Learn Swift, iOS 8 SDK, and Cocoa Touch to start creating exciting iPhone and iPad apps

Learn iOS 8 App Development

SECOND EDITION

James Bucanek

Apress®

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.

Apress®

Contents at a Glance

Introduction

I'm standing on a street corner in San Francisco, a city I visit far too infrequently. In my hand I hold an electronic device. The device is receiving status updates about the city's public transportation system in real time. It is telling me that the F-line rail will arrive at the Market and 5th Street station in seven minutes. It displays a map of the city and, by timing radio waves it receives from outer space, triangulates and displays my exact location on that map. A magnetometer determines which direction I'm holding the device and uses that information to indicate the direction I should walk to meet the rail car in time to board it. My friends call me, wondering when I will arrive. A tiny video camera and microphone share my image and voice with them as I walk. I'm meeting them at a gallery opening. It's an exhibition of new artwork, by artists from all over the world, created entirely using devices similar to the one I hold in my hand. When I arrive, I use my device to share my experiences with friends and family back home, exchange contact information with people I meet, and look up restaurant suggestions for where we might eat later.

This is a true story. A couple of decades ago, it would have been science fiction.

We live in a time in which personal electronics are changing how we work, travel, communicate, and experience the world. A day doesn't go by without someone discovering another novel use for them. And while I'm sure you enjoy benefiting from this new technology, you're reading this book because you want to participate in this revolution. You want to create apps.

You've come to the right place.

Who Is This Book For?

This book is for anyone who wants to learn the basic tools and techniques for creating exciting, dynamic applications for Apple products that run the iOS operating system. As of this writing, that includes the iPad, iPhone, and iPod Touch.

This book assumes you are new to developing iOS apps and that you have limited programming experience. If you've been learning Swift—Apple's new programming language—that's perfect. If you know Objective-C, C, Java, C#, or C++, you shouldn't have too much trouble following along, and there's a Swift primer in Chapter 20 that you'll want to read. If you are completely new to

programming computers, I suggest getting a basic Swift programming book—say, *Swift for Absolute Beginners* by Gary Bennett and Brad Lees—and read that first or in parallel. iOS app are developed using Objective-C or Swift, but this book uses Swift exclusively.

This book will explain the fundamentals of how iOS apps are designed, built, and deployed. You'll pick up some good design habits, get some core programming skills, and learn your way around the development tools used to create apps.

This book is not an in-depth treatise on any one technology. It's designed to stimulate your imagination by giving you a firm grounding in building apps that use a variety of device capabilities, such as finding your location on a map, using the accelerometer, taking pictures with the built-in camera, creating dynamic animation, participating in social networks, and storing information in the cloud. From there, you can leap beyond these examples to create the next great iOS app!

Too Cool for School

I'm an "old-school" programmer. I learned programming from the bit up (literally). At the risk of dating myself, the first program I wrote was on a 4-bit microcontroller using toggle switches to input the machine instructions. So, I pretty much knew everything there was to know about machine code before I started to program in "high-level" languages like BASIC and C. All that was well behind me before I wrote my first graphical user interface (GUI) application for the (then-revolutionary) Macintosh computer.

While I value this accumulated knowledge, and much of it is still useful, I realize that a strict "ground-up" approach isn't necessary to develop great apps for iOS today. Many of the advances in software development over the past few decades have been in insulating the developer—that's you—from the nitty-gritty details of CPU instructions, hardware interfaces, and software design. This frees you to concentrate on harnessing these technologies to turn your idea into reality, rather than spending all of your time worrying about register allocations and memory management.

So, the exciting news is that you can jump right in and create full-featured iOS apps with only a minimal knowledge of computer programming or the underlying technologies that make them possible. And that's what this book is going to do in the first couple of chapters—show you how to create an iOS app without any traditional programming whatsoever.

That's not to say you don't need these skills in order to master iOS development. On the contrary; the more you know about programming, performance, and memory management, the more proficient you're going to be. What's changed is that these skills aren't the prerequisites that they once were. Now, you can learn them in parallel while you explore new avenues of iOS development.

How to Use This Book

This book embraces an "explore as go" approach. Some chapters will walk you through the process of creating an iOS app that uses the camera or plays music. These chapters may gloss over many of the finer details. In between, you'll find chapters on basic software development skills. There are chapters on good software design, making the most of sample code and the Swift programming language.

So, instead of the "traditional" order of first learning all of the basic skills and then building apps using those skills, this book starts out building apps and then explores the details of how that happened.

You can read the chapters in any order, skipping or returning to chapters as you need. If you really want to know more about objects in an earlier chapter, jump ahead and read the chapter on objects. If you're already familiar with the Model-View-Controller design pattern, skip that chapter when you get to it. Treat this book as a collection of skills to learn, not a series of lessons that have to be taken in order.

Here's a preview of the chapters ahead:

- *Got Tools?* shows you how to download and install the Xcode development tools. You'll need those.
- *Boom! App* walks you through the core steps for creating an iOS app—no programming needed.
- *Spin a Web* creates an app that leverages the power of iOS's built-in web browser.
- *Coming Events* discusses how events (touches, gestures, and movement) get from the device into your app and how you use them to make your app respond to the user.
- *Table Manners* shows you how data gets displayed in an app and how it gets edited.
- *Object Lesson* dishes the straight dope on objects and object-oriented programming.
- *Smile!* shows you how to integrate the camera and photo library into your app.
- *Model Citizen* explains the magic incantation that software engineers call Model-View-Controller.
- *Sweet, Sweet Music* jazzes up your mix by showing you how to add music, sounds, and iTunes to your apps.
- *Got Views?* takes you on a brief survey of the view objects (buttons, sliders, and so on) available in the Cocoa Touch framework.
- *Draw Me a Picture* shows you how to create your own views, unlocking the power to draw just about anything in an iOS app.
- *There and Back Again* lays out the basics of app navigation: how your users get from one screen to another and back again.
- *Sharing Is Caring* is all about getting your content out to the 'net, through services such as Twitter, SMS, and e-mail.
- *Game On!* dishes out a fun game with real-time animation.
- *If You Build It . . .* explains some of the magic behind Interface Builder.
- *Apps with Attitude* shakes up your apps with the accelerometer.

- *Where Are You?* draws you a map—literally.
- *Remember Me?* shows you how user preferences are set and saved and how to share them with other iOS devices using iCloud.
- *Doc, You Meant Storage* explains how app documents are stored, read, and updated.
- *See Swift, See Swift Run* is a crash course on the Swift programming language.
- *Frame Up* escapes the confines of your app to create an extension and teaches you a little about frameworks in the process.

Chapter 1

Got Tools?

If you want to build something, you are probably going to need some tools: hammer, nails, laser, crane, and one of those IKEA hex wrenches. Building iOS apps requires a collection of tools called *Xcode*.

This chapter will show you how to get and install Xcode and give you a brief tour of it, so you'll know your way around. If you've already installed and used Xcode, check the "Requirements" section to make sure you have everything you need, but you can probably skip most of this chapter.

Requirements

In this book, you will create apps that run on iOS version 8. Creating an app for iOS 8 requires Xcode version 6. Xcode 6 requires OS X version 10.9 (a.k.a. Mavericks), which requires an Intel-based Mac. Did you get all of that? Here's your complete checklist:

- Intel-based Mac
- OS X 10.9 (or newer)
- A few gigabytes of free disk space
- An Internet connection
- At least one iOS device (iPad Touch, iPhone, or iPad) running iOS 8.0 (or newer)

Make sure you have an Intel-based Mac computer with OS X 10.9 (Mavericks), or newer, installed, enough disk space, and an Internet connection. You can do all of your initial app development right on your Mac, but at some point you'll want to run your apps on a real iOS device (iPhone, iPod Touch, or iPad), and for that you'll need one.

> **Note** As a general rule, newer versions are better. The examples in this book were developed for iOS 8.0, built using Xcode 6.1, running on OS X 10.10 (Yosemite). By the time you read this, there will probably be a newer version of all of these, and that's OK. Read the Xcode and iOS release notes to see what has changed.

Installing Xcode

Apple has made installing Xcode as easy as possible. On your Mac, launch the App Store application—not to be confused with the App Store for iOS, which you find in iTunes. Find the Developer Tools category or just search for *Xcode*. Figure 1-1 show the Xcode app in the Mac App Store.

Figure 1-1. *Xcode in the App Store*

Click the Install button to start downloading Xcode. This will take a while (see Figure 1-2). You can monitor its progress from the Purchases tab of the App Store. Be patient. Xcode is huge, and even with a fast Internet connection, it will take some time to install.

***Figure 1-2.** Downloading Xcode*

While Xcode is downloading, let's talk about it and some related topics.

What Is Xcode?

So, what is this huge application you're downloading?

Xcode is an *integrated development environment* (IDE). Modern software development requires a dizzying number of different programs. To build and test an iOS app, you're going to need editors, compilers, linkers, syntax checkers, cryptographic signers, resource compilers, debuggers, simulators, performance analyzers, and more. But you don't have to worry about that; Xcode orchestrates all of those individual tools for you. All you have to do is use the Xcode interface to design your app, and Xcode will decide what tools need to be run and when. In other words, Xcode puts the *I* in IDE.

As well as including all of the tools you'll need, Xcode can host a number of *software development kits* (SDKs). An SDK is a collection of files that supply Xcode with what it needs to build an app for a particular operating system, like iOS 8. Xcode downloads with an SDK to build iOS apps and with an SDK to build OS X apps, for the most recent versions of each. You can download additional SDKs as needed.

An SDK will consist of one or more *frameworks*. A framework tells Xcode exactly how your application can use an iOS service. This is called an *application programming interface* (API). While it's possible to write code in your app to do just about anything, much of what it will be doing is making requests to iOS to do things that have already been written for you: display an alert, look up a word in the dictionary, take a picture, play a song, and so on. Most of this book will be showing you how to use those built-in services.

> **Note** A framework is a bundle of files in a folder, much like the app bundles you'll be creating in this book. Instead of containing an app, however, a framework contains the files your app needs to use a particular segment of the operating system. For example, all of the functions, constants, classes, and resources needed to draw things on the screen are in the Core Graphics framework. The AVFoundation framework contains classes that let you record and playback audio. Want to know where you are? You'll need the functions in the CoreLocation framework. There are scores of these individual frameworks.

Wow, that's a lot of acronyms! Let's review them.

- *IDE*: Integrated development environment. Xcode is an IDE.
- *SDK*: Software development kit. The supporting files that let you build an app for a particular operating system, like iOS 8.
- *API*: Application programming interface. A published set of functions, classes, and definitions that describe how your app can use a particular service.

You don't need to memorize these. It's just good to know what they mean when you hear them or talk to other programmers.

Becoming an iOS Developer

The fact that you're reading this book makes you an iOS developer—at least in spirit. To become an official iOS developer, you need to join Apple's iOS Developer program.

You must be an iOS developer if you want to do any of the following:

- Sell, or give away, your apps through Apple's App Store
- Gain access to Apple's Developer Forums and support resources
- Give your apps to people directly (outside of the App Store)
- Develop apps that use Game Kit, in-app purchases, push notifications, or other technologies that depend on Apple-operated services
- Test your apps on a real iOS device

The first reason is the one that prompts most developers to join the program and is probably the reason you'll join. You don't, however, have to join to build, test, and run your apps in Xcode's simulator. If you never plan to distribute your apps through the App Store or run your app on an iOS device, you may never need to become an official iOS developer. You can get through most of this book without joining.

Another reason for joining is to gain access to the iOS Developer's community and support programs. Apple's online forums contain a treasure trove of information. If you run into a problem and can't find the answer, there's a good chance someone else has already bumped into the same problem. A quick search of the Developer Forums will probably reveal an answer. If not, post your question and someone might have an answer for you.

Even if you don't plan to sell or give away your masterpiece on the App Store, there are a couple of other reasons to join. If you want to install your app on a device, Apple requires that you become a registered developer. Apple will then generate special files that will permit your app to be installed on an iOS device.

As a registered developer, Apple will also allow you to install your apps on someone else's device directly (that is, not through the App Store). This is called *ad hoc distribution*. There are limits on the number of people you can do this for, but it is possible, and it's a great way to test your app in the field.

Finally, some technologies require your app to communicate with Apple's servers. Before this is allowed, you must register yourself and your app with Apple, even just to test them. For example, if you plan to use Game Kit in your app—and this book includes a Game Kit example—you'll need to be an official iOS developer.

As I write this book, the cost of becoming an iOS developer is $99. It's an annual subscription, so there's no point in joining until you need to join. Go to `http://developer.apple.com/` to find more information about Apple's developer programs.

So, is there anything at `http://developer.apple.com/` that's free? There's quite a lot actually. You can search through all of Apple's published documentation, download example projects, read technology guides, find technical notes, and more—none of which requires you to be an iOS developer. Some activities may require you to log in with your Apple ID. The Apple ID you use with iTunes or your iCloud account will work, or you can create a new Apple ID for free.

Paid registration also gives you the opportunity to buy tickets to the World Wide Developers Conference (WWDC) held by Apple each year. It's a huge gathering, and it's just for Apple developers.

Getting the Projects

Now would be a good time to download the project files for this book. There are numerous projects used throughout this book. Many can be re-created by following the steps in each chapter, and I encourage you to do that whenever possible so you'll get a feel for building your apps from scratch. There are, however, a number of projects that don't explain every detail, and some projects include binary resources (image and sound files) that can't be reproduced in print.

Go to this book's page at `www.apress.com` (you can search for it by name, ISBN, or author name). Below the book's description, you'll see some folder tabs, one of which is labeled Source Code/ Downloads. Click that tab. Now find the link that downloads the projects for this book. Click that link and a file named `Learn iOS Development Projects.zip` will download to your hard drive.

Locate the file `Learn iOS Development Projects.zip` in your Downloads folder (or wherever your browser saved it). Double-click the file to extract its contents, leaving you with a folder named `Learn iOS Development Projects`. Move the folder wherever you like.

Launching Xcode the First Time

After the Xcode application downloads, you will find it in your `Applications` folder. Open the Xcode application, by double-clicking it, using Launchpad, or however you like to launch apps. I recommend adding Xcode to your Dock for easy access.

Xcode will present a licensing agreement (see Figure 1-3), which you are encouraged to at least skim over but must agree to before proceeding. Xcode then may then request an administrator's account and password to finish its installation. Once it has authorization, it will finish its installation, as shown on the right in Figure 1-3.

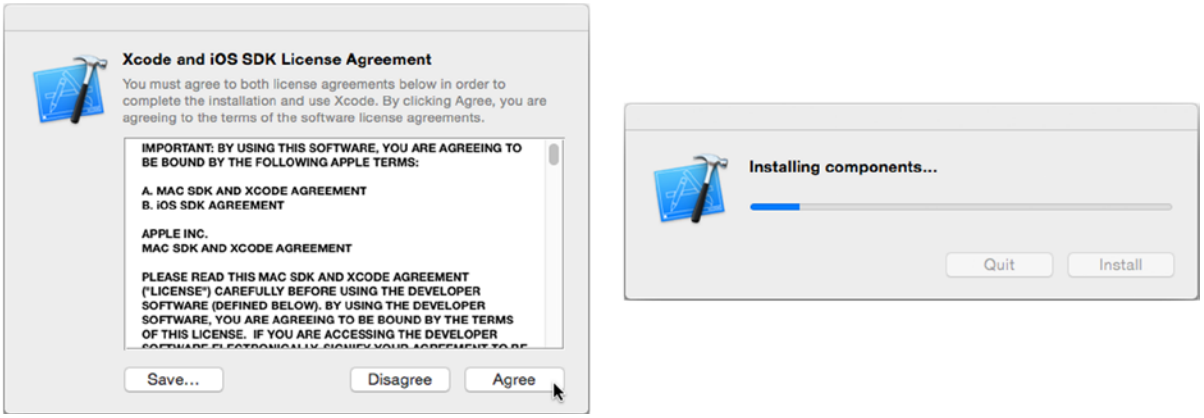


Figure 1-3. *License agreement*

Once you've gotten through all of the preliminaries and Xcode has put everything where it belongs, you'll see Xcode's startup window, as shown in Figure 1-4.

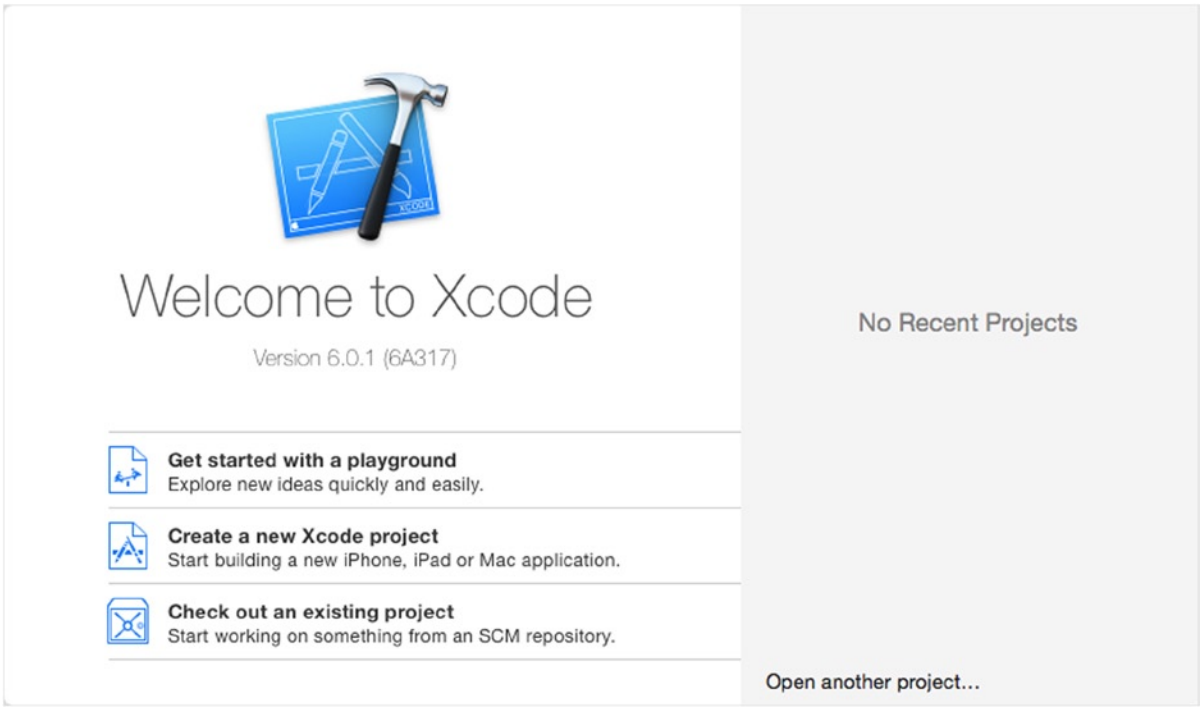


Figure 1-4. *Xcode's startup window*

The startup window has several self-explanatory buttons to help you get started. It also lists the projects you've recently opened.

A new feature of Xcode 6 is playgrounds. A *playground* is a blank page where you can try code in Swift, the language you'll be using in this book. You don't have to create a project or run a compiler; just type some Swift code and Xcode will show you what it did or why it didn't. You'll use playgrounds in Chapter 20 to explore the Swift language in detail, but don't be shy about creating a playground whenever you want to try some code.

The interesting parts of Xcode don't reveal themselves unless you have a project open, so start by creating a new project. Click the Create a new Xcode project button in the startup window (or choose File ➤ New ➤ Project from the menu). The first thing Xcode will want to know is what kind of project you want to create, as shown in Figure 1-5.

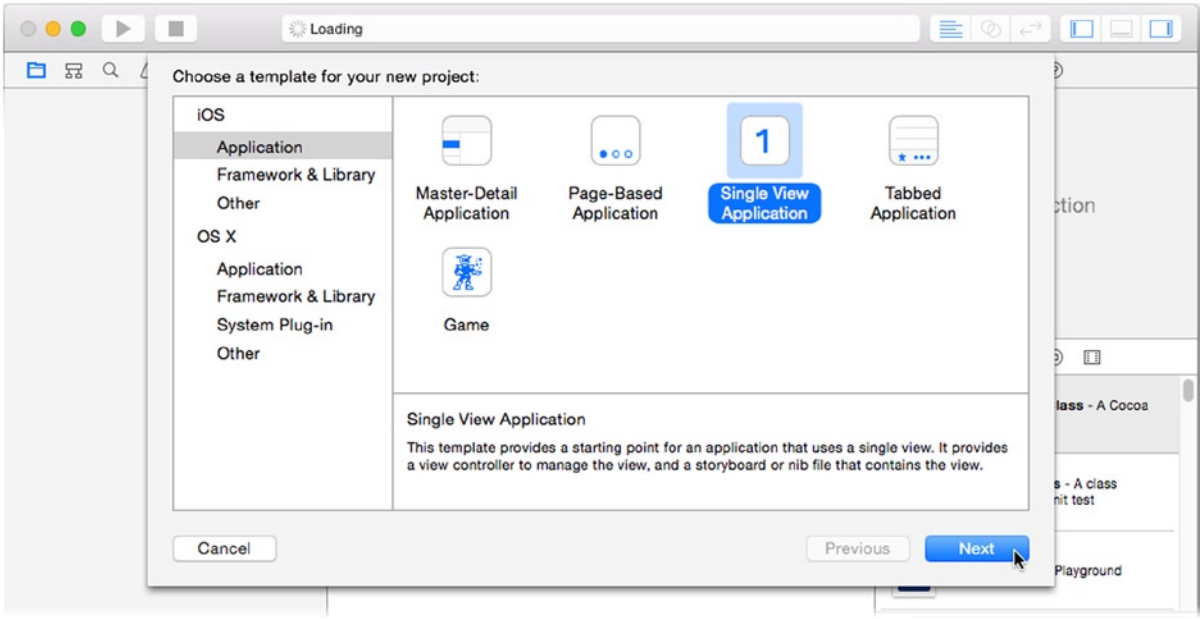


***Figure 1-5.** Project template browser*

The template browser lets you select a project template. Each template creates a new project preconfigured to build something specific (application, library, plug-in, and so on) for a particular platform (iOS or OS X). While it's possible to manually configure any project to produce whatever you want, it's both technical and tedious; save yourself a lot of work and try to choose a template that's as close to the final "shape" of your app as you can.

In this book, you'll only be creating iOS apps, so choose the Application category under the iOS section—but feel free to check out some of the other sections. As you can see, Xcode is useful for much more than just iOS development.

With the Application section selected, click the Single View Application template, and then click the Next button. In the next screen, Xcode wants some details about your new project, as shown in Figure 1-6. What options you see here will vary depending on what template you chose.

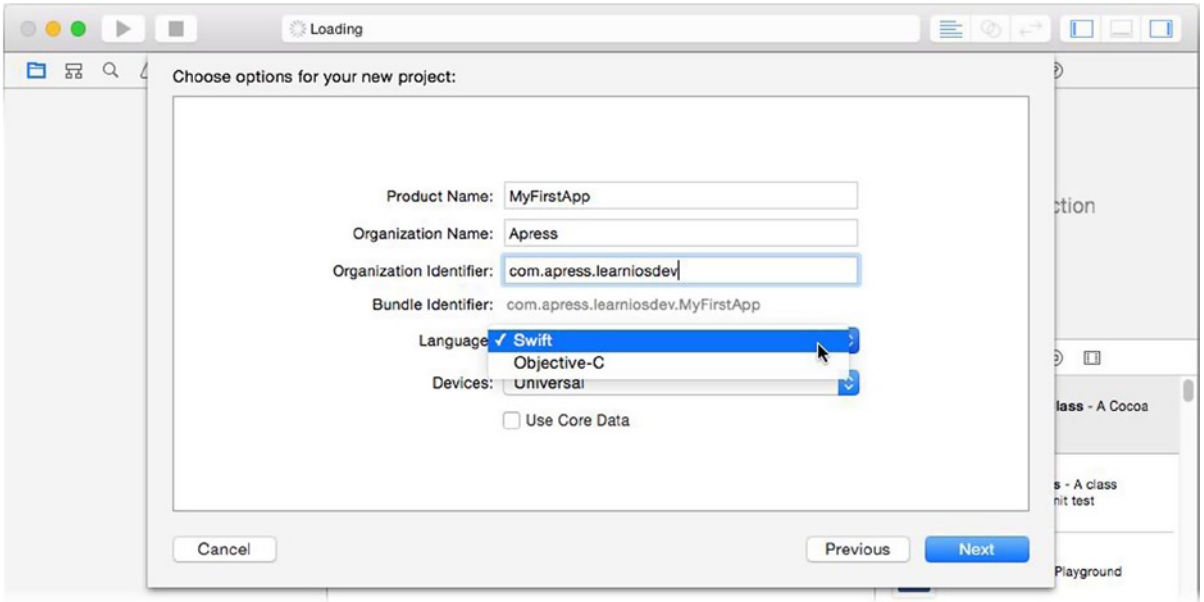


***Figure 1-6.** New project options*

For this little demonstration, give your new project a name in the Product Name field. It can be anything you want—I used `MyFirstApp` for this example—but I recommend you keep the name simple. The organization name is optional, but I suggest you fill in your name (or the company you're working for, if you're going to be developing apps for them).

The organization identifier and product name, together, create a *bundle identifier* that uniquely identifies your app. The organization identifier is a reverse domain name, which you (or your company) should own. It isn't important right now because you'll be building this app only for yourself, so use any domain name you like. When you build apps that you plan to distribute through the App Store, these values will have to be legitimate and unique.

Finally, choose Swift for the project's language. Objective-C is the traditional language for iOS (and most OS X) development. At the 2014 World Wide Developers Conference, Apple unveiled the Swift language. Swift is a, highly efficient, succinct computer language with a lot of features that promote effortless, bug-free development. Swift is the future of iOS development, and it's the language you'll be using throughout this book.

> **Note** While you've chosen Swift as the project's language, you're not limited to just Swift. Xcode and iOS can seamlessly mix Swift, Objective-C, C++, and C in the same project. You can always add Objective-C to a Swift project, and vice versa.

The rest of the options don't matter for this demonstration, so click the Next button. The last thing Xcode will ask is where to store your new project (see Figure 1-7) and if you want to create a source control repository. Source control is a way to maintain the history of your project. You can later go back and review what changes you've made. Xcode's preferred source control system is Git, and Xcode will offer to create a local Git repository for your project. If you don't know what to do, say "yes"; it doesn't really cost anything, and it's easier to do now than later. Xcode also supports remote Git repositories as well older source control systems, like Subversion. Now let's get back to creating that project.

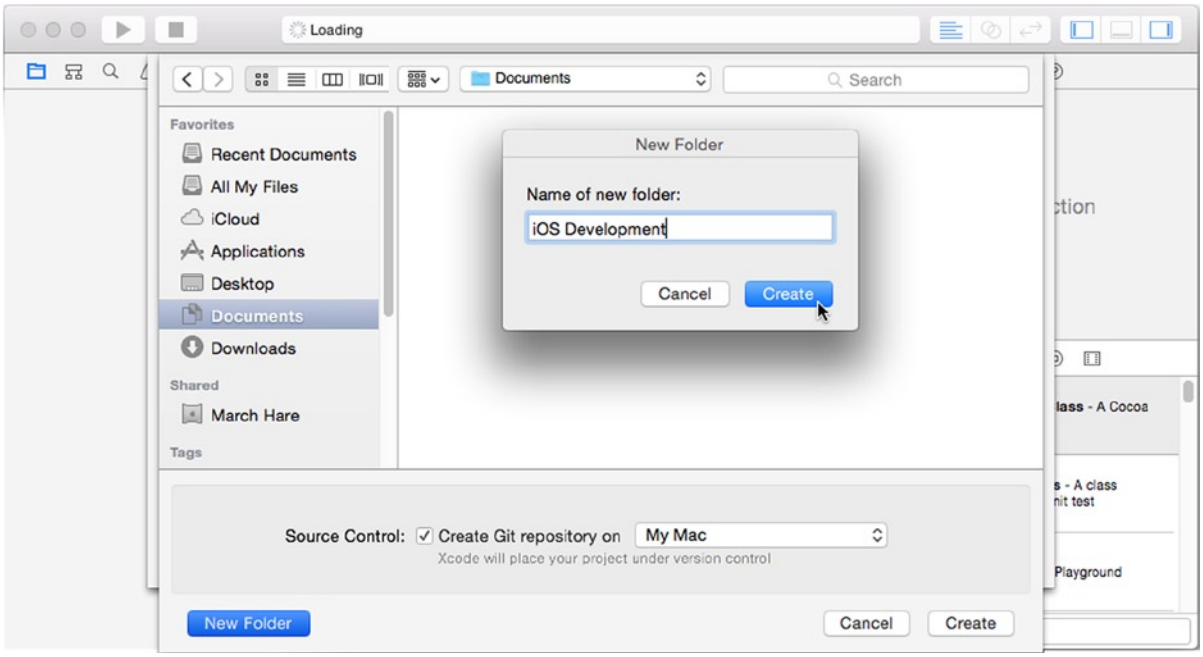


***Figure 1-7.** Creating a new project*

Every project creates a *project folder*, named after your project. All of the documents used to create your app will (should) be stored in that project folder. You can put your project folder anywhere (even on the Desktop). In this example (see Figure 1-7), I'm creating a new iOS Development folder so that I can keep all of my project folders together.

Welcome to Xcode

With all of the details about your new project answered, click the Create button. Xcode will create your project and open it in a *workspace window*. Figure 1-8 shows an exploded view of a workspace window. This is where the magic happens and where you'll be spending most of your time in this book.

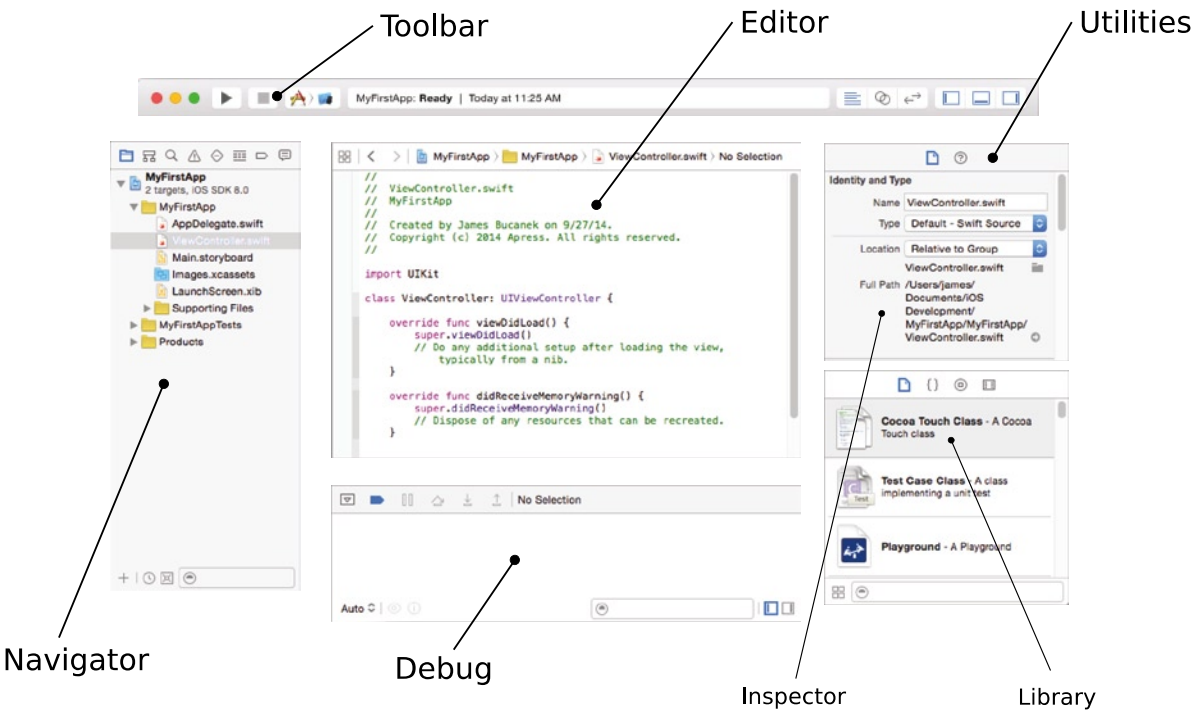


***Figure 1-8.** Xcode workspace window*

A workspace window has five main parts.

- Navigator area (left)
- Editor area (center)
- Utility area (right)
- Debug area (bottom)
- Toolbar (top)

You can selectively hide everything except the editor area, so you may not see all of these parts. Let's take a brief tour of each one, so you'll know your way around.

Navigation Area

The navigators live on the left side of your workspace window. There are eight navigators:

- Project
- Symbol
- Find
- Issue

- Test
- Debug
- Breakpoint
- Report

Switch navigators by clicking the icons at the top of the pane or from the View ➤ Navigator submenu. You can hide the navigators using the View ➤ Navigator ➤ Hide Navigator command (Command+0) or by clicking the left side of the View button in the toolbar (as shown on the right in Figure 1-9). This will give you a little extra screen space for the editor.

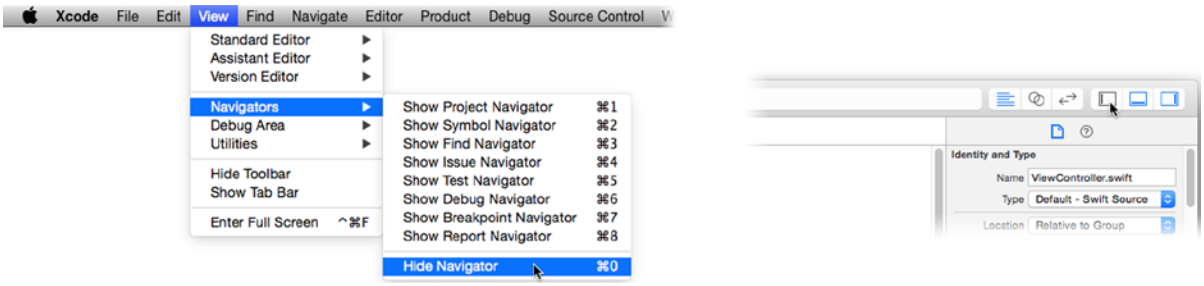


***Figure 1-9.** Navigator view controls*

The project navigator (see Figure 1-8) is your home base and the one you'll use the most. Every source file that's part of your project is organized in the project navigator, and it's how you select a file to edit.

> **Note** A *source file* is any original document used in the creation of your app. Most projects have multiple source files. The term is used to distinguish them from *intermediate files* (transient files created during construction) and *product files* (the files of your finished app). Your product files appear in a special `Products` folder, at the bottom of the project navigator.

The symbol navigator keeps a running list of the symbols you've defined in your project. The search navigator will find text in multiple files. The issues, debug, breakpoint, and report navigators come into play when you're ready to build and test your app.

Editor Area

The editor area is where you create your app—literally. Select a source file in the project navigator, and it will appear in the editor area. What the editor looks like will depend on what kind of file it is.

> **Note** Not all files are editable in Xcode. For example, image and sound files can't be edited in Xcode, but Xcode will display a preview of them in the editor area.

What you'll be editing the most are program source files, which you edit like any text file (see Figures 1-8 and 1-10), and Interface Builder files, which appear as graphs of objects (see Figure 1-11) that you connect and configure.

The editor area has three modes.

- Standard editor
- Assistant editor
- Version editor

The standard editor edits the selected file, as shown in Figure 1-10. The assistant editor splits the editor area and (typically) loads a *counterpart* file on the right side. For example, you can edit your interface design in the left pane and preview what that interface will look like on different devices in the right pane. When editing a Swift source file, you can elect to have the source for its superclass automatically appear in the right, and so on.

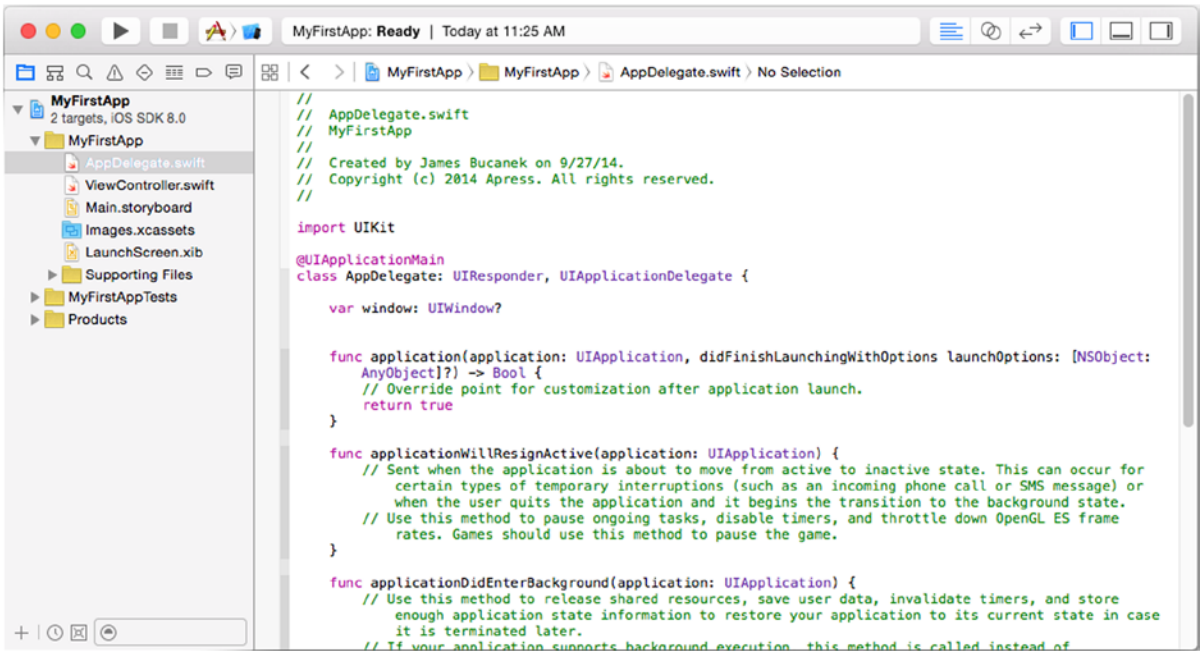


Figure 1-10. *The editor*

The version editor is used to compare a source file with an earlier version. Xcode supports several version control systems. You can check in files into your version control system or take a "snapshot" of your project. You can later compare what you've written against an earlier version of the same file. We won't get into version control in this book. If you're interested, read the section "Save and Revert Changes to Projects" in the Xcode Users Guide.

To change editor modes, click the Editor control in the toolbar or use the commands in the View menu. You can't hide the editor area.

Utility Area

On the right side of your workspace window is the utility area. As the name suggests, it hosts a variety of useful tools, as shown on the right in Figure 1-11.

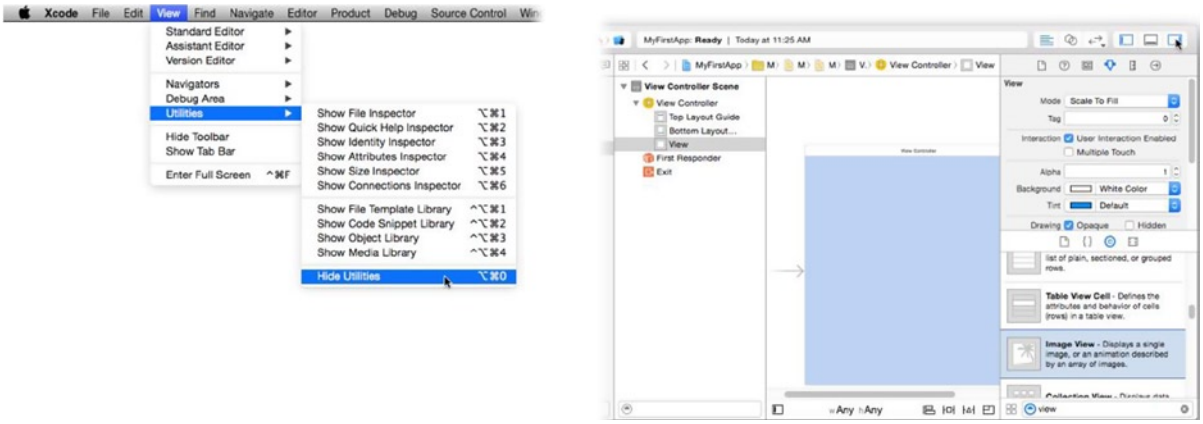


Figure 1-11. *Editing an Interface Builder file*

At the top of the utilities area are the *inspectors*. These will change depending on what kind of file is being edited and possibly what you have selected in that file. As with the navigators, you can switch between different inspectors by clicking the icons at the top of the pane or from the View ➤ Utilities submenu (shown on the left of Figure 1-11). You can hide the utility area using the View ➤ Utilities ➤ Hide Utilities command or by clicking the right side of the View control in the toolbar (also shown at the upper right of Figure 1-11).

At the bottom of the utility area is the library. Here you'll find ready-made objects, resources, and code snippets that you can drag into your project. These too are organized into tabs by type: file templates, code snippets, interface objects, and media assets.

Debug Area

The debug area is used to test your app and work out any kinks. It usually doesn't appear until you run your app—and isn't much use until you do. To make it appear, or disappear, use the View ➤ Debug Area ➤ Show/Hide Debug Area command. You can also click the close drawer icon in the upper-left corner of the debug pane.

Toolbar

The toolbar contains a number of useful shortcuts and some status information, as shown in Figure 1-12.

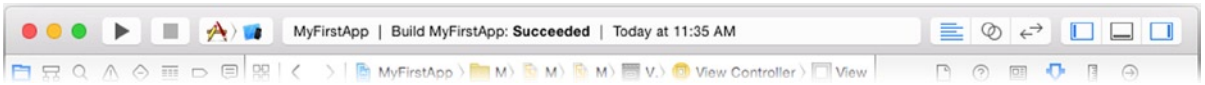


Figure 1-12. Workspace window toolbar

You've already seen the Editor and View buttons on the right. On the left are buttons to run (test) and stop your app. You will use these buttons to start and stop your app during development.

Next to the Run and Stop buttons is the Scheme control. This multipart pop-up menu lets you select how your project will be built (called a *scheme*) and your app's destination (a simulator, an actual device, the App Store, and so on).

In the middle of the toolbar is your project's status. It will display what activities are currently happening, or have recently finished, such as building, indexing, and so on. If you've just installed Xcode, it is probably downloading additional documentation in the background, and the status will indicate that.

You can hide the toolbar, if you want, using the View ➤ Show/Hide Toolbar command. All of the buttons and controls in the toolbar are just shortcuts to menu commands, so it's possible to live without it. This book, however, will assume that it's visible.

If you're interested in learning more about the workspace window, the navigators, editor, and inspectors, you will find all of that (and more) in the Xcode Overview, under the Help menu.

Running Your First App

With your workspace window open, click on the Scheme control and choose one of the iPhone choices from the submenu, as shown in Figure 1-13. This tells Xcode where you want this app to run when you click the Run button.

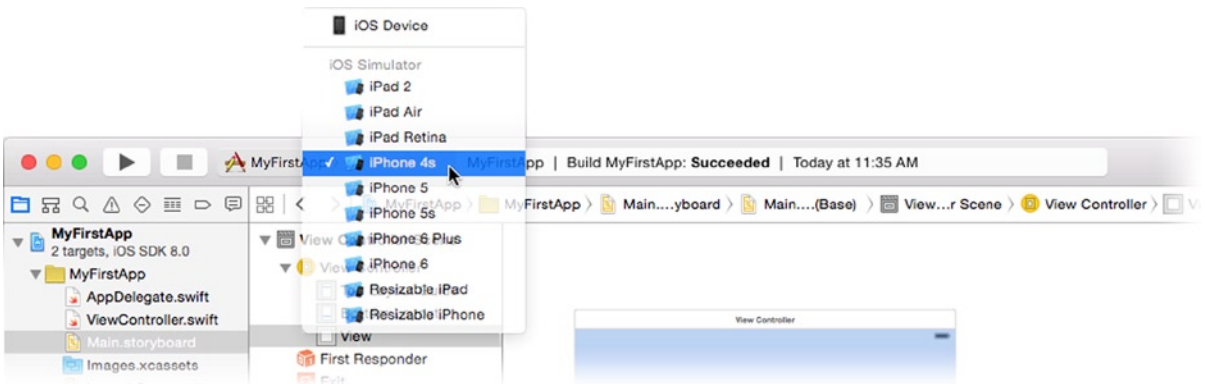


Figure 1-13. Choosing the scheme and target

Click the Run button. OK, there's probably one more formality to attend to. Before you can test an application, Xcode needs to be granted some special privileges. The first time you try to run an app, Xcode will ask if it can have those (see Figure 1-14). Click Enable and supply your administrative account name and password.

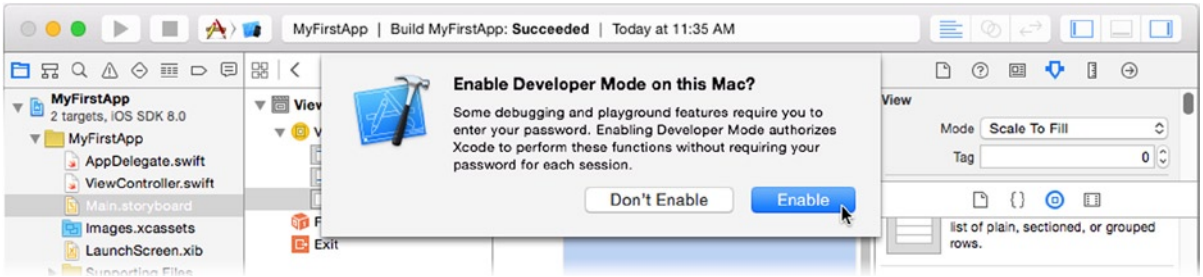


***Figure 1-14.** Enabling developer mode*

Once you're past the preliminaries, Xcode will assemble your app from all of the parts in your project—a process known as a *build*—and then run your app using its built-in iPhone simulator, as shown on the left in Figure 1-15.

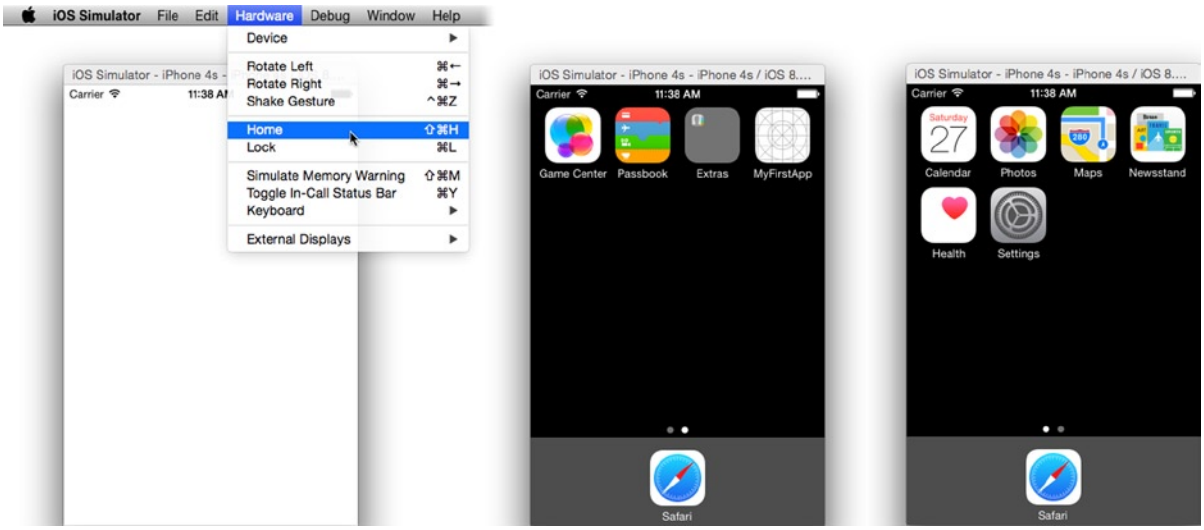


***Figure 1-15.** The iPhone simulator*

The simulator is just what it sounds like. It's a program that pretends—as closely as possible—to be a real iPhone or iPad. The simulator lets you do much of your iOS app testing right on your Mac, without ever having to load your app into a real iOS device. It also allows you to test your app on different kinds of devices, so you don't have to go buy one of each.

Congratulations, you just created, built, and ran an iOS app on a (simulated) iPhone! This works because Xcode project templates always create a runnable project; what's missing is the functionality that makes your app do something wonderful. That's what the rest of this book is about.

While you're here, feel free to play around with the iPhone simulator. Although the app you created doesn't have any functionality—beyond that of a lame "flashlight" app—you'll notice that you can simulate pressing the home button using the Hardware ➤ Home command (shown on the left in Figure 1-15) and return to the springboard (the middle and right in Figure 1-15). There you'll find your new app, the Settings app, Game Center, and more, just as if this were a real iPhone. Sorry, it won't make telephone calls.

When you're finished, switch back to the workspace window and click the Stop button (next to the Run button) in the toolbar.

Summary

You now have all of the tools you need to develop and run iOS apps. You've learned a little about how Xcode is organized and how to run your app in the simulator.

The next step is to add some content to your app.

Chapter 2

Boom! App

In this chapter you'll create an iOS app that does something. Not much—these are early days—but enough to call it useful. In the process, you will do the following:

- Use Xcode's Interface Builder to design your app
- Add objects to your app
- Connect objects
- Customize your objects to provide content
- Add resource files to your project
- Use storyboards to create segues
- Control the layout of visual elements using constraints

Amazingly, you're going to create this app without writing a single line of computer code. This is not typical, but it will demonstrate the flexibility of Xcode.

The app you're going to create presents some interesting facts about women surrealists of the 20th century. Let's get started.

Design

Before firing up Xcode and typing furiously, you need to have a plan. This is the design phase of app development. Over the lifetime of your app, you may revise your design several times as you improve it, but before you begin you need a basic idea of what your app will look like and how you want it to work.

Your design may be written out formally, sketched on a napkin, or just be in your head. It doesn't matter, as long as you have one. You need to, at the least, be able to answer some basic questions. What kinds of devices will your app run on (iPhone/iPod, iPad, or both)? Will your app run in portrait mode, sideways, or both? What will the user see? How will the user navigate? How will they interact with it?

Figure 2-1 shows a rough sketch of this app. The app is simple, so it doesn't require much in the way of initial design. The surrealist app will have an opening screen containing portraits of famous women surrealists. Tapping one will transition to a second screen showing a representative painting and a scrollable text field with information about the artist's life. You've decided this is going to run only on an iPhone or iPod Touch and only in portrait orientation. This will simplify your design and development.

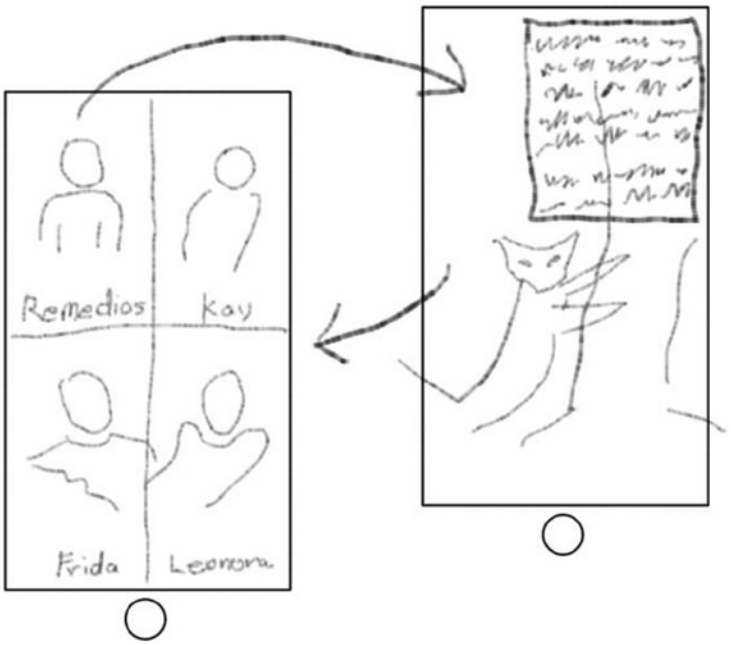


Figure 2-1. *Sketch of Surrealist app*

Creating the Project

The first step is to create your project. Click the New Project button in the startup window or choose the File ➤ New Project command. Review the available templates, as shown in Figure 2-2.

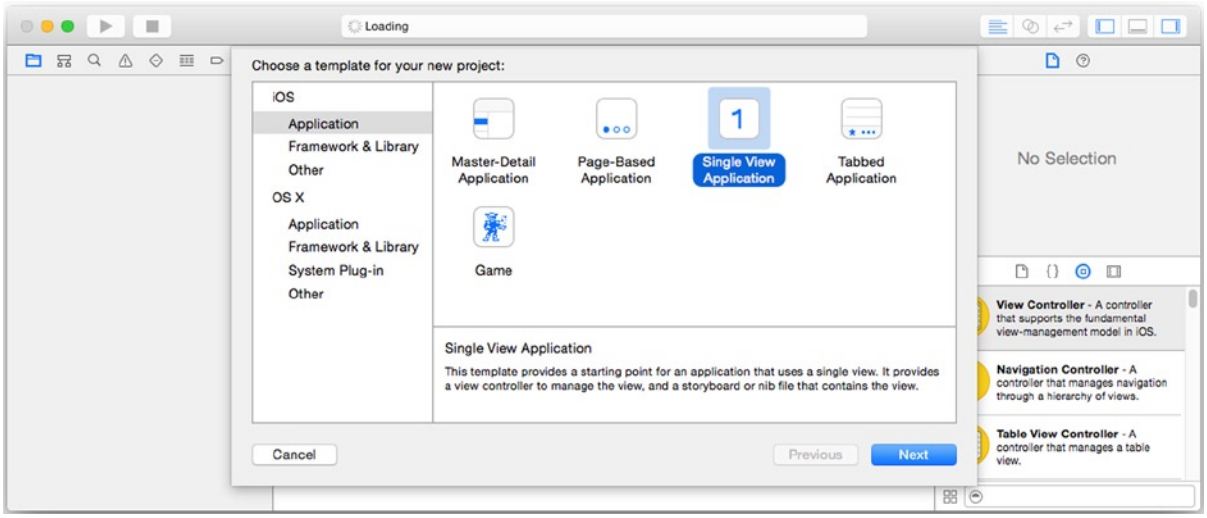


Figure 2-2. iOS project templates

Your design gives you a basic idea of how your app will work, which should suggest which Xcode project template to start with. Your app's design isn't a perfect fit with any of these, so choose the Single View Application template—it's the simplest template that already has a view. Click the Next button.

The next step is to fill in the details about your project (see Figure 2-3). Name the project Surrealists and fill in your organization name and identifier. Consistent with your design choices, change the Devices option from Universal to iPhone, as shown in Figure 2-3. The choice of language doesn't matter since, as I already mentioned, you won't be writing any code for this app.

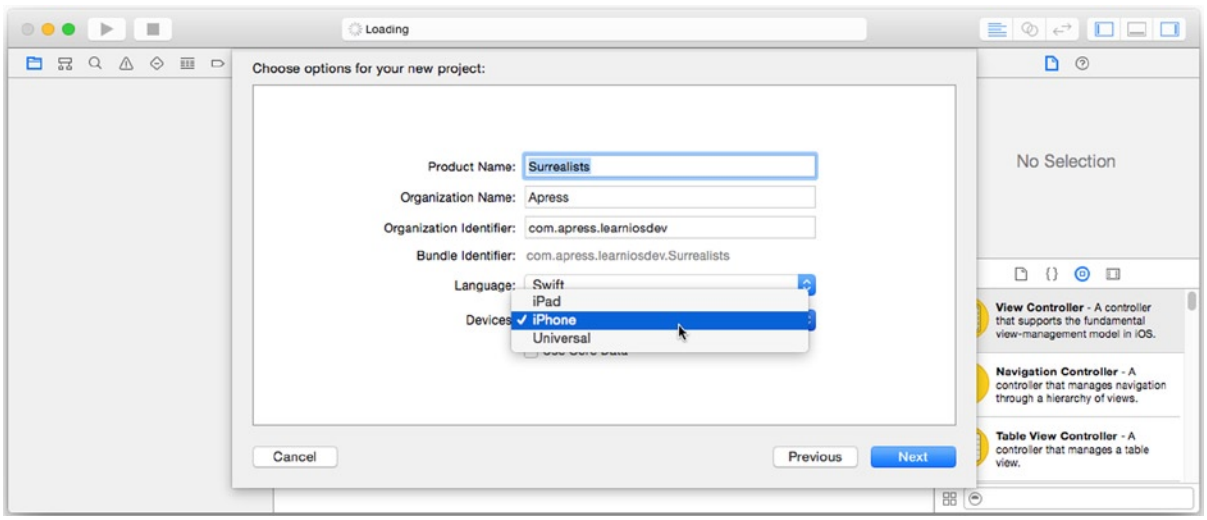


Figure 2-3. Setting the project details

> **Note** Developing for the iPhone is the same as developing for the iPod Touch (unless your app uses features available only on the iPhone). From here on, I'll mention the iPhone only, but please remember that this also includes the iPod Touch.

Click the Next button. Pick a location on your hard drive to save the new project and click Create.

Setting Project Properties

You now have an empty Xcode project; it's time to start customizing it. Begin with the project settings by clicking the project name (Surrealists) in the project navigator, as shown in the upper left of Figure 2-4. The editor area will display all of the settings for this project. If your project's targets are collapsed, pick the Surrealist target from the pop-up menu, in the upper-left corner of the editor (see the left side of Figure 2-4). If your project's targets are expanded, simply select the target from the list, as shown in the middle of Figure 2-4. Once you've picked the target, select the General tab in the middle, as shown on the right in the same figure.

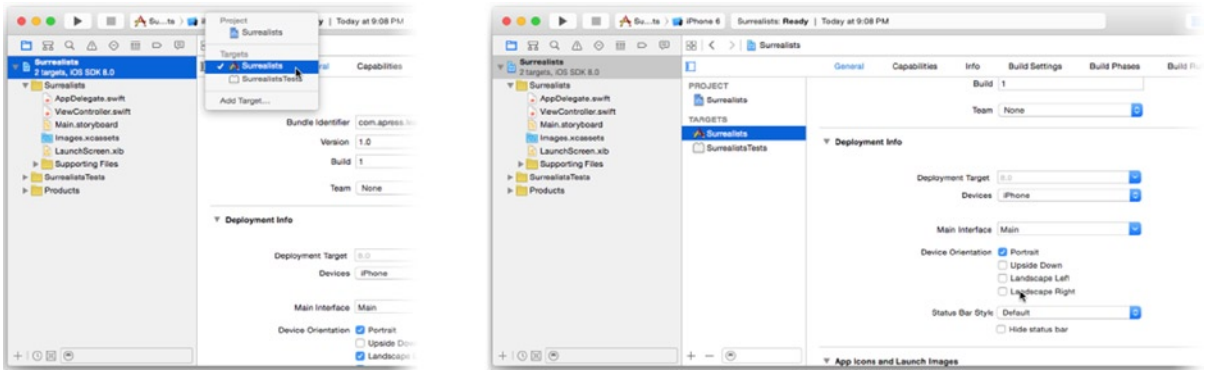


Figure 2-4. *Target settings*

Scroll down the target settings until you find the Deployment Info section. Uncheck the Landscape Left and Landscape Right boxes in Device Orientation so that only the Portrait orientation is checked.

To review, you've created an iPhone-only app project that runs exclusively in portrait orientation. You're now ready to design your interface.

Building an Interface

Click the `Main.storyboard` file in the project navigator. Xcode's Interface Builder editor appears in the edit area, as shown in Figure 2-5.

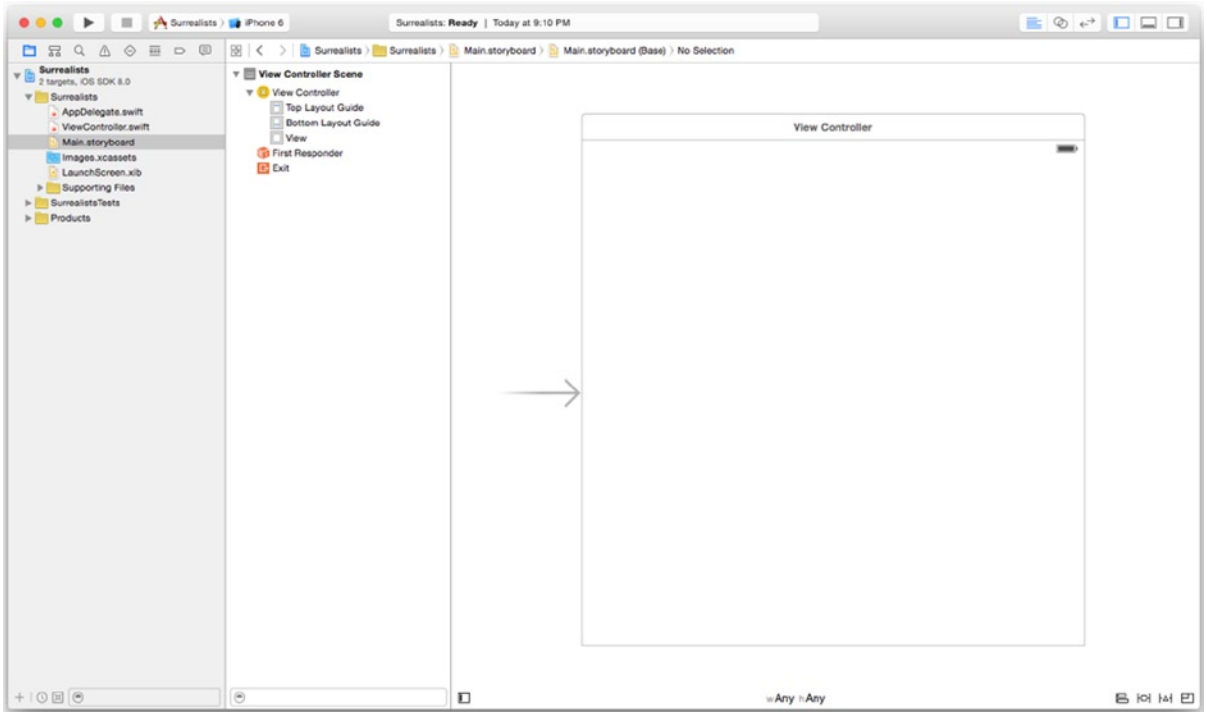


Figure 2-5. *Interface Builder*

Interface Builder is the secret sauce in Apple's app kitchen. In a nutshell, it's a tool that adds, configures, and interconnects objects within your app—without writing any code. You can define most of the visual elements of your app in Interface Builder. Interface Builder edits storyboard, xib, and (legacy) nib files.

> **Note** Modern Interface Builder files have extensions of `.xib` or `.storyboard`. Legacy Interface Builder files have a `.nib` (pronounced "nib") extension, and you'll still hear programmers refer to all of them generically as "nib" files. The NIB acronym stands for Next Interface Builder because the roots of Xcode, Interface Builder, and the Cocoa Touch framework stretch all the way back to Steve Job's "other" company, NeXT. Later in this book, you'll see a lot of class names that begin with `NS`, which is an abbreviation for NeXTStep, the name of NeXT's operating system.

Interface Builder displays the objects in the file in two views. On the left (see Figure 2-5) are the objects organized into a hierarchical list, called the *outline*. Some objects can contain other objects, just as folders can contain other folders, and the outline reflects this. Use the disclosure triangles to reveal contained objects.

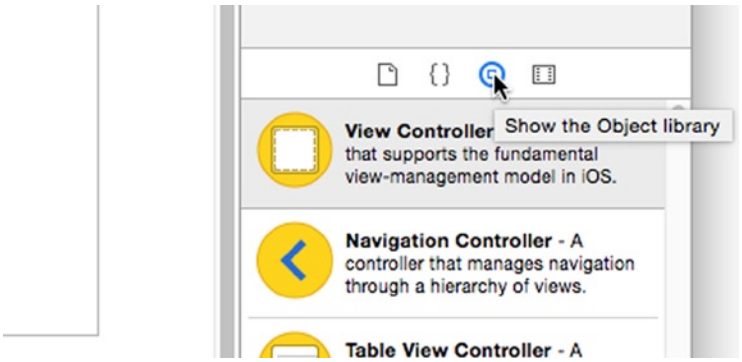
> **Tip** The outline can be collapsed to a *dock* of just the top-level object icons. Click the expand button, in the lower left of the canvas pane (bottom center in Figure 2-5), to toggle between the two views.

The view on the right is called the *canvas*. Here you'll find the visual objects in your Interface Builder file. Only visual objects (such as buttons, labels, images, and so on) appear in the canvas. Objects that don't have a visual aspect will be listed only the outline. If an object appears in both, it doesn't matter which one you work with—they're the same object.

> **Note** If you've been learning an object-oriented programming language, then you know what an object is. If you don't know what an object is, don't panic. For now, just think of objects as Lego bricks—a discrete bundle that performs a specific task in your app and can be connected to others to make something bigger. Feel free to skip ahead to Chapter 6 if you want to learn about objects right now.

Adding Objects

You get new objects from the library. Choose the View ➤ Utilities ➤ Show Object Library command. This will simultaneously make the utility area on the right visible and switch to the object library (the little round thing), like this:



To add an object to your app, drag it from the library and drop it into the Interface Builder editor. Your app needs a navigation controller object, so scroll down the list of objects until you find the Navigation Controller. You can simplify your search by entering a term (like *nav*) into the search field at the bottom of the library pane (see Figure 2-6).