

iPhone, iPad, and Mac Programming Made Easy



Swift

for Absolute Beginners

Gary Bennett | Brad Lees

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.



Contents at a Glance

About the Authors.....	xv
About the Technical Reviewer	xvii
Acknowledgments	xix
Introduction	xxi
■ Chapter 1: Becoming a Great iOS Developer	1
■ Chapter 2: Programming Basics.....	11
■ Chapter 3: It's All About the Data	23
■ Chapter 4: Making Decisions, Program Flow, and App Design.....	37
■ Chapter 5: Object-Oriented Programming with Swift.....	61
■ Chapter 6: Learning Swift and Xcode	79
■ Chapter 7: Swift Classes, Objects, and Methods	101
■ Chapter 8: Programming Basics in Swift	125
■ Chapter 9: Comparing Data	151
■ Chapter 10: Creating User Interfaces	167

■ Chapter 11: Storing Information..... 189

■ Chapter 12: Protocols and Delegates 217

■ Chapter 13: Introducing the Xcode Debugger 231

■ Chapter 14: A Swift iPhone App 249

Index..... 269

Introduction

Over the past three years, we've heard the following countless times:

- “I've never programmed before, but I have a great idea for an iPhone/iPad app.”
- “Can I really learn to program the iPhone or iPad?”

To the latter we answer, “Yes, but you have to believe you can.” Only you are going to tell yourself you can't do it.

For the Newbie

This book assumes you may have never programmed before. The book is also written for someone who may have programmed before but never using object-oriented programming (OOP) languages. There are several Swift books out there, but all of these books assume you have programmed before and know OOP and computer logic. We wanted to write a book that takes readers from knowing little or nothing about computer programming and logic to being able to program in Swift. After all, Swift is a native programming language for the iPhone, iPad, and Mac.

Over the past six years, we have taught thousands of students at xcelMe.com to be iPhone/iPad (iOS) developers. Many of our students have developed some of the most successful iOS apps in their category in the iTunes App Store. We have incorporated what we have learned in our first two courses, Introduction to Object-Oriented Programming and Logic and Swift for iPhone/iPad Developers, into this book.

For the More Experienced

Many developers who programmed years ago or programmed in a non-OOP language need a background in OOP and logic before they dive into Swift. This book is for you. We gently walk you through OOP and how it is used in iOS development to help make you a successful iOS developer.

How This Book Is Organized

You'll notice that we are all about successes in this book. We introduce the OOP and logic concepts in playgrounds and then move those concepts to Xcode and Swift. Many students are visual learners or learn by doing. We use both techniques. We'll walk you through topics and concepts with visual examples and then take you through step-by-step examples that reinforce the concepts.

We often repeat topics in different chapters to reinforce what you have learned and apply these skills in new ways. This enables new programmers to reapply development skills and feel a sense of accomplishment as they progress. Don't worry if you feel you haven't mastered a topic. Keep moving forward!

The Formula for Success

Learning to program is an interactive process between your program and you. Just like learning to play an instrument, you have to practice. You must work through the examples and exercises in this book. Understanding the concept doesn't mean you know how to apply it and use it.

You will learn a lot from this book. You will learn a lot from working through the exercises in this book. However, you will really learn when you debug your programs. Spending time walking through your code and trying to find out why it is not working the way you want is an unparalleled learning process. The downside of debugging is that a new developer can find it frustrating. If you have never wanted to throw your computer out the window, you will. You will question why you are doing this and whether you are smart enough to solve the problem. Programming is humbling, even for the most experienced developer.

Like a musician, the more you practice, the better you get. By practicing, we mean programming! You can do some amazing things as a programmer. The world is your oyster. Seeing your app in the iTunes App Store is one of the most satisfying accomplishments. However, there is a price, and that price is time spent coding and learning.

Having taught many students to become iOS developers, we have put together a formula for what makes students successful. Here is our formula for success:

- Believe you can do it. You'll be the only one who says you can't do this. So, don't tell yourself that.
- Work through all the examples and exercises in this book.
- Code, code, and keep coding. The more you code, the better you'll get.
- Be patient with yourself. If you were fortunate enough to have been a 4.0 student who could memorize material just by reading it, this will not happen with Swift coding. You are going to have to spend time coding.
- You learn by reading this book. You really learn by debugging your code.
- Use the free xcelMe.com webinars and YouTube videos explained at the end of this introduction. The free live and recorded training videos will be invaluable in quickly becoming a successful iOS developer.
- Don't give up!

The Development Technology Stack

We will walk you through the development process for your iOS apps and what technology you need. However, briefly looking at all the technology pieces together is helpful. These are the key iOS development technologies you will need to know in order to build a successful app and get it on the App Store:

- Apple's developer website
- iTunes Connect
- Xcode
- Swift
- Object-oriented programming and logic
- Debugging
- Performance tuning

We know this is a lot of technology. Don't worry, we will go through it, and you will become comfortable using it.

Required Software, Materials, and Equipment

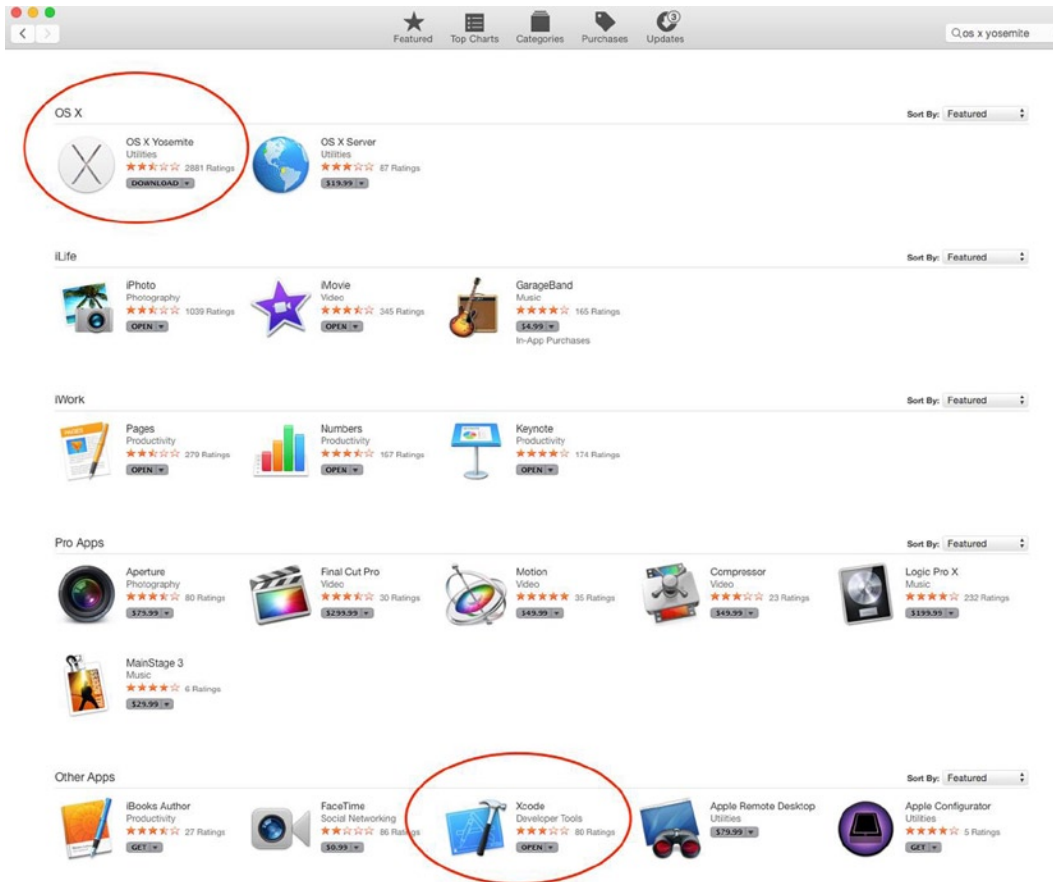
One of the great things about developing iOS apps is that everything you need to develop your app is free.

- Xcode
- Swift
- OSX 10.10 Yosemite
- Integrated development environment
- iPhone and iPad simulators

All you need to get started is a Mac and knowledge of where to download everything. We will cover this.

Operating System and IDE

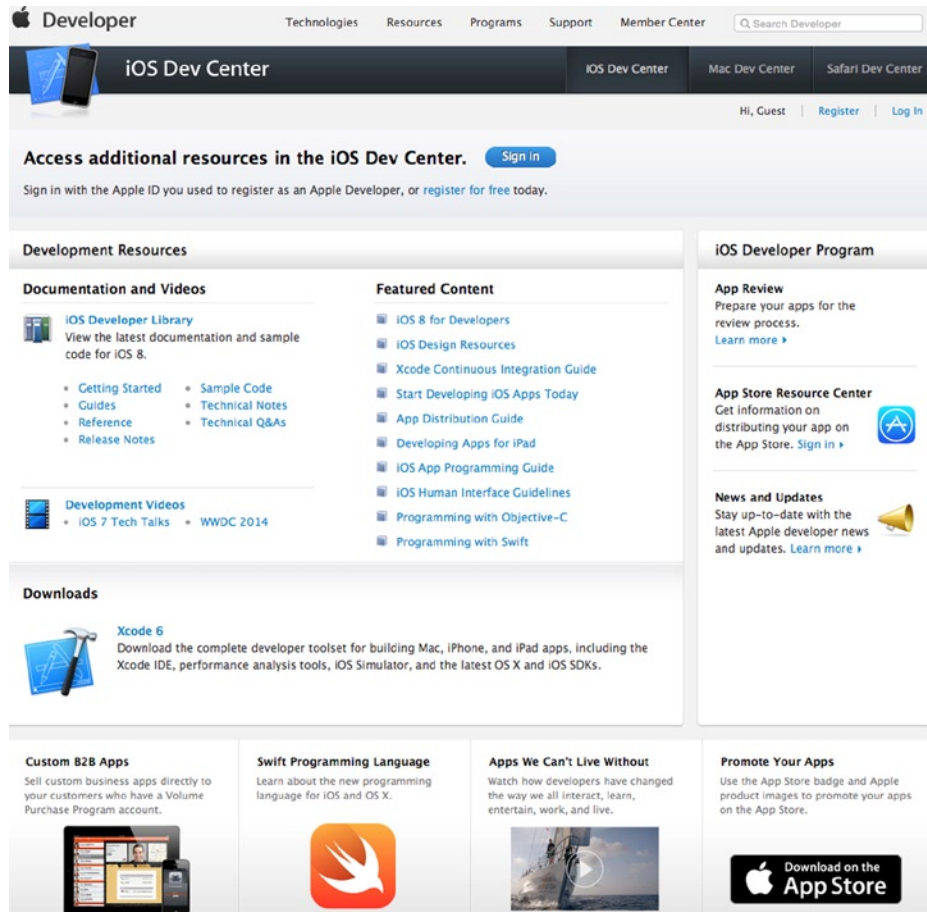
When developing iOS apps, you have to use Xcode and Mac OS X. You can download both of these for free from the Mac App Store.



Software Development Kits

You will need to register as an iOS developer. You can do this for free at <http://developer.apple.com/iphone>.

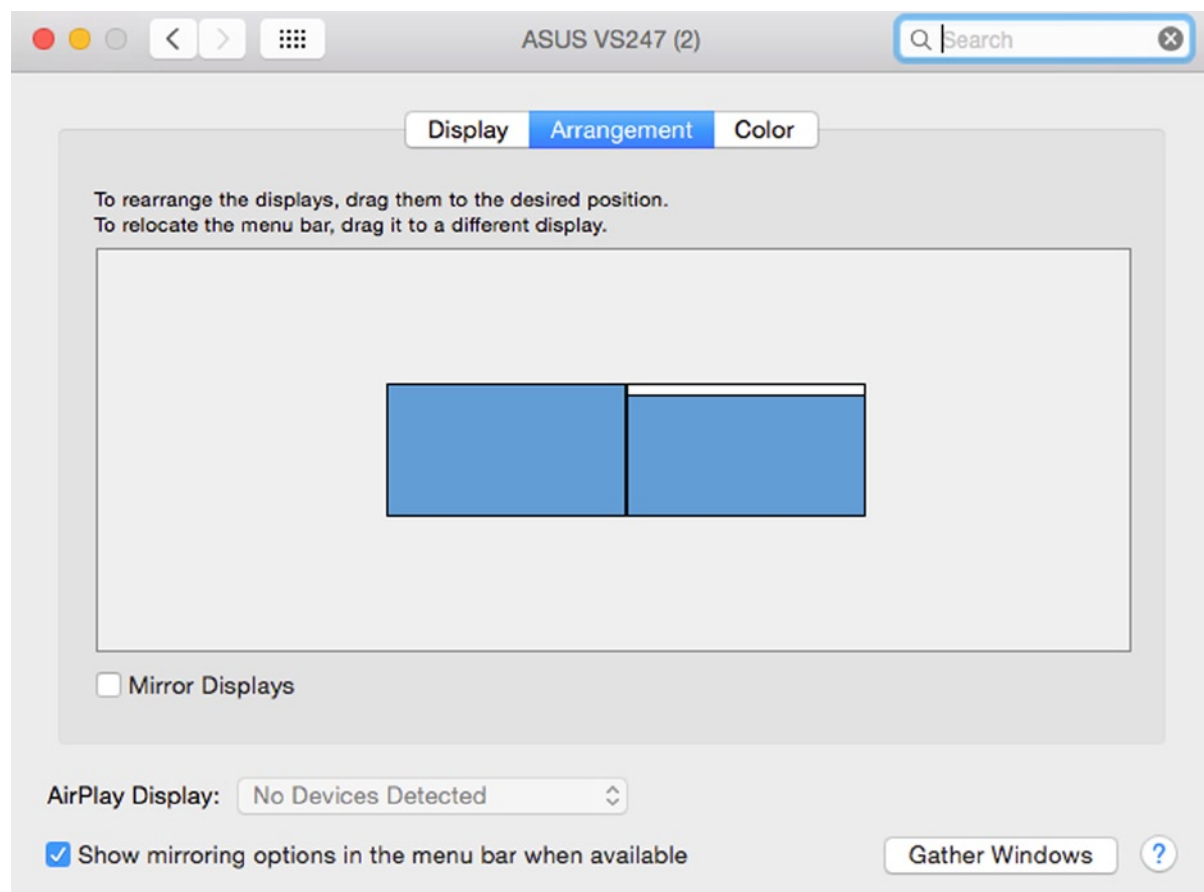
When you are ready to upload your app to the iTunes App Store, you will need to pay \$99 per year in order to access.



Dual Monitors

We recommend developers have a second monitor connected to their computers. It is great to step through your code and watch your output window and iOS simulator at the same time on dual independent monitors.

Apple hardware makes this easy. Just plug your second monitor into the display port of any Mac, with the correct Mini DisplayPort adapter, and you have two monitors working independently of one another. Note that dual monitors are not required. You will just have to organize your open windows to fit on your screen if you don't.



FREE LIVE WEBINARS, Q&A, AND YOUTUBE VIDEOS

Every Monday night at 5:30 p.m. Pacific time, we have live webinars and discuss a topic from the book or a timely item of interest. These webinars are free, and you can register for them at www.xcelme.com/latest-videos/.

LATEST VIDEOS

Home > Latest Videos

Free Swift iOS Webinars

Every Monday night at 5:30 PM Pacific time xcelMe.com is providing FREE webinars.

Gary Bennett discusses Swift, xCode, Interface Builder, iOS, Maker topics, and answers your programming questions. Webinars are recorded and available on his **YouTube channel**. Make sure you subscribe to his channel to be notified when new videos are uploaded.

To register for the FREE webinar, **click HERE**.

Once registered you will receive an email confirming registration with information you need to join the Webinar.

Upcoming Live Swift Tutorials

- Mon, Jan 12, 2015 5:30 PM – 5:45 PM PST Introduction and Chapter 1 – Using Swift Playgrounds
- Mon, Jan 19, 2015 5:30 PM – 5:45 PM PST Chapter 1 – More on Swift Playgrounds
- Mon, Jan 12, 2015 5:30 PM – 5:45 PM PST Chapter 2 – Programming Basics
- Mon, Jan 19, 2015 5:30 PM – 5:45 PM PST Chapter 3 – It's all About the Data
- Mon, Jan 26, 2015 5:30 PM – 5:45 PM PST Chapter 4 – Making Decisions, Program Flow, and App Design
- Mon, Feb 2, 2015 5:30 PM – 5:45 PM PST Chapter 5 – Object-Oriented Programming with Swift
- Mon, Feb 9, 2015 5:30 PM – 5:45 PM PST Chapter 6 – Learning Swift and Xcode
- Mon, Feb 16, 2015 5:30 PM – 5:45 PM PST Chapter 7 – Swift Classes, Objects, and Methods
- Mon, Feb 23, 2015 5:30 PM – 5:45 PM PST Chapter 8 – Programming Basics in Swift
- Mon, Mar 2, 2015 5:30 PM – 5:45 PM PST Chapter 9 – Comparing Data
- Mon, Mar 9, 2015 5:30 PM – 5:45 PM PDT Chapter 10 – Creating User Interfaces
- Mon, Mar 16, 2015 5:30 PM – 5:45 PM PDT Chapter 11 – Storing Information
- Mon, Mar 23, 2015 5:30 PM – 5:45 PM PDT Chapter 12 – Protocols and Delegates
- Mon, Mar 30, 2015 5:30 PM – 5:45 PM PDT Chapter 13 – Introducing the Xcode Debugger
- Mon, Apr 6, 2015 5:30 PM – 5:45 PM PDT Chapter 14 – A Swift iPhone App

At the end of the webinars, we do a Q&A. You can ask a question on the topic discussed or on any topic in the book.

Additionally, all these webinars are recorded and available on YouTube. Make sure you subscribe to the YouTube channel so you are notified when new recordings are uploaded.

Free Book Forum

We have developed an online forum for this book at <http://forum.xcelme.com>, where you can ask questions while you are learning Swift and get answers from the authors. Also, Apple makes frequent changes to the programming language and SDK. We try our best to make sure any changes affecting the book get updated on the forum along with any significant text or code changes.

You can download the source code from the chapters on this forum too.

FORUM	TOPICS	POSTS
How To Access Your Course Webinars And How To Use The Forum New students need to download the attached pdf and follow instructions to register for your webinars after you purchase the class. Additionally, there are directions and updates on how to access your course and forum, post questions, navigate the message board, watch training videos, etc. Moderator: gary.bennett	3	12
Book -> Swift for Absolute Beginners: iPhone and Mac Programming Made Easy This forum contains all the questions readers may have for each chapter and and chapter or code changes. Moderator: gary.bennett	16	16
Book -> Objective C for Absolute Beginners- (2nd Edition) iPhone and Mac Programming Made Easy This forum contains all the assignments and questions readers may have for each chapter. Moderator: gary.bennett	20	224
Free Live Webinars for iPhone Developers This forum lists the schedule for upcoming live webinars for iPhone developers. Webinars are live and have limited seats. Current and former students get first notifications. Seats for all others is first-come-first serve. The sessions are recorded and will be made available to current and former students on this forum. Moderator: gary.bennett	1	9
Current Student & Alumni Recorded Webinars and More This Forum is for current and former students Moderator: gary.bennett	0	0
Student/Instructor AppStore Applications Applications that xcelme instructors and students have successfully posted on iTunes AppStore. Moderator: gary.bennett	38	61
Swift Course 1 - Intro to OOP and Logic Swift Course 1 - Intro to OOP and Logic Moderator: gary.bennett	11	14
Swift Course 2 - Swift for iOS Developers Swift Course 2 - Swift for iOS Developers Moderator: gary.bennett	11	11
Swift Course 3 - Cocoa Touch for iOS Developers Swift Course 3 - Cocoa Touch for iOS Developers Moderator: gary.bennett	6	6
Swift Course 4 - iPhone and iPad Programming Part 1 Swift Course 4 - iPhone and iPad Programming Part 1	2	2

Chapter 1

Becoming a Great iOS Developer

Now that you're ready to become a software developer and have read the introduction of this book, you need to become familiar with several key concepts. Your computer program will do exactly what you tell it to do—no more and no less. It will follow the programming rules that were defined by the operating system and the Swift programming language. Your program doesn't care if you are having a bad day or how many times you ask it to perform something. Often, what you think you've told your program to do and what it actually does are two different things.

Key to success If you haven't already, take a few minutes to read the introduction of this book. The introduction shows you where to go to access the free webinars, forums, and YouTube videos that go with each chapter. Also, you'll better understand why this book uses the Swift playground programming environment and how to be successful in developing your iOS apps.

Depending on your background, working with something absolutely black and white may be frustrating. Many times, programming students have lamented, "That's not what I wanted it to do!" As you begin to gain experience and confidence in programming, you'll begin to think like a programmer. You will understand software design and logic, and you will experience having your programs perform exactly as you want and the satisfaction associated with this.

Thinking like a Developer

Software development involves writing a computer program and then having a computer execute that program. A *computer program* is the set of instructions that you want the computer to perform. Before beginning to write a computer program, it is helpful to list the steps that you want your program to perform in the order you want them accomplished. This step-by-step process is called an *algorithm*.

If you want to write a computer program to toast a piece of bread, you would first write an algorithm. This algorithm might look something like this:

1. Take the bread out of the bag.
2. Place the bread in the toaster.
3. Press the toast button.
4. Wait for the toast to pop up.
5. Remove the toast from the toaster.

At first glance, this algorithm seems to solve the problem. However, the algorithm leaves out many details and makes many assumptions. Here are some examples:

- What kind of toast does the user want? Does the user want white bread, wheat bread, or some other kind of bread?
- How does the user want the bread toasted? Light or dark?
- What does the user want on the bread after it is toasted: butter, margarine, honey, or strawberry jam?
- Does this algorithm work for all users in their cultures and languages? Some cultures may have another word for toast or not know what toast is.

Now, you might be thinking this is getting too detailed for making a simple toast program. Over the years, software development has gained a reputation of taking too long, costing too much, and not being what the user wants. This reputation came to be because computer programmers often start writing their programs before they have really thought through their algorithms.

The key ingredients to making successful applications are *design requirements*. Design requirements can be formal and detailed or simple like a list on a piece of paper. Design requirements are important because they help the developer flush out what the application should do and not do when complete. Design requirements should not be completed in a programmer's vacuum but should be produced as the result of collaboration between developers, users, and customers.

Note If you take anything away from this chapter, take away the importance of considering design requirements and user interface design before starting software development. This is the most effective (and least expensive) use of time in the software development cycle. Using a pencil and eraser is a lot easier and faster than making changes to code because you didn't have others look at the designs before starting to program.

Another key ingredient to your successful app is the *user interface* (UI) design. Apple recommends you spend more than 50 percent of the entire development process focusing on the UI design. The design can be done using simple pencil and paper or using Xcode's storyboard feature to lay out your screen elements. Many software developers start with the UI design, and after laying out all the screen elements and having many users look at paper mock-ups, they then write the design requirements from their screen layouts.

After you have done your best to flush out all the design requirements, laid out all the user interface screens, and had the clients or potential customers look at your design and give you feedback, coding can begin. Once coding begins, design requirements and user interface screens can change, but the changes are typically minor and easily accommodated by the development process.

See Figures 1-1 and 1-2.



Figure 1-1. This is a UI mock-up of the account balance screen for an iPhone mobile banking app before development begins. This UI design mock-up was completed using OmniGraffle

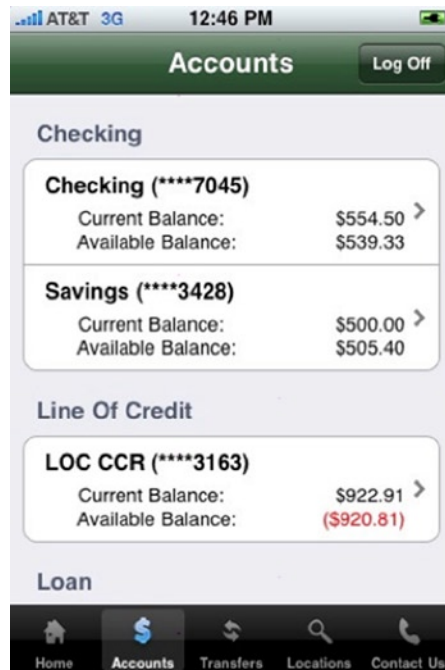


Figure 1-2. This is a completed iPhone mobile banking application as it appeared on the iTunes App Store. This app is called *Woodforest Mobile Banking*

Figure 1-1 shows a mock-up of a mobile banking app screen prior to development. Developing mock-up screens along with design requirements forces developers to think through many of the application's usability issues before coding begins. This enables the application development time to be shortened and makes for a better user experience and better reviews on the iTunes App Store. Figure 1-2 shows how the view for the mobile banking app appears when completed.

Completing the Development Cycle

Now that you have the design requirements and user interface designs and have written your program, what's next? After programming, you need to make sure your program matches the design requirements and user interface design and ensure that there are no errors. In programming vernacular, errors are called *bugs*. Bugs are undesired results of your programming and must be fixed before the app is released to the App Store. The process of finding bugs in programs and making sure the program meets the design requirements is called *testing*. Typically, someone who is experienced in software testing methodology and who didn't write the app performs this testing. Software testing is commonly referred to as *quality assurance* (QA).

Note When an application is ready to be submitted to the iTunes App Store, Xcode gives the file an `.app` or `.ipa` extension, for example, `appName.app`. That is why iPhone, iPad, and Mac applications are called *apps*. This book will use *program*, *application*, and *app* to mean the same thing.

During the testing phase, the developer will need to work with QA staff to determine why the application is not working as designed. The process is called *debugging*. It requires the developer to step through the program to find out why the application is not working as designed. Figure 1-3 shows the complete software development cycle.

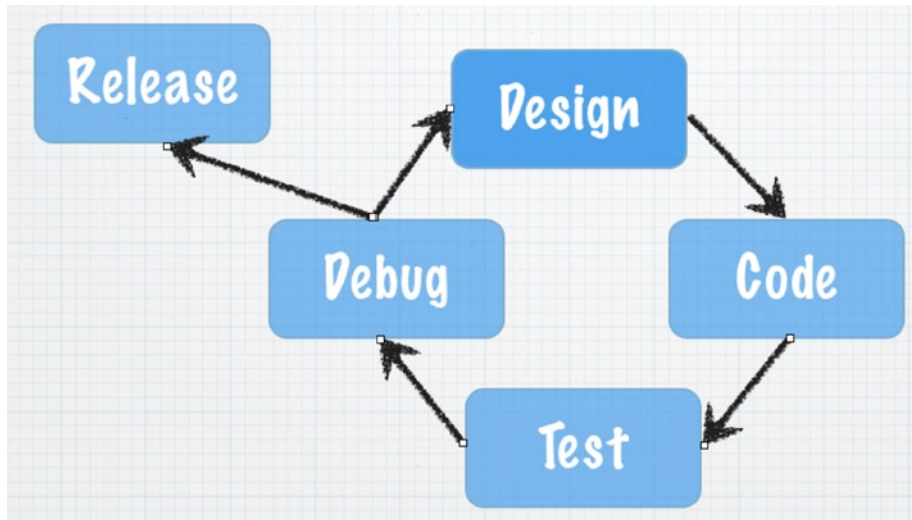


Figure 1-3. The typical software development cycle

Frequently during testing and debugging changes to the requirements (design) must occur to make the application more usable for the customer. After the design requirements and user interface changes are made, the process begins over again.

At some point, the application that everyone has been working so hard on must be shipped to the iTunes App Store. Many considerations are taken into account when this happens:

- Cost of development
- Budget
- Stability of the application
- Return on investment

There is always the give-and-take between developers and management. Developers want the app perfect, and management wants to start realizing revenue from the investment as soon as possible. If the release date were left up to the developers, the app would likely never ship to the App Store. Developers would continue to tweak the app forever, making it faster, more efficient, and more usable. At some point, however, the code needs to be pried from the developers' hands and uploaded to the App Store so it can do what it was meant to do.

Introducing Object-Oriented Programming

As discussed in detail in the introduction, playgrounds enable you to focus on *object-oriented programming* (OOP) without having to cover all the Swift programming syntax and complex Xcode development environment in one big step. Instead, you can focus on learning the basic principles of OOP and using those principles quickly to write your first programs.

For decades, developers have been trying to figure out a better way to develop code that is reusable, manageable, and easily maintained over the life of a project. OOP was designed to help achieve code reuse and maintainability while reducing the cost of software development.

OOP can be viewed as a collection of objects in a program. Actions are performed on these objects to accomplish the design requirements.

An *object* is anything that can be acted on. For example, an airplane, person, or screen/view on the iPad can all be objects. You may want to act on the plane by making the plane bank. You may want the person to walk or to change the color of the screen of an app on the iPad. Actions are all being applied to these objects; see Figure 1-4.

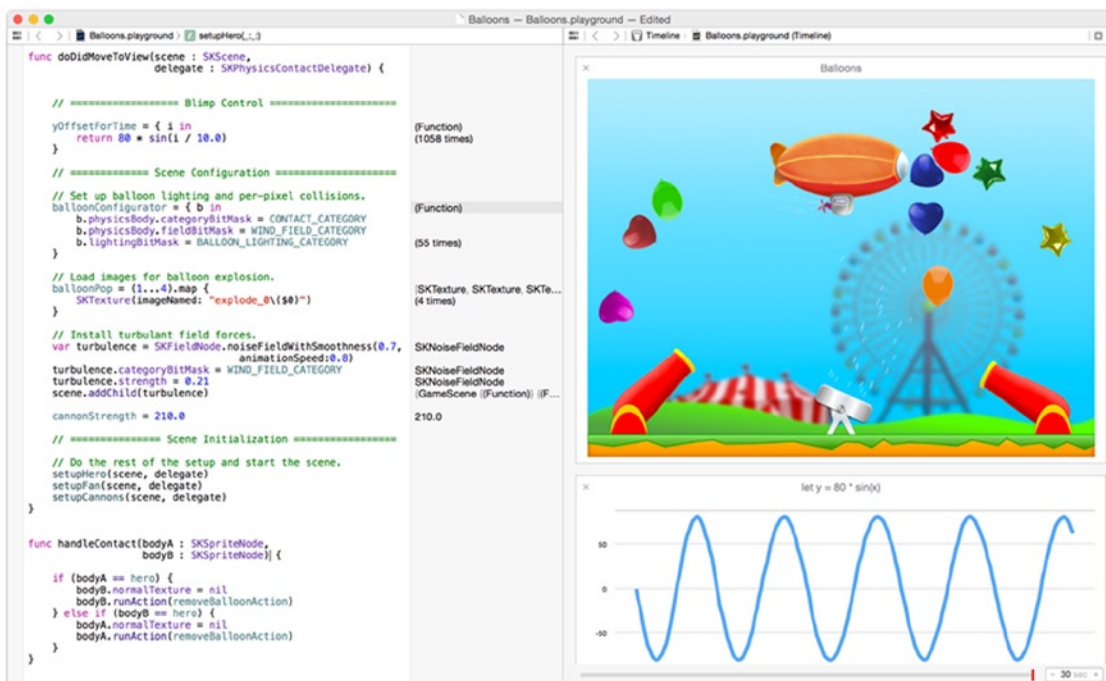


Figure 1-4. There are multiple objects in this view: cannons, balloons, and blimps. All objects can have actions applied—raise, lower, shoot, and so on

Playgrounds execute your code as you complete each line, such as the one shown in Figure 1-4. When you run your playground applications, the user can apply actions to the objects in your application. Xcode is an *integrated development environment* (IDE) that enables you to run your application from within your programming environment. You can test your applications on your computers first before running them on your iOS devices by running the apps in Xcode's simulator, as shown in Figure 1-5.

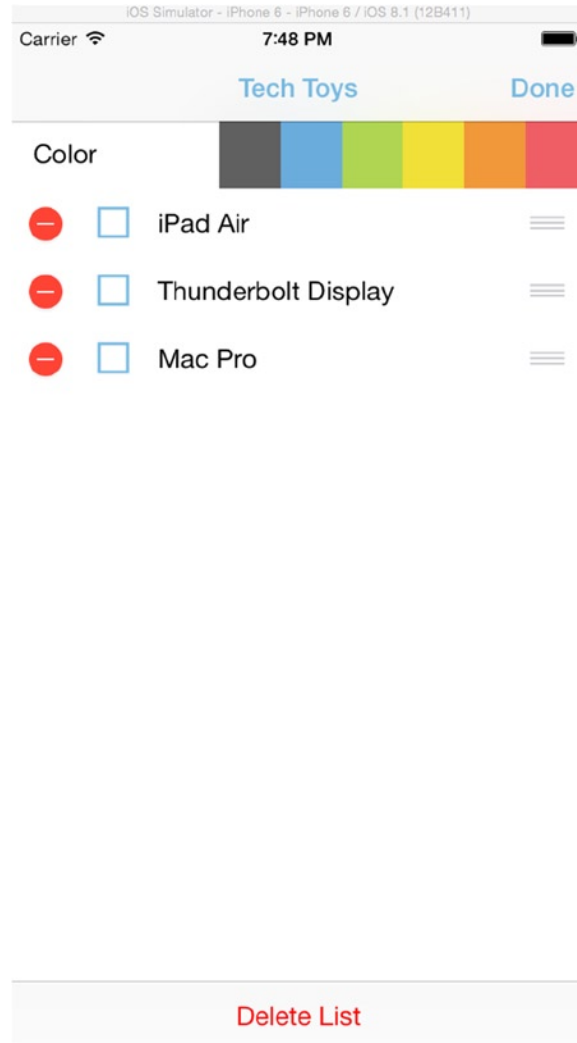


Figure 1-5. This sample iPhone app contains a table object to organize a list of tech toys. Actions such as “rotate left” or “user did select row 3” can be applied to this object

Actions that are performed on objects are called *methods*. Methods manipulate objects to accomplish what you want your app to do. For example, for a jet object you might have the following methods:

```
goUp  
goDown  
bankLeft  
turnOnAfterBurners  
lowerLandingGear
```

The table object in Figure 1-5 is actually called `UITableView` when you use it in a program, and it could have the following methods:

```
numberOfRowsInSection  
cellForRowAtIndexPath  
canEditRowAtIndexPath  
commitEditingStyle  
didSelectRowAtIndexPath
```

All objects have data that describes those objects. This data is defined as *properties*. Each property describes the associated object in a specific way. For example, the jet object's properties might be as follows:

```
altitude = 10,000 feet  
heading = North  
speed = 500 knots  
pitch = 10 degrees  
yaw = 20 degrees  
latitude = 33.575776  
longitude = -111.875766
```

For the `UITableView` object in Figure 1-5, the following might be the properties:

```
backgroundColor = Red  
selectedRow = 3  
animateView = No
```

An object's properties can be changed at any time when your program is running, when the user interacts with the app, or when the programmer designs the app to accomplish the design requirements. The values stored in the properties of an object at a specific time are collectively called the *state of an object*.

State is an important concept in computer programming. When teaching students about state, we ask them to go over to a window and find an airplane in the sky. We then ask them to snap their fingers and make up some of the values that the plane's properties might have at that specific time. Those values might be as follows:

```
altitude = 10,000 feet  
latitude = 33.575776  
longitude = -111.875766
```

Those values represent the state of the object at the specific time that they snapped their fingers.

After waiting a couple minutes, we ask the students to find that same plane, snap their fingers again, and record the plane's possible state at that specific point in time.

The values of the properties might then be something like the following:

```
altitude = 10,500 feet  
latitude = 33.575665  
longitude = -111.875777
```

Notice how the state of the object changes over time.

Working with the Playground Interface

Playgrounds offer a great approach in using the concepts just discussed without all the complexity of learning Xcode and the Swift language at the same time. It takes only a few minutes to familiarize yourself with the playground interface and begin writing a program.

Technically speaking, the playground interface is not a true IDE like you will be using to write your iOS apps, but it is pretty close and much easier to learn in. A true IDE combines code development, user interface layout, debugging tools, documentation, and simulator/console launching for a single application; see Figure 1-6. However, playgrounds offer a similar look, feel, and features to the Xcode IDE you develop apps with.

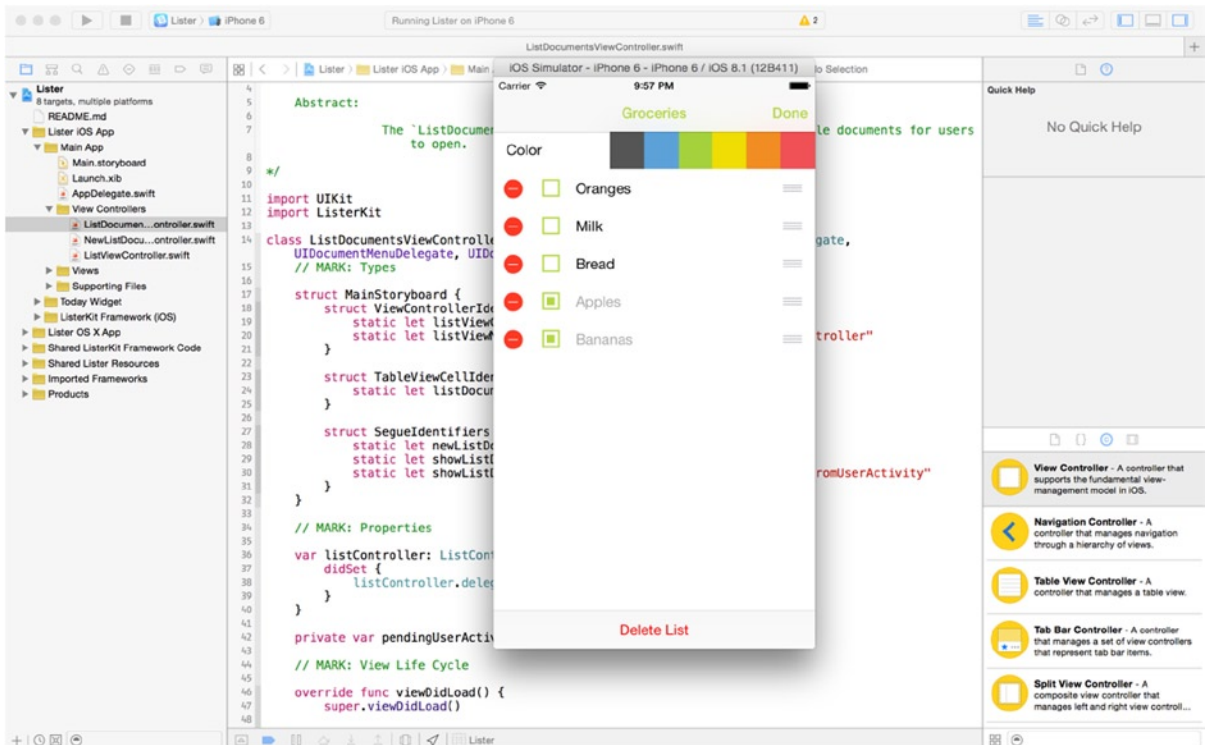


Figure 1-6. The Xcode IDE with the iPhone Simulator

In the next chapter, you will go through the playground interface and write your first program.

Summary

Congratulations, you have finished the first chapter of this book. It is important that you have an understanding of the following terms because they will be reinforced throughout this book:

- Computer program
- Algorithm
- Design requirements
- User interface
- Bug
- Quality assurance (QA)
- Debugging
- Object-oriented programming (OOP)
- Object
- Property
- Method
- State of an object
- Integrated development environment (IDE)

What's Next

The next 13 chapters provide the information you need to learn Swift and write iOS applications. Terms and concepts are introduced and reinforced over and over so you will begin to get more comfortable with them. Keep going and be patient with yourself.

Exercises

- Answer the following questions:
 - Why is it so important to spend time on your user requirements?
 - What is the difference between design requirements and an algorithm?
 - What is the difference between a method and a property?
 - What is a bug?
 - What is state?
- Write an algorithm for how a soda machine works from the time a coin is inserted until a soda is dispensed. Assume the price of a soda is 80 cents.
- Write the design requirements for an app that will run the soda machine.

Programming Basics

This chapter focuses on the building blocks that are necessary to become a great Swift programmer. This chapter covers how to use the playground user interface, how to write your first Swift program, and how to use the Xcode integrated development environment (IDE).

Note We will introduce you to using playgrounds, which will enable you to program right away without worrying about the complexities of Xcode. We have used this approach for the last five years and know that it helps you learn the concepts quickly, without discouragement, and gives you a great foundation to build upon.

Touring Xcode

Xcode and playgrounds make writing Swift code incredibly simple and fun. Type a line of code, and the result appears immediately. If your code runs over time, for instance through a loop, you can watch its progress in the timeline area. When you've perfected your code in the playground, simply move that code into your Swift iOS project. With Xcode, you can do the following:

- Design a new algorithm, watching its results every step of the way
- Create new tests, verifying that they work before promoting into your test suite
- Experiment with new APIs to hone your Swift coding skills

First you'll need to learn a little more about the Xcode user interface. When you open an Xcode iOS project, you are presented with a screen that looks like [Figure 2-1](#).

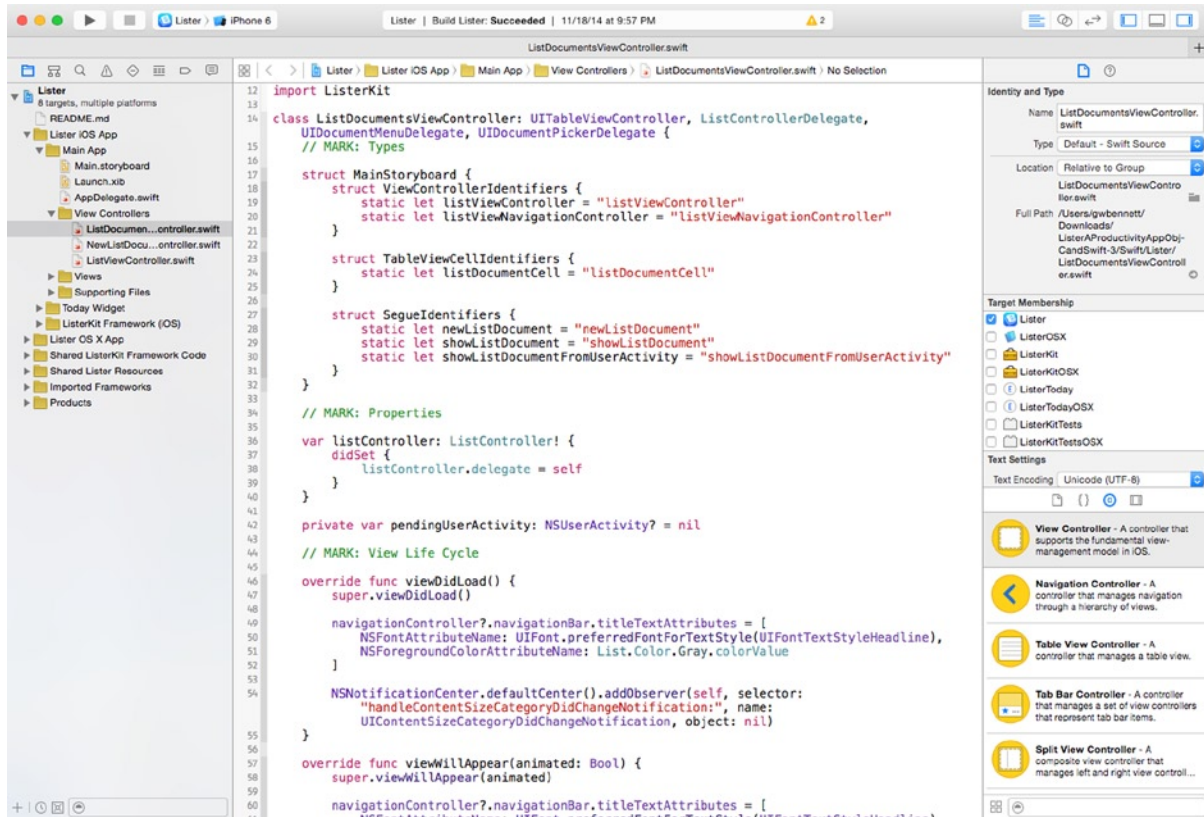


Figure 2-1. Opening screen in Xcode with a Swift project

The Xcode user interface is set up to help you efficiently write your Swift applications. The user interface for playgrounds is similar to the user interface for an iOS application. You will now explore the major sections of Xcode's IDE workspace and playgrounds.

Exploring the Workspace Window

The workspace window, shown in Figure 2-2, enables you to open and close files, set your application preferences, develop and edit or app, and view text output and the error console.

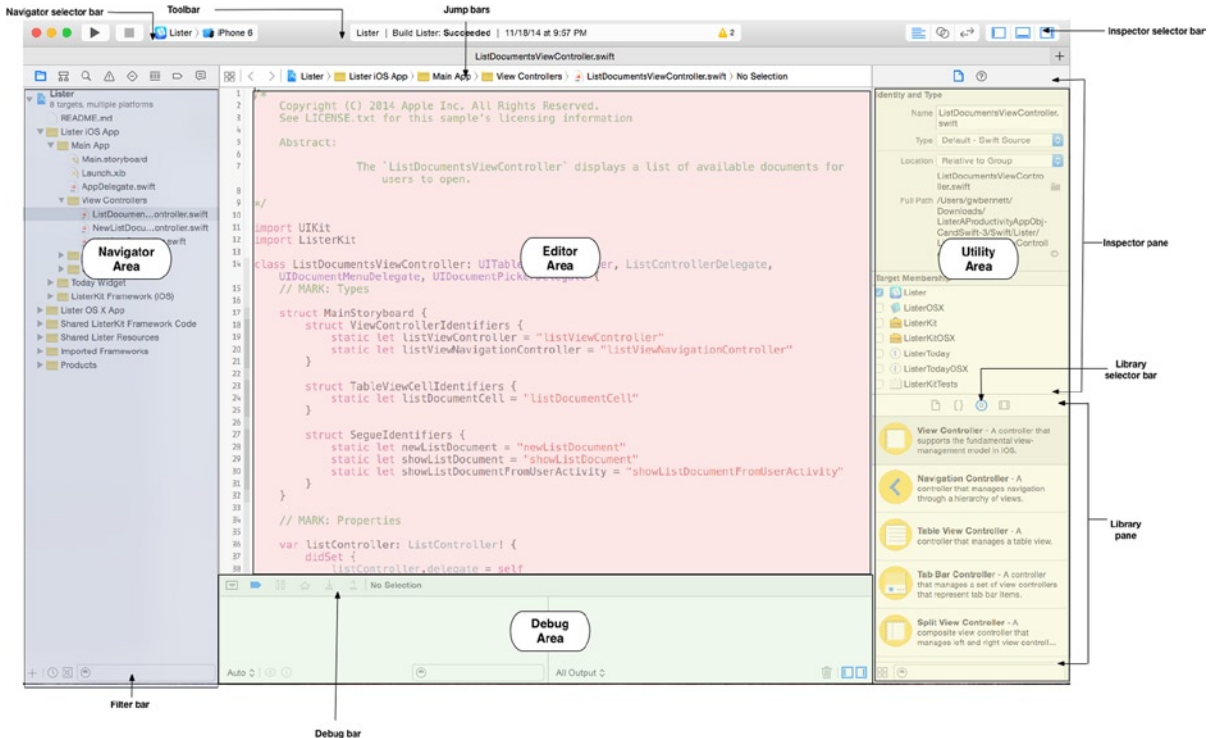



Figure 2-2. Xcode's workspace window

The workspace window is your primary interface for creating and managing projects. The workspace window automatically adapts itself to the task at hand, and you can further configure the window to fit your work style. You can open as many workspace windows as you need.


The workspace window has four main areas: Editor, Navigator, Debug, and Utility.

When you select a project file, its contents appear in the Editor area, where Xcode opens the file in the appropriate editor.

You hide or show the other three areas by using buttons in the view selector in the toolbar.

 Clicking this button shows or hides the Navigator area. This is where you view and maneuver through files and other facets of your project.

 Clicking this button shows or hides the Debug area. This is where you control program execution and debug code.

 Clicking this button shows or hides the Utilities area. You use the Utilities area for several purposes, most commonly to view and modify attributes of a file and to add ready-made resources to your project.

Navigating Your Workspace

You can access files, symbols, unit tests, diagnostics, and other facets of your project from the Navigator area. In the navigator selector bar, you choose the navigator suited to your task. The content area of each navigator gives you access to relevant portions of your project, and each navigator's filter bar allows you to restrict the content that is displayed.

Choose from these options in the navigator selector bar:



Project navigator. Add, delete, group, and otherwise manage files in your project, or choose a file to view or edit its contents in the editor area.



Symbol navigator. Browse the class hierarchy of the symbols in your project.



Find navigator. Use search options and filters to quickly find any string within your project.



Issue navigator. View issues such as diagnostics, warnings, and errors found when opening, analyzing, and building your project.



Test navigator. Create, manage, run, and review unit tests.



Debug navigator. Examine the running threads and associated stack information at a specified point or time during program execution.



Breakpoint navigator. Fine-tune breakpoints by specifying characteristics such as triggering conditions.



Report navigator. View the history of your builds, app console output, continuous integration, and source control tasks.

Editing Your Project Files

Most development work in Xcode occurs in the Editor area, which is the main area that is always visible within the workspace window. The editors you will use most often are as follows:

- *Source editor*: Write and edit Swift source code.
- *Interface Builder*: Graphically create and edit user interface files (see Figure 2-3).

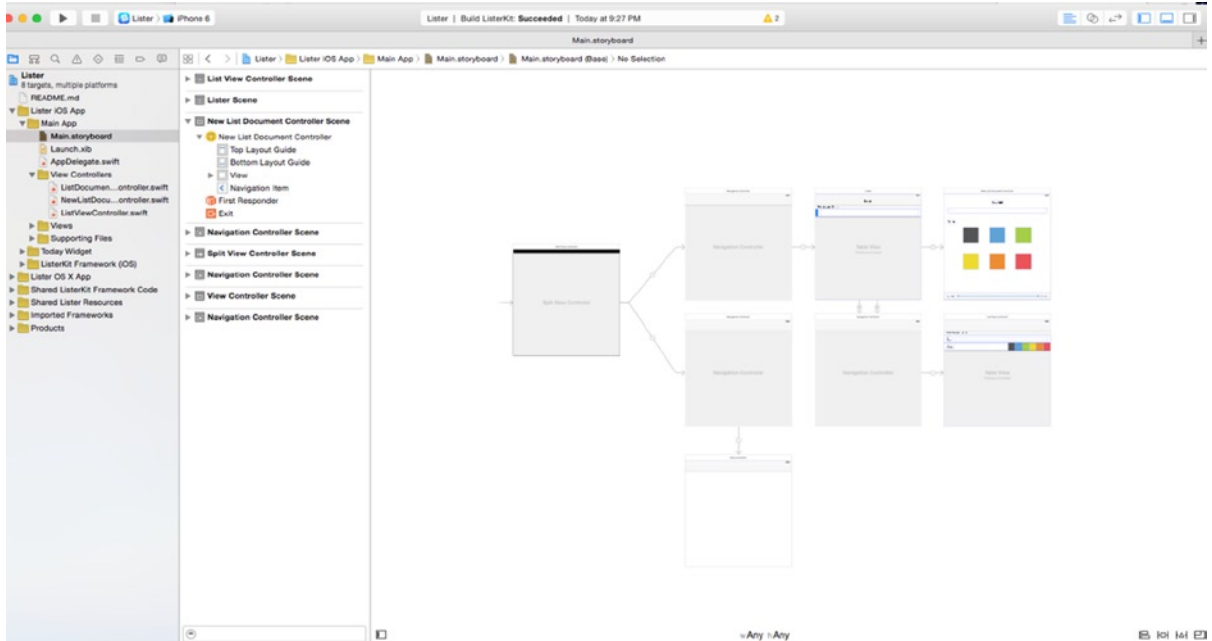


Figure 2-3. Xcode's Interface Builder

- **Project editor:** View and edit how your apps should be built, such by specifying build options, target architectures, and app entitlements.

When you select a file, Xcode opens the file in an appropriate editor. In Figure 2-3, the file `Main.storyboard` is selected in the Project navigator, and the file is open in Interface Builder.

The editor offers three controls:



Clicking this button opens the Standard editor. You will see a single editor pane with the contents of the selected file.



Clicking this button opens the Assistant editor. You will see a separate editor pane with content logically related to that in the Standard editor pane.



Clicking this button opens the Version editor. You will see the differences between the selected file in one pane and another version of that same file in a second pane.

Creating Your First Swift Playground Program

Now that you have learned a little about Xcode, it's time to write your first Swift playground program and begin to understand the Swift language, Xcode, and some syntax. First you have to install Xcode.

Installing and Launching Xcode 6

Xcode 6 is available for download from the Mac App Store for free, as shown in Figure 2-4, and from the iOS Dev Center, as shown in Figure 2-5.

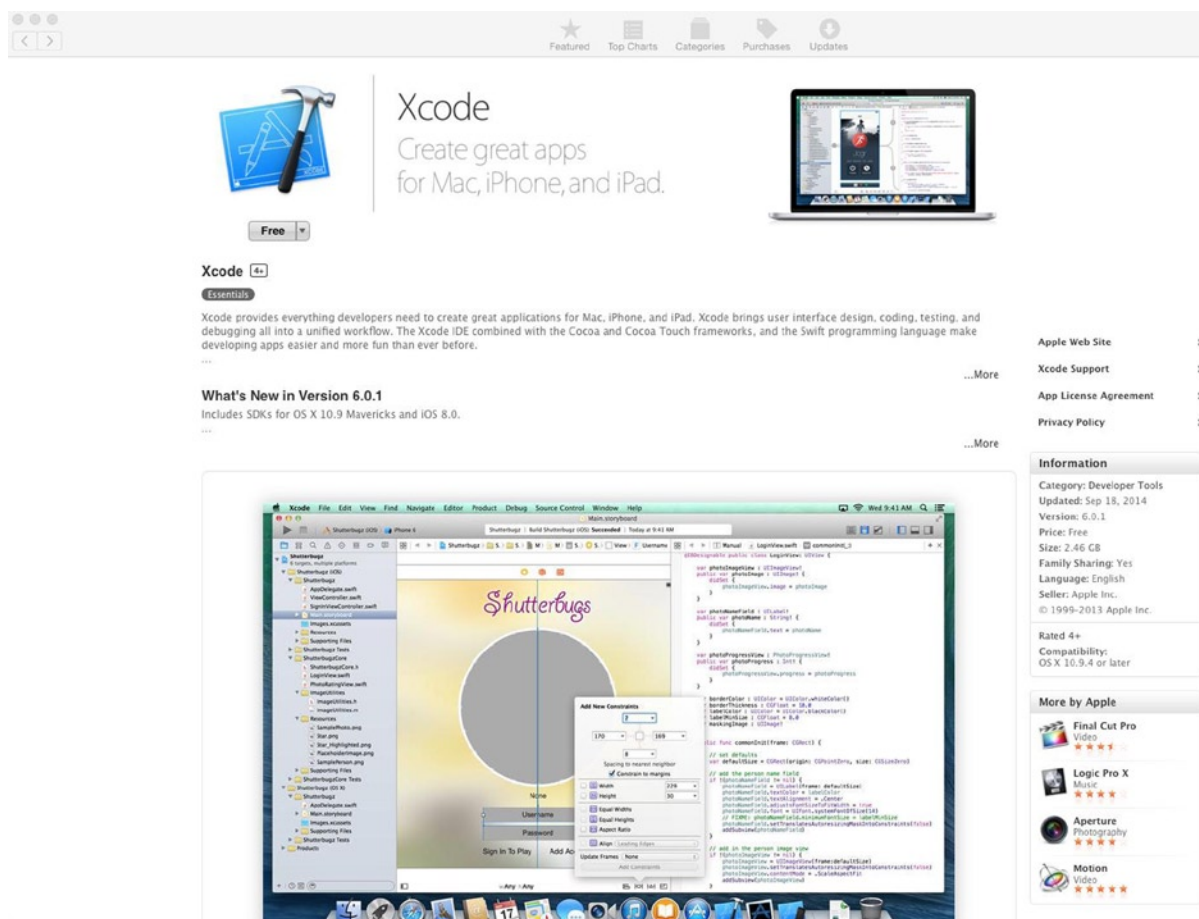


Figure 2-4. Xcode 6 is available for download from the Mac App Store for free

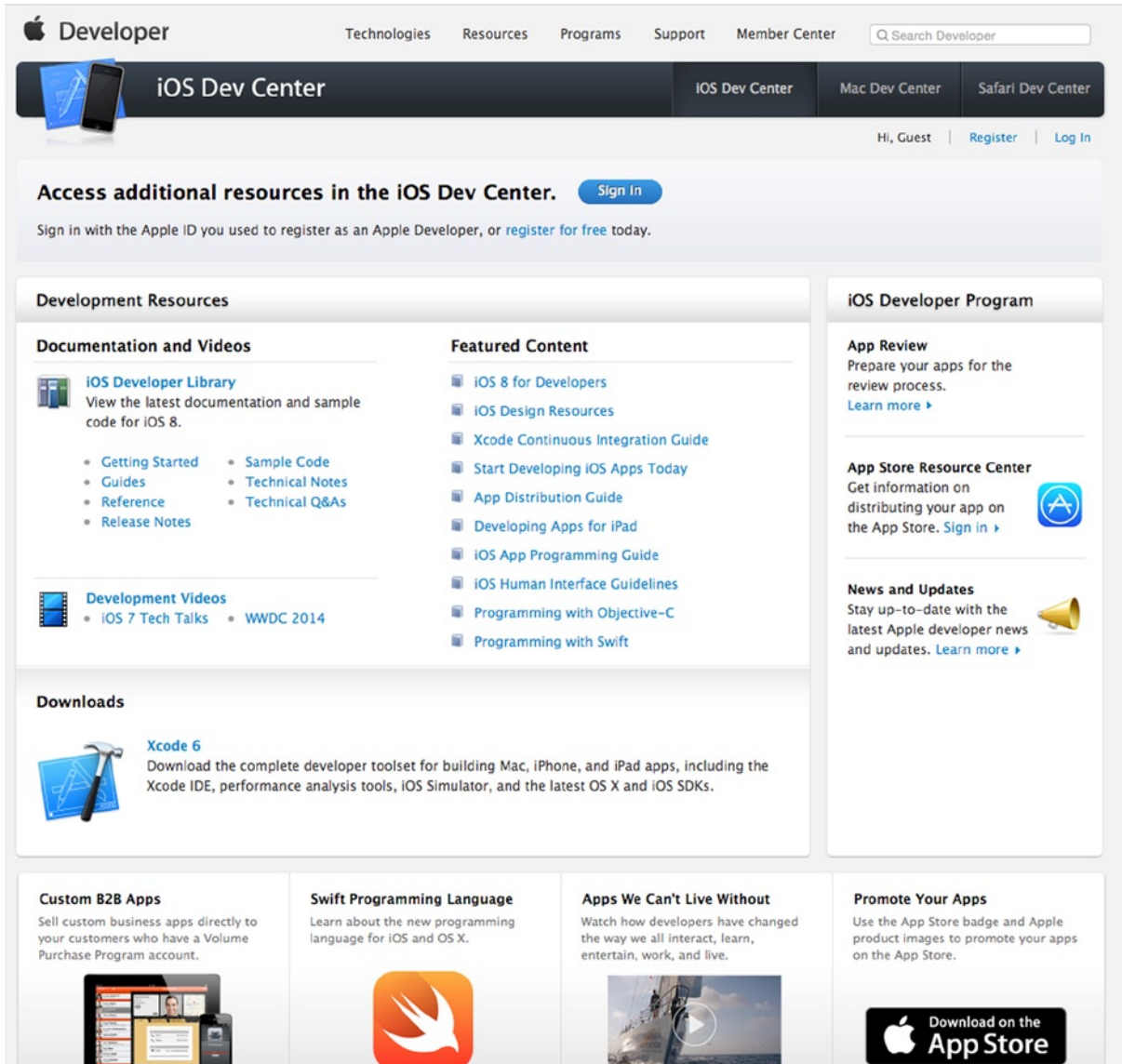


Figure 2-5. The iOS Dev Center

Note This package has everything you need to write iOS apps. To develop iPhone apps, you will need to apply for the iPhone Developer Program and pay \$99 (when ready to test on your iOS device and submit to the App Store). See <http://developer.apple.com>.

Now that you have installed Xcode, let's begin writing a Swift playground. Launch Xcode and click "Get started with a playground," as shown in Figure 2-6.



Figure 2-6. Creating your first Swift playground

Using Xcode 6

After launching Xcode, follow these steps:

1. Let's name the playground **HelloWorld** and select iOS as the platform, as shown in Figure 2-7. Then click Next and save your app in the directory of your choice.