

# iOS 5 Core Frameworks

## DEVELOP AND DESIGN

Working with graphics, location, iCloud, and more



Shawn Welch

# iOS 5 Core Frameworks

DEVELOP AND DESIGN

Working with graphics, location, iCloud, and more

**Shawn Welch**



## **iOS 5 Core Frameworks: Develop and Design**

Shawn Welch

### **Peachpit Press**

1249 Eighth Street  
Berkeley, CA 94710  
510/524-2178  
510/524-2221 (fax)

Find us on the Web at: [www.peachpit.com](http://www.peachpit.com)

To report errors, please send a note to [errata@peachpit.com](mailto:errata@peachpit.com)

Peachpit Press is a division of Pearson Education

Copyright © 2012 by Shawn Welch

**Editor:** Nancy Peterson

**Production editor:** Myrna Vladic

**Development editor:** Margaret S. Anderson/Stellarvisions

**Copyeditor and proofreader:** Jan B. Seymour

**Technical editor:** Scott Fisher

**Cover design:** Aren Howell Straiger

**Cover production:** Jaime Brenner

**Interior design:** Mimi Heft

**Compositor:** David Van Ness

**Indexer:** Jack Lewis

### **Notice of Rights**

All rights reserved. No part of this book may be reproduced or transmitted in any form by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. For information on getting permission for reprints and excerpts, contact [permissions@peachpit.com](mailto:permissions@peachpit.com).

### **Notice of Liability**

The information in this book is distributed on an "As Is" basis, without warranty. While every precaution has been taken in the preparation of the book, neither the author nor Peachpit Press shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the instructions contained in this book or by the computer software and hardware products described in it.

### **Trademarks**

iOS is a trademark of Apple Inc., registered in the United States and other countries.

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 Peachpit Press was aware of a trademark claim, the designations appear as requested by the owner of the trademark. All other product names and services identified throughout this book are used in editorial fashion only and for the benefit of such companies with no intention of infringement of the trademark. No such use, or the use of any trade name, is intended to convey endorsement or other affiliation with this book.

ISBN 13: 978-0-321-80350-4

ISBN 10: 0-321-80350-7

9 8 7 6 5 4 3 2 1

Printed and bound in the United States of America

*To my brothers, Eric, Danny, and Kyle Welch.*

*Thank you for keeping me humble and  
reminding me of the humor in life.*

## ACKNOWLEDGMENTS

---

A book is one of those things that involves so many people besides the author listed on the cover. This book would not exist without the hard work of all those individuals. To all of the fine folks at Peachpit Press, thank you for your time and energy in this project.

Margaret Anderson, Nancy Peterson, and Jan Seymour read through my early writings and helped me turn them into the book you're reading today. Without their guidance and expertise in communication, this book would not have been possible. I am truly blessed to have worked with such a solid, professional, and savvy team. I hope to work again with all of them in the future. Scott "Fish" Fisher, my tech editor, played an equally important role of double-checking my code samples to be sure they were accurate, simple, and to the point. Thanks, Fish.

For people not directly involved in this book, I want to thank the folks at Flipboard for their help answering questions. Also, thanks to Charles Ying for reading some early drafts and serving as a test audience. You guys are a top notch team and I love your work.

As a side note, I wouldn't be where I am today without the teaching efforts of Evan Doll and Alan Cannistraro. Thank you, guys.

Finally, I would like to thank everyone over at Kelby Media Group and those who use my apps. iOS is a platform that is ever changing. For this reason I am constantly learning and applying knowledge to new apps. Kelby Media Group, specifically Scott Kelby and Dave Moser, have offered me so many opportunities to continue to work with their team and perfect my craft. To the fine users of NAPP who download my apps, thank you for your feedback. Without users, an app developer's life is pretty boring.

—Shawn Welch  
@shawnwelch

# CONTENTS

|                  |                                                         |          |
|------------------|---------------------------------------------------------|----------|
|                  | Acknowledgments .....                                   | iv       |
|                  | Welcome to iOS Core Frameworks .....                    | xii      |
| <b>CHAPTER 1</b> | <b>iOS FRAMEWORKS .....</b>                             | <b>2</b> |
|                  | Before We Begin .....                                   | 4        |
|                  | Prerequisites .....                                     | 4        |
|                  | My Goals for This Book .....                            | 6        |
|                  | iOS Frameworks Crash Course .....                       | 8        |
|                  | Understanding the Impact of a Multicore Processor ..... | 11       |
|                  | The Need for Concurrency .....                          | 12       |
|                  | Operation Queues vs. Dispatch Queues .....              | 13       |
|                  | Blocks .....                                            | 14       |
|                  | Using Blocks in iOS Frameworks .....                    | 16       |
|                  | The iOS 5 Top Ten Technologies .....                    | 18       |
|                  | Wrapping Up .....                                       | 20       |

---

## PART I: YOUR DATA AND THE CLOUD

|                  |                                                   |           |
|------------------|---------------------------------------------------|-----------|
| <b>CHAPTER 2</b> | <b>CORE DATA .....</b>                            | <b>22</b> |
|                  | Getting Started with Core Data .....              | 24        |
|                  | What Is Core Data? .....                          | 24        |
|                  | Core Data Stack .....                             | 25        |
|                  | Setting Up Core Data in Xcode .....               | 34        |
|                  | Creating a Managed Object Model .....             | 35        |
|                  | Creating a New Persistent Store Coordinator ..... | 37        |
|                  | Adding New Persistent Stores .....                | 38        |
|                  | Creating a New Managed Object Context .....       | 39        |
|                  | Interacting with Core Data .....                  | 41        |
|                  | Adding New Objects .....                          | 41        |
|                  | Fetching and Modifying Objects .....              | 42        |
|                  | Deleting Objects .....                            | 44        |
|                  | Undo, Redo, Rollback, and Reset .....             | 45        |
|                  | Wrapping Up .....                                 | 49        |

|                  |                                                 |           |
|------------------|-------------------------------------------------|-----------|
| <b>CHAPTER 3</b> | <b>iCLOUD</b>                                   | <b>50</b> |
|                  | Getting Started With iCloud                     | 52        |
|                  | How Does iCloud Work?                           | 53        |
|                  | Before You Begin                                | 56        |
|                  | iCloud Storage Containers                       | 60        |
|                  | Special Considerations when Working with iCloud | 61        |
|                  | Key-Value Storage                               | 63        |
|                  | Using the Ubiquitous Key-Value Store            | 64        |
|                  | Adding and Removing Objects                     | 65        |
|                  | Responding to Change Notifications              | 67        |
|                  | Syncing Core Data                               | 70        |
|                  | Determining if iCloud Is Available              | 71        |
|                  | Setting Up iCloud Syncing Persistent Store      | 72        |
|                  | Core Data Persistent Store Change Notifications | 74        |
|                  | iCloud Document Storage                         | 76        |
|                  | Wrapping Up                                     | 78        |

## PART II: LOCATION AND ACCOUNT SERVICES

|                  |                                                                 |           |
|------------------|-----------------------------------------------------------------|-----------|
| <b>CHAPTER 4</b> | <b>CORE LOCATION AND MAP KIT</b>                                | <b>80</b> |
|                  | Getting Started with Core Location and Map Kit                  | 82        |
|                  | How Location Is Determined                                      | 83        |
|                  | Location Permissions                                            | 86        |
|                  | The Core Location Manager                                       | 91        |
|                  | Standard Location Service                                       | 91        |
|                  | Significant Location Change Monitoring                          | 94        |
|                  | Heading Monitoring                                              | 96        |
|                  | Region Monitoring                                               | 98        |
|                  | Responding to New Information from the<br>Core Location Manager | 100       |
|                  | Forward and Reverse Geocoding                                   | 102       |
|                  | Geocoding Benefits                                              | 102       |
|                  | Geocoding Drawbacks                                             | 103       |
|                  | Forward Geocoding                                               | 103       |
|                  | Reverse Geocoding                                               | 106       |

|                  |                                                                         |            |
|------------------|-------------------------------------------------------------------------|------------|
|                  | Working with Map Kit .....                                              | 108        |
|                  | Tracking Location with Map Kit .....                                    | 108        |
|                  | Wrapping Up .....                                                       | 110        |
| <b>CHAPTER 5</b> | <b>SYSTEM ACCOUNTS AND NATIVE TWITTER APIS .....</b>                    | <b>112</b> |
|                  | Getting Started with System Accounts and Twitter .....                  | 114        |
|                  | Understanding OAuth Services .....                                      | 115        |
|                  | The Accounts Workflow .....                                             | 116        |
|                  | Using the Accounts Framework .....                                      | 118        |
|                  | New Apps and the Accounts Framework .....                               | 118        |
|                  | Accessing Accounts in the Accounts Framework .....                      | 121        |
|                  | Migrating Users from Existing Apps into the<br>Accounts Framework ..... | 122        |
|                  | Special Considerations .....                                            | 125        |
|                  | Easy Twitter: Tweet Compose View Controller .....                       | 128        |
|                  | Using the Tweet Compose View Controller .....                           | 128        |
|                  | Interacting with the Twitter API .....                                  | 132        |
|                  | Creating a TWRequest Object .....                                       | 132        |
|                  | Performing a TWRequest .....                                            | 134        |
|                  | Handling a TWRequest Response .....                                     | 136        |
|                  | Wrapping Up .....                                                       | 138        |

---

## PART III: GRAPHICS, IMAGES, AND ANIMATION

|                  |                                          |            |
|------------------|------------------------------------------|------------|
| <b>CHAPTER 6</b> | <b>CORE GRAPHICS .....</b>               | <b>140</b> |
|                  | Getting Started with Core Graphics ..... | 142        |
|                  | Core Graphics and Quartz 2D .....        | 144        |
|                  | Points vs. Pixels .....                  | 145        |
|                  | The Graphics Context .....               | 147        |
|                  | Why Use Core Graphics? .....             | 148        |
|                  | Understanding Core Graphics .....        | 149        |
|                  | Drawing Custom UIViews .....             | 149        |
|                  | Graphics Context Stack .....             | 151        |

|                  |                                                       |            |
|------------------|-------------------------------------------------------|------------|
|                  | Paths, Gradients, Text, and Images .....              | 155        |
|                  | <i>Paths</i> .....                                    | 155        |
|                  | <i>Gradients</i> .....                                | 158        |
|                  | <i>Text</i> .....                                     | 159        |
|                  | <i>Images</i> .....                                   | 160        |
|                  | Wrapping Up .....                                     | 161        |
| <b>CHAPTER 7</b> | <b>CORE IMAGE</b> .....                               | <b>162</b> |
|                  | Getting Started with Core Image .....                 | 164        |
|                  | <i>Why Use Core Image?</i> .....                      | 165        |
|                  | Understanding the Core Image Framework .....          | 168        |
|                  | <i>Core Image Still Images and Video Frames</i> ..... | 170        |
|                  | <i>Core Image Filters</i> .....                       | 173        |
|                  | <i>Core Image Context</i> .....                       | 178        |
|                  | <i>Analyzing Images</i> .....                         | 182        |
|                  | <i>Don't Block the Main Thread!</i> .....             | 184        |
|                  | Example: Core Image Editor .....                      | 187        |
|                  | Wrapping Up .....                                     | 191        |
| <b>CHAPTER 8</b> | <b>CORE ANIMATION</b> .....                           | <b>192</b> |
|                  | Getting Started with Core Animation .....             | 194        |
|                  | Core Animation Using UIKit .....                      | 197        |
|                  | <i>UIKit Animations with Animation Contexts</i> ..... | 197        |
|                  | <i>UIKit Animations with Animation Blocks</i> .....   | 199        |
|                  | Understanding Custom Core Animation Effects .....     | 201        |
|                  | <i>Core Animation Layer (CALayer)</i> .....           | 201        |
|                  | <i>Implicit vs. Explicit Animations</i> .....         | 202        |
|                  | <i>Core Animation Object (CAAnimation)</i> .....      | 203        |
|                  | <i>My First Custom Animation</i> .....                | 204        |
|                  | Core Animation Examples .....                         | 206        |
|                  | <i>Keyframe Animations</i> .....                      | 206        |
|                  | <i>3D Transforms</i> .....                            | 209        |
|                  | <i>Particle Emitters</i> .....                        | 213        |
|                  | Wrapping Up .....                                     | 217        |

---

## PART IV: MULTIMEDIA: AUDIO AND VIDEO

|                   |                                                       |            |
|-------------------|-------------------------------------------------------|------------|
| <b>CHAPTER 9</b>  | <b>CORE AUDIO</b>                                     | <b>220</b> |
|                   | Multimedia Frameworks                                 | 222        |
|                   | Getting Started with Audio                            | 224        |
|                   | <i>Why Is Audio Important?</i>                        | 225        |
|                   | Using the iPod Music Library                          | 227        |
|                   | <i>Media Picker Controller</i>                        | 227        |
|                   | <i>Music Player Controller</i>                        | 229        |
|                   | <i>Music Player Notifications</i>                     | 230        |
|                   | Using Audio from Other Sources                        | 232        |
|                   | <i>AV Foundation Audio Session</i>                    | 233        |
|                   | <i>AV Audio Player</i>                                | 237        |
|                   | Wrapping Up                                           | 241        |
| <b>CHAPTER 10</b> | <b>AV FOUNDATION</b>                                  | <b>242</b> |
|                   | Getting Started with AV Foundation                    | 244        |
|                   | <i>Why Use AV Foundation?</i>                         | 244        |
|                   | <i>AV Foundation and Other Media-based Frameworks</i> | 246        |
|                   | Using Out-of-the-Box Solutions                        | 248        |
|                   | <i>UIImagePickerController</i>                        | 248        |
|                   | <i>Using MPMoviePlayerController</i>                  | 256        |
|                   | Creating a Custom Media Capture Solution              | 262        |
|                   | <i>The AVCaptureSession</i>                           | 262        |
|                   | <i>The AVCaptureVideoPreviewLayer</i>                 | 262        |
|                   | <i>Setting Up a Custom Image Capture</i>              | 263        |
|                   | Wrapping Up                                           | 271        |

---

## PART V: iOS 5 NEWSSTAND APPS

|                   |                                                                      |            |
|-------------------|----------------------------------------------------------------------|------------|
| <b>CHAPTER 11</b> | <b>NEWSSTAND KIT</b>                                                 | <b>274</b> |
|                   | Getting Started with Newsstand Kit                                   | 276        |
|                   | Setting Up a Newsstand App                                           | 277        |
|                   | Understanding Newsstand Apps                                         | 282        |
|                   | Newsstand Kit Library and Issues                                     | 282        |
|                   | Downloading Newsstand Kit Content                                    | 284        |
|                   | Starting a New Download                                              | 285        |
|                   | Handling Download Progress and Finished Downloads                    | 287        |
|                   | Updating the Appearance of a Newsstand App<br>to Reflect New Content | 290        |
|                   | Notifying Newsstand Apps                                             | 291        |
|                   | Using Apple Push Notification Service                                | 291        |
|                   | Registering for Newsstand Update Notifications                       | 293        |
|                   | Newsstand Push Notification Format                                   | 294        |
|                   | Responding to Remote Notifications                                   | 295        |
|                   | Special Considerations with Newsstand Apps                           | 296        |
|                   | Newsstand Apps Waking Up from Background                             | 296        |
|                   | Reconnecting Abandoned Asset Downloads                               | 297        |
|                   | Wrapping Up                                                          | 299        |
|                   | Index                                                                | 300        |



i

**WELCOME TO  
iOS 5 CORE  
FRAMEWORKS**

## WELCOME TO iOS 5 CORE FRAMEWORKS

In June 2007, Steve Jobs introduced the iPhone and changed our thinking about what is and should be possible with mobile devices. A year later Apple offered this uniquely powerful operating system to third-party app developers. Each release has taken it further and in the summer of 2010 it was re-branded as iOS. With iOS 5, Apple has integrated technologies previously reserved for desktop computers. With that in mind, here are a few things you should be familiar with before we get started.

### THE TOOLS

Because working with iOS apps requires a specific set of tools and resources, you must have access to the following resources before you can implement the examples presented in this book.



#### iOS DEVELOPER REGISTRATION

Some of the new technologies introduced in iOS 5 require testing on actual iOS hardware. Before you can install and run apps on iOS hardware, however, you must be a registered developer at [developer.apple.com](http://developer.apple.com) and you must pay the \$99 registration fee. For more information, visit [developer.apple.com](http://developer.apple.com).



#### XCODE

Free to registered iOS developers, Xcode is Apple's primary IDE (Integrated Development Environment). When you download and install Xcode, that install process will also include the iOS 5.0 SDK. These will be your primary development tools when working with frameworks in iOS 5.



#### iOS DEVICE

It might go without saying, but because certain examples presented in this book require iOS hardware, you should have access to at least one iOS device for testing purposes. Further, when working with iCloud, it might be necessary to have access to more than one device since iCloud syncing is designed to sync content between devices.

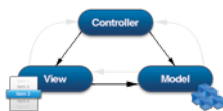


#### iCLOUD

Chapter 3 will focus primarily on iCloud, a cloud-based technology that services your apps with automatic synchronization and management of data between devices. Before you can use iCloud in your apps, however, you must have an iCloud enabled Apple ID. iCloud is free for all users (5 GB of storage) and registration can be completed at [icloud.com](http://icloud.com).

## THE CONCEPTS

*iOS 5 Core Frameworks* will depend heavily on the following concepts throughout examples and teaching narratives. While some explanation is given in the text, it would be helpful to familiarize yourself with these concepts beforehand.



### MODEL-VIEW-CONTROLLER

As with any software development, it's a good idea to be familiar with the Model-View-Controller (MVC) design pattern before you get started. This book will teach you about various core frameworks while assuming an understanding of this paradigm—especially when dealing with frameworks such as Core Data, Core Graphics, and even Newsstand Kit.



### GRAND CENTRAL DISPATCH

Grand Central Dispatch (GCD) is a multi-tasking library designed to take advantage of multicore processors. In iOS 5, most new frameworks will use GCD because of the optimizations it provides. We'll cover some of the basics as needed by this book, however, a familiarity with the concepts and challenges of GCD will be helpful.



### APPLE PUSH NOTIFICATION SERVICE

Apple Push Notification Service (APNS) is used to send notifications to devices so that applications can perform specific actions, even if they're not running when the notification is received. We will use APNS to deliver content update notifications to Newsstand Kit apps. While not covered in this book, a tutorial on APNS is available on [iOSSCoreFrameworks.com](http://iOSSCoreFrameworks.com).

---

**4**

# **CORE LOCATION AND MAP KIT**



One of the obvious benefits of iOS is that it's a mobile platform. iOS devices move throughout the world; calling upon a device's location while utilizing Core Location and Map Kit helps you provide a better context for the data in your app. For example, if a user is in San Francisco and does a search for "Bart," they're probably looking for information on the Bay Area Rapid Transit system (aka the BART), whereas in other parts of the world that same search may be looking for a pop culture reference. By pulling a user's location, you make the data in your app more relevant to your users. New to iOS 5 are enhancements to location services including forward and reverse geocoding, placemarks, and regions.

## GETTING STARTED WITH CORE LOCATION AND MAP KIT

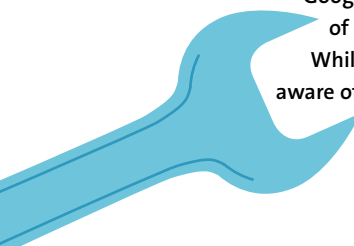
Core Location is a set of Objective-C classes built into the Core Services layer of iOS. Core Location was designed to simplify the process of working with location by providing a set of APIs that facilitate location monitoring and various location-data conversions (such as latitude/longitude coordinates to human readable addresses and vice versa). The Core Location framework is data oriented and can be used to obtain relevant location information as needed for check-in services, user tracking, nearby searches, and more.

The Core Location Manager (CLLocationManager) manages this flow of data, and controls your app's interaction with the physical hardware used to determine location. The location manager passes new location data that is retrieved from hardware to its delegate and then encapsulates it in a CLLocation object. This object contains a determination of the latitude and longitude coordinates of the device as well as information about the accuracy of the determination. The location manager also calculates speed and heading (based on the observed change in location), and likewise encapsulates it in the CLLocation object.

Unlike the Core Location framework, the Map Kit framework is visually oriented—it communicates location data back to a user through maps. Using Map Kit, you can seamlessly embed various map views into your app using Google Maps data as a service provider. Map Kit also has some very handy (and entirely automated) APIs designed for visually based real-time user tracking on a map.

When used in combination, Core Location and Map Kit allow you to create feature-rich, location-aware apps for all iOS devices.

**NOTE:** As mentioned by the Apple Terms of Service agreement and the Apple iOS Developer agreement, because the Map Kit framework uses Google Services, the use of this framework and data holds you liable to the Google Maps/Google Earth API terms of service. For more information about these terms of service, visit <http://code.google.com/apis/maps/iphone/terms.html>. While this will not be an issue for most apps, it's something you should be aware of when making your apps.

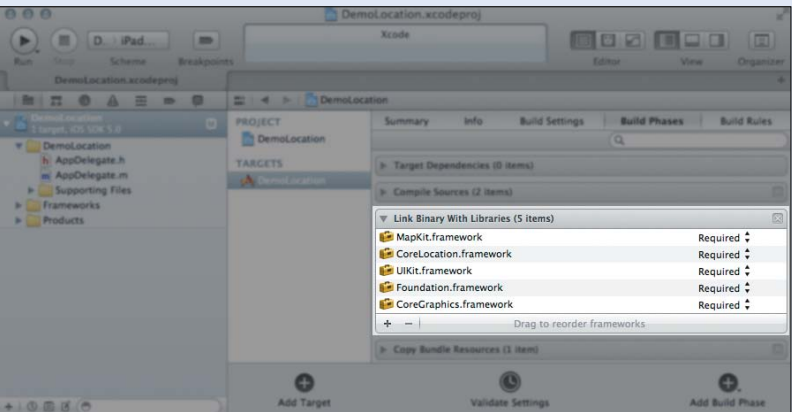


## LINKING CORE LOCATION AND MAP KIT FRAMEWORKS

Before you can make your app location aware, you must first link the Core Location framework to your project. If you plan to use map services (for example, to show locations on a map), you should also link the Map Kit framework. These frameworks are represented by the libraries *CoreLocation.framework* and *MapKit.framework*.

To add the Core Location and Map Kit frameworks to your project, refer to the procedures in Chapter 1, To Link New Frameworks in an Xcode Project, and add *CoreLocation.framework* and *MapKit.framework* (Figure 4.1). Next, import the following code in the appropriate header (.h) files:

```
1 #import <CoreLocation/CoreLocation.h>
2 #import <MapKit/MapKit.h>
```



**FIGURE 4.1** Core Location and Map Kit frameworks linked to your Xcode project.

## HOW LOCATION IS DETERMINED

When an iOS device attempts to pinpoint its location, it relies on three data sources to make the determination. Each of these data sources provides a different level of speed (the time it takes to make the determination), performance (the amount of power used to make the determination), and accuracy (the +/- distance in meters). **Table 4.1** (on the next page) highlights the three technologies used and ranks them based on their relative properties.

**TABLE 4.1** Location Determination Sources on iOS Devices

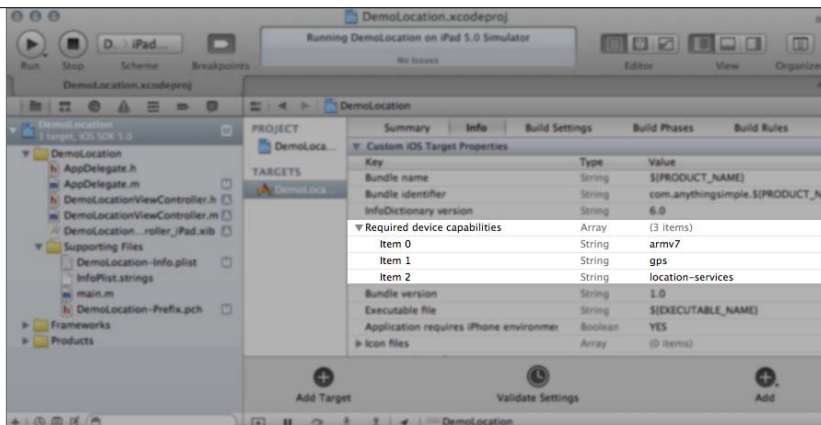
| SOURCE       | SPEED   | INTENDED ACCURACY    | POWER                                                                               |
|--------------|---------|----------------------|-------------------------------------------------------------------------------------|
| Cell Tower * | Fastest | City or region       | Fairly low, since 3G devices stay connected to towers.                              |
| Wi-Fi        | Medium  | City block or better | More than cell, but still low. Requires Wi-Fi to perform a scan of nearby networks. |
| GPS *        | Slowest | +/- 5m or better     | Fairly high compared to other methods, especially during continuous tracking.       |

\* Indicates this location data source is not available on all iOS hardware configurations.

As you can see, there are varying levels of performance, speed, and accuracy between each location source. Determining location through cell phone towers is very fast and very efficient but it's not the most accurate determination. This is not always an issue. For example, if you're making a taxi app and you want to list the phone numbers for all of the taxi services in the area, you probably only need to know what city someone is in. Since taxi services will drive a ways to pick you up, it's not necessarily relevant that a person is standing on the corner of Arlington and Boylston.

At the other end of the spectrum, a fitness app that tracks your running progress or a turn-by-turn GPS app would require more accurate location data. Your users would be annoyed if their turn-by-turn app missed a turn by about 100 meters. In between these extremes, where accuracy is important but not as much as turn-by-turn, would be a location check-in service. In this case it's not critical to your app's function that a person be in the exact location their device claims to be, so you can trade off the accuracy for better battery performance.

Another important take-away from Table 4.1 is that not every location data source is available on every iOS device. Only devices configured with a cellular radio (iPhones and 3G-enabled iPads) are able to determine location through cell towers. Additionally, GPS is only available on the iPhone models 3G and later and all 3G-enabled iPads. If accurate location data is *critical to the operation of your app* (such as for turn-by-turn navigation or Find My Friends), then you should configure your app's info property list to require the appropriate hardware.



**FIGURE 4.2** Location-based hardware requirements added to an app's info property list.

You can add two levels of restrictions for location-based hardware capabilities. When added to the `UIRequiredDeviceCapabilities` array in your info property list, these keys provide the following restrictions (**Figure 4.2**):

- `location-services`: Requires that device has some form of location service available. Used as a general restriction.
- `gps`: Requires device with GPS hardware.

Remember, add these keys to your app only if your app is unable to operate without them. If location is simply a nice feature that improves user experience, then your app should not require specific hardware. For example, a movie theatre app might work best when the app can automatically determine your location using hardware. But this app would also work if a user simply types in their ZIP code for nearby theaters. In this case, the app should not include location-services as required hardware.

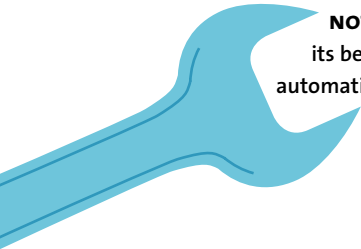
**NOTE:** The required hardware capability “armv7” in Figure 4.2 simply indicates that the app must run on an iOS device and will be in your required capabilities list by default when a new iOS app is created in Xcode.

Fortunately, while it is important for you to be aware of the various source hardware capabilities, it is not necessary for you to specify which piece of hardware your application should use—iOS chooses the hardware automatically. When working with the Core Location Manager to manage updates from hardware, you simply specify your desired accuracy. The desired accuracy of the location manager

is measured in meters and can be set using a `CLLocationAccuracy` constant. These constants are defined by the iOS SDK and indicate by name their intended use (Table 4.2).

TABLE 4.2 Core Location Accuracy Constants

| CONSTANT                                          | INTENDED USE                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|---------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>kCLLocationAccuracyBest</code>              | The default value for the location manager desired accuracy. In this condition, iOS does its best to provide the best location possible with location-based hardware.                                                                                                                                                                                                                                                                                   |
| <code>kCLLocationAccuracyBestForNavigation</code> | This condition is the most accurate of all the available configurations and should only be used in apps where absolute precision is necessary (turn-by-turn). iOS actually achieves better than “best” in this condition by using additional sensors beyond location-based hardware to provide highly-accurate data at all times. This condition is fairly power intensive and is designed to operate while the device is plugged in to a power source. |
| <code>kCLLocationAccuracyNearestTenMeters</code>  | Set the desired accuracy to 10 meters. This condition works well for check-in type applications.                                                                                                                                                                                                                                                                                                                                                        |
| <code>kCLLocationAccuracyHundredMeters</code>     | Set the desired accuracy to 100 meters. This condition works well for nearby services that operate under the assumption your user is walking (such as nearby restaurants or friends close by).                                                                                                                                                                                                                                                          |
| <code>kCLLocationAccuracyKilometer</code>         | Set the desired accuracy for 1 kilometer. This condition works well for city-based searches such as the nearest movie theater.                                                                                                                                                                                                                                                                                                                          |
| <code>kCLLocationAccuracyThreeKilometers</code>   | Set the desired accuracy for 3 kilometers. This condition works well for city-based searches where you’re looking for services available in that city and are not necessarily sorting by the closest service.                                                                                                                                                                                                                                           |



**NOTE:** While the accuracy can be defined, it is not a guarantee. iOS will do its best to optimize accuracy based on the conditions in the table and will automatically switch between available sources to reach the desired accuracy level.

LOCATION PERMISSIONS

I don’t know about you, but I can’t count how many times I’ve launched an app and was surprised to be asked for access to my location. Nine times out of ten, if I wasn’t expecting to provide an app with my location, I won’t allow it.



**FIGURE 4.3** A location services permission dialog.

*Custom purpose string configured by the Core Location Manager.*

The moral of this story is when you use location in apps, you have to ask for permission first—there’s no way around it. The harsh truth about location-aware apps is that many users don’t like providing apps with their location data. Not everyone will enable location for your app, even if it makes your app super awesome. So you need to be prepared to handle conditions where your app does not have permission to use the services you planned on using.

## CONTROLLING LOCATION PERMISSIONS

The first time your app attempts to determine a device’s location, iOS will prompt a permission dialog to the user that indicates your action. This action occurs whether you’re using a `CLLocationManager` (Core Location) or an `MKMapView` (Map Kit) configured to show the device’s location. By default, this dialog will simply say, “*Your App Would Like to Use Your Current Location*,” with the choices of *Don’t Allow* and *OK*. When you’re determining location using the `CLLocationManager`, you have the option of setting a purpose string, which is your opportunity to explain in the permission dialog *why* your app needs access to a user’s location. (Figure 4.3).

We'll get into the specifics of the `CLLocationManager` in the next section; however, while we're on the subject of permissions you can configure the custom purpose message of a `CLLocationManager` by setting its managed property purpose (Figure 4.3).

```
1  [locationManager setPurpose:@"My Custom Purpose Message..."];
```

#### DETERMINING LOCATION SERVICE AVAILABILITY

Before you attempt to use location services in your app, you should first check to see if they're available. There are many reasons why location services might be unavailable. First and foremost, the necessary hardware might be disabled because a device is in airplane mode or because the user has turned off location services globally for all apps. Second, a user might have disallowed access to your app specifically either in the location services permission dialog mentioned in the previous section or in the Settings app. Finally, the Parental Controls section of the Settings app on every iOS device allows parents the choice to prevent apps from using location data. This condition should be handled separately from the permission dialog because in this case your users will never be presented with a dialog asking for permission.

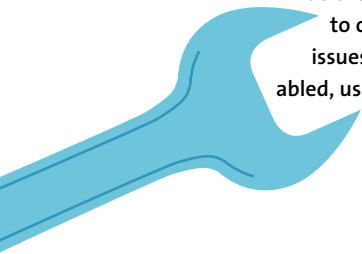
With these conditions in mind, the `CLLocationManager` offers two class methods that allow you to determine first, whether or not location services are enabled, and second, the authorization status of your specific app. These class methods are `[CLLocationManager locationServicesEnabled]` and `[CLLocationManager authorizationStatus]`, with the conditions and possible values demonstrated in the following code block:

```
1  // Check to see if location services are enabled
2  if([CLLocationManager locationServicesEnabled]){
3
4      NSLog(@"Location Services Enabled");
5
6      // Switch through the possible location
7      // authorization states
8      switch([CLLocationManager authorizationStatus]){
9          case kCLErrorAuthorizationStatusAuthorized:
10             NSLog(@"We have access to location services");
11             break;
12          case kCLErrorAuthorizationStatusDenied:
13             NSLog(@"Location services denied by user");
14             break;
15          case kCLErrorAuthorizationStatusRestricted:
16             NSLog(@"Parental controls restrict location services");
17             break;
18          case kCLErrorAuthorizationStatusNotDetermined:
19             NSLog(@"Unable to determine, possibly not available");
20     }
21 }
22 else{
23     // locationServicesEnabled was set to NO
24     NSLog(@"Location Services Are Disabled");
25 }
```

This code block is fairly straightforward. Functionally, we're not doing much more than printing log messages based on the possible location services enabled and location authorization states. In line 2 we first check to see if location services are enabled. If this condition results to NO, we jump down to line 22 and handle our disabled condition. This condition would result as NO if the device were in airplane mode or if location services were disabled globally in the Settings app. In lines 8 through 20 we handle the condition that location services *are* enabled by evaluating a switch statement based on the possible authorization status values. The possible values for the location authorization status are

- **kCLLocationAuthorizationStatusAuthorized:** Your app is able to use location services.
- **kCLLocationAuthorizationStatusDenied:** The user has chosen to deny your app access to location services.
- **kCLLocationAuthorizationStatusRestricted:** You do not have access to location services because availability is restricted by parental controls. This means the user will never be presented a permission dialog.
- **kCLLocationAuthorizationStatusNotDetermined:** Your app was unable to determine if location services are authorized. This authorization state is most likely caused by location services being disabled or some other fringe case caused by errors. In our code block, we would probably never reach this condition because we first check to see if location services are enabled. But, if you were to check this value outside of our code block while services are disabled, the status would be unknown.

**NOTE:** Even though the unknown authorization status is most likely the cause of services being disabled, you should not use this status value as a condition in your app indicating services are disabled. This status could also be the cause of some unknown error iOS experienced when attempting to check on your app's status (possibly caused hardware or software issues, uncorrectable by you or the user). To indicate services are disabled, use the `locationServicesEnabled` Boolean.



## THE CORE LOCATION MANAGER

As the first step in working with location data, we'll focus on Core Location. Remember, Core Location is primarily a data-oriented framework. This means we'll be dealing with coordinates, text strings, and number values instead of visual location information like maps. Later in the Map Kit section, we'll discuss how to use some of the data sources we learn about with Core Location in combination with Map Kit, and how to visually represent location information on a map.

I've mentioned the Core Location Manager (`CLLocationManager`) a few times. The `CLLocationManager` is responsible for controlling the flow and frequency of location updates provided by location hardware. Simply put, the location manager generates location objects (`CLLocation` objects) and passes them to its delegate whenever a certain set of criteria is met. These criteria are determined by how you configure and start your location manager.

The `CLLocationManager` is typically used to generate location data while working with one of the following services.

- Standard Location Service
- Significant Location Change Monitoring
- Heading Monitoring
- Region Monitoring

**NOTE:** You can have as many `CLLocationManager` objects as needed by your application, but each location manager should be used to monitor one service. In the case of regions, any region being monitored during the runtime lifecycle of your app will be added as a member of the `NSSet` property `monitoredRegions` in all `CLLocationManager` objects.

### STANDARD LOCATION SERVICE

The standard location service is one of the most common uses of the location manager. Used to determine a user's current location as needed for nearby searches or check-in locations, the standard location service can be configured with a desired accuracy and distance filter (which is the threshold used to determine when a new location should be generated). When a device moves beyond the configured distance filter, the standard location service triggers a new location and calls the necessary delegate methods. After creating the location manager and configuring the desired properties, call `startUpdatingLocation` to begin location services. The

following code block demonstrates how to set up a new location manager using the standard location service:

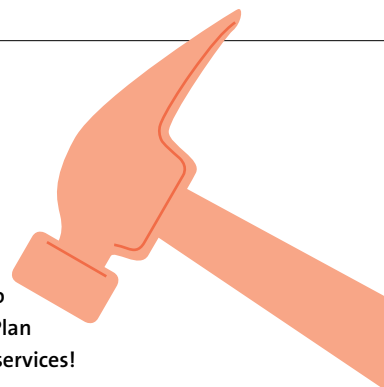
```
1 //Create a new location manager
2 locationManager = [[CLLocationManager alloc] init];
3
4 // Set Location Manager delegate
5 [locationManager setDelegate:self];
6
7 // Set location accuracy levels
8 [locationManager setDesiredAccuracy:kCLLocationAccuracyKilometer];
9
10 // Update again when a user moves distance in meters
11 [locationManager setDistanceFilter:500];
12
13 // Configure permission dialog
14 [locationManager setPurpose:@"My Custom Purpose Message..."];
15
16 // Start updating location
17 [locationManager startUpdatingLocation];
```

In this code block, line 2 allocates a new `CLLocationManager` and saves it as an instance variable named `locationManager`. In line 5 we set the location manager delegate as `self`, which means this class must observe the `CLLocationManagerDelegate` protocol (covered in the section below, Responding to New Information from the Core Location Manager). Next, in line 8 we set the desired accuracy of our location manager to 1 kilometer, and in line 11 we set the distance filter of our location manager to 500 meters.

While the distance filter can be almost any value, I personally have found that setting your distance filter to half the distance of your desired accuracy typically generates a fairly accurate sample of locations as needed by the location accuracy.

In line 14 we set the custom purpose message. Remember, this message will be shown in the location permission dialog as seen in Figure 4.3 and should be used to describe why your app needs access to a user's location—especially when it's not readily apparent. Finally, in line 17 we start updates on the location manager by calling `startUpdatingLocation`.

**TIP:** The permission dialog is presented once and only once to the user the first time your app calls `startUpdatingLocation`. Plan accordingly and be prepared for the user to disallow location services!



## USING STANDARD LOCATION SERVICES AS A BACKGROUND PROCESS

By default, your app will not run the standard location service as a background process. The standard location service significantly impacts your user's battery life if left running. Even if the location manager is not sending new locations to the delegate, the standard location service still continuously monitors a user's location to determine when the distance filter threshold is crossed. Unless the information generated is relevant to a user's current task, it's recommended that you disable this service for performance reasons. If you're doing a simple calculation for search purposes, you should turn off the standard location service as soon as you receive your first location update.

But perhaps location services are vital to the function of your app, such as a fitness app that continues to track a user's run in the background while they exit the app to select a new music playlist. In this case, you can add the mode *location* to the `UIBackgroundModes` array in your app's info property list.

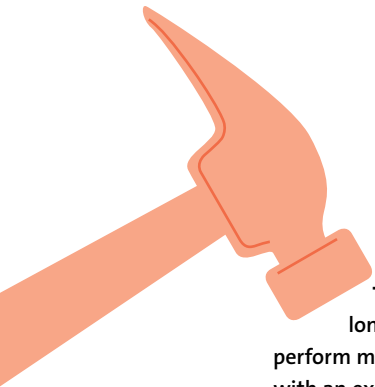
If your app needs location awareness while running in the background and you do not need the high-sample rate generated by standard location services, it's recommended that for better user experience you use the significant location change service described in the next section. For most apps, the accuracy and frequency of location updates provided by significant location change monitoring is sufficient for background needs.

**NOTE:** You can download a sample app that demonstrates the standard location service (with background support) at this book's website, [ioscoreframeworks.com/download#chapter-4](https://ioscoreframeworks.com/download#chapter-4).



## SIGNIFICANT LOCATION CHANGE MONITORING

The location manager of a *significant location change service* sends new locations to the delegate whenever it detects the device has significantly changed position. The location manager provides a starting location as soon as this service is started and future locations are only calculated and sent to the delegate when the device detects new Wi-Fi hotspots or cellular towers. While slightly similar to the standard location service in functionality, this method is much more aggressive in power management and is highly efficient. While the standard service continuously monitors a user's location to determine when the distance filter threshold is crossed, the significant location change disables location services between new location events (because location events are determined when new connections are located by the cellular and Wi-Fi antennae). As an added bonus, unlike the standard location service, the significant change monitoring service will wake up an app that's suspended or terminated and allow the app to execute any necessary background processes.

An orange hammer is shown from a side profile, pointing towards the right. It has a simple, stylized design with a flat head and a handle.

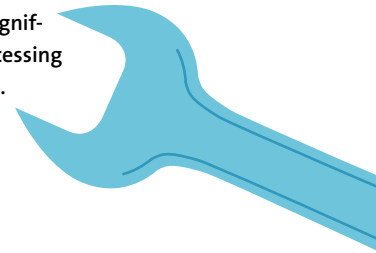
**TIP:** Your app only stays active in the background for a few seconds, long enough to perform simple calculations and updates. If you need to perform more complicated tasks, consider setting up a long background process with an expiration handler. For more information on long background processes, visit [iOSCoreFrameworks.com/reference#long-background-process](https://developer.apple.com/documentation/corelocation/ioscoreframeworks-com/reference/long-background-process).

The code needed to set up a significant location change monitoring is much simpler, essentially because the significant location change service is largely handled by the core operating system. Remember, the significant location change monitoring service automatically generates new locations when the radio antennae detect new connection sources (cellular or Wi-Fi). That means the significant location change service will ignore the accuracy and distance filter properties of the location manager (the accuracy will always be the best available by the sources used). The following code block demonstrates how to set up a location manager that monitors significant location changes:

```
1 //Create a new location manager
2 locationManager = [[CLLocationManager alloc] init];
3
4 // Set Location Manager delegate
5 [locationManager setDelegate:self];
6
7 // Configure permission dialog
8 [locationManager setPurpose:@"My Custom Purpose Message..."];
9
10 // Start updating location
11 [locationManager startMonitoringSignificantLocationChanges];
```

Very similar to the previous code block, starting the significant location change service simply involves creating a new location manager (line 2), setting the delegate (line 5), configuring an optional custom purpose message (line 8), and then calling the method, `startMonitoringSignificantLocationChanges` in line 11 (instead of `startUpdatingLocation`). Just like the standard location service, the significant location change service interacts with its delegate using the same methods, which is covered in the section that follows, Responding to New Information from the Core Location Manager.

**NOTE:** Download a complete project demonstrating the significant location change service and relevant background processing methods at [iOSCoreFrameworks.com/download#chapter-4](https://ioscoreframeworks.com/download#chapter-4).



---

## HEADING MONITORING

Heading information is a little different than the other location-based data types generated by the location manager. Unlike the previous services, the heading monitoring service only generates new heading information (direction information relative to magnetic north or true north). The heading object (CLHeading) is created using the device's magnetometer (compass) and does not contain a reference to the latitude and longitude coordinates of the device.

Just like with other location services, not all devices are equipped with a magnetometer, especially older generation models. You should first check to see if heading services are available by calling `[CLLocationManager headingAvailable]`, and if heading services are required for your app's function (such as a compass app) you should add the value `magnetometer` to your app's info property list.

One of the reasons heading monitoring exists as a separate service—besides separate hardware—is because doing so allows additional performance optimization. In most location-aware apps, you don't need to know the device heading with incredible accuracy. The majority of apps are able to get by on the generic speed and heading information generated in the standard and significant location change services. In these cases, the course and speed values managed by the `CLLocation` object are simply extrapolated based on the previous location coordinate (*distance* moved, *direction* moved). This means if your user is standing still, the `CLLocation` object is likely to hold invalid heading information.

Because heading monitoring is a separate service, however, you can give your users the option of turning on additional heading information as needed. This practice is observed in the native Google Maps app on the iPhone and iPad. When a user taps the location button once, the map zeros in on their current location. If they tap the location button again, the Google Maps app enables heading monitoring to indicate the direction the device is facing.

Starting a heading monitoring service is just like starting updates on a standard location service. The process involves creating a new location manager, assigning the delegate, setting your desired accuracy and threshold filter (in degrees changed), and calling `startUpdatingHeading`. Because the heading is dependent on the orientation of your device (landscape versus portrait), the location manager also allows you to set the desired heading orientation. The following code block demonstrates how to set up a new heading monitoring service:

---

```

1  if([CLLocationManager headingAvailable]){
2
3      // Create a new Location Manager and assign delegate
4      headingManager = [[CLLocationManager alloc] init];
5      [headingManager setDelegate:self];
6
7      //Send all updates, even minor ones
8      [headingManager setHeadingFilter:kCLHeadingFilterNone];
9
10     // Set heading accuracy
11     [headingManager setDesiredAccuracy:kCLLocationAccuracyBest];
12
13     // Set expected device orientation
14     [headingManager setHeadingOrientation:
15                                     CLDeviceOrientationLandscapeLeft];
16
17     // Start updating headings
18     [headingManager startUpdatingHeading];
19 }
20 else
21     NSLog(@"Heading not available");

```

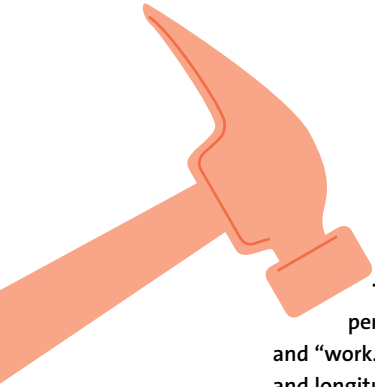
You'll notice this code block is similar to the standard location service. The first thing we do is check to see if heading services are available by calling the `CLLocationManager` class method `headingAvailable` in line 1. Next, in lines 4 and 5 we create a new `CLLocationManager` object and assign the delegate to `self`. In line 8 we set up our desired heading filter. This value specifies the minimum heading change in degrees needed to trigger a new heading event. In line 8 we set this option to the constant, `kCLHeadingFilterNone`. This simply sets the filter to nothing allowing us to obtain every change detected (no matter how minor) from the magnetometer. By default, this filter value is set to 1 degree.

In line 14 we set the expected orientation of our device to landscape left. The orientation will default to portrait, and if your device allows for rotation you should detect device rotations and reassign the heading orientation when appropriate. Finally, in line 17 we start updating our heading information. This begins calling the delegate method, `locationManager:didUpdateHeading:` when the filter threshold condition is met.

## REGION MONITORING

One of the newest features available in iOS 5 is the ability to add region-based monitoring services to a location manager. Region monitoring allows you to monitor a device's interaction with the bounds of defined areas or regions; specifically, the location manager will call `didEnterRegion` and `didExitRegion` on its assigned delegate when the bounds of monitored regions are crossed.

This new functionality allows for all sorts of app opportunities from auto-check-in services to real-time recommendations (for example, you're walking past a good coffee shop and an app on your phone knows that you like coffee). In fact, the new Reminders app for iOS 5 uses this functionality in combination with Siri (the iPhone 4S digital assistant) to carry out requests such as *"Remind me when I get home that I need to take out the trash,"* or *"Remind me when I leave the office that I need to call my wife and tell her I'm on my way."* In these examples, Siri simply defines a region in a core location manager for the locations *home* and *the office* and sets up a reminder to trigger when those regions detect the appropriate `didExitRegion` or `didEnterRegion` events.



**TIP:** When Siri sets up the regions, "she" will actually read your personal address card and look for an address labeled as "home" and "work." If detected, Siri will convert your home address to a latitude and longitude coordinate using the forward geocoding APIs and then set up a region based on that coordinate. More about this in the section below on geocoding.

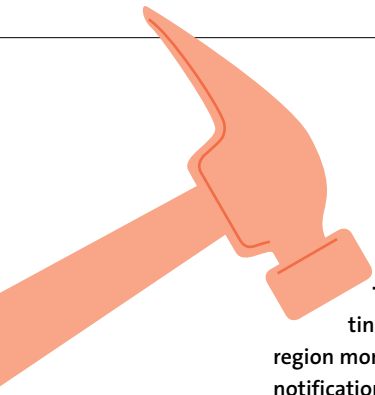
---

The process for monitoring regions is very similar to the other services we monitored. Instead of setting up distance filters or depending on cell towers to trigger new location events, however, we define a specific circular region (or regions) based on a set of latitude and longitude coordinates and a radius in meters.

The following code block demonstrates how to monitor for a region. This example assumes that you already know the latitude and longitude coordinates of your target region. Later, we'll cover how to generate these values using human-readable address strings, but for now, let's just assume you've memorized that Apple's main campus is located at the latitude and longitude coordinates of (37.331691, -122.030751).

```
1  // Create a new location manager
2  locationManager = [[CLLocationManager alloc] init];
3
4  // Set the location manager delegate
5  [locationManager setDelegate:self];
6
7  // Create a new CLRegion based on the lat/long
8  // position of Apple's main campus
9  CLLocationCoordinate2D appleLatLong =
    CLLocationCoordinate2DMake(37.331691, -122.030751);
10 CLRegion *appleCampus = [[CLRegion alloc]
    initWithCircularRegionWithCenter:appleLatLong
    radius:100
    identifier:@"Apple"];
11
12 // Start monitoring for our CLRegion using best accuracy
13 [locationManager startMonitoringForRegion:appleCampus
    desiredAccuracy:kCLLocationAccuracyBest];
```

In this example, we set up the location manager and delegate in lines 2 through 5. In line 9 we create a new `CLLocationCoordinate2D` using the latitude and longitude coordinates for Apple's main campus. Next, in line 10 we allocate a new



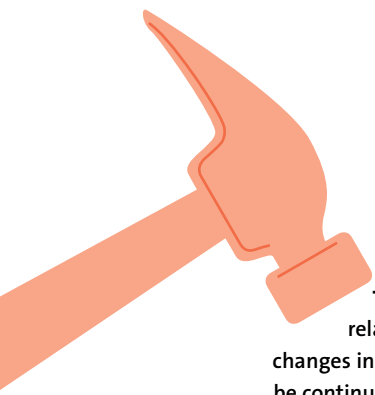
CLRegion. Notice we initialize this region as a circular region with the radius of 100 meters. This method also allows us to assign an identifier we can use to refer to the region at a later time (in the event you're monitoring more than one region in your location manager). Finally, in line 13 we simply start the monitoring service for our CLRegion by calling `startMonitoringForRegion:desiredAccuracy`.

**TIP:** In this example we used the `kCLLocationAccuracyBest` setting because our region radius is only 100 meters. The accuracy of the region monitoring will help eliminate false positives and prevent duplicate notifications by adding a small buffer zone. Make sure your accuracy radius is not too high compared to the radius of your region. For example, if you had a 50m radius defined, you wouldn't want your accuracy to be calculated using the 3 kilometer accuracy setting.

## RESPONDING TO NEW INFORMATION FROM THE CORE LOCATION MANAGER

As you've learned, the location manager is delegate based. This means the location manager calls methods on its assigned delegate whenever new location, heading, or region information is available. These delegates are defined in the protocol `CLLocationManagerDelegate`.

**Table 4.3** outlines the delegate methods used in the standard location service, significant location change monitoring service, heading monitoring service, and the region monitoring service described in this chapter. By implementing these methods in the class used as the assigned delegate, you can update your UI or save relevant course information as needed by your app.



**TIP:** Notice the last delegate method in Table 4.3 is not actually related to the return of `CLLocation` objects from hardware but rather to changes in the authorization status of location services in your app. You should be continually aware of any changes in your app due to permissions with core location services. While the method is optional, it's best to implement it in case something changes while you're using location.

**TABLE 4.3** Core Location Manager Delegate Protocol Methods

| METHOD                                                             | DESCRIPTION                                                                                                                                                                                                                                                                                                          |
|--------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>locationManager:didUpdateToLocation:fromLocation:</code>     | Called by both the standard location service and significant location change service when new <code>CLLocation</code> objects are generated. Both of these services pass in the new <code>CLLocation</code> object ( <code>toLocation</code> ) as well as the previous location object ( <code>fromLocation</code> ) |
| <code>locationManager:didFailWithError:</code>                     | Called by the standard location service and the significant location change service when an error occurs. An error could be the result of conditions such as bad hardware or an interruption in service during a location call.                                                                                      |
| <code>locationManager:didUpdateHeading:</code>                     | Called by the heading monitoring service whenever a new heading is generated based on the heading filter threshold. The heading object passed to this delegate ( <code>CLHeading</code> ) contains relative directions to both true and magnetic north along with the x, y, and z components of that heading.        |
| <code>locationManager:didEnterRegion:</code>                       | Called by the location manager when a device crosses into a monitored region.                                                                                                                                                                                                                                        |
| <code>locationManager:didExitRegion:</code>                        | Called by the location manager when a device exits a monitored region.                                                                                                                                                                                                                                               |
| <code>locationManager:monitoringDidFailForRegion:withError:</code> | Called when region monitoring fails due to an error.                                                                                                                                                                                                                                                                 |
| <code>locationManager:didChangeAuthorizationStatus:</code>         | Called when the location permissions for this app are changed.                                                                                                                                                                                                                                                       |

**NOTE:** You can download a complete project that demonstrates all of the core location manager services demonstrated in this chapter by visiting [ioscoreframeworks.com/download#chapter-4](https://ioscoreframeworks.com/download#chapter-4). For more information on the hardware requirements and various capabilities of different iOS models, visit [developer.apple.com](https://developer.apple.com) or [ioscoreframeworks.com/reference#core-location](https://ioscoreframeworks.com/reference#core-location).



## FORWARD AND REVERSE GEOCODING

Geocoding is the process of going from a set of latitude and longitude coordinates to a human readable address and vice versa. Forward geocoding means you start with an address or location (such as Boston, MA) and end up with latitude and longitude coordinates. Reverse geocoding is the process of going from latitude and longitude coordinates back to a human-readable address.

Before iOS 5, developers only had access to reverse geocoding APIs available in Map Kit. With the introduction of iOS 5, however, the Map Kit APIs have been deprecated and Apple engineers added both forward *and* reverse geocoding to the Core Location framework. Not only does iOS 5 provide unique access to forward geocoding APIs, but there is no longer a dependency on Map Kit for these processes.

### GEOCODING BENEFITS

One of the major advantages of using the iOS 5 geocoding APIs is the fact that they are inherently locale based. For example, if my phone is set to Japanese as my native language and I'm visiting a friend in the United States, when I perform a geocoding operation to convert coordinates to a physical address, the result is returned in the native language of my phone (Japanese). This involves not only translating the language but also reformatting the order in which addresses are communicated.

Additionally, the forward geocoding APIs are form agnostic, meaning they really don't care what language or format an address is entered in. The geocoding APIs will automatically handle any known format based on the language settings of the device and handle the conversion as necessary.

As a developer working with the geocoding APIs, you don't have to do anything special to make your app work with different geocoding languages.

---

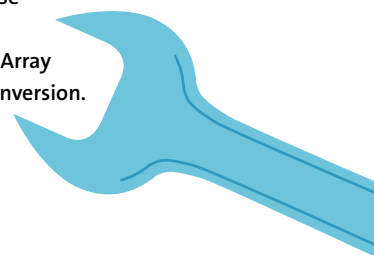
## GEOCODING DRAWBACKS

One of the biggest drawbacks to the geocoding API stems from one of its great advantages. All of the geocoding operations are handled in the cloud, meaning the conversions do not happen on the device. Now, this is undeniably an advantage because your device is not wasting precious power and resources to handle the conversion. Additionally, as new conversion information and techniques become more accurate, Apple can simply update their APIs in the cloud giving your app even better performance down the road. The drawback is your app must have an Internet connection to use the geocoding APIs. That means if your app is running in airplane mode or on a Wi-Fi-only device that's not connected to a Wi-Fi hotspot, you won't have access to geocoding services and should plan accordingly.

**NOTE:** Because geocoding operations are asynchronous, the callbacks of these services are handled using completion handler blocks. When the geocoding operation is complete, the geocoder will execute this block and pass in an `NSArray` of possible placemarks and an `NSError` object indicating the success of the conversion.

## FORWARD GEOCODING

Forward geocoding means you're starting with an address and are seeking coordinates. This can be used to create the coordinates of a region, as needed by the previous example on region monitoring, or to derive the coordinates of nearby locations based on address information (such as a check service or nearby restaurants). There are three ways to forward geocode. Two of these methods involve simple string conversion while the third supports an address dictionary.



## WORKING WITH STRINGS

The first, and most simple, geocoding operation converts a single string to an array of possible CLPlacemark objects.

```
1 // Geocode a simple string using a completion handler
2 [fgeo geocodeAddressString:@"Boston, MA"
   completionHandler:^(NSArray *placemarks, NSError *error){
3
4     // Make sure the geocoder did not produce an error
5     // before continuing
6     if(!error){
7
8         // Iterate through all of the placemarks returned
9         // and output them to the console
10        for(CLPlacemark *placemark in placemarks){
11            NSLog(@"%@",[placemark description]);
12        }
13    }
14    else{
15        // Our geocoder had an error, output a message
16        // to the console
17        NSLog(@"There was a forward geocoding error\n%@",
18            [error localizedDescription]);
19    }
20 }];
```

In this code block we convert the simple string, “Boston, MA”, to a CLPlacemark using forward geocoding. The returned array of CLPlacemarks contains all of the possible placemarks for the given address. Obviously, the more information you provide in the address string, the more reliable the returned placemarks will be. As mentioned before, one of the advantages of the geocoding APIs is they’re form independent. It’s not necessary that you add delimiters like commas or tabs between your address values.

**TIP:** The CLPlacemark object simply contains a CLLocation, CLRegion, and NSDictionary of available address component strings. For example, if you have an incomplete address (say you’re missing a ZIP code), you can convert the address using forward geocoding and pull the completed address from the CLPlacemark object.

**NOTE:** While an Internet location is required for geocoding operations, the forward geocoder is able to determine high level address information (for example, country origin) without an Internet connection based on local device information.

The second geocoding operation is similar, but allows for further optimization by limiting the conversion to a specified CLRegion. If you want to help iOS with the conversion process, you can define a CLRegion (if known) to limit the scope of search and improve result speed and accuracy. This method is handled just as before, except we define a CLRegion as an additional parameter, as seen in the following code block:

```
1  [fgeo geocodeAddressString:@"Boston, MA"
    inRegion:myRegion
    completionHandler:^(NSArray *placemarks, NSError *error){
2      //handle results
3  }
4  ];
```

## WORKING WITH ADDRESS DICTIONARIES

The third method used to forward geocode address information operates within the context of an address dictionary. Using the Address Book framework, you have full access to the contact cards and their relevant address information. When pulled from the address book, this information is returned as an `NSDictionary` object with various keys and values based on the information available.

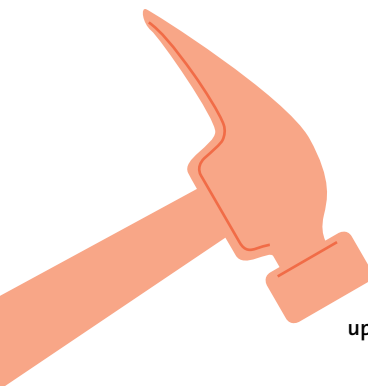
Using the `geocodeAddressDictionary` method on a geocoder object, you can easily convert this address book dictionary into a `CLPlacemark`. This is exactly the process Siri uses to convert address book information for labels like *home* or *the office* into region monitoring services using the location manager. The following code block demonstrates how to convert an address book dictionary using the `CLGeocoder` class. For a complete example on how to pull these address dictionaries from the Address Book using the `ABPeoplePicker`, visit [iOSCoreFrameworks.com/download#chapter-4](https://developer.apple.com/documentation/corelocation/ioscoreframeworks-com/download#chapter-4).

```
1  [fgeo geocodeAddressDictionary:myAddressDictionary
2      completionHandler:^(NSArray *placemarks, NSError *error){
3      //handle results
4      }
5  ];
```

## REVERSE GEOCODING

Reverse geocoding is the process of converting a `CLLocation` into a `CLPlacemark`. Remember that the `CLPlacemark` contains the `CLLocation`, `CLRegion`, and an `NSDictionary` for the address. So while both geocoding techniques create a `CLPlacemark`, the geocoding process `CLGeocoder` simply fills in the blanks.

The following example demonstrates how to convert a `CLLocation` into a `CLPlacemark` using reverse geocoding. Remember, because the monitoring services return `CLLocation` objects when a new update is performed, you can easily obtain an address for a user's location by starting the standard location service, obtaining their current location, and then reverse geocoding that location with the `CLGeocoder`.



**TIP:** Don't forget to turn off location updates when you're finished!

---

```

1  // Reverse Geocode a CLLocation to a CLPlacemark
2  [fgeo reverseGeocodeLocation:myLocationObject
    completionHandler:^(NSArray *placemarks, NSError *error){
3
4      // Make sure the geocoder did not produce an error
5      // before continuing
6      if(!error){
7
8          // Iterate through all of the placemarks returned
9          // and output them to the console
10         for(CLPacemark *placemark in placemarks){
11             NSLog(@"%@",[placemark description]);
12         }
13     }
14     else{
15         // Our geocoder had an error, output a message
16         // to the console
17         NSLog(@"There was a reverse geocoding error\n%@",
            [error localizedDescription]);
18     }
19 }
20 ];

```

## WORKING WITH MAP KIT

Now let's turn from working with data oriented location objects to maps. The Map Kit framework is rather extensive and provides the necessary views and controls for displaying map data. The primary view in the Map Kit framework is `MKMapView`, which is a subclass of `UIView`, and automatically renders Google Maps data based on the relative location of a visible map view rect.

**NOTE:** Map Kit allows you to add a variety of overlays and annotations (such as push pins and location indicators), all of which are incredibly useful for creating rich map data, but not directly relevant to our conversation about location. Because we don't have enough space in this book to go into the finer details of Map Kit, I've put together an online tutorial explaining the ins-and-outs of Map Kit overlays and annotations, available at [iOSScoreFrameworks.com/tutorial#map-kit](http://iOSScoreFrameworks.com/tutorial#map-kit).

### TRACKING LOCATION WITH MAP KIT

So you know that the `MKMapView` renders map data and provides the same gesture-based interaction seen in the native Maps application. You also know that you can use the standard location service to track a user's location. Fortunately, tracking a user's position is a common enough use case that both Map Kit and Core Location offer this capability. The benefit of Map Kit's tracking services is they will automatically track a user's location and indicate that location on the map using the famous Google Maps blue tracking ball seen in the native Maps app. As accuracy changes, the region circle around this ball will automatically adjust just as it does in the native app.

To enable tracking on an `MKMapView`, simply set the Boolean property `showsUsersLocation`. When set to YES, the `MKMapView` will first prompt the user with the same Core Location permission dialog. If authorization is approved, the `MKMapView` will automatically animate the changes in a user's location and the accuracy of the determination.

---

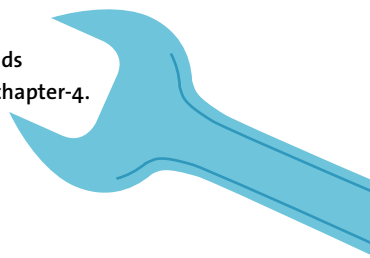
The `MKMapView` also manages a delegate property that it uses to communicate location update information through various methods. These methods are defined in the protocol `MKMapViewDelegate` and can be used to update necessary map information (such as reload overlays and annotations). The delegate method relevant to location updates is `mapView:didUpdateUserLocation:` which passes in an `MKUserLocation` object.

The `MKUserLocation` object is very handy. Unlike monitoring location with Core Location, the `MKMapView` can be configured to provide both heading and motion in a single delegate method based on the tracking mode defined by its `userTrackingMode` property. The possible values of this property are

- `MKUserTrackingModeNone`
- `MKUserTrackingModeFollow`
- `MKUserTrackingModeFollowWithHeading`

When the tracking mode is set to follow with heading, the `MKUserLocation` object will contain both a `CLLocation` object and a `CLHeading` object. If the tracking mode is set to just follow, the `MKUserLocation` object passed to the delegate will only contain the location.

**NOTE:** For a complete project example demonstrating the power of `MKMapView` and the Map Kit framework—along with other downloads available in this chapter—visit, [ioscoreframeworks.com/download#chapter-4](https://ioscoreframeworks.com/download#chapter-4).



## WRAPPING UP

Core Location and Map Kit are an incredibly powerful set of tools and APIs that give you full access to available location metadata. Using Core Location directly, through the location manager, you can monitor a user's location using standard services, significant change services, or region monitoring. Additionally, the core location manager allows direct access to heading information relative to either true north or magnetic north. Using Core Location you can determine where a person is and where they're going.

Beyond specific device location information, Core Location offers powerful (locale aware) address conversion APIs. These APIs let you forward and reverse geocode location information into a `CLPlacemark`. Placemarks contain a completed form of the address including a `CLLocation`, `CLRegion`, and `NSDictionary` of address values.

Finally, using Map Kit you can easily track a user's location by toggling a single Boolean property, `showsUserLocation`. Once enabled, the `MKMapView` will automatically animate and track the location while communicating that information back to the user on the map.

Don't forget, when working with location it's always important to check and monitor relevant permissions! It doesn't matter if you have the coolest location app in the world, there are users who will download your app and not enable location services. Be prepared for error conditions and blocked access.

*This page intentionally left blank*

## NUMBERS

- 3D (three dimensional) transforms
  - adding perspective, 211–212
  - in CALayer, 201
  - Core Animation, 209–210
- 3G (three-G) data connection, 62

## A

- ACAccountCredential
  - defined, 118
  - migrating accounts into
    - Accounts, 124
- ACAccounts
  - accessing, 121
  - defined, 118
- ACAccountStore
  - defined, 118–120
  - maintaining separation, 127
  - populating with available
    - accounts, 121
- ACAccountStoreDidChangeNotification, 126, 138
- ACAccountType
  - accessing accounts, 121–122
  - defined, 119
- ACAccountTypeIdentifierTwitter, 118–119
- access tokens
  - in Accounts framework, 121–122
  - migrating into Accounts, 122–124
  - OAuth services, 115–116
- accessGranted, 121
- Accounts
  - accessing accounts in, 121–122
  - defined, 18
  - getting started, 114–117
  - migrating users from existing apps
    - into, 122–124
  - new apps and, 118–120
  - overview, 113
  - special considerations, 125–127
  - using, 118
- accountsd process, 121
- accountsDidChange:, 126
- accuracy
  - constants, 85–86
  - for region monitoring, 100
- addArcWithCenter, 156–157
- addCurveToPoint, 156
- adding
  - iCloud entitlements, 58–60
  - new objects in Core Data, 41–42
  - new persistent stores, 38
- addLineToPoint, 156–157
- address dictionaries, 106
- airplane mode and geocoding, 103
- alpha, 195

- Ambient mode
  - AVAudioPlayer, 239
  - AVAudioSession, 234
- analyzing images
  - with CIDetector, 168
  - with Core Image, 182–184
  - overview, 166
- Angry Birds, 225
- animateWithDuration:animations:, 200
- animation. *see* Core Animation
- animation blocks, 199–200
- animation contexts, 197–198
- animationDidStop, 197
- animationWithKeyPath, 209
- Aperture, 164
- APNS (Apple Push Notification Service)
  - missed notifications, 296–297
  - overview, 291–298
- App ID, 57–58
- App Store, 275–276
- appearance of Newsstand app, 290
- Apple
  - audio session documentation, 237
  - Core Animation, 194
  - Core Image, 164
  - document storage, 78
  - Mac OS X. *see* Mac OS X
  - multicore processor, 11
  - Twitter and Accounts, 113
  - video streaming guidelines, 257
- Apple Push Notification Service (APNS)
  - missed notifications, 296–297
  - overview, 291–298
- Application Music Player, 229–230
- Application presents content in
  - Newsstand, 280
- applicationDidLaunch, 293
- applyingFilter, 190
- apps
  - Accounts and new, 118–120
  - challenges of audio, 224
  - document storage, 76–77
  - iCloud storage between platforms, 61
  - iCloud use case, 55
  - migrating users from existing into
    - Accounts, 122–124
  - Newsstand. *see* Newsstand Kit
  - Top Ten Technologies of iOS 5, 18–19
  - Twitter. *see* Twitter
  - understanding Newsstand, 282–283
  - using iOS frameworks, 6–7
  - using multicore processors, 11
  - why use Core Graphics?, 148
- ARC (Automatic Reference Counting), 6
- architecture of Core Data, 25
- arcWithCenter, 156–157
- assets
  - Assets Library, 247
  - defined, 244
  - downloading, 284–290

- ATOM feed, 281
- attributes. *see also* properties
  - defined, 26
  - filter, 173–176
- attributes dictionary
  - for complex filters, 176
  - key-value coding and, 168
  - for Sepia-tone filter, 175
- audio. *see* Core Audio
- audio capture. *see* capturing media
- audio sessions, 233–237
- Audio Toolbox, 226
- Audio Unit, 226
- audioPlayerBeginInterruption:, 240
- audioPlayerdecodeErrorDidOccur:error:, 240
- audioPlayerdidFinishPlaying:success
  - fully:, 240
- audioPlayerEndInterruption:, 240
- AudioSessionGetProperty(), 236
- authentication
  - in Accounts workflow, 116–117
  - migrating into Accounts, 122–124
  - performing TWRequest, 134–135
  - understanding OAuth services, 115–116
- authorization status, 88–90
- author's note, 299
- auto adjustment filters, 183
- automatic filter enhancement, 166
- Automatic Reference Counting (ARC), 6
- AV Foundation
  - Audio Session, 233–237
  - AVCaptureSession, 262
  - AVCaptureVideoPreviewLayer, 262–263
  - custom image capture, 263–269
  - defined, 19
  - getting started, 244–247
  - in-camera effects and video, 269–270
  - linking to project, 226
  - MPMoviePlayerController, 256–261
  - multimedia frameworks, 222–223
  - out-of-the-box solutions, 248
  - overview, 243
  - UIImagePickerController, 248–255
- availability
  - checking iCloud, 71, 78
  - checking Twitter, 131
  - determining location service, 88–90
  - filter, 173–174
  - Newsstand content, 293
- AVAsset, 244
- AVAudioPlayer, 232, 237–240
- AVAudioSession, 233–237
- AVCaptureConnection, 267–269
- AVCaptureOutputs, 269–270
- AVCaptureSession, 262–267
- AVCaptureVideoPreviewLayer, 262–263, 265–266
- AVPlayerLayer, 244

## B

### background processes

- AVAudioPlayer, 237–239
- AVAudioSession, 233–235, 237
- Newsstand app, 291
- Newsstand app properties, 280
- Newsstand apps waking up, 296–297
- rendering and filtering images, 184–186
- standard location services as, 93
- using expiration handler, 295–296

### backgroundColor

- custom animation, 204–205
- keyframe animation, 206–207
- UIKit animation, 195

### beginAnimation, 197–198

### beginGeneratingPlaybackNotification, 231

### beginInterruption, 236

### blocks

- in multicore processor, 14–16
- running filters in background thread, 184–186
- UIKit animations with, 199–200
- using in iOS frameworks, 16–17

### bounds, 195

### built-in filters, 173–174

### bundle ID

- defined, 54
- enabling iCloud, 57–58
- using wildcards in, 57

## C

### CAAnimation

- customization, 204–205
- defined, 203

### CABasicAnimation, 203

### Caches folder

- defined, 284
- downloading Newsstand content to, 285–286
- moving downloaded files to, 289

### CAEmitterCells, 213–217

### CAEmitterLayer, 215–217

### CAKeyframeAnimation

- along paths, 208
- color change animation, 206–207
- defined, 203

### CALayer

- AVPlayerLayer and, 244
- defined, 201–202

### Calendar, 148

### cameras

- adding to AVCaptureSession, 264–267
- capture mode, 255
- creating recorder with  
    UIImagePickerController,  
    251–254
- filters. *see* filters

### images. *see* images

### in-camera effects and video, 269–270

- UIImagePickerController source  
    types, 250

### canSendTweet, 131

### captureMode, 255

### captureStillImageAsynchronously

- FromConnection:, 267–269

### capturing media

- AVCaptureSession, 262
- AVCaptureVideoPreviewLayer, 262–263
- custom, 263–269
- in-camera effects and video, 269–270
- using UIImagePickerController,  
    250–254

### CATransform3D, 211–212

### CATransitionAnimation, 203

### cell emitters, 215–217

### cell towers

- location information, 84
- significant location change monitoring,  
    94

### center, 195

### CFBundleIcons, 280

### CGImageRef

- CImage and, 170–172
- rendering images, 181
- rendering in background thread,  
    185–186

### CGPDFGraphicsContext, 147

### change monitoring, 94–95

### change notifications

- Core Data persistent store, 74–75
- iCloud, 53
- observing account database, 126
- responding to iCloud, 67–70
- syncing, 70

### CIContext

- defined, 168–169
- rendering images, 178–181
- rendering in background thread,  
    185–186

### CIDetector

- defined, 168–169
- image analysis, 183–184

### CIFaceFeature, 168

### CIFeature, 168

### CIFilter, 168–169

### CImage

- defined, 168–169
- UIImage and, 170–172

### CIVector, 169

### classes

- defined, 8
- generating managed object, 27

### CLLocationManager

- heading monitoring, 96–98
- overview, 91
- region monitoring, 98–100

### responding to new information for,

100–101

### significant location change monitoring,

94–95

### standard location service, 91–93

- CLLocationManager.authorizationStatus,  
    88–90

- CLLocationManager.locationServicesEnabled,  
    88–90

### CLPlacemark

- forward geocoding, 104–105
- reverse geocoding, 106–107

### CMMotionManager, 16–17

### Cocoa Touch, 5

### coding. *see* Xcode

### coding, key-value. *see* key-value coding

### colors

- animating particle emitters, 215–216
- custom animation, 204–205
- keyframe animation, 206–207
- UIKit animation, 195

### commitAnimation, 197–198

### compass, 96

### completion block handlers

- capturing still image with, 269
- executing code on main thread from,  
    125–126
- GCD, 119–120, 125
- handling TWRequest response, 136–137
- tweet compose view controller, 131

### complex filters, 176

### concurrency

- in Core Data, 33
- in GCD, 13
- in multicore processor, 11–12
- when creating managed object  
    context, 39

### condition change notification, 69

### connectionDidFinishDownload

- handling finished downloads, 287–289
- updating app appearance, 290

### connectionDidFinishLoading, 134

### connections

- AVCaptureConnection, 267–269

- geocoding requirements, 103

### content

- downloading Newsstand app, 284–290
- loading into MPMoviePlayerController,  
    257–261
- Newsstand updated notifications, 293
- responding to remote notifications,  
    295–296

### contentStretch, 195

### context, animation, 197–198

### context, Core Image, 178–181

### context, graphics. *see* graphics context

### context, managed object. *see* managed     object context

### control events, 47

coordinate system of Core Graphics, 145–146

#### Core Animation

3D transforms, 209–210  
adding perspective, 211–212  
animating along paths, 207–208  
custom effects, 201–205  
defined, 19  
getting started, 194–196  
keyframe animations, 206–207  
overview, 193  
particle emitters, 213–217  
using UIKit, 197–200

#### Core Audio

AVAudioPlayer, 237–240  
AVAudioSession, 233–237  
defined, 19  
getting started, 224–226  
multimedia frameworks, 222–223  
overview, 221  
using audio from other sources, 232  
using iPod Music Library, 227–231

#### Core Data

adding new objects, 41–42  
concurrency in, 33  
defined, 18, 24–25  
deleting objects, 44–45  
fetching and modifying objects, 42–44  
getting started, 24  
managed object context, 30–32  
managed object model, 26–27  
overview, 23  
persistent store and store file, 28–30  
persistent store coordinator, 28  
setting up in Xcode, 34–40  
stack, 25  
syncing entitlements, 58–59  
syncing with iCloud, 69–75  
undo, redo, rollback and reset, 45–48

#### Core Data Model Editor

creating managed object model, 35–36  
defined, 26

#### Core Graphics

defined, 18  
getting started, 142–148  
linking to AV Foundation project, 247  
overview, 141  
paths, gradients, text, and images, 154–160  
understanding, 149–154

#### Core Image

analyzing images, 182–184  
Core Image context, 178–181  
defined, 19  
don't block main thread, 184–186  
editor, 187–190  
filters, 173–178

getting started, 164–167  
linking to AV Foundation project, 247  
overview, 163  
still images and video frames, 170–172  
understanding, 168–169

#### Core Image context, 178–181

##### Core Location

defined, 18  
forwarding and reverse geocoding, 102–107  
getting started, 82–90  
manager. *see* CLLocationManager  
overview, 81  
working with Map Kit, 108–109

##### Core Media, 247

##### Core OS, 5

##### Core Services

Core Location. *see* Core Location  
defined, 5  
GCD in, 14

cover updating, 290

CPU rendering, 178–179

##### customization

animation, 204–205  
animation effects, 201–205  
camera overlay, 251–254  
with Core Graphics, 148  
UIView, 149–151

CVPixelBuffer, 171

## D

data management. *see* Core Data

default iCloud settings, 54

defaultStore, 64

##### delegate methods

animation contexts, 197  
AVAudioPlayer, 240  
AVAudioSession, 233–234  
capturing video, 269–270  
CLLocationManager, 100–101  
downloading content, 285–287  
handling audio interruptions, 236  
Map Kit, 109  
media picker controller, 227–229  
NSURLConnectionDownloadDelegate, 287–289  
reconnecting abandoned asset  
downloads, 297–298  
registering for Newsstand update  
notifications, 293

deleting objects, 44–45

detecting audio sessions, 235–236

detecting faces. *see* face detection

device orientation, 268

dictionaries, address, 106

dictionaries, attribute. *see* attributes

dictionary

didEnterRegion, 98

didExitRegion, 98

didReceiveRemoteNotification, 294, 295

dispatch queues vs. operation queues, 13–14

document apps, 55

document storage

iCloud, 76–77

iCloud entitlements, 58–59

downloading Newsstand content

overview, 284–290

reconnecting abandoned, 297–298

downloadIssueWithInfo, 296

downloadWithDelegate, 285–287

drawing in Core Graphics, 155–160

drawRect:

Core Graphics image, 160

customizing UIView, 149–150

nesting save and restore calls, 152

overriding, 147

## E

EAGLContext, 180

EaseIn, 208

EaseInEaseOut, 208

EaseOut, 208

editing images. *see* Core Image

##### effects

cell emitter, 213–217

filter. *see* filters

Flipboard page-turn, 211–212

in-camera effects and video, 269–270

Elliott, Bryan, 164

emitterCells, 213–215

emitterMode, 213

emitterPosition, 213–214

emitterShape, 213

enabling iCloud

in provisioning portal, 57–58

quick starting, 54

before you begin, 56–57

enabling Newsstand app, 281

endInterruptionWithFlags:, 236

##### entities

creating managed object model, 36

defined, 26

managed object context, 30–32

##### entitlements

adding required to iCloud, 58–60

determining availability of iCloud, 71

exception, unknown key, 177

expiration handler, 295–296

explicit animations, 202

## F

- face detection
  - with CIDetector, 168
  - defined, 166
  - image analysis, 182–184
- favorite tracking, 65–67
- featuresInImage:options:, 182–183
- fetches properties
  - creating managed object model, 36
  - defined, 27
- fetching objects
  - in Core Data, 42–44
  - managed object context, 32
- file coordinator object, 76
- filters
  - AV Foundation, 270
  - CIFilter object, 168–169
  - in CIImage, 172
  - Core Image, 173–178
  - Core Image editor, 187–190
  - image analysis, 182–184
  - overview of Core Image, 164
  - running in background thread, 184–186
- filtersInCategory:, 187
- filtersNamesInCategory:, 174
- filterWithName:, 173–174
- fireworks effect, 213–217
- flash simulation, 267
- flip animation, 209–210
- Flipboard page-turn effect, 211–212
- fonts, 159
- forKey, 204–205
- format of APNS, 294
- forward geocoding, 102–106
- Foundation
  - AV. *see* AV Foundation
  - iOS crash course, 8
  - iOS prerequisites, 4
  - objects, 136
- frame, 195
- frameworks, 8

## G

- Garage Band, 148
- GCD (Grand Central Dispatch)
  - completion block handlers, 120
  - concurrency in Core Data, 33
  - defined, 13–14
  - running filters in background thread, 184–186
  - UIKit animations with blocks, 199–200
- geocodeAddressDictionary, 106
- geocoding, 102–107
- GET method, 133
- goals for book, 6–7
- Google Maps, 96

- Google Services, 82
- gotchas
  - Core Image editor, 187–190
  - Newsstand app, 296–298
- GPS, 84–85
- GPU rendering
  - vs. CPU rendering, 178–179
  - image, 181
- gradients, 158–159
- Grand Central Dispatch (GCD). *see* GCD (Grand Central Dispatch)
- graphics. *see* Core Graphics
- graphics context
  - defined, 147
  - properties, 150–151
  - stack, 151–154

## H

- handler block, 16–17
- hardware requirements
  - AVCaptureSession, 266
  - camera device capability, 254
  - for location-based apps, 84–85
- HD (high-definition) recording, 266
- heading monitoring
  - with CLLocationManager, 96–98
  - delegate methods, 101
- high-definition (HD) recording, 266
- HTTP Live Streaming, 257, 261

## I

- iCloud
  - adding required entitlements, 58–60
  - before you begin, 56
  - defined, 18
  - document storage, 76–77
  - enabling in iOS Provisioning Portal, 57–58
  - getting started, 52
  - how it works, 53–55
  - key-value storage, 63–69
  - overview, 51
  - special considerations, 61–62
  - storage containers, 60–61
  - syncing Core Data, 69–75
  - use case, 55
- iCloud Container Identifier, 71
- iCloud Daemon, 67
- Icon Files, 280
- icons, Newsstand app
  - aspect ratio, 276
  - styles, 280
  - updating appearance, 290
- identifiers
  - account, 121

- maintaining account store
  - separation, 127
  - team prefix. *see* team prefix identifiers
- ubiquity container, 54, 71
- image picker controller, 248–255
- images
  - in AV Foundation. *see* AV Foundation
  - in Core Graphics, 160
  - in Core Image. *see* Core Image
  - custom capture, 263–269
  - still images and video frames, 170–172
  - uploading with tweet compose view controller, 130
- implicit animations, 202
- importing
  - Accounts and Twitter to project, 114
  - audio frameworks, 226
  - Core Graphics, 143
  - Core Image, 167
  - Core Location and Map Kit, 83
  - framework into header files, 10
  - Newsstand Kit, 278
  - Quartz Core, 196
- in-camera effects, 269–270
- info property list, 279–280
- inputImage, 175
- inputIntensity, 175–177
- instance variables
  - preventing filter lag, 188–190
  - setting CIContext as, 180, 186
- interruptions, handling audio, 236, 240
- iOS frameworks
  - crash course, 8–10
  - enabling iCloud in Provisioning Portal, 57–58
  - goals for book, 6–7
  - overview, 3
  - prerequisites, 4–6
  - top ten technologies, 18–19
  - understanding impact of multicore processor, 11–17
- iOS Provisioning Portal, 54, 57–58
- iPad
  - multicore processor, 11
  - resolution, 145
- iPhone
  - multicore processor, 11
  - resolution, 145
- iPod Music Library, 227–231
- iPod Music Player, 229–230
- issues, Newsstand Kit
  - defined, 282–283
  - downloading content, 285–286
  - handling finished download and progress, 288–289
- issueWithName, 44–45
- iTunes Connect, 281
- iWork applications, 55

## J

Java, 12  
Jobs, Steve, 51

## K

Key Value Observing (KVO), 267, 269  
keyframe animations  
    CAKeyframeAnimation, 203  
    Core Animation, 206–208  
keys  
    location-based app, 85  
    Newsstand app, 279–280  
key-value coding  
    attributes dictionary and, 177  
    custom animation, 205  
    defined, 168  
    format of APNS, 294  
key-value storage  
    with iCloud, 63–69  
    iCloud entitlements, 58–59  
    iCloud use case, 55  
    quick starting iCloud, 54  
    responding to change notifications,  
        67–69  
KVO (Key Value Observing), 267, 269

## L

launch site, 213  
launchOptions, 296–297  
layers  
    graphics context, 151  
    iOS, 5  
    preview, 262–263  
layers, Core Animation  
    3D transforms, 209–210  
    assigning animations to, 204–205  
    CAEmitterLayer, 215–217  
    CALayer, 201–202  
LibDispatch, 178  
libraries, photo  
    capturing still image, 267–269  
    selecting photos from, 249–250  
library, Newsstand Kit, 282–283, 297–298  
linear gradients, 158–159  
Liner, 208  
linking  
    Accounts and Twitter to project, 114  
    audio frameworks, 226  
    AV Foundation to project, 247  
    Core Data framework, 35  
    Core Graphics to project, 143  
    Core Image, 167  
    Core Location and Map Kit, 83  
    new frameworks in Xcode project,  
        9–10

    Newsstand Kit to project, 278  
    Quartz Core to project, 196  
Live Streaming, 257, 261  
loading content into  
    MPMoviePlayerController, 257–261  
location manager. *see* CLLocationManager  
location-services, 85

## M

Mac OS X  
    Core Image, 163  
    enabling iCloud, 56  
    iCloud storage between platforms, 61  
    native frameworks in iOS, 4  
main dispatch queue  
    creating managed object context, 39  
    in GCD, 13–14  
managed object context  
    in Core Data, 30–32  
    creating new, 39–40  
    deleting objects in, 44–45  
    fetching and modifying objects, 42–44  
    tips and tricks, 48  
    undo, redo, rollback and reset, 47  
managed object model  
    Core Data, 26–27  
    creating in Xcode, 35–36  
Map Kit  
    Core Location and, 91  
    forward and reverse geocoding, 102  
    working with, 108–109  
maximum duration of video, 255  
Media  
    AV Foundation, 243  
    Core Animation, 193  
    Core Audio, 221  
    Core Graphics, 141  
    Core Image, 163  
    defined, 5  
media capture. *see* capturing media  
media picker controller, 227–229  
Media Player  
    linking to project, 226, 247  
    MPMediaPickerController, 227–229  
    MPMoviePlayerController, 245, 256–261  
    MPMoviePlayerViewController, 261  
    MPMusicPlayerController, 229–230  
    multimedia frameworks, 222–223  
medialibraryButtonPressed:, 227–229  
mergeChangesFromContextDidSave  
    → Notification:, 70  
message composition, 130  
methods  
    Core Graphics image, 160  
    creating CGContext, 179–180  
    for creating UIImage, 171  
    delegate. *see* delegate methods  
text, 159

    UIBezierPath, 156  
migrating users in Accounts, 122–124  
mirroring data with iCloud, 53, 55  
MKMapView, 108–109  
Mobile Core Services, 247  
mobile platforms, 3  
model layer of Core Animation, 202  
Model-View-Controller (MVC) design  
    paradigm, 24–25  
modifying objects in Core Data, 42–44  
monitoring  
    heading, 96–98  
    region, 98–100  
    significant location change, 94–95  
motion manager, 16–17  
moveToPoint, 156–157  
movie player controller, 256–261  
moviePlayer, 261  
MPMediaPickerController, 227–229  
MPMoviePlayerController, 245, 256–261  
MPMoviePlayerViewController,  
    256–257, 261  
MPMusicPlayerController, 229–230  
multicore processors, 11–17  
multimedia frameworks  
    AV Foundation, 242–246  
    Core Audio, 222–223  
multithreading  
    in multicore processor, 12  
    UIKit animations with blocks, 199–200  
Music Library, 227–231  
music player controller, 229–230  
MVC (Model-View-Controller) design  
    paradigm, 24–25

## N

naming  
    downloads, 288–289  
    Newsstand issues, 283  
    persistent stores, 72–73  
newCover, 290  
Newsstand Content Availability, 291–292  
Newsstand Kit  
    defined, 19  
    downloading content, 284–290  
    getting started, 276–281  
    notifications, 291–296  
    overview, 275  
    special considerations, 296–298  
    understanding apps, 282–283  
NKAssetDownload  
    defined, 282  
    downloading content, 285–287  
    handling finished download and  
        progress, 287–289  
    reconnecting abandoned, 297–298  
NKDontThrottleNewsstandContent  
    → Notifications, 293

- NKIssue
  - defined, 282–283
  - downloading content, 285–286
  - handling finished download and progress, 288–289
- NKLibrary
  - defined, 282–283
  - reconnecting abandoned asset downloads, 297–298
- notifications
  - change. *see* change notifications
  - missed APNS, 296–297
  - music player, 230–231
  - Newsstand Kit, 291–296
- Now Playing
  - challenges of mobile device audio, 223
  - using iPod Music Library, 227
- NSBinaryStorageType, 30
- NSConfinementConcurrencyType, 33
- NSData, 171
- NSInMemoryStoreType, 30
- NSMainQueueConcurrencyType, 33
- NSManagedObject
  - managed object context, 30
  - objectID, 40
- NSNotificationCenter, 53
- NSNumber, 69
- NSObject, 5
- NSPersistentStoreDidImportUbiquitous
  - ContentChangesNotification, 74
- NSPredicate, 43–44
- NSPrivateQueueConcurrencyType, 33
- NSSQLiteStoreType, 30
- NSThread, 184
- NSUbiquitousKeyValueStore
  - adding and removing objects, 65–67
  - quick starting, 54
  - using, 64
- NSUbiquitousKeyValueStoreDidChange
  - ExternallyNotification, 67–68
- NSUndoManager, 45–47
- NSUnknownKeyException, 177
- NSURLConnection
  - creating CImage with, 171
  - download delegate, 287–289
  - handling TWRequest response, 136–137
  - performing TWRequest, 134–135
- NSUserDefaults
  - adding and removing objects, 65–67
  - key-value storage and, 63

## O

- OAuth services
  - in Accounts workflow, 116–117
  - migrating into Accounts, 122–124
  - Twitter API, 132
  - understanding, 115–116

- objectForID
  - defined, 40
  - tips and tricks, 48
- objectForKey:, 175
- objectID
  - defined, 40
  - tips and tricks, 48
- objects
  - adding new in Core Data, 41–42
  - CAAnimation, 203
  - CLLocation, 82
  - CLPlacemarks, 104–105
  - in Core Data, 25
  - Core Image, 168
  - creating TWRequest, 132–134
  - deleting in Core Data, 44–45
  - fetching and modifying in Core Data, 42–44
  - location, 91
  - in managed object model, 26–27
  - NKLibrary, 282–283
  - storing as keys, 63
  - UIBezierPath, 155–157
- observers, 267
- online materials
  - for Core Data, 49
  - iOS, 7
- OpenAL, 226
- OpenGL ES 2.0, 178
- operation queues, 13–14
- OS X
  - Core Image, 163
  - enabling iCloud, 56
  - iCloud storage between platforms, 61
  - native frameworks in iOS, 4
- outputs of AVCaptureSession, 269–270

## P

- particle emitter animation, 213–217
- paths
  - animating along, 207–208
  - in Core Graphics, 155–157
- performance
  - CPU vs. GPU rendering, 178–179
  - GCD and, 14
  - of implicit vs. explicit animation, 202
  - location data source, 83–84
  - multicore processor, 11
  - storage types, 58
- performBlock
  - change notifications, 74–75
  - managed object context, 33
- performBlockAndWait
  - creating managed object context, 39
  - defined, 33
- performRequestWithHandler
  - handling TWRequest response, 136–137
  - performing TWRequest, 134–135
- performSelectorOnMainThread, 125–126
- permissions
  - Accounts workflow, 117
  - location, 86–88
  - OAuth services, 115
- persistent store coordinator
  - Core Data, 28
  - creating in Xcode, 37
- persistent stores
  - adding new in Core Data, 38
  - store file and, 28–30
  - syncing with iCloud, 70–75
- persistentStoreDidChange:, 74–75
- perspective, 211–212
- photo editing. *see* Core Image
- photo library
  - capturing still image, 267–269
  - selecting photos from, 249–250
- pixels
  - modifying with filters, 173–178
  - vs. points, 145–146
- placemarks
  - forward geocoding, 104–105
  - reverse geocoding, 106–107
- platforms, iCloud storage on, 61
- Play and Record mode, 234
- playback. *see* AV Foundation; Core Audio
- playback, 240
- Playback mode, 234
- player layer, 244
- points coordinate display, 145–146
- polar coordinate system, 215
- preferences, 62
- prepareToPlay, 238–239
- presentation layer of Core Animation, 202
- preview layer
  - adding to AVCaptureSession, 265–266
  - AVCaptureVideoPreviewLayer, 262–263
- properties
  - accessGranted, 121
  - account-type, 114
  - animating, 203
  - AVAudioSession, 237
  - CALayer, 201
  - CImageRef, 170
  - creating managed object model, 36
  - custom animation, 204–205
  - graphics context, 150–151
  - key-value coding, 168
  - moviePlayer, 261
  - Newsstand app info list, 279–280
  - NKIssue, 283
  - objectID, 40
  - observers, 267
  - particle emitter, 213–217
  - showsUserLocation, 108
  - UIImagePickerController video, 255
  - UIKit animations, 197–198
  - UIView animatable, 195
  - userTrackingMode, 109

protocols, Foundation, 5  
public timeline, 133  
publications. *see* Newsstand Kit  
push notifications, 291–298

## Q

quality of video, 255  
Quartz 2D, 144  
Quartz Core  
    Core Animation, 193  
    Core Graphics and, 144  
    linking in Xcode project, 9–10  
    linking to AV Foundation project, 247  
    linking to project, 196  
queues  
    managed object context, 48  
    operation vs. dispatch, 13–14  
    when creating managed object  
        context, 39

## R

radial gradients, 158–159  
reason keys, 68–69  
reconnecting abandoned asset downloads,  
    297–298  
Record mode, 234  
recording HD, 266  
recording undo events, 47  
rectangle parameter for image  
    rendering, 181  
redo, 45–48  
region monitoring  
    CLLocationManager, 98–100  
    geocoding, 105  
registerForRemoteNotificationTypes:,  
    293  
registering for APNS  
    Newsstand update notifications, 293  
    overview, 292  
registering iOS apps, 56  
relationships  
    creating managed object model, 36  
    defined, 26  
    key-value coding, 168  
remote notifications  
    iOS Simulator and, 292  
    missed notifications, 296–297  
    registering for Newsstand update  
        notifications, 293  
    responding to, 295–296  
removeObjectForKey:, 65–67  
rendering images  
    CIImage object, 168  
    with Core Image context, 178–181

    running in background thread,  
        184–186  
requests  
    creating TWRequest, 132–134  
    handling TWRequest, 136–137  
    performing in Twitter, 134–135  
Required background modes, 280  
requirements  
    downloading content, 287–289  
    hardware. *see* hardware requirements  
    iCloud entitlements, 58–60  
reset, 45–48  
resolution, 145–146  
response to TWRequest, 136–137  
restoring graphics context, 152–154  
retina display, 145–146  
reverse geocoding, 102–103, 106–107  
rocket animation, 213–217  
rollback, 45–48  
root classes, 5  
rotationMode, 208

## S

Saved Photos album, 250  
saving  
    in Core Data, 48  
    graphics context, 152–154  
SDK (Software Development Kit), 9  
selecting photos from photo library,  
    249–250  
Sepia-tone filter  
    gotcha, 188  
    overview, 174–177  
serial queues, 13  
sessions, audio, 233–237  
setAnimationDidStopSelector, 206  
setContentURL, 257  
setNeedsDisplay, 150  
setNewsstandIconImage, 290  
setObject:forKey:, 65–66  
Settings app, 116–117  
setValue:forKey:, 175, 177  
shake events, 47  
showPhotoLibrary, 249  
showsUserLocation, 108, 110  
significant location change monitoring  
    defined, 94–95  
    delegate methods, 101  
sizing fonts, 159  
sliders  
    Core Image editor, 187–188  
    working with filters and, 176  
Software Development Kit (SDK), 9  
Solo Ambient mode  
    AVAudioPlayer, 239  
    AVAudioSession, 234

sound. *see* Core Audio  
soundURL, 239  
source types for UIImagePickerController,  
    249–250  
standard location service  
    defined, 91–93  
    delegate methods, 101  
startMonitoringForRegion:desired  
    → Accuracy, 100  
startMonitoringSignificantLocation  
    → Changes, 95  
startUpdatingHeading, 96–97  
startUpdatingLocation, 91–93  
startVideoCapture, 255  
state notifications, 230–231. *see also*  
    notifications  
still images. *see also* images  
    vs. video capture, 269–270  
    video frames and, 170–172  
Stocks, 148  
stopVideoCapture, 255  
storage. *see* key-value storage  
    account, 118–120  
    containers, 60–61, 71  
    document, 76–77  
    issue, 284  
    persistent store and store file, 28–30  
streaming video guidelines, 257  
strings  
    attributes dictionary and, 177  
    forward geocoding, 104–105  
    key-value coding, 168  
sublayer hierarchy of Core Animation, 201  
synchronization  
    Core Data with iCloud, 69–75  
    with key-value storage, 64  
    responding to notifications, 68  
    setting up iCloud persistent store, 72–73  
system accounts. *see* Accounts

## T

team prefix identifiers  
    defined, 54  
    determining availability of iCloud, 71  
    enabling iCloud, 57–58  
    iCloud storage between platforms, 61  
technologies in iOS 5, 18–19  
terms of service, Map Kit, 82  
testing iCloud, 78  
text, 159  
three dimensional (3D) transforms  
    adding perspective, 211–212  
    in CALayer, 201  
    Core Animation, 209–210  
three-G (3G) data connection, 62  
Tiny Wings, 225

- tokens, access. *see* access tokens
- Top Ten Technologies of iOS 5, 18–19
- touch events, 47
- tracking
  - favorites, 65–67
  - issues, 282–283
  - location, 108–109
- transform, 195
- transforms. *see* 3D (three dimensional) transforms
- transition animation, 203
- Twitter
  - Accounts. *see* Accounts
  - framework in iOS, 18
  - getting started, 115–116
  - interacting with API, 132–137
  - overview, 113
  - tweet compose view controller, 128–131
- TWRequest
  - creating, 132–134
  - performing, 134–135
- TWTweetComposeViewController, 128–131

## U

- ubiquitous key-value store. *see* key-value storage
- ubiquity container
  - document storage and, 76–77
  - quick starting iCloud, 54
- ubiquity container identifier
  - defined, 54
  - team prefix identifier and, 71, 78
- UIBackgroundModes, 280
- UIBezierPath, 155–157
- UIEvent, 142
- UIGraphicsGetCurrentContext(), 147
- UIImage, 170–172
- UIImagePickerController, 245, 248–255
- UIImageView, 181
- UIKit
  - Core Animation using, 197–200
  - Core Graphics and, 142
  - getting started with Core Animation, 194–195
  - iOS crash course, 8
  - iOS prerequisites, 4
  - linking to project, 247
  - UIImage and UIImage, 170
  - UIImagePickerController, 248–255
- UINewsstandApp, 280
- UIRemoteNotificationTypeNewsstand
  - ContentAvailability, 293
- UITweetComposeViewController, 128

- UIView
  - animatable properties, 195
  - animation with UIKit, 197–200
  - custom camera overlay, 251–254
  - customization, 149–151
- undo, 45–48
- UNIX, 12
- unknown authorization status, 90
- updating
  - heading, 96–97
  - location, 91–93
  - Newsstand appearance, 290
  - notifications, 293
- URLs
  - creating AVAudioPlayer, 238
  - creating UIImage, 171
  - creating TWRequest, 132–134
  - downloading issue content, 289
  - loading content into
    - MPMoviePlayerController, 257
  - syncing persistent stores, 72–73
- user accounts. *see* Accounts
- user defaults
  - adding and removing objects, 65–67
  - key-value storage and, 63
- users
  - events, 47
  - migrating into Accounts, 122–124
- userTrackingMode, 109

## V

- values. *see also* properties
  - in CALayer, 201
  - complex filter, 175–176
  - in Core Data, 25
  - fetches properties and, 27
  - keyframe animation, 207
  - for location authorization, 90
  - obtaining image analysis, 182–183
  - perspective, 211–212
  - setting Sepia-tone filter, 177
  - userTrackingMode, 109
- video
  - capture. *see* capturing media
  - in-camera effects and, 269–270
  - loading into MPMoviePlayerController, 257–261
  - still images and frames, 170–172
  - working with UIImagePickerController, 254–255
- videoMaximumDuration, 255
- videoQuality, 255
- viewDidLoad
  - animation contexts, 197–198

- creating MPMoviePlayerController, 260
- Twitter availability, 131
- views
  - CALayer, 201
  - custom camera overlay, 251–254
  - customizing UIView, 149–151
  - Map Kit, 108–109
  - MPMoviePlayerViewController, 256–257, 261
  - tweet compose view controller, 128–131
  - UIImagePickerController, 250
  - UIView. *see* UIView

## W

- Weather
  - 3D transforms, 209
  - Core Graphics, 148
- web services authorization, 115
- whitelist, 284
- Wi-Fi
  - geocoding access, 103
  - location information, 84
  - significant location change monitoring, 94
- wildcards
  - in bundle ID, 57
  - iCloud storage between platforms, 61
- workflows
  - Accounts, 116–117
  - image processing, 169
  - OAuth, 115

## X

- Xcode
  - ARC, 6
  - info property list, 279–280
  - iOS crash course, 8–10
  - linking Accounts and Twitter to project, 114
  - linking audio frameworks to project, 226
  - linking AV Foundation to project, 247
  - linking Core Graphics to project, 143
  - linking Core Image to project, 167
  - linking Newsstand Kit to project, 278
  - linking Quartz Core to project, 196
  - setting up Core Data in, 34–40

## Z

- Z value, 201