints, 508
 - scope of, 508-509
 - symbolic breakpoints, 508

C

- caching achievements (games), 89-90**
- Carmageddon*, 3
- CloudKit, 217-218, 220, 222, 235**
 - account setup, 217-219
 - assets, 222
 - CloudTracker sample app, 218, 228
 - containers, 220
 - dashboard and data management, 233-235

- databases, 221
- iCloud capabilities, enabling, 220
- push notifications, 227
- record identifiers, 222
- record zones, 222
- records, 221-222
 - creating, 224-226
 - fetching, 223
 - saving, 224-226
 - updating, 226
- subscriptions to data changes, 227-228
- user discovery/management, 229-233
- CloudTracker sample app, 218, 228**
- coders/keyed archives and persistent data, 272**
- collection views, 395-396, 417**
 - animations, 395, 413
 - change animations, 416-417
 - layout animations, 414-416
 - layout changes, 413-414
 - custom layouts, creating, 408-413
 - data source methods, 398-401
 - delegate methods, 401-402
 - flow layouts, 395-396, 403-408
 - organizing, 395
 - PhotoGallery sample app, 395-396
 - setup, 397-398
- collisions (physics simulations) and UIKit Dynamics, 3-6**
- Combined Leaderboards, 64**
- completion banners (achievements), 93**
- concurrent operations, running, 351-352**
- configuring**
 - Handoff, 251-252
 - HomeKit, 162-179
 - leaderboards, 64

contacts (Address Book)

- creating, 122-125
- customizing, 120
- editing, 120-121
- viewing, 120-121

containers (CloudKit), 220**content specific highlighting and TextKit, 427-431****Continuity and Handoff, 249, 257**

- advertisements, 249-251
- configuring, 251-252
- continuation, 250-251
- document-based apps, implementing in, 255-257
- HandOffNotes sample app, 249
- implementing, 251-257
- introduction to, 249-251
- testing, 251
- user activity
 - continuing, 253-255
 - creating, 252-253

continuous gesture recognizers, 435**Cook, Tim, 244****coordinate systems in Map apps, 25****Core Data, 271-273, 303**

- default data setup, 282
 - data model version migrations, 284
 - inserting new managed objects, 282-284
 - loading data from Web services/API, 284
- environment of, 275-278
- EOF and, 271
- features of, 271
- fetchd results controller, 292, 298-299
 - deleting rows, 298
 - inserting new sections, 297

- inserting rows, 298
- integrating table view with, 294-296
- moving rows, 298
- preparations for, 292-294
- removing rows, 298
- removing sections, 297-298
- responding to content changes, 296-299
- updating rows, 298

managed object models, building, 278-279

- adding attributes to, 280
- creating entities, 280
- customized subclasses, 282
- establishing relationships, 281

managed objects, 299

- adding, 299-300
- creating fetch requests, 285-287
- displaying, 285-291
- displaying object data, 288-290
- editing, 301
- fetching by object ID, 287
- predicates, 290-291
- removing, 300-301

- rolling back changes, 301-303
- saving changes, 301-303

MyMovies sample app, 273

- displaying object data, 288-290
- friend chooser, 285-287
- movie display view, 287
- movie list view controller, 292-299
- predicates, 290-291
- Shared Movies tab, 291

projects, starting, 274-278**SQLite, 271**

- table view, integrating with fetched results controller, 294-296

Core Image filters, 383, 394

- face detector, 391
 - processing facial features, 392-394
 - setup, 391-392
- filters
 - attributes of, 386-388
 - categories of, 383-386
 - chaining, 390-391
- images
 - initializing, 388-389
 - rendering filtered images, 389-390

Core Location, 15

- FavoritePlaces sample app
 - purpose of, 15
 - user location requests, 16-24
- geofencing (regional monitoring), 43
 - boundary definitions, 44-45
 - monitoring changes, 45-46
 - regional monitoring capability checks, 43-44
- importing, 16
- user location requests, 16
 - location services checks, 19
 - parsing location data, 22-23
 - permissions, 16-19
 - requirements, 16-19
 - significant location change notifications, 23
 - starting requests, 19-22
 - testing locations, 23-24
 - understanding data, 22-23

Core Text, 419

- coupons (Passbook), 469-471
- CSV (Comma Separated Values) and persistent data, 273
- Custom Keyboard Extensions, 238

customizing

- achievements (games), 105-107
- breakpoints (debugging), 507-508
- flow layouts (collection views), 403
 - basic customizations, 403-404
 - decoration views, 405-408
- leaderboards, 81-82
- People Picker (Address Book), 120

D

dashboard (CloudKit) and data manager, 233-235**data security. See security****databases**

- CloudKit, 221
- object databases. *See* Core Data

debugging, 503, 519-520

- breakpoints, 506
 - customizing, 507-508
 - exception breakpoints, 508
 - scope of, 508-509
 - symbolic breakpoints, 508
- first computer bug, 504
- Instruments, 510-511, 519
 - interface of, 511-514
 - Leaks instrument, 516-518
 - Time Profiler instrument, 514-516
- introduction to, 503-504
- Xcode, 504-505, 509-519

decoration views (collection views), 405-408**developers (game) and physics simulations, 3****development provisioning profiles and push notification tests, 203-207****Development Push SSL Certificates, 200-203**

dictionaries (Keystone sample app),
 securing, 368-370

Dijkstra, Edsger W., 503

Direct SQLite and persistent data, 273

directions, getting via Maps.app, 47-51

discrete gesture recognizers, 435

dispatch queues and GCD (Grand Central
Dispatch), 357, 361

 concurrent dispatch queues, 357-359

 serial dispatch queues, 359-361

Document Provider Extensions, 238

duplexing (printing), 262-263

Dylan, Bob, 143

dynamic behavior and UIKit Dynamics, 2

Dynamic Link Detection and TextKit,
 423-424

Dynamic Type and TextKit, 432

E

earned/unearned achievements (games),
 98-99

embedded frameworks (extensions),
 creating, 243-244

entities, creating for managed object
 models in Core Data, 280

EOF (Enterprise Object Framework) and
 Core Data, 271

error codes (Keychain sample app), 372

error handling when printing, 264

events (Passbooks), 469, 471

exception breakpoints (debugging), 508

exclusion paths and TextKit, 425-426

expiration handlers and background-task
 processing, 337

extensions, 237, 247

 Action Extensions, 238

 API limitations, 239

 Apple Watch Extensions, 244-247

 creating, 240-241

 Custom Keyboard Extensions, 238

 Document Provider Extensions, 238

 embedded frameworks, creating,
 243-244

 functionality of, 238-239

 host apps, sharing information with,
 243-244

 Photo Editing Extensions, 238

 Share Extensions, 238

 Today Extensions, 237, 240, 242

 WatchKit, 244-247

F

face detector (Core Image filters), 391

 processing facial features, 392-394

 setup, 391-392

Facebook and Social Framework, 305, 331

 Facebook app, creating, 315-316

 logins, 306-308

 permissions

 basic Facebook permissions,
 317-318

 publishing to stream permissions,
 319-320

 posting to

 Facebook, 311, 315

 streams, 320-321

 SLComposeViewController, 308-310

 SocialNetworking sample app, 305-306

 user timelines, accessing, 322, 327-331

FavoritePlaces sample app

 annotations, 28

 adding, 28-31

 custom views, 31-33

 displaying, 31-33

 draggable views, 34

 standard views, 31-33

- displaying maps, 25
 - coordinate systems, 25
 - Mercator Projection, 25
- geocoding addresses, 36-40
- geofencing (regional monitoring), 43
 - boundary definitions, 44-45
 - monitoring changes, 45-46
 - regional monitoring capability checks, 43-44
- map view, 28
- MKMapKit, configuring/customizing, 25-26
- overlays, 28, 35-36
- purpose of, 15
- reverse-geocoding addresses, 36, 40-43
- user interactions, responding to, 27-28
- user location requests, 16
 - location services checks, 19
 - parsing location data, 22-23
 - permissions, 16-19
 - requirements, 16-19
 - significant location change notifications, 23
 - starting requests, 19-22
 - testing locations, 23-24
 - understanding data, 22-23

fetchd results controller (Core Data), 292

filters (Core Image filters), 383

- attributes of, 386-388
- categories of, 383-386
- chaining, 390-391
- rendering filtered images, 389-390

fitness/health apps. See HealthKit

flow layouts (collection views), 395-396

- customizing, 403-404
- decoration views, 405-408

font settings (text), changing in TextKit, 432

foreground app, 333

formatting scores in Whack-a-Cac sample app, 65-66

frameworks (embedded), creating for extensions and host apps, 243-244

G

Game Center

- achievements, 85, 87, 107
 - Achievement Challenges, 94-97
 - adding to iTunes Connect, 86
 - authenticating, 88
 - caching, 89-90
 - completion banners, 93
 - creating, 85-86
 - customizing, 105-107
 - displaying achievements, 87-88
 - earned/unearned achievements, 98-99
 - Hidden property, 87
 - hooks, 92-93
 - localization information, 87
 - multiple session achievements, 101-102
 - partially earned achievements, 99-100
 - piggybacked achievements, 102-103
 - Point Value attribute, 87
 - reporting, 90-92
 - resetting, 104-105
 - storing achievement precision, 102-103
 - timer-based achievements, 103-104
 - Whack-a-Cac sample app, 97-104
- Game Center Manager, 66-68, 88

iTunes Connect

- adding achievements to, 86
- configuring Game Center behavior in, 63-64

leaderboards, 53, 83

- Apple's limit on number of leaderboards, 65
- authenticating, 68-73
- Combined Leaderboards, 64
- configuring, 64-65
- configuring behavior in iTunes Connect, 63-64
- customizing leaderboard systems, 81-82
- deleting, 64
- formatting scores, 65-66
- localization information, 66
- presenting, 77-79
- Single Leaderboards, 64
- sort-order option, 66

scores

- Game Center Challenges, 79-81
- submitting, 73-76

sort-order option, 66

Whack-a-Cac sample app, 53-55, 63

- achievement hooks, 92-93
- achievements, 97-104
- configuring leaderboards, 65
- displaying life, 60-61
- displaying score, 60
- Game Center Manager and, 66-68
- hooks (achievements), 92-93
- interacting with cacti (cactus), 58-60
- pausing games, 62
- resuming games, 62
- spawning cacti (cactus), 55-58

game developers and physics simulations, 3**GarageBand, custom sound and notifications, 208-209****GCD (Grand Central Dispatch) and performance, 345, 361**

- dispatch queues, 357, 361
 - concurrent dispatch queues, 357-359
 - serial dispatch queues, 359-361
- LongRunningTasks sample app, 345-346
 - background-task processing, 349-351
 - running in operation queues, 351-357
 - running main threads, 347-349

operation queues, running in, 361

- cancelling operations, 354-355
- concurrent operations, 351-352
- custom operations, 355-357
- serial operations, 353-354

queues, 347

generic passes (Passbooks), 469, 471-472**geocoding addresses in Map apps, 36-40.**

See also reverse-geocoding in Map apps

geofencing (regional monitoring), 43

- boundaries, defining, 44-45
- monitoring
 - changes, 45-46
 - regional monitoring capability checks, 43-44

gesture recognizers, 435, 448

- basic usage, 436
- continuous gesture recognizers, 435
- custom UIGestureRecognizer subclasses, 448
- discrete gesture recognizers, 435

- event sequence of a recognizer, 443-444
- failures, requiring, 446-447
- Gesture Playground sample app, 437
 - pinch gesture recognizers, 440-441
 - tap gesture recognizers, 438-440
- multiple recognizers, using per view, 441-445
- pinch gesture recognizers, 440-441
- tap gesture recognizers, 436, 438-440
- types of, 435

GPS (Global Positioning System) in Map apps, 22

GPX (GPS Exchange Format) files, testing locations in Map apps, 23-24

graphics. See image handling; photo library

gravity (physics simulations), 3-4

H

Handoff, 249, 257

- advertisements, 249-251
- configuring, 251-252
- continuation, 250-251
- document-based apps, implementing in, 255-257
- HandOffNotes sample app, 249
- implementing
 - configurations, 251-252
 - continuing user activity, 253-255
 - creating user activity, 252-253
 - document-based apps, 255-257
- introduction to, 249-251
- testing, 251
- user activity
 - continuing, 253-255
 - creating, 252-253

Harvard University, 504

Health.app

- Dashboard, 146
- introduction to, 146
- reading characteristic data, 152

HealthKit, 145, 160

- framework guide website, 145
- ICFFever sample app, 147
 - adding HealthKit to, 148-149
 - permission requests, 150
 - reading/writing data, 152-154
- introduction to, 145-146
- new projects, adding to, 148-149
- permission requests, 149-151
- privacy, 145-146
- reading/writing data
 - basic data, 152-154
 - body temperature data, 155-160
 - characteristic data, 152
 - complex data, 155-160
- WWDC 2014, 145

Hidden property (achievements), 87

highlighting (content specific) and TextKit, 427-431

hit detection and TextKit, 424-425

HomeKit, 161, 181

- Accessories
 - Accessory Simulator tests, 179-180
 - configuring, 170-175
 - discovering, 162
 - first time setups, 162
- action sets, 162, 178-179
- actions, 178-179
 - scheduling, 181
 - triggers, 181
- capability setup, 163-164
- configuring, 162-179
- data access, 162

- developer account setup, 163
 - enabling, 162
 - Home Manager, 164-168
 - HomeNav sample app, 161
 - Accessory configuration, 171-175
 - adding homes to, 166-168
 - iCloud setup, 165-166
 - introduction to, 162
 - Rooms, 162, 168-169
 - Service Groups, 176-178
 - Services, 176-178
 - triggers, 181
 - Zones, 169-170
 - hooks (achievements), 92-93**
 - Hopper, Grace Murray, 504**
 - horizontal accuracy in Map apps, 22**
 - HTML (rendered), printing, 268-269**
-
- |
- ICFFever sample app, 147**
 - adding HealthKit to, 148-149
 - permission requests, 150
 - reading/writing data
 - basic data, 152-154
 - body temperature data, 155-160
 - complex data, 155-160
 - iCloud**
 - CloudKit, 217-218, 220, 222, 235
 - account setup, 217-219
 - assets, 222
 - containers, 220
 - creating records, 224-226
 - dashboard and data management, 233-235
 - databases, 221
 - enabling iCloud capabilities, 220
 - fetching records, 223
 - push notifications, 227
 - record identifiers, 222
 - record zones, 222
 - records, 221-222
 - saving records, 224-226
 - subscriptions to data changes, 227-228
 - updating records, 226
 - user discovery/management, 229-233
 - CloudTracker sample app, 218, 228
 - components of, 217
 - HandOffNotes sample app, 249
 - HomeKit setup, 165-166
 - Key-Value Storage, 272
 - Photo Stream, 464
 - image handling, 375-376, 394**
 - Core Image filters, 383, 394
 - chaining filters, 390-391
 - face detector, 391-394
 - filter attributes, 386-388
 - filter categories, 383-386
 - initializing images, 388-389
 - rendering filtered images, 389-390
 - Image Picker, 379-382
 - ImagePlayground sample app, 375
 - images**
 - displaying, 377-379
 - initializing (Core Image Filters), 388-389
 - instantiating, 376-377
 - rendering filtered images (Core Image Filters), 389-390
 - resizing, 382-383
 - photo library, 449
 - PhotoLibrary sample app, 449-450
 - Photos framework, 449-450

Instruments (Xcode), 510-511, 519

- interface of, 511-514
- Leaks instrument, 516-518
- Time Profiler instrument, 514-516

iOS

- background-task processing, 333
- Continuity, 249
- foreground app, 333
- Handoff, 249
- Message Board sample app, 184-189
- provisioning profiles and push notification tests, 203-207

iPhones and music libraries, 127**item properties (physics simulations) and UIKit Dynamics, 11-13****iTunes Connect**

- achievements, adding to, 86
- Game Center, configuring behavior in, 63-64
- new apps, submitting to, 63

J

Jobs, Steve, 127**JSON (JavaScript Object Notation), 183, 193**

- benefits of, 183-184
- Message Board sample app, 184
- messages, posting, 189-191
- parsing, 186-187
- persistent data and, 273
- servers, getting JSON from, 185
 - building requests, 185-186
 - displaying data, 187-189
 - inspecting responses, 186
 - parsing JSON, 186-187
- servers, sending JSON to, 191-193
- website, 184

K

keyboards and Custom Keyboard Extensions, 238**Keychain sample app, 363-364, 374**

- apps, sharing between, 370-371
- attribute keys, 367
- dictionaries, securing, 368-370
- error codes, 372
- items, resetting, 370
- PIN, storing/retrieving, 366-367
- setup, 365-366
- updating, 363

keyed archives/coders and persistent data, 272

L

labels (Address Book), 115-116**latitude and longitude in Map apps, 22**

- geocoding addresses, 36-40
- reverse-geocoding addresses, 36, 40-43

leaderboards, 53, 83

- Apple's limit on number of leaderboards, 65
- authenticating, 68-73
- Combined Leaderboards, 64
- configuring, 64
- deleting, 64
- Game Center
 - authenticating leaderboards, 68-73
 - configuring behavior in iTunes Connect, 63-64
 - presenting leaderboards in, 77-79
 - score challenges, 79-81
 - submitting scores to, 73-76
- leaderboard systems, customizing, 81-82
- localization information, 66

- scores
 - formatting, 65-66
 - score challenges, 79-81
 - submitting to Game Center, 73-76
 - Single Leaderboards, 64-65
 - sorting, 66
 - Leaks instrument, 516-518**
 - life, displaying in Whack-a-Cac sample app, 60-61**
 - links, Dynamic Link Detection and UITextView, 423-424**
 - local notifications, 195-196, 216**
 - custom sound setup, 208-209
 - scheduling, 211-212
 - testing, 212
 - localization information**
 - achievements, 87
 - leaderboards, 66
 - locations (maps), 15**
 - annotations, 28
 - adding, 28-31
 - custom views, 31-33
 - displaying, 31-33
 - draggable views, 34
 - standard views, 31-33
 - Apple Maps, 15
 - Core Location, 15
 - importing, 16
 - user location requests, 16-24
 - geocoding addresses, 36-40
 - geofencing (regional monitoring), 43
 - boundary definitions, 44-45
 - monitoring changes, 45-46
 - regional monitoring capability checks, 43-44
 - GPS, 22
 - horizontal accuracy, 22
 - latitude and longitude, 22
 - map view, 28
 - MapKit, 15
 - displaying maps, 25-28
 - importing, 16
 - Maps.app, getting directions, 47-51
 - overlays, 28, 35-36
 - reverse-geocoding addresses, 36, 40-43
 - testing, 23-24
 - logging into Social Framework, 306-308**
 - longitude and latitude in Map apps, 22**
 - geocoding addresses, 36-40
 - reverse-geocoding addresses, 36, 40-43
 - LongRunningTasks sample app, 345-346**
 - background-task processing, 349-351
 - custom operations, 355-357
 - main thread, running, 347-349
 - operation queues, running in, 351
 - cancelling operations, 354-355
 - concurrent operations, 351-352
 - serial operations, 353-354
-
- ## M
- manifests (passes), 488**
 - MapKit, 15**
 - annotations, 28
 - adding, 28-31
 - custom views, 31-33
 - displaying, 31-33
 - draggable views, 34
 - standard views, 31-33
 - displaying maps, 25
 - coordinate systems, 25
 - Mercator Projection, 25
 - geocoding addresses, 36-40
 - importing, 16
 - map view, 28

MKMapKit, configuring/customizing, 25-26
 overlays, 28, 35-36
 reverse-geocoding addresses, 36, 40-43
 user interactions, responding to, 27-28

maps, 15

annotations, 28
 adding, 28-31
 custom views, 31-33
 displaying, 31-33
 draggable views, 34
 standard views, 31-33
 Apple Maps, 15
 Core Location, 15
 importing, 16
 user location requests, 16-24
 geocoding addresses, 36-40
 geofencing (regional monitoring), 43
 boundary definitions, 44-45
 monitoring changes, 45-46
 regional monitoring capability checks, 43-44
 GPS, 22
 horizontal accuracy, 22
 latitude and longitude, 22
 map view, 28
 MapKit
 displaying maps, 25-28
 importing, 16
 Maps.app, getting directions, 47-51
 overlays, 28, 35-36
 reverse-geocoding addresses, 36, 40-43
 testing locations, 23-24

Mark II Aiken Relay Calculator, 504

Media Picker feature (music libraries), 138-141

memory management and NARC (New, Allow, Retain, Copy), 113-114

Mercator Projection in Map apps, 25

Message Board sample app, 184

MobileMe, 217

multiple session achievements (games), 101-102

multitasking and background-task processing, 335

music, playing in a background, 340-342

music libraries, 127, 144

Media Picker, 138-141
 playback engines, 129
 handling state changes, 132-137
 playback duration, 137-138
 registering notifications, 129-130
 repeat feature, 138
 shuffle feature, 138
 timers, 137-138
 user controls, 131-132
 Player sample app, 127-128
 handling state changes, 132-137
 playback duration, 137-138
 repeat feature, 138
 shuffle feature, 138
 timers, 137-138
 user controls, 131-132

Programmatic Picker, 141

 playing random songs, 141-142
 predicate song matching, 142-143

MyMovies sample app, 273

 displaying object data, 288-290
 friend chooser, 285-287
 movie display view, 287
 movie list view controller, 292-299
 predicates, 290-291
 Shared Movies tab, 291

N

NARC (New, Allow, Retain, Copy) and memory management, 113-114

NeXT EOF (Enterprise Object Framework) and Core Data, 271

notifications, 195

APN, 195-196, 216

Apple documentation, 214

feedback, 215

CloudTracker sample app, 228

custom sound setup, 208-209

local notifications, 195-196, 216

custom sound setup, 208-209

scheduling, 211-212

testing, 212

push notifications, 195-196, 216

APN, 195-196, 200, 214

App ID, 196-199

app setup, 196-199

CloudKit, 227

custom sound setup, 208-209

development provisioning profiles, 203-207

Development Push SSL Certificates, 200-203

iOS provisioning profiles, 203-207

sending, 214-215

servers, 213-214

testing, 203-207, 212

receiving, 212-213

registering for, 209-211

ShoutOut sample app, 196

receiving push notifications, 215

registering for notifications, 209-211

NSDictionaries, 367

NSLayoutManager (UIKit), 420-421

NSLayoutManagerDelegate, 423

NSTextContainer, 423

NSTextStore, 421

NSUserDefaults and persistent data, 272

O

object databases. See Core Data

operation queues and GCD (Grand Central Dispatch), 351, 361

cancelling operations, 354-355

concurrent operations, running, 351-352

custom operations, 355-357

serial operations, 353-354

OS X Yosemite

Continuity, 249

Handoff, 249-250

overlays in Map apps, 28, 35-36

P

page ranges, setting for printing, 263-264

parsing JSON, 186-187

partially earned achievements (games), 99-100

Passbook, 467, 502

Pass Test sample app, 468

passes

adding, 494-497

app interactions, 491-494

barcode information, 477

boarding passes, 469

building, 474-481

coupons, 469-471

customizing appearance of, 468-478

designing, 468-474

- events, 469, 471
- fields, 478-481
- generic passes, 469, 471-472
- identification, 476
- manifests, 488
- packaging, 489
- Pass Type ID, 481-483
- presenting, 473-474
- removing, 500-501
- relevance, 476-477
- showing, 499
- signing, 489
- signing certificates, 483-488
- simulating updates, 497-499
- store cards, 469, 472-473
- testing, 489-490
- types of, 469
- updating, 497-499, 501

PassKit, 467, 502

password security. See security

pausing games, Whack-a-Cac sample app, 62

PDF (Portable Document Format), printing, 269-270

People Picker (Address Book), 118-120

contacts

- creating, 122-125
- editing, 120-121
- viewing, 120-121

customizing, 120

performance and GCD (Grand Central Dispatch), 345, 361

- dispatch queues, 357, 361
 - concurrent dispatch queues, 357-359
 - serial dispatch queues, 359-361

- LongRunningTasks sample app, 345-346
 - running in operation queues, 351-357
 - running main threads, 347-349
- operation queues, running in, 361
 - cancelling operations, 354-355
 - concurrent operations, 351-352
 - custom operations, 355-357
 - serial operations, 353-354
- queues, 347

permissions

- HealthKit permission requests, 150
- photo library, 451-453

persistent data

- coders/keyed archives, 272
- Core Data, 271-273, 299, 303
 - adding managed objects, 299-300
 - building managed object models, 278-282
 - default data setup, 282-284
 - displaying managed objects, 285-291
 - editing managed objects, 301
 - environment of, 275-278
 - EOF and, 271
 - features of, 271
 - fetchd results controller, 292-299
 - MyMovies sample app, 273
 - removing managed objects, 300-301
 - rolling back changes to managed objects, 301-303
 - saving changes to managed objects, 301-303
- SQLite, 271
- starting projects, 274-278

- CSV, 273
- Direct SQLite, 273
- iCloud Key-Value Storage, 272
- JSON, 273
- MyMovies sample app, 273
 - displaying object data, 288-290
 - friend chooser, 285-287
 - movie display view, 287
 - movie list view controller, 292-299
 - predicates, 290-291
 - Shared Movies tab, 291
- NSUserDefaults, 272
- plist (Property List), 272
- structured text files, 273
- Photo Editing Extensions, 238**
- photo library, 449, 451, 459, 465**
 - asset collections, 453-457, 459-461
 - assets, 457-458, 462-464
 - permissions, 451-453
 - Photo Stream, 464
 - PhotoLibrary sample app, 449-450
 - Photos framework, 449-450
 - PHAsset, 450
 - PHAssetCollection, 450
 - PHFetchResult, 450
 - PHImageManager, 450
 - PHPhotoLibrary, 450
- PhotoGallery sample app, 395-396**
- physics simulators and UIKit Dynamics, 1, 3, 14**
 - attachments, 7-8
 - classes of, 2
 - collisions, 3-6
 - dynamic behavior, 2
 - gravity, 3-4
 - introduction to, 2
 - item properties, 11-13
 - push forces, 10-11
 - sample app, 1
 - snaps, 9
 - springs, 8-9
 - UIAttachmentBehavior class, 2
 - UICollisionBehavior class, 2
 - UIDynamicAnimator, 2-3, 13
 - UIDynamicAnimatorDelegate, 13
 - UIDynamicItem protocol, 1, 12
 - UIDynamicItemBehavior class, 2
 - UIGravityBehavior class, 2
 - UIPushBehavior class, 2
 - UISnapBehavior class, 2
- pictures. See image handling; photo library**
- piggybacked achievements (games), 102-103**
- pinch gesture recognizers, 440-441**
- playlist engines**
 - playlist duration, 137-138
 - repeat feature, 138
 - shuffle feature, 138
 - state changes, handling, 132-137
 - timers, 137-138
 - user controls, 131-132
- playlist engines (music libraries), 129-130**
- Player sample app (music libraries), 127-128**
 - playlist duration, 137-138
 - repeat feature, 138
 - shuffle feature, 138
 - state changes, handling, 132-137
 - timers, 137-138
 - user controls, 131-132
- plist (Property List) and persistent data, 272**
- Point Value attribute (achievements), 87**
- predicates, displaying managed objects in Core Data, 290-291**

printing

- AirPrint, 259, 270
 - error handling, 264
 - page ranges, 263-264
 - Print Center app, 266-267
 - printer compatibility, 259
 - Printer Simulator tool, 259, 265
 - printing PDF, 269-270
 - printing rendered HTML, 268-269
 - printing text, 261-265
 - starting jobs, 264-265
 - testing, 259, 261
 - UIPrintInteractionController-Delegate, 267
- duplexing, 262-263
- Print Center app, 266-267
- Print sample app, 260
- Printopia, 259

privacy

- Address Book, 110
- HealthKit, 145-146

Programmatic Picker feature (music libraries), 141, 144

- predicate song matching, 142-143
- random songs, playing, 141-142

properties of items (physics simulations) and UIKit Dynamics, 11-13**Property List (plist) and persistent data, 272****protecting data**

- Keychain sample app, 363-364, 374
 - attribute keys, 367
 - error codes, 372
 - resetting items, 370
 - securing dictionaries, 368-370
 - setup, 365-366
 - sharing between apps, 370-371

- storing/retrieving PIN, 366-367
- updating, 363

Touch ID, 374

- error codes, 373
- implementing, 372-373

push forces (physics simulations) and UIKit Dynamics, 10-11**push notifications, 195-196, 216**

- APN, 195-196
 - Apple documentation, 214
 - Development Push SSL Certificates, 200
- App ID, 196-199
- app setup, 196-199
- CloudKit, 227
- custom sound setup, 208-209
- development provisioning profiles, 203-207
- Development Push SSL Certificates, 200-203
- iOS provisioning profiles, 203-207
- sending, 214-215
- servers, 213-214
- testing, 203-207, 212

Q

queues and GCD (Grand Central Dispatch), 347

- dispatch queues, 357, 361
 - concurrent dispatch queues, 357-359
 - serial dispatch queues, 359-361
- operation queues, running in, 351, 361

R

receiving notifications, 212-213**record identifiers (CloudKit), 222****record zones (CloudKit), 222**

records (CloudKit), 221-222

- creating, 224-226
- fetching, 223
- saving, 224-226
- updating, 226

regional monitoring. See geofencing

relationships, establishing in managed object models in Core Data, 281

remote notifications. See push notifications

rendered HTML, printing, 268-269

repeat feature (playback engines), 138

reporting achievements (games), 90-92

resetting achievements (games), 104-105

resizing images, 382-383

resuming (pausing) games, Whack-a-Cac sample app, 62

reverse-geocoding addresses in Map apps, 36, 40-43. See also geocoding addresses in Map apps

Rooms (HomeKit), 162, 168-169

Ruby on Rails and Message Board sample app, 184

- JSON, encoding, 189-191
- server access, 184

S

saving records (CloudKit), 224-226

scheduling

- actions (HomeKit), 181
- local notifications, 211-212

scores

- Game Center
 - customizing leaderboard systems, 81-82
 - score challenges, 79-81
 - submitting to, 73-76

- Whack-a-Cac sample app
 - adding scores to, 76-77
 - displaying, 60
 - formatting, 65-66

security

- Keychain sample app, 363-364, 374
 - attribute keys, 367
 - error codes, 372
 - resetting items, 370
 - securing dictionaries, 368-370
 - setup, 365-366
 - sharing between apps, 370-371
 - storing/retrieving PIN, 366-367
 - updating, 363

Touch ID, 374

- error codes, 373
- implementing, 372-373

serial operations, running, 353-354

Service Groups (HomeKit), 176-178

Services (HomeKit), 176-178

Share Extensions, 238

Shared Movies tab (MyMovies sample app), 291

sharing information between host apps and extensions, 243-244

ShoutOut sample app, 196

- notifications, registering for, 209-211
- push notifications, receiving, 215

shuffle feature (playback engines), 138

Sina Weibo and Social Framework, 305

Single Leaderboards, 64-65

sizing images, 382-383

SLComposeViewController, 308-310

snaps (physics simulations) and UIKit Dynamics, 9

Social Framework, 305, 331

- logins, 306-308
- posting to
 - Facebook, 311, 315-321
 - Twitter, 311
- SLComposeViewController, 308-310
- SocialNetworking sample app, 305-306
- user timelines, accessing
 - Facebook timelines, 322, 327-331
 - Twitter timelines, 322-327

songs in Programmatic Picker (music libraries)

- predicate song matching, 142-143
- random songs, playing, 141-142

sort-order option (Game Center), 66

sound (custom) and notifications, 208-209

springs (physics simulations) and UIKit Dynamics, 8-9

SpriteKit, 2

SQLite

- Core Data and, 271
- Direct SQLite and persistent data, 273

SSL (Secure Socket Layer) and Development Push SSL Certificates, 200-203

store cards (Passbooks), 469, 472-473

storing

- achievement precision (games), 102-103
- iCloud Key-Value Storage and persistent data, 272
- PIN in Keychain sample app, 366-367

street addresses, handling in Address Book, 116-117

structured text files and persistent data, 273

subclasses, customized in managed object models in Core Data, 282

submitting

- new apps to iTunes Connect, 63
- scores to Game Center, 73-76

subscribing to data changes in CloudKit, 227-228

symbolic breakpoints (debugging), 508

T

tap gesture recognizers, 436, 438-440

temperature (body), reading/writing data in HealthKit, 155-160

testing

- Accessory Simulator tests (HomeKit), 179-180
- AirPrint, 259, 261
- Handoff, 251
- local notifications, 212
- passes (Passbook), 489-490
- push notifications, 203-207

text

- AirPrint, printing text via, 261-262
 - configuring print info, 262-263
 - duplexing, 262-263
 - error handling, 264
 - page ranges, 263-264
 - Printer Simulator tool, 265
 - starting print jobs, 264-265
- Core Text, 419
- UIKit, 419, 433
 - changing font settings (text), 432
 - content specific highlighting, 427-431
 - Dynamic Link Detection, 423-424
 - Dynamic Type, 432
 - exclusion paths, 425-426
 - hit detection, 424-425
 - NSLayoutManager, 420-423
 - sample app, 420

Time Profiler instrument, 514-516

timers

- playback engines, 137-138
- timer-based achievements (games), 103-104

Today Extensions, 237, 240, 242

Touch ID, 374

- error codes, 373
- implementing, 372-373

triggers (HomeKit), 181

Twitter and Social Framework, 305, 331

- logins, 306-308
- posting to Twitter, 311-315
- SLComposeViewController, 308-310
- SocialNetworking sample app, 305-306
- user timelines, accessing, 322-327

U

UIAttachmentBehavior class, 2

UICollisionBehavior class, 2

UIDynamicAnimator, 2, 13

- creating, 3
- multiple instances of, 3

UIDynamicAnimatorDelegate, 13

UIDynamicItem protocol, 1, 12

UIDynamicItemBehavior class, 2

UIGravityBehavior class, 2

UIKit Dynamics, 1, 14

- attachments, 7-8
- classes of, 2
- collisions, 3-6
- dynamic behavior, 2
- gravity, 3-4
- introduction to, 2
- item properties, 11-13
- push forces, 10-11

sample app, 1

snaps, 9

springs, 8-9

UIAttachmentBehavior class, 2

UICollisionBehavior class, 2

UIDynamicAnimator, 2, 13

creating, 3

multiple instances of, 3

UIDynamicAnimatorDelegate, 13

UIDynamicItem protocol, 1, 12

UIDynamicItemBehavior class, 2

UIGravityBehavior class, 2

UIPushBehavior class, 2

UISnapBehavior class, 2

UIPrintInteractionControllerDelegate, 267

UIPushBehavior class, 2

UISnapBehavior class, 2

unearned/earned achievements (games), 98-99

updating

passes (Passbook), 497-499, 501

records (CloudKit), 226

UTI (Uniform Type Indicators), Handoff and document-based app implementations, 256

V - W

WatchKit, 244-247

Whack-a-Cac sample app, 53-55, 63

achievements, 97-98

earned/unearned achievements, 98-99

hooks, 92-93

multiple session achievements, 101-102

partially earned achievements, 99-100

piggybacked achievements, 102-103

- storing achievement precision, 102-103
- timer-based achievements, 103-104
- cacti (cactus)
 - interaction with, 58-60
 - spawning, 55-58
- Game Center Manager and, 66-68
- leaderboards, configuring, 65
- life, displaying, 60-61
- pausing games, 62
- resuming games, 62
- score, displaying, 60
- scores, submitting, 76-77

WWDC 2014 and HealthKit, 145

X

Xcode

- background-task processing
 - executing tasks, 335-336
 - types of background activities, 339-340
- CloudKit
 - account setup, 217-219
 - enabling iCloud capabilities, 220
- Core Data
 - building managed object models, 278-282
 - fetching results controller, 292-299
 - starting projects, 274-278
- debugging, 504-505, 509-520
- HomeKit
 - capability setup, 163-164
 - developer account setup, 163
- Instruments, 510-511, 519
 - interface of, 511-514
 - Leaks instrument, 516-518
 - Time Profiler instrument, 514-516
- testing locations in Map apps, 23-24

Y

Yosemite (OS X)

- Continuity, 249
- Handoff, 249-250

Z

Zones (HomeKit), 169-170