Learning iOS Development
This page intentionally left blank
Learning iOS Development

A Hands-on Guide to the Fundamentals of iOS Programming

Maurice Sharp
Erica Sadun
Rod Strougo

Addison-Wesley
To my wife, Lois, and my daughter, Karli. They gave me the time I needed to work on the book, even though it effectively meant a second job on top of my day one. You did it with love and compassion and still had energy for when I could be there.

Maurice
Contents at a Glance

Foreword    xvi
Preface     xx

Chapter 1   Hello, iOS SDK      1
Chapter 2   Objective-C Boot Camp   21
Chapter 3   Introducing Storyboards   65
Chapter 4   Auto Layout     117
Chapter 5   Localization  183
Chapter 6   Scrolling      225
Chapter 7   Navigation Controllers I: Hierarchies and Tabs   253
Chapter 8   Table Views I: The Basics  275
Chapter 9   Introducing Core Data  317
Chapter 10  Table Views II: Advanced Topics   341
Chapter 11  Navigation Controllers II: Split View and the iPad  371
Chapter 12  Touch Basics  427
Chapter 13  Introducing Blocks   453
# Table of Contents

Foreword  xvi  
Preface xx  

1 Hello, iOS SDK  1  
   Installing Xcode  1  
   About the iOS SDK  2  
      What You Get for Free  3  
      iOS Developer Program (Individual and Company)  4  
      Developer Enterprise Program  4  
      Developer University Program  5  
      Registering  5  
      iTunes U and Online Courses  5  
      The iOS SDK Tools  6  
   Testing Apps: The Simulator and Devices  7  
      Simulator Limitations  8  
      Tethering  10  
      iOS Device Considerations  11  
   Understanding Model Differences  15  
      Screen Size  15  
      Camera  16  
      Audio  16  
      Telephony  16  
      Core Location and Core Motion Differences  17  
      Vibration Support and Proximity  17  
      Processor Speeds  17  
      OpenGL ES  18  
      iOS  18  
   Summary  19  

2 Objective-C Boot Camp  21  
   Building Hello World the Template Way  21  
   Creating the Hello World Project  21  
   A Quick Tour of the Xcode Project Interface  25  
   Adding the Hello World Label  28
### 4 Auto Layout  117

- Auto Layout Basics  117
- Constraints  120
- Perfecting Portrait  131
  - Thinking in Constraints  132
  - What Makes a Complete Specification  133
- Adding/Viewing Cars: Designing and Implementing the Constraints  134
- Edit Car: An Initial Look  155
- Adding Landscape  156
  - Adding and Viewing Cars: Designing the Landscape Constraints  158

### 5 Localization  183

- Localization Basics  183
- Redirection  184
- Formats  187
- Preparing the App for Localization  189
  - Setting Up Localization for the Add/View Car Scene  191
- German Internationalization  203
  - Adding the German Locale  203
  - Changing the Device Language  206
  - Updating the German Localizable.strings  207
- Changing Label Constraints  209
- Formatting and Reading Numbers  213
- Right-to-Left: Arabic Internationalization  215
  - Adding Arabic Strings  215
  - Making Dates and Numbers Work  219
- Text Alignment  222

### 6 Scrolling  225

- Scrolling Basics  225
- Bounce Scrolling  227
  - Adding a Scroll View to the View/Edit Scene  227
Handling the Keyboard 230
  Adding the Scroll View 231
  Resizing for the Keyboard 234
  Adding Resizing 239
Scrolling Through Content 240
  Populating the Scroll View 241
  Adding Paging 243
  Adding Zoom 245
  Rotation 248
  What Car Is This? 249
Summary 250
Challenges 251

7 Navigation Controllers I: Hierarchies and Tabs 253
  Navigation Controller 254
    Navigation Controller Classes 256
    Message-Based Navigation 263
    A Bit of Color 264
  Tab Bar Controller 267
    How the Tab Bar Works 268
    CarValet: Adding a Tab Bar 270
    Car Valet: Moving Info 272
Summary 273
Challenges 274

8 Table Views I: The Basics 275
  Introduction to Table Views 275
    Project TableTry 277
  Phase I: Replacing the Add/View Scene 283
    Adding a Car View Cell 285
    Adding New Cars 287
    Removing Cars 288
  Phase II: Adding an Edit Screen Hierarchy 291
    Adding a View Car Scene 292
    Populating the View Car Scene with Data 294
    Editing Data 296
    Editing the Year 307
Summary 314
Challenges 315
9 Introducing Core Data  317
   Introduction to Core Data  318
   Moving CarValet to Core Data  320
       Adding the CDCar Model  321
       Adding Core Data Boilerplate Code  324
       Converting CarTableViewController  326
   Easier Tables: NSFetchedResultsController  332
       Part 1: Integrating NSFetchedResultsController  333
       Part 2: Implementing NSFetchedResultsControllerDelegate  336
   Summary  339
   Challenges  340

10 Table Views II: Advanced Topics  341
   Custom Table View Cells  341
       Adding the Custom Cell Visual Elements  343
   Sections and Sorting  345
       Section Headers  346
       Enabling Changing of Section Groups  349
   Adding an Index  355
       Showing the Year in an Index  357
   Searching Tables  358
       Adding Searching  361
   Summary  369
   Challenges  370

11 Navigation Controllers II: Split View and the iPad  371
   Split View Controller  372
   Adding a Split View Controller  374
       Adding the Split View Controller  376
       Adding App Section Navigation  379
   Adding About  382
       Creating MainMenuViewController  383
       Polishing Menu Images  385
   Accessing the Menu in Portrait  387
       Implementing the DetailController Singleton  388
Replacing a Protocol 462
   Step 1: Changing ViewCarTableView-Controller 463
   Step 2: Updating CarTableViewCellController 464
   Step 3: Modifying CarDetailViewController 465
   Step 4: Updating MainMenuViewController 466

Summary 466
Challenges 467

14 Instruments and Debugging 469
   Instruments 469
   Templates and Instruments 471
   An Example Using the Time Profiler 472
   A Last Word on Instruments 478
   The Debugger 479
   Debug Gauges: Mini “Instruments” 481
   Breakpoints, and Actions, and Code...Oh My! 483
   Bug Hunt: Instruments and the Debugger 486
   Starting with Zombies 486
   Moving On to the Debugger 489

Summary 491
Challenges 491

15 Deploying Applications 493
   Certificates, Profiles, and Apps 493
   Generating a Development Certificate and Profile 495
   App ID and Provisioning 498
   Prelaunch 506
   Bug Reporting 506
   Metrics 508
   Quality Assurance Testing 509
   Marketing 512
   Uploading and Launching 513
   App Details 515
   Uploading to the App Store 521
   Some Things to Watch Postlaunch 526
Foreword

It’s been an amazing five years since the first edition of the *iPhone Developer’s Cookbook* debuted for the new Apple iPhone SDK. Since then, new APIs and new hardware have made the task of keeping on top of iOS development better suited for a team than for an individual. By the time the iOS 5 edition of the *Cookbook* rolled around, the book was larger than a small baby elephant. We had to publish half of it in electronic form only. It was time for a change.

This year, my publishing team sensibly split the *Cookbook* material into several manageable print volumes. This volume is *Learning iOS Development: A Hands-on Guide to the Fundamentals of iOS Programming*. My coauthors, Maurice Sharp and Rod Strougo, moved much of the tutorial material that used to comprise the first several chapters of the *Cookbook* into its own volume and expanded that material into in-depth tutorials suitable for new iOS developers.

In this book, you’ll find all the fundamental how-to you need to learn iOS development from the ground up. From Objective-C to Xcode, debugging to deployment, *Learning iOS Development* teaches you how to get started with Apple’s development tool suite.

There are two other volumes in this series as well:

- The *Core iOS Developer’s Cookbook* provides solutions for the heart of day-to-day development. It covers all the classes you need for creating iOS applications using standard APIs and interface elements. It offers the recipes you need for working with graphics, touches, and views to create mobile applications.

- The *Advanced iOS 6 Developer’s Cookbook* focuses on common frameworks such as Store Kit, Game Kit, and Core Location. It helps you build applications that leverage these special-purpose libraries and move beyond the basics. This volume is for those who have a strong grasp of iOS development and are looking for practical how-to information for specialized areas.

It’s been a pleasure to work with Maurice and Rod on *Learning iOS Development*. They are technically sharp, experienced developers, and they’re genuinely nice guys. It’s difficult to hand over your tech baby to be cared for by someone else, and these two have put a lot of effort into turning the dream of *Learning iOS Development* into reality. Maurice, who wrote the bulk of this volume, brings a depth of personal experience and an Apple background to the table.

iOS has evolved hugely since the early days of iPhone, both in terms of APIs and developer tools. *Learning iOS Development* is for anyone new to the platform, offering a practical, well-explored path for picking up vital skills. From your first meeting with Objective-C to App Store deployment, *Learning iOS Development* covers the basics.

Welcome to iOS development. It’s an amazing and exciting place to be.

—Erica Sadun, April 2013
Acknowledgments

What do acknowledgments have to do with learning iOS development? I used to be likely to skim or skip this section of a book—and you might be tempted to do that as well. Who are these people? Why do I care? You care because your ability to get things done really depends on who you know. And I am about to thank people who have helped me, many of whom enjoy helping. You may know some of them or know someone who does. I am often surprised how close I am to the truly great people on LinkedIn. So read on, note the names, and see how close you are to someone who may be able to help you solve your most pressing problem.

First, my deep thanks to Erica Sadun (series editor and code goddess) and Trina MacDonald (editor) for the opportunity to write most of this book. When they asked me to contribute, my first thought was “I have never written anything this big, but how hard could it be?” I found out, and their support, along with that of Rod Strougo, Chris Zahn (please correct my grammar some more), Jovana Shirley (so that is production editing), Kitty Wilson (are you sure you do not know how to code?), Anne Goebel (may I use might, or might I use may?), both Olivia Basegio and Betsy Gratner (if only I were that organized), and the entire production staff (I fed them sketches; they produced the beautiful diagrams). All of you started my journey of learning to be an author. I have always been a helper. Developer Technical Support enabled me to help thousands. This book is an opportunity to help a wider audience. Thank you, all.

I am also deeply grateful to friends old and new for answering technical questions: Mike Engber, a superstar coder at Apple, showed me the light on blocks as well as answering other questions. Others took time to answer questions or talk about possible solutions: Thanks to Tim Burks, Lucien Dupont, Aleksey Novicov, Jeremy Olson (@jerols, inspired UX!), Tim Roadley (Mr. Core Data), and Robert Shoemate (Telerik/TestStudio). Thanks also to Marc Antonio and Mark H. Granoff for reviewing every chapter and giving suggestions and corrections on things technical. And an extra shout out to Gemma Barlow and Scott Gruby for checking the iOS 7/ Xcode 5 changes in addition to all the other feedback.

Contributors are not limited to engineers. German translations are from Oliver Drobnik and David Fedor, a longtime friend. Arabic is from Jane Ann Day...take her course; she is very good. Glyphish, aka Joseph Wain, provided the beautiful icons and user interface (UI) element graphics. Get some great icons for your app at www.glyphish.com. The 11 car photos first used in Chapter 6, “Scrolling,” are courtesy of Sunipix.com.

Thanks to those at Couchsurfing (www.couchsurfing.org) for giving me time to work on this book, including our CEO Tony Espinoza, my friend Andrew Geweke, and the whole mobile design and development team: Gemma Barlow, David Berrios, Evan Lange, Hass Lunsford, Nicolas Milliard, Nathaniel Wolf, and Alex Woolf. You are a joy to serve.

Thanks to those who have taught, inspired, and challenged me along my technical journey. Listing them all would take a whole book, but some include the faculty and fellow students at the University of Calgary computer science department, as well as Dan Freedman, Scott Golubock, Bruce Thompson, Jim Spohrer, Bob Ebert, Steve Lemke, Brian Criscuolo, and many more from Apple, Palm, eBay, Intuit, Mark/Space, ShopWell, and Couchsurfing.
Then there is one man who taught me how to be a steward (some say leader or manager): Gabriel Acosta-Mikulasek, a coworker, then manager, and now close friend: Querido hermano. He now teaches leadership and living, and you could not ask for a better coach. Find him at www.aculasek.com.

Oddly, I’d like to also thank our kittens (now cats), who continually tried to rewrite content, typing secret cat code such as “vev uiscmr///l’64.” And many thanks to my family, who stood beside me and gave me the time to work, and even provided content. My 10-year-old daughter drew the r graphic used in Chapter 12, “Touch Basics.”

—Maurice Sharp
About the Authors

Maurice Sharp is a 21-year veteran of mobile development at companies both large and small, ranging from Apple, Palm, and eBay to ShopWell and Couchsurfing. Maurice got his start as an intern developing the Newton ToolKit prototype, then as a Developer Technical Support (DTS) Engineer helping make the world safe and fun for Newton then Palm developers. After mastering the DTS side, he went back to coding, and he currently manages and does mobile development at Couchsurfing; runs his own consulting company, KLM Apps; and is ex officio technical advisor to some mobile-focused startups. When not living and breathing mobile, Maurice spends his time being a husband, and a father (to a precocious 10-year-old girl)—his two most important roles.

Erica Sadun is a bestselling author, coauthor, and contributor to several dozen books on programming, digital video and photography, and web design, including the widely popular The Core iOS 6 Developer’s Cookbook, now in its fourth edition. She currently blogs at TUAW.com and has blogged in the past at O’Reilly’s Mac Devcenter, Lifehacker, and Ars Technica. In addition to being the author of dozens of iOS-native applications, Erica holds a Ph.D. in computer science from Georgia Tech’s Graphics, Visualization, and Usability Center. A geek, a programmer, and an author, she’s never met a gadget she didn’t love. When not writing, she and her geek husband parent three geeks-in-training, who regard their parents with restrained bemusement when they’re not busy rewiring the house or plotting global domination.

Rod Strougo is an author, instructor, and developer. Rod’s journey in iOS and game development started way back with an Apple, writing games in Basic. From his early passion for games, Rod’s career moved to enterprise software development, and he spent 15 years writing software for IBM and AT&T. These days, Rod follows his passion for game development and teaching, providing iOS training at the Big Nerd Ranch (www.bignerdranch.com). Originally from Rio de Janeiro, Rod now lives in Atlanta, Georgia, with his wife and sons.
Preface

“Mobile is the future” is a phrase you hear more and more these days. And when it comes to mobile, nobody has more user-friendly devices than Apple.

You want to add iOS development to your set of skills, but where do you begin? Which resources do you need and choose? It depends on how you learn. This book is hands-on. The goal is to get you doing things as soon as possible. You start with small things at first and then build on what you already know.

The result is a book that gives you the skills you need to write an app in an easily digestible format. You can go as fast or slow as you wish. And once you are creating apps, you can turn back to specific parts of the book for a refresher.

So find a comfortable place, have your Mac and your iOS handheld nearby, and dig in!

What You’ll Need

You will need a few things before you go any further in learning iOS development:

- A modern Mac running the current or previous generation of Mac OS—As of the writing of this book, Mac OS X Mountain Lion (v. 10.8) is the latest version with Mavericks just around the corner (not used for this book). Before Mountain Lion was Mac OS X Lion (v. 10.7). Ideally you want to use the latest OS, have at least 8GB of RAM, and lots of disk space.

- An iOS device—Although Xcode includes a desktop simulator for developing apps, you will need to run your app on an actual device to make sure it works correctly. It is helpful to have the same kinds of units your target customers are likely to use to make sure your app works well on all of them.

- An Internet connection—You will need to be able to download development resources. At some point, you might also want to test wireless app functionality. And of course, you will want to ship your app.

- Familiarity with Objective-C—You create native applications for iOS by using Objective-C. The language is based on ANSI C, with object-oriented extensions, which means you also need to know a bit of C. If you have programmed with Java or C++ and are familiar with C, you’ll find that moving to Objective-C is easy. There is a short intro to Objective-C in Chapter 2, “Objective-C Boot Camp,” but a broader understanding will help you learn more quickly.

You also need Xcode, the development tool, and some sort of Apple developer account, as discussed in Chapter 1, “Hello, iOS SDK.”

Your Roadmap to iOS Development

One book can’t be everything to everyone. Try as we might, if we were to pack everything you need to know into this book, you wouldn’t be able to pick it up. There is, indeed, a lot you need to know to develop for the Mac and iOS platforms. If you are just starting out and don’t
have any programming experience, your first course of action should be to take a college-level course in the C programming language.

When you know C and how to work with a compiler (something you’ll learn in that basic C course), the rest should be easy. From there, you can hop right on to Objective-C and explore how to program with it alongside the Cocoa frameworks. The flowchart shown in Figure P-1 shows you key titles offered by Pearson Education that provide the training you need to become a skilled iOS developer.

Figure P-1  A roadmap to becoming an iOS developer
When you know C, you have a few options for learning how to program with Objective-C. If you want an in-depth view of the language, you can either read Apple’s documentation or pick up one of these books on Objective-C:

- **Objective-C Programming: The Big Nerd Ranch Guide** by Aaron Hillegass (Big Nerd Ranch, 2012)
- **Learning Objective-C: A Hands-on Guide to Objective-C for Mac and iOS Developers** by Robert Clair (Addison-Wesley, 2011)

With the language behind you, next up is tackling Cocoa and the developer tools, otherwise known as Xcode. For that, you have a few different options. Again, you can refer to Apple’s documentation on Cocoa and Xcode. See the **Cocoa Fundamentals Guide** (http://developer.apple.com/mac/library/documentation/Cocoa/Conceptual/CocoaFundamentals/CocoaFundamentals.pdf) for a head start on Cocoa, and for Xcode, see **A Tour of Xcode** (http://developer.apple.com/mac/library/documentation/DeveloperTools/Conceptual/A_Tour_of_Xcode/A_Tour_of_Xcode.pdf). Or if you prefer books, you can learn from the best. Aaron Hillegass, founder of the Big Nerd Ranch in Atlanta (www.bignerdranch.com), is the coauthor of **iOS Programming: The Big Nerd Ranch Guide**, second edition, and author of **Cocoa Programming for Mac OS X**, soon to be in its fourth edition. Aaron’s book is highly regarded in Mac developer circles and is the most recommended book you’ll see on the cocoa-dev mailing list. And to learn more about Xcode, look no further than Fritz Anderson’s **Xcode 4 Unleashed** from Sams Publishing.

**Note**

There are plenty of other books from other publishers on the market, including the bestselling **Beginning iPhone 4 Development** by Dave Mark, Jack Nutting, and Jeff LaMarche (Apress, 2011). Another book that’s worth picking up if you’re a total newbie to programming is **Beginning Mac Programming** by Tim Isted (Pragmatic Programmers, 2011). Don’t just limit yourself to one book or publisher. Just as you can learn a lot by talking with different developers, you can learn lots of tricks and tips from other books on the market.

To truly master Apple development, you need to look at a variety of sources: books, blogs, mailing lists, Apple’s documentation, and, best of all, conferences. If you get the chance to attend WWDC (Apple’s Worldwide Developer Conference), you’ll know what we’re talking about. The time you spend at conferences talking with other developers, and in the case of WWDC, talking with Apple’s engineers, is well worth the expense if you are a serious developer.

**How This Book Is Organized**

The goal of this book is to enable you to build iOS apps for iOS handheld and tablet devices. It assumes that you know nothing about iOS development but are familiar with Objective-C. (Although there is a boot camp in Chapter 2, you will find it easier to learn from this book if you are more familiar with the language.) Each chapter introduces new concepts and, where appropriate, builds on knowledge from previous chapters.
Most chapters cover extra material in addition to their core content. The additional material doesn’t necessarily fit with the heart of a particular chapter, but it is important in creating apps. Extra material shows you how to use specific UI elements, provides tips and tricks, explains coding practices, and provides other helpful information.

Here is a summary of each chapter:

- **Chapter 1, “Hello, iOS SDK”—**Find out about the tools, programs, and devices used for creating iOS apps. You start by installing Xcode and also learn about the Apple developer programs and how to sign up. The last two sections help when you design your app. The first covers how limitations of handheld devices inform various iOS technologies. And the last gives a tour of model differences.

- **Chapter 2, “Objective-C Boot Camp”—**An Xcode project is a container for an app’s code, resources, and meta-information. In this chapter, you create your first project. You also get a quick refresher on Objective-C, the language of app development.

- **Chapter 3, “Introducing Storyboards”—**A user of your app sees only the interface. You might implement app behaviors by using incredible code, but the user sees only the effects. In this chapter, you start creating the interface by using a storyboard, a way to see all your app screens at once. You add screens and hook them together and to underlying code. The skills you get from this chapter are a core part of creating iOS apps.

- **Chapter 4, “Auto Layout”—**So far, iOS handheld devices have two different screen sizes and two different orientations for each screen size. Supporting four screen variations can be challenging. In this chapter, you learn and use auto layout, Apple’s constraint-based layout engine, to more easily support multiple screen sizes. You even use it to change layouts when the screen rotates.

- **Chapter 5, “Localization”—**iOS devices are available in at least 155 countries and many different languages. As you go through the chapter, you create one app that supports three languages and many countries. You build on Chapter 4, using auto layout to adjust interface elements for different localized string lengths. You also implement language- and country-specific formatting of dates and times as well as left-to-right and right-to-left writing.

- **Chapter 6, “Scrolling”—**You typically want to present more information than fits on a handheld screen. Sometimes the best way to navigate is to scroll through content. Starting with the simplest use case, you use the built-in scroll view UI element to go from simply bouncing a screen to scrolling through elements. You add pan and zoom as well as display item numbers based on scroll position.

- **Chapter 7, “Navigation Controllers I: Hierarchies and Tabs”—**Navigating complex information can be challenging, especially on a phone’s relatively small screen size. iOS provides navigation controllers to make the job easier. You start by using UINavigationController for moving through a hierarchy of information. Then you use more advanced features providing further customization. Next, you use a tab bar for moving between different kinds of information, and you learn how to work with view controllers that are not on the storyboard.
Chapter 8, “Table Views I: The Basics”—Table views are an important part of apps on both the iPhone and iPad. After learning how they work, you create a table of cars and then implement addition and deletion of items. You go deeper, using a variation of a table for car details. While doing this, you use a picker view for dates and protocols for communicating data and state between scenes.

Chapter 9, “Introducing Core Data”—Core Data provides full data management for a relatively small amount of work. In this chapter, you create a Core Data model for the app and use that data for the list of cars and car detail. Next, you use built-in objects to make managing the table view of cars even easier. You also learn ways to convert a project to use Core Data, and you become familiar with common errors.

Chapter 10, “Table Views II: Advanced Topics”—There are several advanced features of table views for adding polish to apps. As the chapter progresses, you implement different features, including custom cells, sections, sorting, a content index, and searching. You also learn about UISegmentedControl, a bit more on debugging, and a good way to use #define.

Chapter 11, “Navigation Controllers II: Split View and the iPad”—Apps for the iPad usually require a different design than ones for the iPhone. In this chapter, you create a universal app, one that works on both the iPhone and iPad. You build a separate interface using the iPad-only UISplitViewController. You learn how to adapt iPhone views to iPad and how to choose when to use them and when to create something new. In addition, you implement a singleton, a special object that can have only one instance, learn the usefulness of accessor methods, and implement custom transition animations.

Chapter 12, “Touch Basics”—Almost everything a user does on iOS devices involves gestures with one or more fingers. Some features, like buttons, are easy to add. Others take more work. In this chapter, you learn the basics of gesture recognizers and add swiping through car detail views. Then you go deeper, creating a custom gesture recognizer. Finally, you add a draggable view.

Chapter 13, “Introducing Blocks”—From animating views to error responders, blocks are an important tool for using system calls. You learn how to create and use blocks, and use them to add pulsing to a view. You also learn about variable scope and read-only versus modifiable variables. Finally, you replace a protocol using blocks.

Chapter 14, “Instruments and Debugging”—There are two constants in app development: Initial implementations rarely perform as you expect, and there are always bugs. In this chapter, you start by fixing a performance problem using Instruments, a tool for checking performance, memory use, and other important parts of your app. Next, you learn some advanced features of breakpoints in the debugger. Then, you use both tools to solve one of the hardest types of bugs. In this chapter, you also learn about a process for finding and fixing problems, as well as a way to use background tasks.

Chapter 15, “Deploying Applications”—In the final chapter, you take your app from your machine to the App Store. First, you create any required developer credentials and app security certificates. You add icons and launch images, and then you learn about useful extra functionality for your app, such as metrics and bug reporting, as well as some of the main providers. After a brief look at marketing, you get the App Store ready to receive your application, build it, and upload it. The chapter ends with a summary of resources you can use as you continue your journey of creating great iOS apps.
About the Sample Code

As you progress through this book, you develop and refine an application for valet car parking. The CarValet app is used as a practical implementation for concepts you learn. It is not meant to be an app shipped to the masses, although it could serve as a base for one.

Any chapter that involves creating code usually comes with at least two projects: a starter that incorporates code from any previous chapters in the book and a finished project, including all changes made in the chapter. For most of the book, you can use your own completed project from one chapter as the starter for the next. There are a couple places where this is not the case, and the chapter makes that plain.

Except for the very end, the sample code projects use the same unique bundled identifier: com.mauricesharp.CarValet. As a result, you cannot have multiple versions installed in the simulator or on your device at the same time. If you want to have multiple versions, you can simply add a unique string to the end of the identifier, such as com.mauricesharp.CarValet.CH05.portrait. You’ll learn the significance of the bundle identifier in Chapter 15.

All the code you write and concepts you learn work with iOS 7 or later. By the end of the first day of availability, more than 35% of existing devices were using the iOS 7, the fastest adoption rate ever. That share will only increase. Adoption rates for iOS are usually very fast, typically hitting 80% or higher within a few months.

Getting the Sample Code

All the sample code is on GitHub, at https://github.com/mauricesharp/Learning-iOS-Development. The code is organized by chapter, with most folders containing starter and finished projects. Some also contain projects for interim steps, as well as folders containing new assets such as images. For example, these are the folders for Chapter 6:

- **CH06 CarValet Starter**—The finished project from Chapter 5, with no changes from Chapter 6. Use either this project or your own project from the end of Chapter 5 as a starting place for Chapter 6 additions.
- **CH06 CarValet Finished**—A project with all the changes from Chapter 6. You can use this as a reference for what changes should have been made or as a starter for the next chapter.
- **CH06 Assets CarImages**—An extra folder with image resources used in changes made during the chapter.

The code will be refreshed as needed. If you see something that needs changing, is missing, or even a way to implement something in a better way, feel free to...
Contribute!

Sample code is never a fixed target. It continues to evolve as Apple updates its SDK and the Cocoa Touch libraries. Get involved. You can pitch in by suggesting bug fixes and corrections, as well as by expanding the code that’s on offer. GitHub allows you to fork repositories and grow them with your own tweaks and features, and you can share those back to the main repository using a Pull Request on GitHub. If you come up with a new idea or approach, let us know. We are happy to include great suggestions both at the repository and in the next edition of this book.

Accessing git

You can download this book’s source code by using the git version control system. An OS X implementation of git is available at http://code.google.com/p/git-osx-installer. OS X git implementations include both command-line and GUI solutions, so hunt around for the version that best suits your development needs.

There are third-party git tools, as well—some free and some not. These are two of the most popular:

- SourceTree—A free git hub client tool available at www.sourcetreeapp.com
- Tower—A paid client with a polished UI at www.git-tower.com

Accessing GitHub

GitHub (http://github.com) is the largest git-hosting site, with more than 150,000 public repositories. It provides both free hosting for public projects and paid options for private projects. With a custom web interface that includes wiki hosting, issue tracking, and an emphasis on social networking of project developers, it’s a great place to find new code and collaborate on existing libraries. You can sign up for a free account at http://github.com. When you do that, you can copy and modify the book repository or create your own open-source iOS projects to share with others.

Contacting the Author

If you have any comments, questions, or suggestions about this book, please e-mail me at learningios@mauricesharp.com.

This book was written using developer preview releases of both iOS 7 and Xcode. Several different versions were used, though the majority was done using DP (Developer Preview) 4. Large portions of the book were checked against DP 6, the last preview before the final release, but some earlier code does exist, especially in the CarValet sample. Check the errata for updates.

Now read through these pages, write the code, and do the challenges. By the end, you will know how to create iOS apps for handholds and tablets.
Editor’s Note: We Want to Hear from You!

As the reader of this book, you are our most important critic and commentator. We value your opinion and want to know what we’re doing right, what we could do better, what areas you’d like to see us publish in, and any other words of wisdom you’re willing to pass our way.

You can e-mail or write us directly to let us know what you did or didn’t like about this book—as well as what we can do to make our books stronger.

Please note that we cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail we receive, we might not be able to reply to every message.

When you write, please be sure to include this book’s title and authors as well as your name and phone or e-mail address.

E-mail: trina.macdonald@pearson.com

Mail: Trina MacDonald
Senior Acquisitions Editor
Addison-Wesley Pearson Education, Inc.
75 Arlington Street, Suite 300
Boston, MA 02116 USA
Developing for iOS is a joyful and fun adventure in learning Objective-C and the Apple frameworks. Nowhere else is it so easy and quick to go from an idea to an app you can hold in your hand on an iPhone, iPad, or iPod touch. With your code behind the glass touchscreen, you can turn these devices into anything you can think of. An iOS device can become a flight simulator, an interactive book, or just about anything else you can imagine. In this chapter, you take the first steps in developing for iOS by learning about the iOS Software Development Kit (SDK) and how to get the Xcode toolset installed on your Mac. (It is easy.) In the next chapter, you dive in, create your first iOS app, and get it running on the iOS Simulator.

The iOS family includes the iPhone, the iPad, and the iPod touch. Despite their relatively diminutive proportions compared to desktop systems, they use increasingly powerful multi-core CPUs to run iOS, a first-class version of OS X. iOS comes with a rich and varied SDK that enables you to design, implement, and realize a wide range of applications. For your projects, you can take advantage of the multitouch interface and powerful onboard features using Xcode, Apple’s integrated development environment (IDE). In this chapter, you learn about Apple’s various iOS Developer Programs and how you can join. Ready? Onward to the next step: getting the Xcode application installed on your Mac.

**Installing Xcode**

The first step in developing for the iOS platform is to get Xcode: the IDE from Apple. Xcode is the tool you use for writing Objective-C applications and compiling them for iOS devices. Apple has recently made installing Xcode as easy as possible by providing Xcode as a free download from the Mac App Store, as shown in Figure 1-1.
To install Xcode, follow these steps:

1. Launch the Mac App Store application on your Mac.
2. Search for Xcode.
3. Click the Free button to install Xcode.

While Xcode is downloading and being installed, you can read the rest of this chapter and learn about the iOS SDK. That is all it takes to install Xcode and get on your way. The rest of this chapter covers the iOS SDK, the devices, and the development programs Apple offers. In Chapter 2, “Objective-C Boot Camp,” you start your journey into the Objective-C language and application development in iOS.

**About the iOS SDK**

The iOS SDK comprises all the libraries you need to write iOS apps, as well as the iOS Simulator for you to try out your apps on your Mac. The SDK is included with the Xcode tool, which is used for creating iOS and Mac applications.
You can register for free for the Apple Online Developer Program and download and explore the full iOS SDK programming environment. However, as discussed in the next section, this program doesn’t let you deploy your applications to an actual iOS device, such as the iPhone or iPad. If you want to do that, you need to register and become a member of Apple’s iOS Developer Program. There are four program choices, described in Table 1-1.

<table>
<thead>
<tr>
<th>Program</th>
<th>Cost</th>
<th>Audience</th>
</tr>
</thead>
<tbody>
<tr>
<td>iOS Developer Program–Individual</td>
<td>$99/Year</td>
<td>Individual developers who want to distribute through the App Store. The apps will appear under your name in iTunes.</td>
</tr>
<tr>
<td>iOS Developer Program–Company</td>
<td>$99/Year</td>
<td>For a company or development team that wants to distribute through the App Store. The apps will appear under the company name in iTunes.</td>
</tr>
<tr>
<td>iOS Developer Enterprise Program</td>
<td>$299/Year</td>
<td>Large companies building proprietary software for employees and distributing the apps in-house.</td>
</tr>
<tr>
<td>iOS Developer University Program</td>
<td>Free</td>
<td>Free program for higher education institutions that provide iPhone development curriculum.</td>
</tr>
</tbody>
</table>

Each program offers access to the same iOS SDK, which provides ways to build and deploy your applications. The audience for each program is specific. Keep in mind that if your company wants to deploy apps in the normal App Store, all you need is the iOS Developer Program–Company. The Enterprise option is available to you only if your company wants to deploy apps in a private in-house App Store.

The following sections discuss the various iOS Developer Programs in more detail.

**What You Get for Free**

The free program is for anyone who wants to explore the full iOS SDK programming environment but isn’t ready to pay for further privileges. The free program limits your deployment options to the iOS Simulator. Although you can run your applications in the simulator, you cannot install those applications to a device or sell them in the App Store.

Although each version of the simulator moves closer to representing iOS, you should not rely on it for evaluating your application. An app that runs rock solid on the simulator might be unresponsive or even cause crashes on an actual device. The simulator does not, for example, support vibration or accelerometer readings. These and other features present on devices are not always available in the simulator. A more detailed discussion about the simulator...
and its differences from a real device follows later in this chapter, in the section “Simulator Limitations.”

While you can download Xcode for free and without registering, joining a full program gives you access to much more, including the ability to run your code on devices, access to early releases, and even the ability to ask questions of Apple developer support engineers.

**iOS Developer Program (Individual and Company)**

To receive device and distribution privileges, you pay a program fee, currently $99/year, for the standard iOS Developer Program. You can join as an individual or as a company. When you have paid, you gain access to App Store distribution and can test your software on actual iOS hardware. This program adds ad hoc distribution as well, allowing you to distribute prerelease versions of your application to a set number of registered devices. The standard program provides the most general solution for the majority of iOS programmers who want to be in the App Store. If you intend to conduct business by selling applications, this is the program to sign up for.

The standard iOS Developer Program also offers early access to beta versions of the SDK. This is a huge advantage for developers who need to prepare products for market in a timely manner and to match Apple’s OS and device upgrade dates. As an example, program members gained access to early versions iOS 7 and Xcode 5 in June 2013.

---

**Caution: Going from Individual to Company Is Hard to Do**

Joining the company program currently requires paperwork to prove the company is a valid corporate entity. Changing from individual to company is even harder than starting with a company membership. If you are an individual and expect to become a company, even if only for liability protection, you are better off creating the company first and then joining the Developer Program, or even joining as an individual and then creating a separate company membership later. Joining as a company does take longer, especially with the current requirement for a DUNS (Data Universal Numbering System) number.

---

**Note**

In early 2010, Apple restructured its Macintosh Developer Program to match the success of the iOS Developer Program. Currently costing $99/year, the restructured Mac program offers the same kind of resources as the iOS program: code-level technical support incidents, developer forum membership, and access to prerelease software. Neither program offers hardware discounts. The Mac Developer Program does not give access to iOS software and vice versa.

**Developer Enterprise Program**

The Enterprise Program, currently $299/year, is meant for in-house application distribution. It’s targeted at companies with 500 employees or more. Enterprise memberships do not offer
access to the Apple public App Store. Instead, you can build your own proprietary applications and distribute them to your employees’ hardware through a private storefront. The Enterprise Program is aimed at large companies that want to deploy custom applications such as ordering systems to their employees.

**Developer University Program**

Available only to higher education institutions, the Developer University Program is a free program aimed at encouraging universities and colleges to form an iOS development curriculum. The program enables professors and instructors to create teams with up to 200 students, offering them access to the full iOS SDK. Students can share their applications with each other and their teachers, and the institution can submit applications to the App Store.

**Registering**

Register for a free or paid program at the main Apple developer site: http://developer.apple.com/programs/register.

Regardless of which program you sign up for, you must have access to a Mac running a current version of Mac OS X. It also helps to have at least one—and preferably several—iPhone, iPad, and/or iPod touch units. These are for testing to ensure that your applications work properly on each platform, including legacy units. What better excuse for buying that iPhone, iPad, or iPod touch you’ve been wanting...err, needing for business purposes?

Often, signing up for paid programs involves delays. After registering, it can take time for account approval and invoicing. When you actually hand over your money, it may take another 24 to 72 hours for your access to advanced portal features to go live. There is a very short delay for individual registration, and the delay is longer for companies.

Registering for iTunes Connect, so you can sell your application through the App Store, is a separate step. Fortunately, this is a process you can delay until after you’ve finished signing up for a paid program. With iTunes Connect, you must collect banking information and incorporation paperwork prior to setting up your App Store account. You must also review and agree to Apple’s distribution contracts. Apple offers full details at http://itunesconnect.apple.com. Bear in mind that it can take several days until you are able to upload apps, so do not delay signing up for too long.

**iTunes U and Online Courses**

When you have registered for any level of iOS development with Apple, you will have access to the World Wide Development Conference (WWDC) videos that Apple releases each year. These high-quality presentations, given by Apple’s own engineers, provide great insight into many of the features in iOS and examples of how to use them. In addition, there are many iPhone programming courses available for free on iTunes University (iTunes U inside iTunes) that you can use as a companion to this book.
The iOS SDK Tools

Xcode typically runs a few gigabytes in size and installs an integrated suite of interactive design tools onto your Macintosh. This suite consists of components that form the basis of the iOS development environment and includes the following parts:

- **Project Editor**—This is the heart of Xcode and provides a home for most of the features, including project file and component management, syntax-aware source editing for both the Objective-C language and iOS SDK, as well as a visual editor and a full debugger. A separate window gives access to the full range of documentation for iOS, Xcode, and other supporting documentation.

- **Interface Builder (IB)**—IB is accessed through the project editor and provides a rapid prototyping tool for laying out user interfaces (UIs) graphically, and linking those prebuilt interfaces to your Xcode source code. With IB, you use powerful visual design tools to add the visual elements of your app and then connect those onscreen elements to objects and method calls in your application. In addition to individual screens, you can lay out all your application screens in one document and define the ways each screen moves to the next. You learn about this in Chapter 3, “Introducing Storyboards.” In Chapter 4, “Auto Layout,” you learn how to use IB with another powerful feature of iOS. Auto layout is an advanced rule-based system that enables you to specify the visual relationships between views instead of worrying about pixel-perfect placement. With it, you can create one interface that adapts to different screen orientations and sizes.

- **Simulator**—The iOS Simulator runs on a Macintosh and enables you to create and test iOS apps on your desktop. You can test programs without connecting to an actual iPhone, iPad, or iPod touch. The simulator offers the same API (Application Programming Interface) used on iOS devices and provides a preview of how your concept designs will look and behave. When working with the simulator, Xcode compiles Intel x86 code that runs natively on the Macintosh rather than ARM-based code used on the iPhone. Keep in mind that performance in the simulator is likely very different than on a physical device as it is running with a very different CPU, GPU (graphics processor), and storage/disk format. Your app is likely to be much faster in the simulator and have no memory or communications problems.

- **Performance Tools**—As you run your app in the simulator or on a device, runtime debug gauges give an overview of performance including memory and CPU use. Instruments provides even more detail, profiling how iPhone applications work under the hood. It samples memory usage and monitors performance, enabling you to identify and target problem areas in your applications and work on their efficiency. As you see in Chapter 14, “Instruments and Debugging,” if you tune your app as you develop, you will catch issues early and end up with the best performance. Instruments offers graphical time-based performance plots that show where your applications are using the most resources. It is built around the open-source DTrace package developed by Sun Microsystems and plays a critical role in making sure your applications run efficiently on the iOS platform.

In the simulator or on a device, the static analyzer shows you places where your code might have problems. Simply run the analyzer on a single file or on your whole project to find unused variables, possible logic problems, potential memory leaks, and more.
- **Debugger**—Chapter 14 also covers the debugger. It helps you quickly find and fix problems in your code. With it, you can step through code and inspect values of variables, either in a separate display area or by just hovering the mouse pointer over the source code. You can set rich breakpoints, including conditional triggers and associated actions such as logging messages, playing source, or even running scripts. There is even a console for fine control.

- **Other Features**—Xcode provides a wide array of other features supporting the app development and deployment cycle including built-in support for branching source code control using Git, management of developer certificates and app entitlements, testing device management, and uploading apps to the store.

Together, the components of the iOS SDK enable you to develop your applications. From a native application developer's point of view: You will spend most of your time editing and debugging source, creating the interface, and running your app in the simulator. You will also spend time tuning your code in instruments. In addition to these tools, there's an important piece not on this list. This piece ships with the SDK, but is easy to overlook: Cocoa Touch.

Cocoa Touch is a library of classes provided by Apple for rapid iOS application development. Cocoa Touch, which takes the form of a number of API frameworks, enables you to build graphical event-driven applications with UI elements such as windows, text, and tables. Cocoa Touch and UIKit on iOS is analogous to Cocoa and AppKit on Mac OS X and supports creating rich, reusable interfaces on iOS.

Many developers are surprised by the code base size of iOS applications; they're tiny. Cocoa Touch's library support is the big reason for this. By letting Cocoa Touch handle all the heavy UI lifting, your applications can focus on getting their individual tasks done. The result is compact code, focused on the value provided by your app.

Cocoa Touch lets you build applications with a polished look and feel, consistent with those developed by Apple. Remember that Apple must approve your software. Apple judges applications on the basis of appearance, operation, and even content. Using Cocoa Touch helps you better approximate the high design standards set by Apple's native applications.

Before you start creating apps, make sure you look at the Apple “iOS Human Interface Guidelines” available in the Xcode documentation in the “User Interface” group, or on the web at https://developer.apple.com/appstore/guidelines.html. Also read through the legal agreement you signed for iTunes Connect. Breaking rules is highly likely to result in your app being rejected from the App Store.

### Testing Apps: The Simulator and Devices

A physical iPhone, iPad, or iPod touch is a key component of the SDK. Testing on a device is vital. As simple and convenient as the iOS Simulator is, it is not the same as a real device. You want your apps to run on some or all of the iOS device family, so it's important that they run best in the native environment. An iOS device itself offers the fully caffeinated, un-watered-down testing platform you need.
Apple regularly suggests that a development unit needs to be devoted exclusively to development. Reality has proven rather hit and miss on that point. Other than early betas, releases of iOS have proven stable enough that you can use your devices for both development and day-to-day tasks, including making calls on iPhones. It’s still best to have extra units on hand devoted solely to development, but if you’re short on available units, you can probably use your main iPhone for development; just be aware of the risks, however small. Note that as a developer program member, you have agreed to a non-disclosure agreement (NDA) with Apple. Beware of accidentally showing Apple confidential prereleases to others.

Devices must be proactively set up for development use with Xcode’s Organizer. The Organizer also lets you register your device with Apple, without having to enter its information by hand at the provisioning portal. Chapter 15, “Deploying Applications,” gives detailed information on how to do this.

When developing, it’s important to test on as many iOS platforms as possible. Be aware that real platform differences exist between each model of iPhone, iPad, and iPod touch. For example, two models of the fifth-generation iPod touch offer front- and back-facing cameras; one only offers a front-facing camera. The second-generation iPad and earlier as well as the original iPad-mini do not have retina screens. iPhones all have cameras, which none of the iPod touches offered until the fourth generation. Certain models of the iPad and the iPhone offer GPS technology; other models do not. A discussion of major platform device features along with some device differences follows later in this chapter.

**Note**

iOS developers do not receive hardware discounts for development devices. You pay full price for new devices, and you pay nonsubsidized prices for extra iPhones and iPads with carrier access. You can get significant savings by buying used and refurbished units. Depending on your country and other circumstances, you might be able to deduct the cost of units from your taxes.

**Simulator Limitations**

Each release of the Macintosh-based iOS Simulator continues to improve on previous technology. That said, there are real limitations you must take into account. From software compatibility to hardware, the simulator approximates but does not equal actual device performance.

The simulator uses many Macintosh frameworks and libraries, offering features that are not actually present on the iPhone or other iOS devices. Applications that appear to be completely operational and fully debugged on the simulator might flake out or crash on a device itself due to memory or performance limitations on iOS hardware. Even the smallest Mac nowadays comes with 4GB of RAM, whereas the third-generation iPad has only 1GB of RAM. Instruction set differences might cause apps to crash on older devices when they are built to support only newer versions of the ARM architecture. You simply cannot fully debug any program solely by using the simulator and be assured that the software will run bug-free on iOS devices.
The simulator is also missing many hardware features. You cannot use the simulator to test the onboard camera or to get accelerometer and gyro feedback. Although the simulator can read acceleration data from your Macintosh using its sudden motion sensor (if there’s one onboard, which is usually the case for laptops), the readings will differ from iOS device readings and are not practical for development or testing. The simulator does not vibrate or offer multitouch input (at least not beyond a standard “pinch” gesture).

**Note**

The open-source accelerometer-simulator project at Google Code (http://code.google.com/p/accelerometer-simulator/) offers an iPhone application for sending accelerometer data to your simulator-based applications, enabling you to develop and debug applications that would otherwise require accelerometer input. A similar commercial product called iSimulate is available in the App Store for purchase.

From a software point of view, the basic keychain security system is not available on the simulator. You cannot register an application to receive push notification either. These missing elements mean that certain kinds of programs can be properly tested only when deployed to an iPhone or other iOS device.

Another difference between the simulator and the device is the audio system. The audio session structure is not implemented on the simulator, hiding the complexity of making things work properly on the device. Even in areas where the simulator does emulate the iOS APIs, you might find behavioral differences because the simulator is based on the Mac OS X Cocoa frameworks. Sometimes you have the opposite problem: Some calls do not appear to work on the simulator but work correctly on the device. For example, if you store or access files, the simulator is usually case-insensitive (depending on how the Mac is set up), but iOS is case-sensitive.

That’s not to say that the simulator is unimportant in testing and development. Trying out a program on the simulator is quick and easy, typically much faster than transferring a compiled application to an iOS unit. The simulator lets you rotate your virtual device to test reorientation, produce simulated memory warnings, and try out your UI as if your user were receiving a phone call. It’s much easier to test out text processing on the simulator because you can use your desktop keyboard rather than hook up an external Bluetooth keyboard to your system and you can copy and paste text from local files; this simplifies repeated text entry tasks such as entering account names and passwords for applications that connect to the Internet.

Another area the simulator shines is localization. As you see in Chapter 5, “Localization,” switching languages for your app is as easy as launching the simulator with the right special flag.

In the end, the simulator offers compromise: You gain a lot of testing convenience but not so much that you can bypass actual device testing.
**Note**

The simulator supports Video Out emulation. There’s no actual Video Out produced, but the simulated device responds as if you’ve added a compliant cable to its (nonexistent) connector. You can view the “external” video in a floating simulator window.

Apple encourages new applications to use AirPlay to send the content to the user’s TV via AppleTV instead of relying on cables.

---

**Tethering**

Apple is moving away from tethered requirements in iOS but has not yet introduced a way to develop untethered at the time this book is being written. At this time, all interactive testing is done using a USB cable. Apple provides no way to wirelessly transfer, debug, or monitor applications as you develop. This means you perform nearly all your work tethered over a standard iPhone USB cable.

When you are debugging a tethered unit, try to set things up to reduce accidentally disconnecting the cable. If that happens, you lose the debug session including any interactive debugging, the console, and screenshot features.

You want to invest in a good third-party dock for iPhones or iPod touches and possibly one for iPads. Look for stands that allow the cable to be connected and hold the unit at a comfortable angle for touching the screen. Even better are docks that work in both portrait and landscape. The iPad will work in the Apple doc, though only in portrait. Alternatively, the Apple folding cases that also act as stands work in both orientations.

When tethered, always try to connect your unit to a port directly on your Mac. If you must use a hub, connect to a powered system that supports USB 2.0 or higher. Most modern screens, including Apple’s large display, come with built-in powered USB ports, but it pays to double check.

When it comes to the iPad, if the USB connection does not have sufficient power to charge the device, untether your device between testing periods and plug it directly into the wall using its 10W power adapter. Some USB ports provide sufficient power to charge the iPad while you’re using it, but this is not a universal situation.

---

**Note**

When testing applications that employ Video Out, you can use the Apple-branded component and composite cables or the HDMI digital adapter. These provide both Video Out and USB connections to allow you to tether while running your applications. The Apple-branded VGA cable does not offer this option. You need to redirect any testing output to the screen or to a file because you cannot tether while using VGA output. Another common way to show apps on another device is to use AirPlay screen mirroring. It is a good idea to pick up an AppleTV and test whether your app works well with AirPlay. It can also save money compared to buying adapter cables for both the original 30-pin and newer lightning connectors.
iOS Device Considerations

Designing apps for mobile platforms such as the iPhone or iPad is not the same as designing for the desktop (or laptop). There are several extra considerations such as storage, interaction methods, and battery life. Storage limits, smaller screens, different interaction techniques, and energy consumption are important design considerations when creating your app.

With the iPhone, you are designing for a small touch-based screen with a good, but limited battery life. It is not a desktop with a large screen, a mouse or trackpad, and a physical always-on A/C power supply. Platform realities must shape and guide your development. Fortunately, Apple has done an incredible job designing a platform that leverages flexibility from its set of storage, interaction controls, and constrained battery life.

Storage Considerations

The iPhone hosts a powerful yet compact OS X–based installation. Although the entire iOS fills no more than a few hundred megabytes of space—almost nothing in today’s culture of large operating system installations—it provides an extensive framework library. These frameworks of precompiled routines enable iPhone users to run a diverse range of compact applications, from telephony to audio playback, from e-mail to web browsing. The iPhone provides just enough programming support to create flexible interfaces while keeping system files trimmed down to fit neatly within tight storage limits.

Most modern devices come with at least 16GB of onboard Flash-based storage, and some have considerably more. Some older devices running iOS 7 and later have as little as 4GB. Although application size is limited (see the “Note: App Size”), the space for data is much larger. Having said that, be aware that users can check how much space an app is using and might delete hungrier apps.

Note: App Size

Each application is limited to a maximum size of 2GB. To the best of my knowledge, no application has ever actually approached this size, although there are some navigation apps that are pushing new records of deployment size, such as Navigon (1.5GB) and Tom Tom (1.4GB). Apple currently restricts apps larger than 50MB to Wi-Fi downloading. This bandwidth was set at the time that Apple announced its new iPad device and the possibility of delivering universal applications that could run on both platforms. Apple’s over-the-air restrictions help reduce cell data load when media-intensive applications exceed 50MB and ease the pain of long download times. The 50MB limit is also an important design consideration. Keeping your size below the 50MB cutoff allows mobile users to make impulse application purchases, increasing the potential user base. Check the iTunesConnect guide for the latest maximum size.

Data Access Considerations

Every iOS application is sandboxed. That is, it lives in a strictly regulated portion of the file system. Your program cannot directly access other applications, certain data, and certain folders. Among other things, these limitations require accessing built-in application data using
system APIs including the iTunes library, calendar, photos, location services, notifications, reminders, and built-in social services such as Facebook and Twitter.

Your program can, however, access any data that is freely available over the air when the iOS device is connected to a network—including any iCloud documents it owns. Your app can also access data stored in the shared system pasteboard and data shared using a document interaction controller, which offers a limited way to share document data between applications. Apps that create or download data can send those files to applications that can then view and edit that data. In that situation, the data is fully copied from the originating application into the sandbox of the destination application.

**Memory Considerations**

On iOS, memory management is critical. Apple has not enabled disk swap–based virtual memory for iOS. When you run out of memory, iOS shuts down your application; random crashes are probably not the user experience you were hoping for. With no swap file, you must carefully manage your memory demands and be prepared for iOS to terminate your application if it starts swallowing too much memory at once. You must also take care concerning what resources your applications use. Too many high-resolution images or audio files can bring your application into the auto-terminate zone.

Many parts of the iOS framework cache your image data in order to speed up rendering and application performance. This caching can come at the cost of a larger memory footprint and, on retina devices, if used improperly, can generate more memory pressure on your app. Chapter 14 covers using the Instruments tool to figure out what parts of your application consume too much memory and techniques to address and resolve those issues. It also covers the debug memory gauge, a handy way to see if and when your app is approaching the memory danger zone.

**Interaction Considerations**

For the iPhone and iPod touch, losing physical input devices such as mice and working with a small screen doesn’t mean you lose interaction flexibility. With multitouch and the onboard accelerometer, you can build UIs that defy expectations and introduce innovative interaction styles. The iPhone’s touch technology means you can design applications complete with text input and pointer control, using a virtual screen that’s much larger than the actual physical reality held in your palm.

**Note**

Almost all iOS devices support external keyboards. You can connect Bluetooth and USB keyboards to iOS devices for typing. Only a tiny fraction of devices running versions of iOS older than 3.2 have no external keyboard support.

In addition to the touchscreen, users can interact with your app using a smart autocorrecting onscreen keyboard, built-in microphone (for all units except on the obsolete first-generation iPod touch), and an accelerometer that detects current orientation as well as changes. When
designing text input, look for ways you can make it easier for the user such as splitting up longer inputs into smaller fields or using auto completion. For longer text areas, make sure you use scrolling text views. Most importantly, try your interface without an external keyboard, as most users will not have one.

Focus your design efforts on easy-to-tap interfaces rather than on desktop-like mimicry. Remember to use just one conceptual window at a time—unlike in desktop applications, which are free to use a more flexible multiwindow display system.

**Note**
The iPhone screen supports up to five touches at a time. The iPad screen supports up to about 11 touches at a time. With its larger screen, the iPad invites multihand interaction and gaming in ways that the iPhone cannot, particularly allowing two people to share the same screen during game play. Virtual instruments are another type of app that benefits from lots of fingers. Apple has not specified the maximum number of touches for an iPad at the time of writing this book, but empirical evidence still points to 11. See [http://mattgemmell.com/2010/05/09/ipad-multi-touch/](http://mattgemmell.com/2010/05/09/ipad-multi-touch/).

**Energy Considerations**
For mobile platforms, wise use of the battery is part of any design. Apple’s SDK features help to design your applications to limit CPU use and avoid running down the battery. A smart use of technology (for example, properly suspending themselves between uses) lets your applications play nicely on the iPhone and keeps your software from burning holes in users’ pockets (sometimes almost literally, speaking historically). Some programs, when left running, produce such high levels of waste heat that the phone becomes hot to the touch, and the battery quickly runs down. The Camera application was one notable example.

Heavy users of the battery include the Camera app; communications, especially over phone networks; and high-precision location services that use the GPS hardware instead of Wi-Fi triangulation.

Each new generation of iOS device brings some improvement to battery life. Even so, you should continue to keep energy consumption in mind when developing your applications.

**Application Considerations**
With iOS multitasking, applications can allow themselves to

- Be suspended completely between uses (the default behavior)
- Be suspended with occasional slices of background processing time
- Quit entirely between uses
- Run for a short period of time to finish ongoing tasks
- Create background tasks that continue to run as other applications take control
There is built-in support for background tasks including playing music and other audio, collecting location data, and using Voice over IP (VoIP) telephony. Rather than running a simple background daemon, these tasks are event-driven. Your application is periodically called by iOS with new events, allowing the application to respond to audio, location, and VoIP updates.

Since only the current app can update the UI, Apple supports pushing data from web services. Using Push Notifications sends processing off-device to dedicated web-based services, leveraging their always-on nature to limit on-device processing requirements. Registered services can push badge numbers and messages to users, letting them know that new data is waiting on those servers. Push notifications can allow the user to launch your app or bring it to the foreground, passing a small amount of optional data while doing so.

A special kind of notification gives your app some background execution time for updating changes. And even if you do not use notifications, you can ask the system for regular background processing callbacks. These two mechanisms keep your app up to date before the user brings it into the foreground.

In addition, applications can pass control from one to the other by passing data (using the document interaction controller) and by opening custom URL schemes.

Apple strongly encourages developers to limit the amount of cell-based data traffic used by each application. The tendency of carriers to meter data usage and the overall movement away from unlimited data plans help reinforce this requirement. Applications that are perceived to use too much cell bandwidth might be rejected or pulled from the store. If your application is heavily bandwidth-dependent, you may want to limit that use to Wi-Fi connections.

Almost all device families come with Wi-Fi, mostly 802.11n. For those with cellular connections, many are at least 4G (5.8Mbps HSUPA), and LTE is usually the minimum speed for new devices.

**Note**

According to the iPhone Terms of Service, you may not use Cocoa Touch’s plug-in architecture for applications submitted to the App Store. You can build static libraries that are included at compile time, but you may not use any programming solution that links to arbitrary code at runtime. That means your app cannot download new or replacement code from a server. That means bug fix releases need to be just that, full app releases. It also means extra code-level features available by in-app purchase need to ship with the app.

**User Behavior Considerations**

Although this is not a physical device-based consideration, iPhone users approach phone-based applications sporadically. They enter a program, use it for its intended purpose, and then leave just as quickly. The handheld nature of the device means you must design your applications around short interaction periods and prepare for your application to be interrupted as a user receives a phone call or sticks the phone back into a pocket, purse, or backpack. Keep your application state current between sessions and relaunch quickly to approximately the same task
your user was performing the last time the program was run. This can demand diligence on
the part of the programmer, but payoff in user satisfaction is worth the time invested. Apple
does provide APIs for state restoration, though they are beyond the scope of this book. For
more information, start with the chapter on state preservation and restoration in the iOS App
Programming Guide available with the documentation that comes with Xcode.

**Understanding Model Differences**

When it comes to application development, many iOS apps never have to consider the plat-
form on which they're being run. Most programs rely only on the display and touch input.
They can be safely deployed to all the current family of iOS devices; they require no special
programming or concern about which platform they are running on.

There are, however, real platform differences. The most obvious difference is in screen size
between iPhones/iPod touches and iPads. Other differences are usually feature-based such as the
types of sensors, the presence or absence of cellular-based networking, and a few other items.

These differences can play a role in deciding how you tell the App Store to sell your software
and how you design the software in the first place. Should you deploy your software only to
the iPhone family or only to the iPad? To the iPhone, the iPad, and the second-generation and
later iPod touch? Or can your application be targeted to every platform? You can use APIs and
other techniques to find out what particular features are on a given device and even enable or
disable parts of your app. The next section covers some issues to consider.

**Screen Size**

The most obvious difference is the amount of screen space available on the iPad family versus
iPhone or iPod touch. iPads have a large 1024x768 point resolution enabling the display of
much more content. iPhones and iPod touches have two display geometries: The 3.5-inch
screen used by earlier devices is 480x320 points while the newer 4-inch screen is 568x320.

Notice that the above resolutions are in points, not pixels. Most Apple devices now use a higher
resolution retina display, doubling the number of available pixels and better matching human
vision. Luckily, instead of worrying about whether the device is 480x320 (non-retina) pixels or
960x640 (retina) pixels, you can work in the world of points. For artwork, Xcode makes it easy
to provide any appropriate resolutions and, at runtime, the system automatically chooses the
right one.

The Apple human interface guidelines for iPad differ from those for iPhone/iPod touch.
Developing for the iPad involves creating unified interfaces rather than the staged screen-by-
screen design used by the earlier iPhone and iPod touch units, with their reduced window size.
Applications that rely on the greater screen scope that the iPad provides may not translate well
to the smaller members of the device family.

Although the retina screens on the newer iPhones and iPod touches look great, their screen
dimensions are either 3.5- or 4-inches diagonal. That geometry, combined with the physical
realities of the human hand and fingers, prevents these units from providing the same kind of user interaction experience that is possible on the iPad. The interaction guidelines for the newest units remain in lock step with the earlier members of the iPhone and iPod touch family.

Camera

Most applications can assume there will be at least one camera. In most cases, there will be front- and back-facing cameras, though it is still wise to check at runtime. Although some very early devices had no camera (earlier iPod touches or the first-generation iPad), those devices make up a very small percentage of the market, and none of them run iOS 7. There are also devices with just a back-facing or a front-facing camera. The 16GB fifth-generation iPod touch is an example of the latter.

The cameras are useful. You can have the camera take shots and then send them to Flickr or Twitter. You can use the camera to grab images for direct manipulation, augmented reality, and so forth. The iOS SDK provides a built-in image picker controller that offers camera access to your users. There are also ways to capture still images, capture video, play movies, and stream content.

Audio

All iOS devices have headphone jacks and all but the very oldest have speakers as well. The same is true of microphones. The SDK provides ways to capture and play back audio.

The microphones and speakers are also used for accessibility features such as the VoiceOver screen reader. You can build descriptions into your graphical user interface (GUI) elements to enable your applications to take advantage of VoiceOver, so your interfaces can describe themselves to visually impaired end users.

Telephony

It may seem an overly obvious point to make, but the iPhone's telephony system, which handles both phone calls and SMS messaging, can and will interrupt applications when the unit receives an incoming telephone call. Sure, users can suspend out of apps whenever they want on the iPhone, iPad, and iPod touch platforms, but only the iPhone has to deal with the kind of transition that's forced by the system and not a choice by the user.

In addition to phone calls suspending your app, the user is able to open your app while on a call. When that happens, iOS adds a special top bar indicating the status of the call. Make sure to test your interface with the bar open as well as the bar being open then closing. The simulator lets you toggle the in-call status bar on and off.

Consider how the different kinds of interruptions might affect your application. It’s important to keep all kinds of possible exits in mind when designing software. Be aware that the choice to leave your app might not always come from the user, especially on the iPhone. Applications that use audio need to take special care to restore the correct state after phone call interruptions.
Another fallout of telephony operations is that more processes end up running in the background on iPhones than on iPod touches and iPads, even those iPads that provide cellular data support. These processes do reduce the amount of free memory, though for modern devices, the effect is minimal. Having said that, it still pays to test your app on cellular-enabled devices.

**Core Location and Core Motion Differences**

Core Location depends on three different approaches, each of which might or might not be available on a given platform. These approaches are limited by each device’s onboard capabilities. Wi-Fi location, which scans for local routers and uses their MAC addresses to search a central position database, is freely available on all iPhone, iPad, and iPod touch platforms.

Cell location, however, depends on an antenna that is available on the iPhone and on suitably equipped iPad models. This technology triangulates from local cell towers, whose positions are well defined from their installations by telephone companies.

The final and most accurate strategy, GPS location, depends on appropriate hardware. Most modern iPhones and iPads come with the hardware, though as of the writing of this book, no iPod touches do. You can use built-in calls to check for the presence of the hardware.

The third-generation iPhone 3GS introduced a built-in compass (via a magnetometer) along with the Core Location APIs to support it. The iPhone 4 and iPad 2 added a three-axis gyro, which provides pitch, roll, and yaw feedback, all of which can be solicited via the Core Motion framework. Most modern iPhone and iPad devices have both the compass and gyro. Modern iPod touches have only the gyro as of the writing of this book.

**Vibration Support and Proximity**

Vibration, which adds tactile feedback to many games, is limited to iPhones. The iPad and iPod touch do not offer vibration support. Nor do they include the proximity sensor that blanks the screen when holding an iPhone against your ear during calls. The `UIDevice` class offers direct access to the current state of the proximity sensor.

**Processor Speeds**

All modern devices come with fast Apple-designed ARM processors. The CPU includes a good amount of fast access RAM for code execution. To save power, some devices run the CPU at slower speeds (underclocked), and all have the ability to suspend parts of the hardware. Some earlier devices had relatively slow processors and much less execution space though they make up an ever-decreasing part of the market. Targeting iOS 6 or later will avoid those early devices.

The important thing is to run your app on a representative sample of the kinds of devices you are targeting. Make sure it performs well on the devices your customers will use. This is especially important if you plan to support iPhones prior to the 4 as well as first-generation iPads.

If your application isn’t responsive enough on the older platforms, consider working up your code efficiency. There is no option in the App Store at this time that lets you omit earlier
generation iPhone devices from your distribution base, although setting your minimal required iOS version to 6.0 or higher will automatically exclude most older devices.

There are a few places you can look for an idea of the market share for each version of iOS. When a new version is released, check the Apple-oriented press, such as the following sites:

- MacOSRumors: www.macrumors.com
- MacWorld: www.macworld.com
- TUAW: www.tuaw.com

You can also check with data analysis and mobile information companies, though you might have to dig to find the information:

- Canalys: www.canalys.com
- Chitika: chitika.com
- Flurry: www.flurry.com/index.html
- Gartner: www.gartner.com/technology/home.jsp
- IDC: www.idc.com

Finally, app developer David Smith regularly updates what OS versions are used in his app:

- http://david-smith.org/iosversionstats/

**OpenGL ES**

OpenGL ES offers a royalty-free cross-platform API for 2D- and 3D-graphics development. It is provided as part of the iOS SDK. Most devices support OpenGL ES 2.0 with the newest support version 3.0. Some very early units supported only OpenGL ES 1.1, but you are unlikely to encounter them.

**Note**

Devices and features remain a moving target. Apple continues to introduce new units and make changes to iOS. As new devices are introduced, check Apple’s information pages, especially the technical specs. For iOS, make sure you read the release notes. In addition, you can look for summary pages on the Internet. One good source is Wikipedia: http://en.wikipedia.org/wiki/List_of_iOS_devices.

**iOS**

One obvious difference is the version of iOS running on any given device. iOS device users are quick to upgrade to new releases. It took comparatively little time for most devices to upgrade from iOS 3 to 4, then 4 to 5, and 5 to 6. Although there are some models that cannot upgrade to iOS 7, they make up a rapidly shrinking percentage of the total number of units.
There are definitely differences in functionality between various versions of the OS. For example, in addition to the new look, iOS 7 introduces UI Motion, UI Dynamics, and Text Kit. All three offer ways to increase engagement with your user. Usually it is a decision of supporting the current version plus the one before—in this case, iOS 6 and 7. It is fairly easy to test for the availability of features and enable or disable access in your app. The largest difference is the user experience, though it is fairly easy to create interfaces that work on both 6 and 7 if you use the built-in UI elements.

Ultimately, what you support should depend on what your potential customers are using. If they are all using devices with iOS 7, there is no need to support 6. This book focuses on iOS 7, though with the exception of some specific features, everything will work in iOS 6. In addition, using auto layout, covered in Chapter 4, makes adapting your interfaces to each iOS much easier.

**Note**

Apple expanded the iOS version of Objective-C 2.0 starting with the 4.0 SDK to introduce blocks. Blocks are a technology that have been around for decades in languages such as Scheme, Lisp, Ruby, and Python. Blocks allow you to encapsulate behavior as objects, so you can pass that behavior along to methods as an alternative to using callbacks. This new feature is introduced in Chapter 13, “Introducing Blocks.”

Other features, such as literals, better property declarations, and fast enumeration, make Objective-C even more powerful. You work with all these features as you progress through the book.

**Summary**

In this chapter, you have taken the first steps in learning to create applications for iOS. You have downloaded and set up Xcode and covered some of the basics of the devices and Apple’s developer program. Through the rest of the book, you continue your journey into the world of creating iOS apps. Each chapter focuses on important skills for some area of development. Though the territory might be unfamiliar, the book provides a focused map to guide you through.

In the next few chapters, you learn the Objective-C language and create your first application in Xcode. From there, you continue to expand your knowledge of iOS development, including user interface elements, adapting to screen size and language, performance tuning, debugging, and how to ship your app. When you are ready, turn the page to start writing your first iOS app.
This page intentionally left blank
Symbols & Numerics

#pragma mark, 325
@ symbol, 36
_ (underscore character), 36
{} curly braces, 49
3.5-inch screens, previewing constraints, 144-145

A

about scene (CarValet project), creating, 263-264
about view, adding to universal CarValet app, 383-385
  menu images, polishing, 385-387
abstracting out code, 84
accelerometer-simulator project (Google Code), 9
accessors, 49-50
  dot notation, 53
Accounts pane (Xcode), 495-497
action selectors, implementing, 434-435
actions, 72-73, 484-485
  adding to add/view scene (CarValet project), 74-77
  IBAction identifier, adding to view controller, 112-115
ad hoc testing providers, 510
adapting cars table for iPad, 401-404
adding
  cars to table (CarValet project), 287-288
  color themes to navigation controllers, 264-267
  German locale to CarValet project, 203-206
icons to asset catalog, 257-259
index to table views, 355-358
recognizers to DetailController, 446
references to constraints, 163-166
scroll view to edit scene (CarValet project), 231-234
search capability to tables, 361-369
sections, 347-349
add/view scene (CarValet project)
behaviors, adding, 72-77
actions, 72-77
outlets, 72-73
buttons, localizing, 195-197
car display behaviors, adding, 82-85
localization, 191-199
navigation controllers, 257
new cars, adding, 81-82
replacing with table view controller-based scene, 283-285
scroll view, adding, 227-230
toolbar
adding, 259-261
localization, 261-263
visual elements, adding
dividers, 71-72
labels, 68-70
allocating memory, 77-80
for objects, 38-39
animation, pulsing, 456-460
app ID, generating, 499
app listing
category, selecting, 516
creating, 513-520
description, adding, 517-518
details, adding, 515-520
EULA, 519
-saving, 519-520
screenshots, 519
App Store, uploading apps to, 521-526
configuring the project, 521-522
setting up the project scheme, 522-523
Appearance protocol, 267
Apple iOS Developer Programs, 3
Apple Online Developer Program, 3
apps
bug reporting, 506-507
Core Data, preparing for use in, 323-325
designing, holistic goals, 66
launching in Instruments, 470
testing, 7-15, 479
tethering, 10
uploading to App Store, 521-526
Arabic internationalization, 215-223
Arabic strings, adding, 215-219
dates, formatting, 219-222
numbers, 219-222
text alignment, 222-223
ARC (Automatic Reference Counting), 31
arrows, using in toolbars, 263
asset catalog, adding icons, 257-259, 504-505
assigning blocks, 455-456
assistant editor preview mode, 145-148
atomicity of variables, 57-58
attaching recognizers to a view, 442
attributes of recognizers, 427
audio, differences among platforms, 16
audio system on iOS Simulator, 9
auto layout, 117-131
   constraints, 120-131
      Assistant editor preview mode, 145-148
   bottom layout guides (IB), 176-178
   changing for orientation, 162-163
   completeness of specification, 133-134
   content compression resistance, 150
   content hugging, 150
   creating, 122-123
   dragging out, 130-131
   intrinsic content size, 134
   invisible container views, 137
   previewing, 144-145
   references, adding, 163-166
   relationships, 120-122
   top layout guides (IB), 176-178
   values, changing, 128-130
   issues popup (IB), 154-155

car display behaviors, adding (CarValet project), 82-85
editor behaviors, adding to edit scene (CarValet project), 94-97
of table views, 277
blocks, 453
   assigning, 455-456
   calling, 454
   declaring, 453-454
   defining, 455-456
   pulsing animation, adding, 456-460
   replacing protocols with, 462-466
   variables, 460-462
      scoped variables, modifying, 462
   writing, 455-460
Boolean type (Objective-C), 34
bottom layout guides (IB), 176-178
bounce scrolling, 227-230
   adding to CarValet project scenes, 230
breakpoints, 483-484
   exception breakpoints, 486
   symbolic breakpoints, 485
bug reporting, 506-507
build number, 526
buttons
   localizing in add/view scene (CarValet project), 195-197
Next Car button, adding to CarValet project, 86-89
Previous Car button, adding to CarValet project, 86-89
text color, changing, 266-267
Xcode
   Editor buttons, 27
   Run button, 26
calling
blocks, 454
functions in Objective-C, 32-35
camel case, 36
camera, differences among platforms, 16
canalys.com, 18
Car class, adding to CarValet project, 42-44
car detail controller, iPad-specific, 407-424
closing, 419-420
disabling car editing, 420-421
layout, 409-414
loading cars, 416-417
polishing, 418-419
popover behavior modifications, 421-424
preparing the picker, 414-416
saving cars, 417-418
car display behaviors, adding (CarValet project), 82-85
car image scene (CarValet project), scrolling, 240-249
paging, 243-244
rotation, handling, 248-249
scroll view, populating, 241-243
updating label with index of current car image, 249-250
zoom, adding, 245-248
car view cell, adding to table view controller (CarValet project), 285-287
cars table
adapting for iPad, 401-404
adding cars to, 287-288
converting for Core Data, 326-332
accessing data with managed property context, 327-328
adding and deleting cars, 328-329
adding managed property context, 326
switching to CIDCar object class, 330
index, displaying year in, 357-358
removing cars from, 288-291
searching, 361-369
details for found car, displaying, 365-367
predicate, adding to fetched results controller, 362-365
updating, 306-307
user-initiated editing, 289-291
CarValet project. See also universal CarValet app, creating
about scene, 263-264
accessors, 49-50
add/view scene
behaviors, adding, 72-77
buttons, localizing, 195-197
navigation controllers, 257
new cars, adding, 81-82
replacing with table view controller-based scene, 283-285
scroll view, adding, 227-230
visual elements, adding, 67-72
Arabic internationalization, 215-223
Arabic strings, adding, 215-219
dates, formatting, 219-222
numbers, 219-222
text alignment, 222-223
base initializer, 49
Car class, adding, 42-44
cars
  creating, 53-54
  removing from table, 288-291
cars table
  adding cars to, 287-288
  displaying year in index, 357-358
  updating, 306-307
  user-initiated editing, 289-291
constraints
  adjusting for screen height, 155
  designing, 134-138
  implementing, 141-144
New Car button, 151
top-level view constraints, 138-141
  for Total Cars label, 151
  view car area, 151-154
Core Data, adding CDCar model, 321-324
creating, 41-42
disclosure indicator, adding to car data cell, 291-292
edit scene, 89-106
  resizing scroll view for keyboard, 234-240
  scroll view, adding, 227-230
German internationalization, 203-215
  formatting numbers, 213-215
  German locale, adding, 203-206
  label constraints, changing, 209-213
  Localizable.strings, updating, 207-209
header file, 44-45
HybridCar class, 58-62
  implementation file, 59-61
implementation file, 46-50
landscape orientation, 156-180
  constraints, adding, 169-172
  constraints, creating, 167-169
  constraints, designing, 158-159
  top-level view constraints, 159-162
localization, 189-202
  add/view scene, 191-199
  strings, 189-191
make and model edit scene
  creating, 296-307
  delegate, preparing, 300-303
  transitions, 305-306
  ViewCarProtocol, adding, 303-305
model year edit scene, 307-314
  picker, implementing, 309-312
  year edit protocol, adding, 312-314
  year editor, setting up, 308-309
Next Car button, adding, 86-89
properties, 50-53
  encapsulation, 51
  qualifiers, 55-56
tab bar
  adding, 270
  car images, moving to, 271-272
  dynamically updating items, 272-273
table view controller, adding car view cell, 285-287
view car scene
  creating, 292-294
  populating, 294-296
  swipes, enabling support for, 428-438
categories, 324  
cautions icon (IB), 149  
CDCar model, adding to CarValet project, 321-324  
cells  
car view cell, adding to table view controller (CarValet project), 285-287  
creating, 279-281  
custom cells, 341-345  
  populating, 345  
  visual elements, adding, 343-344  
deletions, 289  
index paths, 282  
certificates, 494  
changing  
constraints  
  rotation, handling, 162-163, 172-176  
  values, 128-130  
device language, 206-207  
table views, groups, 349-355  
Chisnall, David, 419  
chitika.com, 18  
Clair, Robert, 31  
class clusters, 40  
classes, 31, 35-41  
  #pragma mark, 325  
camel case, 36  
categories, 324  
forward references, 95  
implementation file, 35  
  defining, 37-38  
inheritance, 39-40, 59-62  
navigation controller classes, 256-257  
NSCalendar, 189  
NSDate, 189  
NSDateComponents, 189  
NSDateFormatter, 188  
NSLayoutConstraint, 120  
NSNumberFormatter, 188  
NSTimeZone, 189  
objects, creating, 38-39  
prefixes, 24  
reducing dependencies between, 332  
singletons, 387  
superclasses  
  initializing, 40  
  responding to open and closed keyboard, 237-239  
UIGestureRecognizer, 427  
ViewController class, modifying, 110-112  
closed keyboard, responding to, 237-239  
Cocoa Touch, 7  
color themes, adding to navigation controllers, 264-267  
comparing  
iOS device platforms  
  audio, 16  
  camera, 16  
  Core Location, 17  
  Core Motion, 17  
  OpenGL ES, 18  
  processor speeds, 17-18  
  screen size, 15-16  
  telephony, 16-17  
  vibration support, 17
landscape and portrait orientation constraints, 161
VCL and full specification, 168

Conferences, 528-529

Configuring form views, 233-234

Constants, gesture state constants, 440-441

Constraints, 120-131. See also VCL (Visual Constraint Language)

Assistant editor preview mode, 145-148
bottom layout guides (IB), 176-178
completeness of specification, 133-134
content compression resistance, 150
content hugging, 150
creating, 122
in IB, 122-123
designing for CarValet project
adjusting for screen height, 155
edit scene, 155-156
landscape orientation, 156-180
New Car button, 151
portrait orientation, 134-138
top-level view constraints, 138-141, 159-162
Total Cars label, 151
view car area, 151-154
dragging out, 130-131
generating from strings, 170-172
intrinsic content size, 134
invisible container views, 137
landscape orientation
comparing with portrait orientation, 161
creating, 167-169
top-level view constraints, 159-162
troubleshooting, 178-180
portait orientation, 131-132
previewing, 144-145
references, adding, 163-166
relationships, 120-122
for scroll views, 235
top layout guides (IB), 176-178
troubleshooting, 149-150
auto layout issues popup, 154-155
values, changing, 128-130
containers, invisible container views, 137
content compression resistance, 150
content hugging, 150
content views, 233-232
Continuous Flow state (recognizers), 439
controller layer (MVC), 318
converting
cars table for Core Data, 326-332
accessing data with managed property context, 327-328
adding and deleting cars, 328-329
adding managed property context, 326
switching to CDCar object class, 330
between coordinate spaces, 236-237
coordinate spaces, converting between, 236-237

Core Data, 317-320

CDCar model, adding to CarValet project, 321-324
classes, 319
converting cars table for use, 326-332
accessing data with managed property context, 327-328
adding and deleting cars, 328-329
adding managed property context, 326
switching to CDCar object class, 330
entities, 319
fetched results controller, 332-339
NSFetchedResultsController, integrating, 333-335
NSFetchedResultsController-Delegate, implementing, 336-339
initializing for use, 323-325
managed objects, 319-320
stores, 319
Core Location, differences among platforms, 17
Core Motion, differences among platforms, 17
CoreData framework, adding to projects, 320-321
count-based labels, 87-88
Cox, Brad J., 31
creating
app listing, 513-520
details, adding, 515-520
cars for CarValet project, 53-54
cells, 279-281
constraints, 122
in IB, 122-123
landscape orientation, 167-169
distribution provisioning profile, 501-503
format strings, 88-89
Hello World project, 21-25
objects, 38-39
outlets, 73-74
return gesture recognizer, 442-447
table views
cells, 279-281
sections, 281-283
universal CarValet app, 374-382
about view, adding, 382-387
app section navigation, adding, 379-382
car images view controller, adding, 397-400
Cars tab, adding, 400-424
menu, accessing in portrait, 387-396
split view controller, adding, 376-379
custom cells, 341-345
populating, 345
visual elements, adding, 343-344
custom getters, 56-58
custom recognizers, 441-442
return gesture recognizer
creating, 442-447
return gesture recognizer, creating, 442-447
custom setters, 56-58
data access, designing for mobile apps, 11-12
date formats, 187-189
david-smith.org, 18
debugger (iOS SDK), 7, 479-486
actions, 484-485
breakpoints, 483-484
EXC_BAD_ACCESS errors, troubleshooting, 489-491
gauges, 481-482
process view, 480
variables view, 480-481
dynamically updating tab bar items, 272-273
DetailController
adding recognizers to, 446
implementing in universal CarValet project, 388-396
consolidating code, 393-396
singleton, setting up, 391-393
UISplitViewControllerDelegate, adding, 389-391
Developer Enterprise Program, 4-5
Developer University Program, 5
development certificate, generating, 495-497
devices
adding to provisioning profile, 497-498
language, changing, 206-207
dimensions, resizing. See auto layout
disabling
car editing, 420-421
recognizers, 438
disclosure indicator, adding to car data cell, 291-292
displaying
section headers, 346-347
year in cars table index, 357-358
distribution provisioning profile, 497-498
creating, 501-503
dividers, adding to add/view scene (CarValet project), 71-72
documentation, iOS Human Interface Guidelines, 494-495
dot notation, 51-53
double strings, faking localization with, 193-195
drag gesture recognizers, 448-450
dragging out constraints, 130-131
dynamically updating tab bar items, 272-273
debugging strokes, 447
declaring
blocks, 453-454
methods, 36-37
properties, 50
defaults, 79
defining
blocks, 455-456
implementation file, 37-38
delegates, 46
make and model edit scene (CarValet project), preparing for, 300-303
protocols, 107-112
deleting
cell data, 289
sections, 347-349
description, adding to app listing, 517-518
designing for mobile apps
application considerations, 13-14
constraints, 132-133
data access considerations, 11-12
energy considerations, 13
holistic goals, 66
interaction considerations, 12-13
memory considerations, 12
storage considerations, 11
use behavior considerations, 14-15
detail view controller, 372
edit scene (CarValet project), 89-106
   constraints, 155-156
   editor behaviors, adding, 94-97
   localization, 200-202
   scroll view
      adding, 230-240
      resizing for keyboard, 234-240
   scroll view, adding, 227-234
   visual elements, adding, 91
editor behaviors, adding to edit scene
   (CarValet project), 94-97
Editor buttons (Xcode), 27
   enabling support for swipe gestures,
      428-438
   protocols, 107-108
   entities (Core Data), 319
   enumerated types, 303
   EULA (end user license agreement), 519
   event aggregators, 529
EXC_BAD_ACCESS errors, troubleshooting,
   486-491
   with debugger, 489-491
   with Zombies Instrument template,
      486-489
exception breakpoints, 486
exchanging data with protocols, 108-112
fetched results controller, 332-339
   NSFetchedResultsController, integrating, 333-335
   NSFetchedResultsControllerDelegate,
      implementing, 336-339
   predicate, adding, 362-365
   section headers, displaying, 346-347
   finding icons, 259
flurry.com, 18
   form view, configuring for scroll view,
      233-234
format strings, 88-89
   forward references, 95
   frames, 235-239
   functional testing, 509-510
   functions, calling in Objective-C, 32-35
gartner.com, 18
gathering metrics, 508-509
gauges (debugger), 481-482
   generating
      app ID, 499
      constraints from strings, 170-172
      development certificate, 495-497
German internationalization, 203-215
   formatting numbers, 213-215
   German locale, adding, 203-206
   label constraints, changing, 209-213
   Localizable.strings, updating, 207-209
   gestures, 427
      dragging, 448-450
      iPhone action selectors, enabling,
         434-435
faking localization with double strings,
   193-195
recognizers
  adding to DetailController, 446
  attaching to a view, 442
  attributes, 427
  custom recognizers, 441-442
  disabling, 438
  responding to, 446-447
  states, 439-441
  strokes, debugging, 447
  swipes, enabling support for, 428-438
  target/action pairs, 428
getter methods, 36, 56-58
Google Code, accelerometer-simulator project, 9
groups, 345
  changing, 349-355

header files, 35
  CarValet project, 44-45
Hello World project
  creating, 21-25
  labeling, 28-30
hierarchies of content, 254-255
  leaf nodes, 254
  root scene, 255
 holistic goals for app design, 66
hooking scenes together, 98-105
  prepareForSegue:sender method, 103-105
  transitions, 102-103
HybridCar class
  creating for CarValet project, 58-62
  implementation file, 59-61

IB (Interface Builder), 6
  bottom layout guides, 176-178
  constraints
    creating, 122-123
    troubleshooting, 149-150
  Size inspector, 150-151
  toolbar
    auto layout issues popup, 154-155
    constraints, adding, 126-127
    pin popup, 127-128
    top layout guides, 176-178
IBAction identifier, 72-73
  adding to view controller, 112-115
IBOutlet identifier, 72-73
ibtool, localizing storyboard strings with, 198
icons
  adding to asset catalog, 257-259, 504-505
  caution icon (IB), 149
  magnifying glass, adding to indexes, 367-369
  sizes, 503-504
  sources of, 259
idc.com, 18
identifying constraint issues, 149-150
implementation file, 35
  CarValet project, 46-50
  defining, 37-38
  dot notation, 52-53
  encapsulation, 51
  for HybridCar class, 59-61
implementing
   constraints for CarValet project, 141-144
   picker for model year edit scene (CarValet project), 309-312
index paths, 282
indexes
   cars table, displaying year in, 357-358
   magnifying glass, adding, 367-369
   properties, 356
inheritance, 39-40, 59-62
initializing
   Core Data, 323-325
   superclasses, 40
installing Xcode, 1-2
instance variables, properties, 50-53
Instruments, 469-479
   EXC_BAD_ACCESS errors, troubleshooting with Zombies template, 486-489
   fixing problems with, 476-478
   launching apps in, 470
   prefetching, 476-477
   problem isolation, 474-476
   templates, 471
      Time Profiler, 472-478
   tree mining, 475-476
integrating NSFetchedResultsController, 333-335
integration testing, 510-512
interaction, designing for mobile apps, 12-13
interface, Xcode, 25-27
   Editor buttons, 27
   Navigator, 27
   Run button, 26
   status area, 26
   utilities area, 27
internationalization, 189
   Arabic internationalization, 215-223
      Arabic strings, adding, 215-219
      dates, formatting, 219-221
      numbers, 219-222
      text alignment, 222-223
   German internationalization, 203-215
      label constraints, changing, 209-213
      localizable.strings, updating, 207-209
intrinsic content size, 134
invisible container views, 137
iOS Developer Programs, 3
   Developer Enterprise Program, 4-5
   Developer University Program, 5
   registration, 5
   standard iOS Developer Program, 4
iOS Human Interface Guidelines, 7, 494-495
iOS SDK, 2-7
iOS Simulator, 6
   limitations of, 8-10
      audio system, 9
      keychain security, 9
      localization, 9
      testing apps, 7-15
      Video Out emulation, 10
iPad
   returning to default state with custom recognizer, 442-447
universal CarValet app, creating, 374-382
about view, adding, 382-387
app section navigation, adding, 379-382
car images view controller, adding, 397-400
Cars tab, adding, 400-424
menu, accessing in portrait, 387-396
split view controller, adding, 376-379
iPhone action selectors, enabling, 434-435
iSimulate, 9
ISO 639.2 standard, 185
isolating problems with Instruments, 474-476

landscape orientation
CarValet project, 156-180
constraints, designing, 158-159
constraints
adding to CarValet project, 169-172
comparing with portrait orientation, 161
creating, 167-169
troubleshooting, 178-180
VCL, 166-168
language support. See also localization
launch images, 505-506
launching apps in Instruments, 470
leaf nodes in hierarchies of content, 254
limitations of iOS Simulator, 8-10
audio system, 9
keychain security, 9

literals, 31
localization. See also internationalization
Base, 184
CarValet project, 189-202
add/view scene, 191-199
edit scene, 200-202
strings, 189-191
faking with double strings, 193-195
formats, 187-189
on iOS Simulator, 9
ISO 639.2 standard, 185
language preferences, setting, 183-184
redirection, 184-187
string tables, 186-187
toolbars, 261-263
version control, 192
via formats, 187-189

labeling
add/view scene (CarValet project), 68-70
count-based labels, 87-88
Xcode projects, 28-30

Jobs, Steve, 31
keyboard
handling for scroll views, 234-240
support on iOS devices, 12
keywords, @ symbol, 36
KVC (Key Value Coding), 419

ISO 639.2 standard, 185
isolating problems with Instruments, 474-476
Macintosh Developer Program, 4
macosrumors.com, 18
macros, localed string macros, 202
macworld.com, 18
magnifying glass, adding to indexes, 367-369
MainMenuViewController, creating for universal CarValet app, 383-385
make and model edit scene (CarValet project)
    creating, 296-307
delegate, preparing, 300-303
MakeModelEditViewController class, 297-300
transitions, 305-306
ViewCarProtocol, adding, 303-305
MakeModelEditViewController class, 297-300
managed objects, 319-320
managing memory, 77-80
marketing, 511-513
    interest, 513
    open-source projects, 511
master view controller, 372
memory
    allocating for objects, 38-39
    leaks, 79-80
    managing, 77-80
    mobile apps, designing for, 12
message-based navigation, 263-264
messages
    protocols, 46
    sending, 47
    sending with Objective-C, 32-35
methods, 33
    accessor methods, 49-50
    dot notation, 53
camel case, 36
    declaring, 36-37
    inheritance, 39-40
    parameters, 34-35, 45
    prepareForSegue:sender method, 103-105
metrics gathering, 508-509
mobile apps, designing for
    application considerations, 13-14
    constraints, 132-133
    data access considerations, 11-12
    energy considerations, 13
    interaction considerations, 12-13
    memory considerations, 12
    storage considerations, 11
    use behavior considerations, 14-15
modal presentation, 275
model layer (MVC), 318
model year edit scene (CarValet project), 307-314
    picker, implementing, 309-312
    year edit protocol, adding, 312-314
    year editor, setting up, 308-309
modifying
    scoped variables of blocks, 462
    ViewController class, 110-112
moving car images to tab bar, 271-272
multiple string tables for localization, 202
multitasking, 13-14
MVC (Model-View-Controller), 318
    controller layer, 318
    model layer, 318
    view layer, 318
navigation controllers, 253-267
  color themes, adding, 264-267
dynamically updating items, 272-273
hierarchies of content, 254-255
  leaf nodes, 254
message-based navigation, 263-264
tab bar controller, 267-273
  adding to CarValet project, 270
car images, moving to, 271-272
toolbars, 257-263
  arrows, 263
  populating, 259-261
UINavigationController, 256
Navigator (Xcode), 27
NDA (non-disclosure agreement), 8
new cars, adding to table (CarValet project), 287-288
NeXT, 31
Next Car button, adding to CarValet project, 86-89
nonatomic properties, 57-58
nonretina icon sizes, 503-504
notifications, 14
  responding to open and closed keyboard, 237-239
NSCalendar class, 189
NSDate class, 189
NSDateComponents class, 189
NSDateFormatter class, 188
NSFetchedResultsController, 332-339
  integrating, 333-335
NSFetchedResultsControllerDelegate, implementing, 336-339
NSIndexPath object, 282
NSLayoutConstraint class, 120
NSManagedObject class, 319
NSManagedObjectModel class, 319
NSNumberFormatter class, 188
NSPersistentStoreCoordinator class, 319
NSTimeZone class, 189
numbers
  Arabic internationalization, 219-222
  formats, 187-189
  German internationalization, 213-215
Objective-C, 30-41
  @ symbol, 36
  Boolean type, 34
camel case, 36
classes, 35-41
  implementation file, 35
  functions, calling, 32-35
  implementation file, defining, 37-38
  messages, sending, 32-35
  methods, 33
  declaring, 36-37
  inheritance, 39-40
  parameters, 34-35
objects, 35-41
  pointing to, 40-41
selector, 47
underscore character (_), 36
object-oriented programming, 31
objects, 31, 35-41
  creating, 38-39
defaults, 79
delegates, 46
dot notation, 51-53
managed objects, 319-320
memory, allocating, 38-39
NSIndexPath, 282
pointing to, 40-41
targets, 428
observers, responding to open and closed keyboard, 237-239
open keyboard, responding to, 237-239
OpenGL ES, differences among platforms, 18
opening XIB files, 263-264
orientation. See also portrait orientation, constraints; landscape orientation
constraints, changing to handle rotation, 162-163
rotation
constraints, changing to handle rotation, 172-176
scroll views, handling, 248-249
outlets, 72-73
creating, 73-74
entities, 319
fetched results controller, 332-339
managed objects, 319-320
stores, 319
pickers, 308
implementing for model year edit scene, 309-312
pin popup (IB), 127-128
pixels, 120
platforms (iOS devices)
differences in
audio, 16
camera, 16
Core Location, 17
Core Motion, 17
OpenGL ES, 18
processor speeds, 17-18
screen size, 15-16
telephony, 16-17
vibration support, 17
versions of iOS, 18-19
Plausible Crash Reporter project, 506
pointing to classes, 40-41
points, 120
populating
custom cells, 345
scroll views, 241-243
toolbars, 259-261
view car scene (CarValet project), 294-296
popups (IB)
auto layout issues popup, 154-155
creating constraints, 126-128
portrait orientation, constraints, 131-132
   designing for CarValet project, 134-141
   implementing for CarValet project, 141-144
   previewing, 144-145
postlaunch activities, 525
predefined cells, creating (TableTry project), 279
predicate, adding to fetched results controller, 362-365
prefetching, 476-477
prefixes, 24
prelaunch activities
   bug reporting, 506-507
   marketing, 511-513
      interest, 513
      open-source projects, 511
   metrics gathering, 508-509
   QA testing, 509-512
      functional testing, 509-510
      integration testing, 510-512
      unit testing, 509-510
prepareForSegue:sender method, 103-105
preparing
   Core Data for apps, 323-325
   make and model edit scene (CarValet project) for delegate, 300-303
previewing constraints, 144-145
Previous Car button, adding to CarValet project, 86-89
printing cars for CarValet project, 53-54
privacy laws, 509
private keys, 494
process view (debugger), 480
processor speeds, differences among platforms, 17-18
Project Editor, 6
projects
   CarValet. See CarValet project
   localization, 184-185
   TableTry, 277-283
   universal projects, creating, 374-382
   Xcode
      creating, 21-25
      labeling, 28-30
      options, 23-25
properties, 50-53
   declaring, 50
   dot notation, 51-53
   encapsulation, 51
   qualifiers, 55-56
   read-only, 55-56
   setters, 56-58
   of table indexes, 356
variables
   atomicity, 57-58
   temporary, 78
protocols, 46, 107-112
   Appearance protocol, 267
   encapsulation, 107-108
   exchanging data with, 108-112
   replacing with blocks, 462-466
   reusability, 107
   ViewController class, modifying, 110-112
   year edit protocol, adding to model year edit scene, 312-314
provisioning profile, adding devices to, 497-498
public keys, 494
pulsing animation, adding, 456-460
Push notifications, 14

Q
QA (quality assurance) testing, 509-512
integration testing, 510-512
unit testing, 509-510
qualifiers, 55-56
memory-related, 78

R
read-only properties, 55-56
recognizers, 427-428
adding to DetailController, 446
attaching to a view, 442
attributes, 427
custom recognizers, 441-442
return gesture recognizer, creating, 442-447
disabling, 438
drag gesture recognizers, 448-450
responding to, 446-447
states, 439-441
redirection
Base localization, 184
ISO 639.2 standard, 185
project-level localization, 184-185
string tables, 186-187
reducing dependencies between classes, 332
references, adding to changing constraints, 163-166
registering for iOS Developer Programs, 5
removing
cars from table (CarValet project), 288-291
sections, 347-349
replacing
add/view scene (CarValet project) with table view controller-based scene, 283-285
protocols with blocks, 462-466
reporting bugs, 506-507
resizing
dimensions. See auto layout
keyboard for scroll views, 234-240
scroll views, 239-240
resources
books, 525-527
conferences, 528-529
social media, 529-530
websites, 527-528
responding
to open and closed keyboard, 237-239
to recognizers, 446-447
retain cycles, 79-80
retina, 120
icon sizes, 503-504
return gesture recognizer
creating, 442-447
return gesture recognizer, creating, 442-447
reusability of protocols, 107
right-to-left languages, internationalization, 215-223
Roadley, Tim, 318
root views, 255
tabs, 268-273
rotation
  constraints, changing for orientation, 162-163, 172-176
  scroll views, handling, 248-249
rows, creating, 281-282
Run button (Xcode), 26
Sadun, Erica, 456
saving app listing, 519-520
scenes, 66-67
  about scene (CarValet project), 263-264
  add/view scene (CarValet project)
    behaviors, adding, 72-77
    buttons, localizing, 195-197
  car display behaviors, adding, 82-85
  dividers, 71-72
  labels, 68-70
  localization, 191-199
  new cars, adding, 81-82
  replacing with table view controller-based scene, 283-285
  scroll view, adding, 227-230
  toolbar, adding, 259-261
  car image scene (CarValet project), scrolling, 240-249
edit scene (CarValet project), 89-106
  constraints, 155-156
  localization, 200-202
  scroll view, adding, 227-234
hooking together, 98-105
  prepareForSegue:sender method, 103-105
  transitions, 102-103
make and model edit scene (CarValet project)
  creating, 296-307
  delegate, preparing, 300-303
  transitions, 305-306
  ViewCarProtocol, adding, 303-305
model year edit scene (CarValet project), 307-314
  picker, implementing, 309-312
  year edit protocol, adding, 312-314
  year editor, setting up, 308-309
view car scene (CarValet project)
  creating, 292-294
  populating, 294-296
scoped variables, modifying, 462
screen size
  auto layout, 117-131. See also auto layout
  differences among platforms, 15-16
  intrinsic content size, 134
  standard distance, 119
  toggling (view controllers), 145
screens, 65
screenshots, providing for app listing, 519
scroll views
  adding
    to add/view scene (CarValet project), 227-230
    to edit scene (CarValet project), 227-234
constraints, 235
form view, configuring, 233-234
paging, 240-241, 243-244
populating, 241-243
resizing, 239-240
  for keyboard, 234-240
rotation, handling, 248-249
zooming content, 226, 245-248
scrolling, bounce scrolling, 227-230
SDKs
iOS Developer Programs, 3
  Developer Enterprise Program, 4-5
  Developer University Program, 5
Macintosh Developer Program, 4
registration, 5
standard iOS Developer Program, 4
iOS SDK, tools, 6-7
searching table views, 358-369
  index, adding, 367-369
predicate, adding to fetched results
controller, 362-365
sections, 346-349
  adding and deleting, 347-349
creating, 282-283
  in tables, 281-282
groups, changing, 349-355
index paths, 282
segues, 103-106
selecting category for your app, 516
selectors, 47
  iPhone action selectors, enabling,
  434-435
  responding to open and closed key-
  board, 237-239
self-configuring content views, 233-232
sending messages, 47
  with Objective-C, 32-35
setter methods, 36, 56-58
setting language preferences, 183-184
simulator, changing device language,
  206-207
singletons, 387
  DetailController singleton, implement-
  ing, 388-396
Size Inspector, 150-151
sizes of icons, 503-504
Smalltalk, 31
Smith, David, 18
sorting, 345
sources of icons, 259
split view controller, 372-374
  detail view controller, 374
  master view controller, 373
universal CarValet app, creating
  car detail controller, 407-424
  car images view controller, adding,
  397-400
  DetailController singleton, imple-
  menting, 388-396
standard distance, 119
standard iOS Developer Program, 4, 59-61
status area (Xcode), 26
storage
  Core Data, 317-320
  CDCar model, adding to CarValet
  project, 321-324
classes, 319
  converting cars table for use,
  326-332
entities, 319
fetched results controller, 332-339
managed objects, 319-320
mobile apps, designing for, 11
stores (Core Data), 319
storyboards, 65-67
auto layout, 124-126
scenes, 66-67
edit scene (CarValet project), 89-106
hooking together, 98-105
transitions, 102-103
screens, 65
strings, localizing with ibtool, 198
string tables
localization, 186-187
multiple string tables, 202
strings
Arabic strings, adding for internationalization, 215-219
format strings, 88-89
generating constraints from, 170-172
localization
CarValet project, 189-191
faking localization with double strings, 193-195
storyboard strings, localizing with ibtool, 198
VCL, 166-167
strokes, debugging, 447
strong qualifier, 78
subclasses, inheritance, 39-40, 59-62
subscripting, 31

superclasses, 40
initializing, 40
responding to open and closed keyboard, 237-239
swipes, 428-438
iPhone action selectors, enabling, 434-435
recognizers
custom recognizers, 441-442
disabling, 438
states, 439-441
symbolic breakpoints, 485
syntax
methods, 36-37
Objective-C methods, 32-35

T

tab bar controller, 267-273
adding to CarValet project, 270
car images, moving to tab bar, 271-272
dynamically updating items, 272-273
table view controllers, adding car view cell, 285-287
table views, 275-277, 282-283
behaviors, 277
CarValet project
new cars, adding, 287-288
removing cars from, 288-291
user-initiated editing, 289-291
cells
creating, 279-281
deleting, 289
custom cells, 341-345
populating, 345
visual elements, adding, 343-344
groups, 345
changing, 349-355
index paths, 282
indexes
adding, 355-358
properties, 356
rows, creating, 281-282
searching, 358-369
  index, adding, 367-369
  predicate, adding to fetched results
  controller, 362-365
sections, 346-349
  adding and deleting, 347-349
  creating, 281-283
sorting, 345
updating, 306-307
**TableTry project, 277-283**
cells, creating, 279-281
index paths, 282
predefined cells, creating, 279
rows, creating, 281-282
sections, creating, 281-283
**target audience for iOS Developer Programs, 3**
targets, 428
telephony, differences among platforms, 16-17
templates (Instruments), Time Profiler, 472-478
temporary variables, 78
ternary operators, 83

**testing**
apps, 479
tethering, 10
apps with iOS Simulator, 7-15
functional testing, 509-510
unit testing, 509-510
tethering, 10
text alignment, Arabic internationalization, 222-223
text color of buttons, changing, 266-267
text fields, adding to edit scene (CarValet project), 91
third-party bug reporting services, 507
Time Profiler template (Instruments), 472-478
  problem isolation, 474-476
toggling between screen sizes (view controllers), 145
**toolbar (IB)**
  auto layout issues popup, 154-155
  constraints, adding, 126-127
  pin popup, 127-128
**toolbars, 257-263**
  arrows, 263
  color themes, 264-267
  localization, 261-263
  populating, 259-261
**top layout guides (IB), 176-178**
top-level view constraints (CarValet project)
  landscape orientation, 159-162
  portrait orientation, 138-141
touchscreen interaction, designing for mobile apps, 12-13
transitions, 102-103
adding to view cars scene (CarValet project), 305-306
tree mining, 475-476
troubleshooting
constraints, 149-150
 auto layout issues popup, 154-155
 landscape orientation, 178-180
EXC_BAD_ACCESS errors, 486-491
with debugger, 489-491
with Zombies Instrument template, 486-489
with Instruments, 474-476
search code, 361
tuaw.com, 18

U

UI (user interface), Xcode, 25-27
 Editor buttons, 27
 Navigator, 27
 Run button, 26
 status area, 26
 utilities area, 27
UIBarButton class, 256
UIGestureRecognizer class, 427
UILabel, 82
UINavigationBar class, 256
UINavigationController, 253, 256
 hierarchies of content, 254-255
UINavigationItem class, 256
UIPickerView, 307
UIScrollView, 225-226, 240-250
 paging, 243-244
 zooming, 245-248
UISplitViewController, 371
UITabBar class, 269
UITabBarController, 253, 268-273
UITabBarItem class, 269
UITextField, 91
UIToolBar class, 256
underscore character (_), 36
unit testing, 509-510
universal apps, 374-382
universal CarValet app, creating
 about view, adding, 382-387
 MainMenuViewController, creating, 383-385
 menu images, polishing, 385-387
 app section navigation, adding, 379-382
car detail controller, 404-406
 iPad-specific, 407-424
car images view controller, adding, 397-400
Cars tab, adding, 400-424
car table, adapting to iPad, 401-404
 menu, accessing in portrait, 387-396
DetailController singleton, implementing, 388-396
split view controller, adding, 376-379
updating
form views, 233-234
labels of car image scene, 249-250
Localizable.strings, 207-209
tab bar items, 272-273
table views, 306-307
uploading apps to App Store, 521-526
 configuring the project, 521-522
 setting up the project scheme, 522-523
user-initiated editing (cars table), 289-291

utilities

ibtool, localizing storyboard strings with, 198

Instruments, 469-479

utilities area (Xcode), 27

variables, 31

atomicity, 57-58

block access to, 460-462

properties, 50-53

temporary, 78

variables view (debugger), 480-481

VCL (Visual Constraint Language), 166-168

versus full specification, 168

strings, 166-167

version control, 192

versions of iOS, 18-19

vibration, support for in iOS devices, 17

Video Out emulation (iOS Simulator), 10

videos, WWDC, 5

view car area (CarValet project), constraints, 151-154

view car scene (CarValet project)

creating, 292-294

populating, 294-296

swipes, enabling support for, 428-438

view controllers

IBAction identifier, adding, 112-115

responding to open and closed keyboard, 237-239

scenes, 66-67

ViewController class, modifying, 110-112

view layer (MVC), 318

ViewCarProtocol, adding to make and model edit scene (CarValet project), 303-305

views

attaching recognizers to, 442

constraints, 120-131

Assistant editor preview mode, 145-148

bottom layout guides (IB), 176-178

changing for orientation, 162-163

completeness of specification, 133-134

cost content compression resistance, 150

creating, 122-123

dragging out, 130-131

intrinsic content size, 134

previewing, 144-145

references, adding, 163-166

relationships, 120-122

top layout guides (IB), 176-178

values, changing, 128-130

content views, 233-232

frames, 235-239

invisible container views, 137

root view, 255

screen sizes, toggling, 145

scroll views

adding to add/view scene (CarValet project), 227-230
adding to edit scene (CarValet project), 227-230
constraints, 235
paging, 240-241, 243-244
populating, 241-243
resizing, 239-240
zooming content, 226, 245-248
table views
   behaviors, 277
groups, 345
index, adding, 355-358
searching, 358-369
sections, 346-349
sorting, 345
updating, 306-307
UIPickerView, 307
UIScrollView, 225-226
visual elements
   adding
      to custom cells, 343-344
      to edit scene (CarValet project), 91
   add/view scene (CarValet project)
   dividers, 71-72
   labels, 68-70

glyphish.com, 259
idc.com, 18
macosrumors.com, 18
macworld.com, 18
Plausible Crash Reporter project, 506
tuaw.com, 18
workspaces, 21
writing blocks, 455-460
WWDC (World Wide Development Conference) videos, 5

Xcode
   Accounts pane, 495-497
debugger, 7
Hello World project
   creating, 21-25
IB, 6
   bottom layout guides, 176-178
   constraints, creating, 122-123
   top layout guides, 176-178
installing, 1-2
interface, 25-27
   Editor buttons, 27
   Navigator, 27
   Run button, 26
   status area, 26
   utilities area, 27
iOS Simulator, 6
   limitations of, 8-10
   localization, 9
testing apps, 7-15
   Video Out emulation (iOS Simulator), 10

Wain, Joseph, 259
weak qualifier, 78
websites, 527-528
   canalys.com, 18
   chitika.com, 18
   flurry.com, 18
gartner.com, 18
Xcode

performance tools, 6
Project Editor, 6
projects, labeling, 28-30
workspaces, 21
XIB files, opening, 263-264

Y-Z

year editor, setting up for model year edit scene, 308-309
Zombies template (Instruments), troubleshooting EXC_BAD_ACCESS errors, 486-489
zooming content with scroll views, 226, 245-248
Cocoa Programming for Mac OS X, Fourth Edition
Aaron Hillegass and Adam Preble

Effective Objective-C 2.0
Matt Galloway

iOS 6 App Development Fundamentals
LiveLessons Part I (Video Training)
Paul Deitel

Objective-C Advanced Programming
LiveLessons (Video Training)
Jiva DeVoe

iOS Auto Layout Demystified
Erica Sadun

iOS UICollectionView
Ash Furrow

Xcode and Instruments Fundamentals
LiveLessons
Brandon Alexander

For more information and to read sample material, please visit informit.com/learnmac.
Titles are also available at safari.informit.com.
ESSENTIAL REFERENCES FOR
PROGRAMMING PROFESSIONALS

Developer’s Library
informit.com/devlibrary

The Core iOS 6 Developer's Cookbook, Fourth Edition
Erica Sadun

The Advanced iOS 6 Developer's Cookbook
Erica Sadun

Programming in Objective-C, Fifth Edition
Stephen G. Kochan

Other Developer’s Library Titles

<table>
<thead>
<tr>
<th>TITLE</th>
<th>AUTHOR</th>
<th>ISBN-13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test-Driven iOS Development</td>
<td>Graham Lee</td>
<td>978-0-321-77418-7</td>
</tr>
<tr>
<td>Cocoa® Programming Developer’s Handbook</td>
<td>David Chisnall</td>
<td>978-0-321-63963-9</td>
</tr>
<tr>
<td>Cocoa Design Patterns Applications for the iPhone</td>
<td>Erik M. Buck / Donald A. Yacktman</td>
<td>978-0-321-53502-3</td>
</tr>
</tbody>
</table>

Developer’s Library books are available at most retail and online bookstores. For more information or to order direct, visit our online bookstore at informit.com/store.

Online editions of all Developer’s Library titles are available by subscription from Safari Books Online at safari.informit.com.
Register the Addison-Wesley, Exam Cram, Prentice Hall, Que, and Sams products you own to unlock great benefits.

To begin the registration process, simply go to informit.com/register to sign in or create an account. You will then be prompted to enter the 10- or 13-digit ISBN that appears on the back cover of your product.

Registering your products can unlock the following benefits:
- Access to supplemental content, including bonus chapters, source code, or project files.
- A coupon to be used on your next purchase.

Registration benefits vary by product. Benefits will be listed on your Account page under Registered Products.

About InformIT — THE TRUSTED TECHNOLOGY LEARNING SOURCE

INFORMIT IS HOME TO THE LEADING TECHNOLOGY PUBLISHING IMPRINTS Addison-Wesley Professional, Cisco Press, Exam Cram, IBM Press, Prentice Hall Professional, Que, and Sams. Here you will gain access to quality and trusted content and resources from the authors, creators, innovators, and leaders of technology. Whether you’re looking for a book on a new technology, a helpful article, timely newsletters, or access to the Safari Books Online digital library, InformIT has a solution for you.
InformIT is a brand of Pearson and the online presence for the world’s leading technology publishers. It’s your source for reliable and qualified content and knowledge, providing access to the top brands, authors, and contributors from the tech community.

LearnIT at InformIT

Looking for a book, eBook, or training video on a new technology? Seeking timely and relevant information and tutorials? Looking for expert opinions, advice, and tips? InformIT has the solution.

- Learn about new releases and special promotions by subscribing to a wide variety of newsletters. Visit informit.com/newsletters.
- Access FREE podcasts from experts at informit.com/podcasts.
- Read the latest author articles and sample chapters at informit.com/articles.
- Get tips from expert blogs at informit.com/blogs.

Visit informit.com/learn to discover all the ways you can access the hottest technology content.

Are You Part of the IT Crowd?

Connect with Pearson authors and editors via RSS feeds, Facebook, Twitter, YouTube, and more! Visit informit.com/socialconnect.
Try Safari Books Online FREE for 15 days
Get online access to Thousands of Books and Videos

Feed your brain

See it, believe it
Watch hundreds of expert-led instructional videos on today’s hottest topics.

WAIT, THERE’S MORE!

Gain a competitive edge
Be first to learn about the newest technologies and subjects with Rough Cuts pre-published manuscripts and new technology overviews in Short Cuts.

Accelerate your project
Copy and paste code, create smart searches that let you know when new books about your favorite topics are available, and customize your library with favorites, highlights, tags, notes, mash-ups and more.

* Available to new subscribers only. Discount applies to the Safari Library and is valid for first 12 consecutive monthly billing cycles. Safari Library is not available in all countries.