Beginning iPhone Development with Swift 4

Exploring the iOS SDK

Fourth Edition

Molly K. Maskrey

Apress®

Beginning iPhone Development with Swift 4

Exploring the iOS SDK

Fourth Edition

Molly K. Maskrey

Apress®

Beginning iPhone Development with Swift 4: Exploring the iOS SDK

Molly K. Maskrey
Parker, Colorado, USA

ISBN-13 (pbk): 978-1-4842-3071-8 ISBN-13 (electronic): 978-1-4842-3072-5
https://doi.org/10.1007/978-1-4842-3072-5

Library of Congress Control Number: 2017957132

Managing Director: Welmoed Spahr
Editorial Director: Todd Green
Acquisitions Editor: Aaron Black
Development Editor: James Markham
Technical Reviewer: Bruce Wade
Coordinating Editor: Jessica Vakili
Copy Editor: Kim Wimpsett
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springer.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a Delaware corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text is available to readers at www.apress.com. For detailed information about how to locate your book's source code, go to www.apress.com/source-code/.

Printed on acid-free paper

Another year, another revision of this book, as well as one of my own personal tribe.

Still around after more than two years of ups and downs, KP stuck beside me during my most difficult times. No matter what I did, her sanity and silliness kept me going. She was, for a long while, my muse…my inspiration to write and to keep it as fun as possible…never taking myself too seriously. Now, as our paths slightly diverge, the qualities she's left permeate my days and make me a better self.

Mellie has been my rock. We've been in and out of being friends, and I hope she knows I consider her the closest friend I've ever had. I probably wouldn't be doing this if it weren't for the love she's shown me.

Tonya came into my life recently and has been such a godsend to help me become more confident in all I say and do, and I can't thank her enough.

She probably has no clue she's in here, but Lauren kept me going for the past year. When I would've preferred to lay in the dark and sleep and zone out, without any aggressiveness Lauren would give me something in her words to keep to going…to not give up. She—I won't say forced, but she was very persuasive—got me into a program that is, this very day, putting me onto a better path for myself and my future and the future of anyone alongside me for my journey.

Finally, to Jennifer…our lives have radically changed over the past year. But, as partners in business and advocates for each other's happy life, our relationship bond, while admittedly very tumultuous, has solidified into something we both see as pretty damn good.

In my personal struggles over the preceding year, these friends kept me from falling into an abyss so deep I might never have returned. Writing can be a lonely thing, and having a support system such as these beautiful, wonderful women are the only reason this endeavor was a success. A special friend told me last year that some friends are only in your life for a season. I pray that these women are friends for a lifetime.

—MM, August 2017

Contents at a Glance

Contents

About the Author

Molly K. Maskrey started as an electrical engineer in her 20s working for various large aerospace companies including IBM Federal Systems, TRW (now Northrup-Grumman), Loral Systems, Lockheed-Martin, and Boeing. After successfully navigating the first dot-com boom, she realized that a break was in order and took several years off, moving to Maui and teaching windsurfing at the beautiful Kanaha Beach Park.

She moved back to Colorado in 2005 and, with Jennifer, formed Global Tek Labs, an iOS development and accessory design services company that is now one of the leading consulting services for new designers looking to create smart attachments to Apple devices.

In 2014 Molly and Jennifer formed Quantitative Bioanalytics Laboratories, a wholly owned subsidiary of Global Tek to bring high-resolution mobile sensor technology to physical therapy, elder balance and fall prevention, sports performance quantification, and instrumented gait analysis (IGA). In a pivot, Molly changed the direction of QB Labs to a platform-based predictive analytics company seeking to democratize data science for smaller companies.

Molly's background includes advanced degrees in electrical engineering, applied mathematics, data science, and business development. Molly generally speaks at a large number of conferences throughout the year including the Open Data Science Conference (ODSC) 2017 West, advancing the topic of moving analytics from the cloud to the fog for smart city initiatives. What fuels her to succeed is the opportunity to bring justice and equality to everyone whether it's addressing food insecurity with her business partner or looking at options for better management of mental health using empirical data and tools such as natural language processing, speech pattern recognition using neural networks, or perfusion analysis in brain physiology.

About the Technical Reviewer

Bruce Wade is a software engineer from British Columbia, Canada. He started software development when he was 16 years old by coding his first web site. He went on to study computer information systems at DeVry Institute of Technology in Calgary; to further enhance his skills, he studied visual and game programming at the Art Institute of Vancouver. Over the years he has worked for large corporations as well as several startups. His software experience has led him to utilize many different technologies including C/C++, Python, Objective-C, Swift, Postgres, and JavaScript. In 2012 he started the company Warply Designed to focus on mobile 2D/3D and OS X development. Aside from hacking out new ideas, he enjoys spending time hiking with his Boxer Rasco, working out, and exploring new adventures.

Acknowledgments

First, I want to acknowledge all my friends who gave me the support to persevere and go through with writing when it would have been so easy to just give up. Thanks to Sam especially, another writer I met more than a year ago, who was always there to provide support, drink tequila, and suggest fun things to do in order to keep it real. He was my cohost at almost a year of rooftop parties at Galvanize.

Thanks to Children's Hospital Colorado and the Center for Gait and Movement Analysis, both of which have been so generous to let me be part of what they do for young adults with cerebral palsy and other gait disorders. The understanding I've gained drives me to focus my efforts on helping the many who truly need it. Thanks to the CEOs of 10.10.10 Health who let me see what they were doing and to provide my annoying feedback.

Thanks to the clients and friends of Global Tek Labs who so generously allowed me to include some of their projects in this book for illustrative purposes. Thanks to the hundreds of people who have attended my talks over the past year and have given me ideas for what to include, such as John Haley who told me of his personal woes in understanding the Auto Layout feature of Xcode. Those actual experiences helped drive the subject matter I chose to include in this book.

Finally, I want to acknowledge all the authors before me who set the stage for my own little work to fit into a much broader landscape.

CHAPTER 1

Getting to Know the iOS Landscape

Coding for Apple mobile devices provides a rewarding and lucrative career path where you might not only change people's lives with your app (see Figure 1-1) but might also have a great time being with bright, like-minded women and men such as yourself. Though you're bound to find some difficulty in learning the language, tools, and processes, these new associates will help you through the landscape of this new world and will challenge you to be your best and to stand out from the mediocre.

Figure 1-1. *One of the greatest feelings you can experience as an iOS developer is seeing other people using your creation out in the real world*

M. K. Maskrey, *Beginning iPhone Development with Swift 4*, https://doi.org/10.1007/978-1-4842-3072-5_1

For now, think of me as one of those friends along your journey of iOS discovery. I'm so proud to be able to help you by providing this initiation into the world of iOS development, whether it is for iPhone, iPod touch, or the iPad. iOS provides an exciting platform that has seen explosive growth since it first came out in 2007. The proliferation of mobile devices means that people are using software everywhere they go, whether it is a phone or a wearable, such as the Apple Watch. With the release of iOS 11, Xcode 9, Swift 4, and the latest version of the iOS software development kit (SDK), things continue to be exciting and generally have become easier for new developers. What's more, you can now bring the world of data science and predictive analytics to your app with CoreML as well as image and natural language processing frameworks.

About the Book

This book guides you down the path to creating your own iOS applications. I want to get you past the initial difficulties to help you understand the way that iOS applications work and how they are built.

As you work your way through this book, you will create a number of small applications, each designed to highlight specific iOS features and to show you how to control or interact with those features. If you combine the foundation you'll gain through this book with your own creativity and determination and then add in the extensive and well-written documentation provided by Apple, you'll have everything you need to build your own professional iPhone and iPad applications.

■ **Note** Throughout most of this book, I tend to refer to the iPhone and iPad, as they are the devices that you'll most commonly use in the examples. This does not preclude the iPod touch by any means; it is just a matter of convenience.

■ **Tip** The authors of the previous editions of this book have set up a forum, which is a great place to meet like-minded folks, get your questions answered, and even answer other people's questions. The forum is at `http://forum.learncocoa.org`. Be sure to check it out!

Things You'll Need

Before you can begin writing software for iOS, you'll need a few items. For starters, you'll need an Intel-based Macintosh running Sierra (macOS 10.12) or newer. Any recent Intel-based Macintosh computer—laptop or desktop—should work just fine. Of course, as well as the hardware, you'll need the software. You can learn how to develop iOS applications and get the software tools that you'll need as long as you have an Apple ID; if you own an iPhone, iPad, or iPod, then you've almost certainly already have an Apple ID, but if you don't, just visit `https://appleid.apple.com/account` to create one. Once you've done that, navigate to `https://developer.apple.com/develop`. This will bring you to a page similar to the one shown in Figure 1-2.

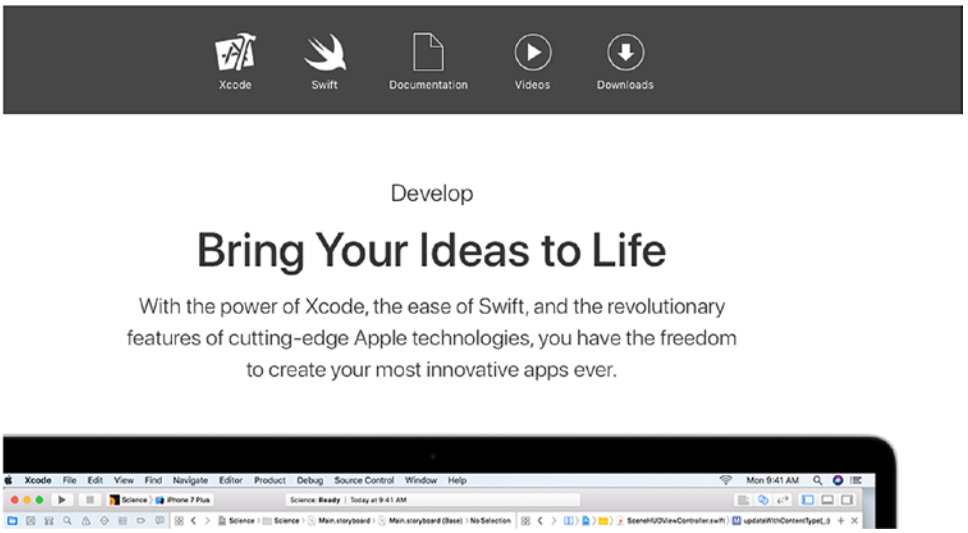

Figure 1-2. *Apple's development resources site*

Click Downloads on the top bar to go to the main resources page (see Figure 1-3) for the current production release and (if there is one) the current beta release of iOS. Here, you'll find links to a wealth of documentation, videos, sample code, and the like—all dedicated to teaching you the finer points of iOS application development. Be sure to scroll to the bottom of the page and check out the links to the Documentation and Videos sections of the web site. You'll also find a link to the Apple Developer Forums, where you can follow discussions on a wide variety of topics covering the whole iOS platform, as well as macOS, watchOS, and tvOS. To post to the forums, you'll need to be a registered Apple developer.

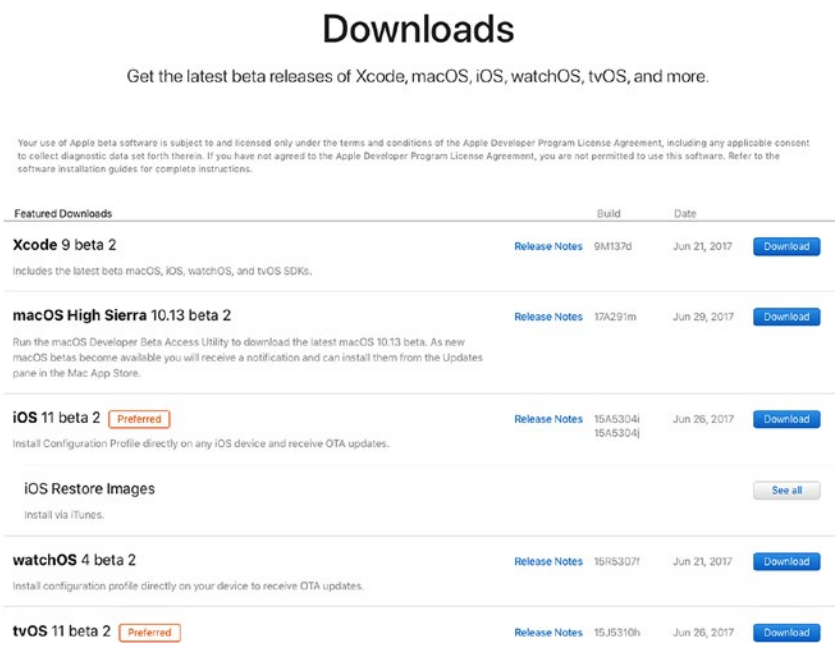

Figure 1-3. *You can download all the production and beta releases of the development tools from the Downloads page. You will need to sign in with your Apple ID.*

■ **Note** At the developer conference WWDC 2016, Apple changed the name of OS X back to the previously used *macOS* to become more in line with the other naming conventions used throughout the four major system platforms.

The most important tool you'll be using to develop iOS applications is Xcode, Apple's integrated development environment (IDE). Xcode includes tools for creating and debugging source code, compiling applications, and performance tuning the applications you've written.

You can download the current beta release of Xcode by following the Xcode link from the developer Downloads page shown in Figure 1-3. If you prefer to use the latest production release, you'll find it in the Mac App Store, which you can access from your Mac's Apple menu.

SDK VERSIONS AND SOURCE CODE FOR THE EXAMPLES

As the versions of the SDK and Xcode evolve, the mechanism for downloading them has changed over the past few years. Apple now publishes the current production version of Xcode and the iOS SDK on the Mac App Store, while simultaneously providing developers with the ability to download preview versions of upcoming releases from its developer site. Bottom line: unless you really want to work with the most recent development tools and platform SDK, you usually want to download the latest released (nonbeta) version of Xcode and the iOS SDK, so use the Mac App Store.

This book is written to work with the latest versions of Xcode and the SDK. In some places, new functions or methods are introduced with iOS 11 that are not available in earlier versions of the SDK.

Be sure to download the latest and greatest source code archive for examples from this book's page at `www.apress.com`. The code is updated as new versions of the SDK are released, so be sure to check the site periodically.

Your Options as a Developer

The free Xcode download includes a simulator that will allow you to build and run iPhone and iPad apps on your Mac, providing the perfect environment for learning how to program for iOS. However, the simulator does *not* support many hardware-dependent features, such as the accelerometer and camera. To test applications that use those features, you'll need an iPhone, iPod touch, or iPad. While much of your code can be tested using the iOS simulator, not all programs can be. And even those that can run on the simulator really need to be thoroughly tested on an actual device before you ever consider releasing your application to the public.

Previous versions of Xcode required you to register for the Apple Developer Program (which is not free) to install your applications on a real iPhone or other device. Fortunately, this has changed. Xcode 7 started allowing developers to test applications on real hardware, albeit with some limitations that I'll cover in this book, without purchasing an Apple Developer Program membership. That means you can run most of the examples in this book on your iPhone or iPad for free! However, the free option does not give you the ability to distribute your applications on Apple's App Store. For those capabilities, you'll need to sign up for one of the other options, which aren't free.

- *The Standard program costs $99/year*: It provides a host of development tools and resources, technical support, and distribution of your applications via Apple's iOS and Mac App Stores. Your membership lets you develop and distribute applications for iOS, watchOS, tvOS, and macOS.

- *The Enterprise program costs $299/year*: It is designed for companies developing proprietary, in-house iOS applications.

For more details on these programs, visit `https://developer.apple.com/programs` (see Figure 1-4). If you are an independent developer, you can definitely get away with just buying the standard program membership. You don't have to do that until you need to run an application that uses a feature such as iCloud that requires a paid membership, you want to post a question to the Apple Developer Forums, or you are ready to deploy your application to the App Store.

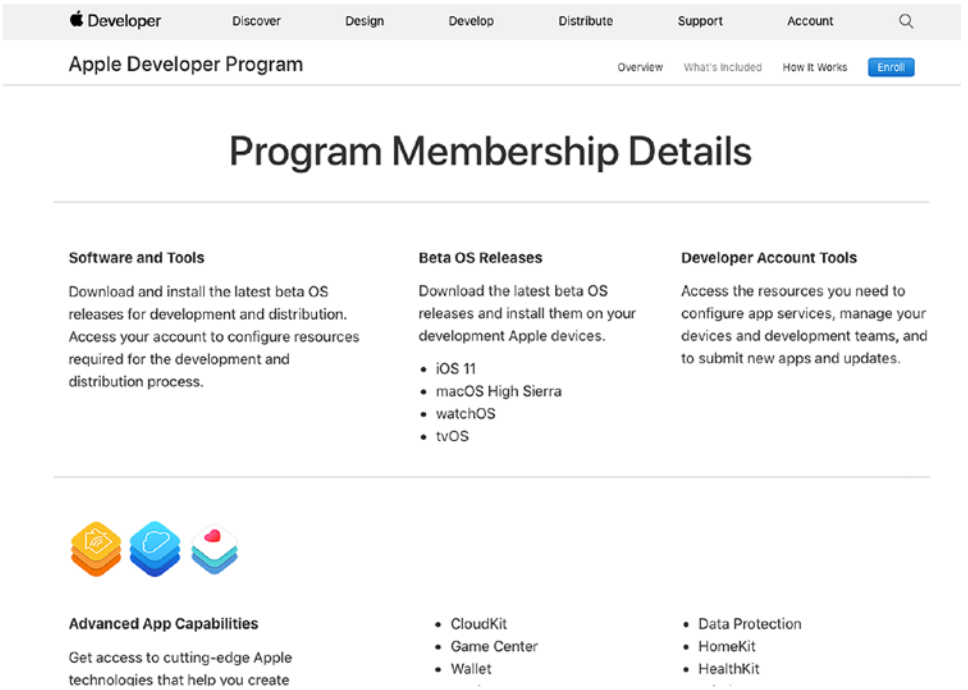

Figure 1-4. *Signing up for a paid membership gives you access to beta and OS tools releases*

Because iOS supports an always-connected mobile device that uses other companies' wireless infrastructures, Apple has needed to place far more restrictions on iOS developers than it ever has on Mac developers (who are able—at the moment, anyway—to write and distribute programs with absolutely no oversight or approval from Apple). Even though the iPod touch and the Wi-Fi-only versions of the iPad don't use anyone else's infrastructure, they're still subject to these same restrictions.

Apple has not added restrictions to be mean but rather as an attempt to minimize the chances of malicious or poorly written programs being distributed and degrading performance on the shared network. Developing for iOS may appear to present a lot of hoops to jump through, but Apple has expended quite an effort to make the process as painless as possible. Also consider that $99 is still much less expensive than buying, for example, any of the paid versions of Visual Studio, which is Microsoft's software development IDE.

Things You Should Know

In this book, I'm going to assume you already have some programming knowledge in general and object-oriented programming in particular (you know what classes, objects, loops, and variables are, for example). But of course, I don't assume you are already familiar with Swift. There's an appendix at the end of the book that introduces you to both Swift and the Playground feature in Xcode that makes it easy to try the features. If you'd like to learn more about Swift after reading the material in the appendix, the best way to do so is to go directly to the source and read *The Swift Programming Language*, which is Apple's own guide and reference to the language. You can get it from the iBooks store or from the iOS developer site at `https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/index.html`.

You also need to be familiar with iOS itself as a user. Just as you would with any platform for which you wanted to write an application, get to know the nuances and quirks of the iPhone, iPad, or iPod touch. Take the time to get familiar with the iOS interface and with the way Apple's iPhone and/or iPad applications look and feel.

Because the different terms can be a little confusing at first, Table 1-1 shows the relationships of IDEs, application programming interfaces (APIs), and languages to the platform operating system for which you are developing.

Table 1-1. *Platform, Tools, Language Relationships*

Develop For…	IDE	API	Language
macOS	Xcode	Cocoa	Objective-C, Swift
iOS	Xcode	Cocoa Touch	Objective-C, Swift

Some Unique Aspects About Working in iOS

If you have never programmed for the Mac using Cocoa, you may find Cocoa Touch—the application framework you'll be using to write iOS applications—a little alien. It has some fundamental differences from other common application frameworks, such as those used when building .NET or Java applications. Don't worry too much if you feel a little lost at first. Just keep plugging away at the exercises and it will all start to fall into place after a while.

■ **Note** You'll see a lot of reference to *frameworks* in this book. Although the term is a little vague and used in a few different ways depending on the context, a *framework* is a collection of "stuff," which may include a library, several libraries, scripts, UI elements, and anything else in a single collection. A framework's stuff is generally associated with some specific function such as location services using the Core Location framework.

If you have written programs using Cocoa, a lot of what you'll find in the iOS SDK will be familiar to you. A great many classes are unchanged from the versions that are used to develop for macOS. Even those that are different tend to follow the same basic principles and similar design patterns. However, several differences exist between Cocoa and Cocoa Touch.

Regardless of your background, you need to keep in mind some key differences between iOS development and desktop application development. These differences are discussed in the following sections.

iOS Supports a Single Application at a Time—Mostly

On iOS, it's usually the case that only one application can be active and displayed on the screen at any given time. Since iOS 4, applications have been able to run in the background after the user presses the Home button, but even that is limited to a narrow set of situations and you must code for it specifically. In iOS 9, Apple added the ability for two applications to run in the foreground and share the screen, but for that, the user needs to have one of the more recent iPads.

When your application isn't active or running in the background, it doesn't receive any attention whatsoever from the CPU. iOS allows background processing, but making your apps play nicely in this situation will require some effort on your part.

There's Only a Single Window

Desktop and laptop operating systems allow many running programs to coexist, each with the ability to create and control multiple windows. However, unless you attach an external screen or use AirPlay and your application is coded to handle more than one screen, iOS gives your application just one "window" to work with. All of your application's interaction with the user takes place inside this one window, and its size is fixed at the size of the screen, unless your user has activated the Multitasking feature, in which case your application may have to give up some of the screen to another application.

For Security, Access to Device Resources Is Limited

Programs on a desktop or a laptop computer pretty much have access to everything that the user who launched it does. However, iOS seriously restricts which parts of the device your program can use.

You can read and write files only from the part of iOS's file system that was created for your application. This area is called your application's *sandbox*. Your sandbox is where your application will store documents, preferences, and every other kind of data it may need to retain.

Your application is also constrained in some other ways. You will not be able to access low-number network ports on iOS, for example, or do anything else that would typically require root or administrative access on a desktop computer.

Apps Need to Respond Quickly

Because of the way it is used, iOS needs to be snappy, and it expects the same of your application. When your program is launched, you need to get your application open, the preferences and data loaded, and the main view shown on the screen as fast as possible—in no more than a few seconds. Your app should have low latency.

■ **Note** By latency, I do not mean speed. Speed and latency are commonly interchanged, but that is not really correct. *Latency* refers to the time between an action is taken and a result happens. If the user presses the Home button, iOS goes home, and you must quickly save everything before iOS suspends your application in the background. If you take longer than five seconds to save and give up control, your application process will be killed, regardless of whether you finished saving. There is an API that allows your app to ask for additional time to work when it's about to go dark, but you've got to know how to use it. So, in general, you want to get things done quickly, which might mean dumping and losing unnecessary information.

Limited Screen Size

The iPhone's screen is really nice. When introduced, it was the highest-resolution screen available on a handheld consumer device, by far. But even today, the iPhone display isn't all that big, and as a result, you have a lot less room to work with than on modern computers. The screen was just 320 × 480 on the first few iPhone generations, and it was later doubled in both directions to 640 × 960 with the introduction of the iPhone 4's Retina display. Today, the screen of the largest iPhone (the iPhone 6/6s Plus) measures 1080 × 1920 pixels. That sounds like a decent number of pixels, but keep in mind that these high-density displays (for which Apple uses the term *Retina*) are crammed into pretty small form factors, which has a big impact on the kinds of applications and interactivity you can offer on an iPhone and even an iPad. Table 1-2 lists the sizes of the screens of all the current commonly used iPhone devices that are supported by iOS 11 at the time of writing.

Table 1-2. *iOS Device Screen Sizes*

Device	Hardware Size	Software Size	Scaling
iPhone 7	750 × 1334	320 × 568	3×
iPhone 7s	1080 × 1920	375 × 667	3×
iPhone 6s	750 × 1334	375 × 667	3×
iPhone 6s Plus	1080 × 1920	414 × 736	3×
iPhone SE	640 × 1136	320 × 568	2×

The hardware size is the actual physical size of the screen in pixels. However, when writing software, the size that really matters is the one in the Software Size column. As you can see, in most cases, the software size reflects only half that of the actual hardware. This situation came about when Apple introduced the first Retina device, which had twice as many pixels in each direction as its predecessor. If Apple had done nothing special, all existing applications would have been drawn at half-scale on the new Retina screen, which would have made them unusable. So, Apple chose to internally scale everything that applications draw by a factor of 2 so that they would fill the new screen without any code changes. This internal scaling by a factor of 2 applies iPhone 6s and iPhone SE, while the 6s Plus, 7, and 7 Plus use a factor of 3×. For the most part, though, you don't need to worry too much about the fact that your application is being scaled—all you need to do is work within the software screen size, and iOS will do the rest.

The only exceptions to this rule center on bitmap images. Since bitmap images are, by their nature, fixed in size, for best results you can't really use the same image on a Retina screen as you would on a non-Retina screen. If you try to do that, you'll see that iOS scales your image up for a device that has a Retina screen, which has the effect of introducing blur. You can fix this by including separate copies of each image for the 2× and 3× Retina screens. iOS will pick the version that matches the screen of the device on which your application is running.

■ **Note** If you look back at Table 1-1, you'll see that it appears that the scale factor in the fourth column is the same as the ratio of the hardware size to the software size. For example, on the iPhone 6s, the hardware width is 750 and software width is 375, which is a ratio of 2:1. Look carefully, though, and you'll see that there's something different about the iPhone 6/6s Plus. The ratio of the hardware width to the software width is 1080/414, which is 2.608:1, and the same applies to the height ratio. So, in terms of the hardware, the iPhone 6s Plus does not have a true 3× Retina display. However, as far as the software is concerned, a 3× scale is used, which means that an application written to use the software screen size of 414 × 736 is first logically mapped to a virtual screen size of 1242 × 2208, and the result is then scaled down a little to match the actual hardware size of 1080 × 1920. Fortunately, this doesn't require you to do anything special because iOS takes care of all the details.

Limited Device Resources

Software developers from just a decade or two ago laugh at the idea of a machine with at least 512MB of RAM and 16GB of storage being in any way resource-constrained, but it's true. Developing for iOS doesn't reside in the same league as trying to write a complex spreadsheet application on a machine with 48KB of memory. But given the graphical nature of iOS and all it is capable of doing, running out of memory happens from time to time. Lately, Apple has significantly boosted RAM to a minimum of 2GB.

The iOS devices available right now have either 2GB (iPad, iPad mini 4, iPhone 6s/6s Plus, and iPhone SE), 3GB (iPhone 7s Plus), or 4GB (both iPad Pro models), though this will likely increase over time. Some of that memory is used for the screen buffer and by other system processes. Usually, no more than half of that memory is left for your application to use, and the amount can be considerably less, especially now that other apps can be running in the background.

Although that may sound like it leaves a pretty decent amount of memory for such a small computer, there is another factor to consider when it comes to memory on iOS. Modern computer operating systems like macOS take chunks of memory that aren't being used and write them to disk in something called a *swap file*. The swap file allows applications to keep running, even when they have requested more memory than is actually available on the computer. iOS, however, will not write volatile memory, such as application data, to a swap file. As a result, the amount of memory available to your application is constrained by the amount of unused physical memory in the iOS device.

Cocoa Touch has built-in mechanisms for letting your application know that memory is getting low. When that happens, your application must free up unneeded memory or risk being forced to quit.

Features Unique to iOS Devices

Since I've mentioned that Cocoa Touch is missing some features that Cocoa has, it seems only fair to mention that the iOS SDK contains some functionality that is not currently present in Cocoa—or, at least, is not available on every Mac.

- The iOS SDK provides a way for your application to determine the iOS device's current geographic coordinates using Core Location.
- Most iOS devices have built-in cameras and photo libraries, and the SDK provides mechanisms that allow your application to access both.
- iOS devices have built-in motion sensors that let you detect how your device is being held and moved.

User Input and Display

Since iOS devices do not have a physical keyboard or a mouse, you interact differently with your user than you do when programming for a general-purpose computer. Fortunately, most of that interaction is handled for you. For example, if you add a text field to your application, iOS knows to bring up a keyboard when the user touches that field, without you needing to write any extra code.

■ **Note** All iOS devices allow you to connect an external keyboard via Bluetooth or the Lightning connector, which provides a nice keyboard experience and saves you some screen real estate. Currently, iOS does not support connecting a mouse.

What's in This Book

When I first started programming applications for iOS, then called iPhone OS, I picked up the original edition of this book based on Objective-C. I became, at least in my mind, a capable and productive app developer, even making some money with my products. So, I want to return the favor by providing this latest and greatest edition to help you achieve that same level of success and more. So, here's what I'm going to be covering:

- In Chapter 2, you'll learn how to use Xcode's user interface (UI) developer tool, Interface Builder, to create a simple visual result, placing some text on the screen.
- In Chapter 3, I'll show you how to start interacting with the user, building an application that dynamically updates displayed text at runtime based on buttons the user presses.
- Chapter 4 continues Chapter 3's topic by introducing you to several more of iOS's standard user interface controls. I'll also demonstrate how to use alerts and action sheets to prompt users to make a decision or to inform them that something out of the ordinary has occurred.
- In Chapter 5, you'll look at handling rotation and Auto Layout, the mechanisms that allow iOS applications to be used in both portrait and landscape modes.
- In Chapter 6, I'll start discussing more advanced user interfaces and explore creating applications that support multiple views. I'll show you how to change which view is shown to the user at runtime, which will greatly enhance the potential of your apps.
- iOS supports tab bars and pickers as part of the standard iOS user interface. In Chapter 7, you'll learn how to implement these interface elements.
- In Chapter 8, I'll cover table views, the primary way of providing lists of data to the user and the foundation of hierarchical navigation–based applications. You'll also see how to let the user search your application data.
- One of the most common iOS application interfaces, the hierarchical list, lets you drill down to see more data or more details. In Chapter 9, you'll learn what's involved in implementing this standard type of interface.

- From the beginning, iOS applications have used table views to display dynamic, vertically scrolling lists of components. A few years ago, Apple introduced a new class called `UICollectionView` that takes this concept a few steps further, giving developers lots of new flexibility in laying out visual components. Chapter 10 introduces you to collection views.
- Chapter 11 shows you how to build master-detail applications and present a list of items (such as the e-mails in a mailbox), allowing the user to view the details of each individual item, one at a time. You'll also work with iOS controls that support this concept, originally developed for the iPad and now also available on the iPhone.
- In Chapter 12, you'll look at implementing application settings, which is iOS's mechanism for letting users set their application-level preferences.
- Chapter 13 covers data management on iOS. I'll talk about creating objects to hold application data and show how that data can be persisted to iOS's file system. I'll present the basics of using Core Data, allowing you to save and retrieve data easily; however, for an in-depth discussion of Core Data, you'll want to check out *Pro iOS Persistence Using Core Data* by Michael Privat and Robert Warner (Apress, 2014).
- Everyone loves to draw, so you'll look at doing some custom drawing in Chapter 14, where I'll introduce you to the Core Graphics system.
- Finally, the appendix introduces the Swift programming language in its current state and covers all the features that you'll need to know to understand the example code in this book.

What's New in This Update?

After the first edition of this book hit the bookstores, the iOS development community grew at a phenomenal rate. The SDK continually evolved, with Apple releasing a steady stream of SDK updates. iOS 11 and Xcode 9 contain many new enhancements. I've been hard at work updating the book to cover the new technologies that you'll need to be aware of to start writing iOS applications.

Swift and Xcode Versions

Though having been out for more than two years now, Swift is still in a state of flux and likely to remain so for some time to come. Interestingly, Apple promised that the compiled binaries for applications written now will work on later versions of iOS, but it is *not* guaranteed that the source code for those same applications will continue to compile. As a result, it is possible that example code that compiled and worked with the version of Xcode that was current when this book was published no longer works by the time you read it. Xcode 6.0 shipped with Swift version 1, Xcode 6.3 had Swift version 1.2, Xcode 7 introduced Swift 2, and Xcode 8 introduced Swift 3. In this book, I'm starting off with a beta 2 release of Xcode 9 and Swift 4.

If you find that some of the example source code no longer compiles with the release of Xcode that you are using, please visit the book's page at `www.apress.com` and download the latest version. If after doing this you are still having problems, please bring it to my attention by submitting an erratum on the Apress web site.

Let's Get Started

iOS provides an incredible computing platform and an exciting new frontier for your development career. You'll likely find programming for iOS to be a new experience—different from working on any other platform. For everything that looks familiar, there will be something alien, but as you work through the book's code, the concepts should all come together and start to make sense.

Keep in mind that the examples in this book are not simply a checklist that, when completed, magically grants you iOS developer expert status. Make sure you understand what you did and why before moving on to the next project. Don't be afraid to make changes to the code. By observing the results of your experimentation, you can wrap your head around the complexities of coding in an environment like Cocoa Touch.

That said, if you've already downloaded and installed Xcode, turn the page and take your next steps to becoming a real iOS app developer.