Exploring Swift Playgrounds

The Fastest and Most Effective Way to Learn to Code and to Teach Others to Use Your Code

Jesse Feiler

Apress®

Exploring Swift Playgrounds

The Fastest and Most Effective Way to Learn to Code and to Teach Others to Use Your Code

Jesse Feiler

Apress®

Exploring Swift Playgrounds: The Fastest and Most Effective Way to Learn to Code and to Teach Others to Use Your Code

Jesse Feiler
Plattsburgh, New York, USA

ISBN-13 (pbk): 978-1-4842-2646-9 ISBN-13 (electronic): 978-1-4842-2647-6
DOI 10.1007/978-1-4842-2647-6

Library of Congress Control Number: 2017938279

Managing Director: Welmoed Spahr
Editorial Director: Todd Green
Acquisitions Editor: Aaron Black
Development Editor: James Markham
Technical Reviewer: Massimo Nardone
Coordinating Editor: Jessica Vakili
Copy Editor: Corbin Collins
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global
Cover image designed by Freepik

Contents at a Glance

Contents

About the Author

Jesse Feiler is a developer, consultant, and author focusing on Apple technologies for small businesses and nonprofit organizations. His projects have included database design and development with FileMaker and Core Data as well as production process control, publishing project management, and social media strategies for clients such as Federal Reserve Bank of New York, Young & Rubicam, Cutter Consortium, and Archipenko Foundation. His books have been published by Wiley, Pearson, Apress, and others. His apps, including Utility Smart, Minutes Machine, Saranac River Trail, and The Nonprofit Risk App, are published by Champlain Arts Corp (`http://champlainarts.com`). He is founder of Friends of Saranac River Trail, Inc. and has served on a variety of boards for libraries and nonprofit cultural organizations. A native of Washington, DC, he has lived in New York City and currently lives in Plattsburgh, New York. He can be reached at `jfeiler@champlainarts.com`.

About the Technical Reviewer

Massimo Nardone has more than 22 years of experiences in Security, Web/Mobile development, Cloud and IT Architecture. His true IT passions are Security and Android.

He has been programming and teaching how to program with Android, Perl, PHP, Java, VB, Python, C/C++ and MySQL for more than 20 years.

He holds a Master of Science degree in Computing Science from the University of Salerno, Italy.

He has worked as a Project Manager, Software Engineer, Research Engineer, Chief Security Architect, Information Security Manager, PCI/SCADA Auditor and Senior Lead IT Security/Cloud/SCADA Architect for many years.

Technical skills include: Security, Android, Cloud, Java, MySQL, Drupal, Cobol, Perl, Web and Mobile development, MongoDB, D3, Joomla, Couchbase, C/C++, WebGL, Python, Pro Rails, Django CMS, Jekyll, Scratch, etc.

He currently works as Chief Information Security Office (CISO) for Cargotec Oyj.

He worked as visiting lecturer and supervisor for exercises at the Networking Laboratory of the Helsinki University of Technology (Aalto University). He holds four international patents (PKI, SIP, SAML and Proxy areas).

Massimo has been reviewing more than 40 IT books for different publishing company and he is the coauthor of *Pro Android Games* (Apress, 2015).

This book is dedicated to Antti Jalonen and his family who are always there when I need them.

Introduction

Once you get beyond the basics of very simple code that doesn't do very much, you quickly discover a conundrum: testing code to do something pretty simple in the context of an app requires you to write a pretty complicated app—in many cases before you can test your simple code. Apple's Swift playgrounds address that issue in many of its guises. With a playground, you can experiment with a simple snippet of code on its own or within a playground that provides the context that your snippet will run in. You don't have to write the whole app in order to test your few lines of code.

You can use a Swift playground as a trainer or teacher: you can build the app context as a playground so that your students can write their snippets inside your playground. Because playgrounds are often used for training and documentation, Apple's Swift playgrounds support their own markup language that lets you format your code and create areas of the playground's code where the user can or must provide their own code. You can even hide some of your playground context so that the user or learner sees only the snippet to be worked with.

Swift playgrounds can be built and run with Playgrounds for iPad or with Xcode for macOS. The code that you write in a playground can be tested in that standalone environment and then copied and pasted into an app being developed with Xcode for macOS, iOS, watchOS, or tvOS.

This book provides an introduction to Swift playgrounds and gets you started either as a developer of playgrounds or a user of playgrounds developed by someone else. As the book progresses, you'll see how to build more and more complex playgrounds.

Playgrounds can provide a powerful and intriguing entry into coding for new coders of any age or background.

Downloading Playgrounds for the Book

You can download playgrounds from the book from the author's website at `northcountryconsulting.com`. Create an account, log in, and use the Downloads section on the left-hand side of the landing page.

CHAPTER 1

Introducing Swift Playgrounds

Swift is Apple's new programming language being used by developers inside and outside Apple to create new apps for macOS, iOS, watchOS, and tvOS. Most Apple operating systems and frameworks were written originally in Objective-C, and there are bridges between the two so that you can write new apps in Swift that use the Objective-C frameworks, sometimes without even knowing it. Examples and demos from Apple on `http://developer.apple.com` and at the Apple Worldwide Developer Conferences (WWDC) and Tech Talks now use Swift.

In and of itself, a new programming language isn't an earth-shaking event. Yes, many people think Swift is a terrific language (count me among them!), but new programming languages have appeared many times over the years since the first programming languages were developed in the 1950s. What *is* revolutionary is the Swift playground. This book provides an introduction to playgrounds and covers how to use them with Swift. (At the moment, Swift is the only language for playgrounds.)

This chapter introduces you to the pieces you'll use to put together apps for the operating systems and frameworks and talks about how they fit together. In different ways, Swift and playgrounds simplify the process, but underneath it all, the components described in this chapter are what make apps run.

Tip If you've used or even just looked at these components in the past, treat this chapter as a review. Things were changing even before Swift and playgrounds came along. The app development process—particularly the management of apps themselves—has been simplified.

Developer Overview

Getting started as an Apple developer has changed a little in the last few years. What hasn't changed is that apps for the App Store (including the Mac App Store) are *curated*—meaning Apple reviews each app and its descriptive materials. Curation helps to enforce basic standards of app quality and security to enhance consumers' confidence in the

J. Feiler, *Exploring Swift Playgrounds*, DOI 10.1007/978-1-4842-2647-6_1

Apple and App Store brands. The only way an app can be installed on an Apple device is through the relevant App Store using a special code that App Store reviewers place in each app to guarantee that it has not been changed since the review.

That said, there are now more ways to distribute your apps on a limited basis without going through the App Store. One important way to share your work with others is to build a *playground* for part of your app. You won't be able to build the next killer game or must-have lifestyle app using only a playground, but you'll be able to build small pieces of it to try out your concept and share it with friends. You can also build a playground to provide a proof-of-concept look at what your app will eventually be and do.

The App Store review and curation process require that you be a registered Apple developer. You can find out more about the programs at `http://developer.apple.com`. Most developers subscribe to the $99 per year membership category, which enables access to the App Store as well as to Developer Technical Support (two incidents per year). There are other development categories for corporations and educational institutions, all described on `http://developer.apple.com`.

Most of the development tools and documentation are available for free through `http://developer.apple.com`. You may need to register with a valid email address to gain access, but for the most part, there is no cost. Where there is a cost involved is for anything that you use for testing on the iOS Simulator or on live devices. For many would-be developers, that is when they pay the $99 fee.

In short, there's no cost involved in getting started programming with the Apple environments.

Xcode

Xcode is the integrated development environment (IDE) used to develop apps. It's enormously powerful: in fact, it's used to develop the operating systems themselves. This power means that it may appear daunting to use it to build something small like a Hello World app. As your development projects in Xcode increase in size and complexity, Xcode's power and features come into play for you. By the time you get up to even a small app with a user interface for iOS or macOS, using Xcode is more efficient than writing out code line by line.

This chapter gives only a very high-level overview of the Xcode development process. Don't worry, there are a lot more details as we move into playgrounds.

Building the Single View Application in Xcode

Let's start with an example of building a simple iOS app with Xcode. This is not an Xcode tutorial, but rather just a quick look at the Xcode process. As you move on in this chapter and through the book, you'll see how playgrounds can become part of that process, saving you time and effort along the way.

We're going to look at the Single View Application project that is built into Xcode. You'll see how pieces of it reappear in a Swift playground as you work with code in both the project and the playground:

1. Launch Xcode and choose New ➤ Project.
2. Select Single View Application, as shown in Figure 1-1.

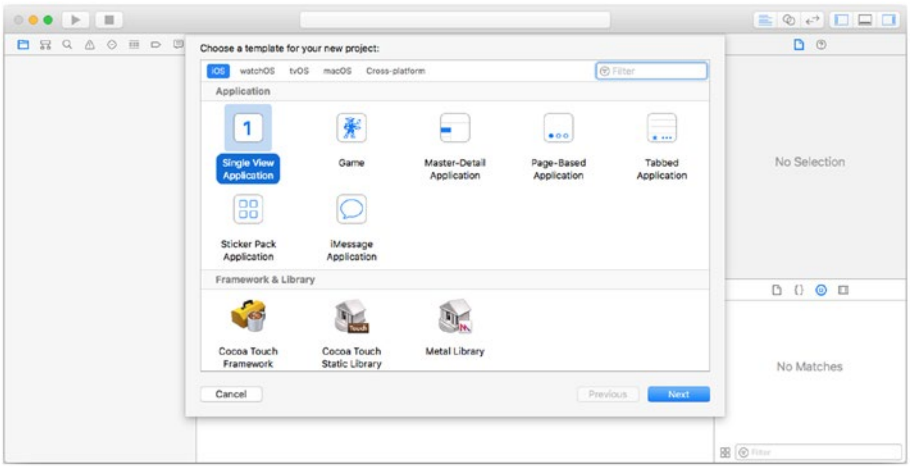
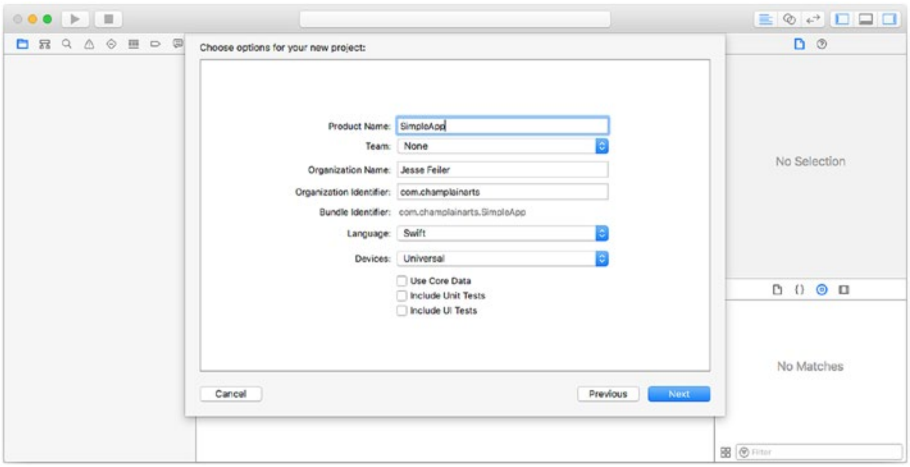


Figure 1-1. *Single View Application has been selected*

3. Click Next and select the options for your project, as shown in Figure 1-2. All that matters right now is the name and that the language is set to Swift.


Choose options for your new project:

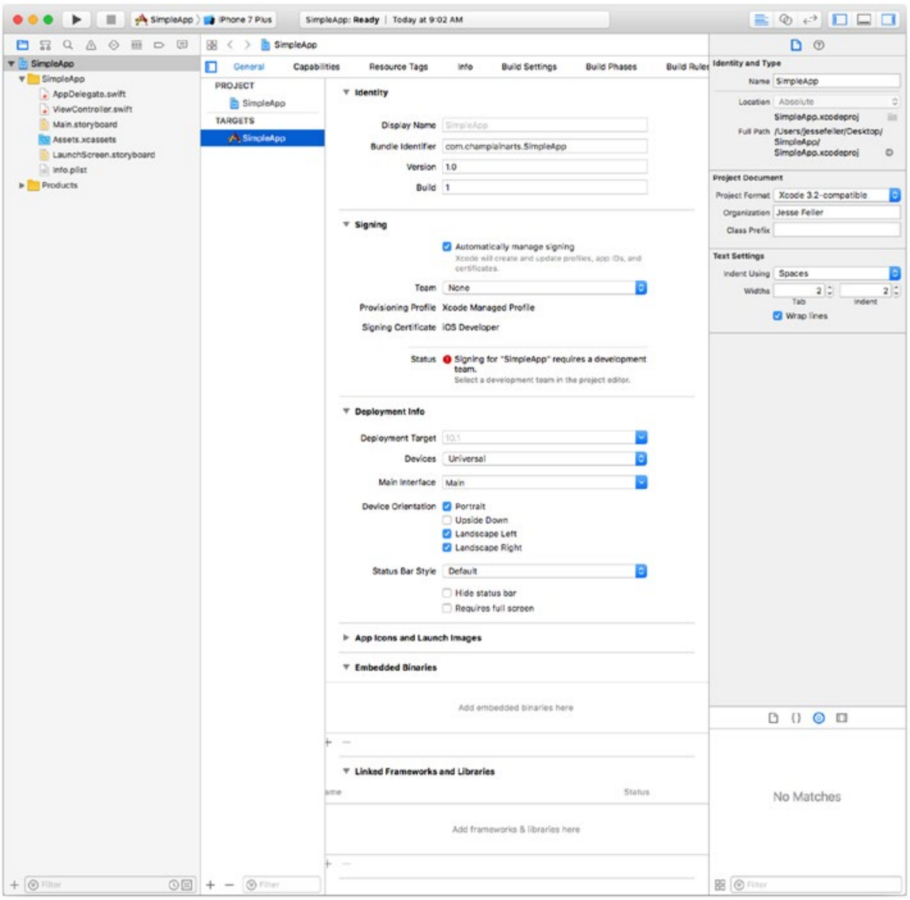
Product Name:	SimpleApp
Team:	None
Organization Name:	Jesse Feiler
Organization Identifier:	com.champlainarts
Bundle Identifier:	com.champlainarts.SimpleApp
Language:	Swift
Devices:	Universal

☐ Use Core Data
☐ Include Unit Tests
☐ Include UI Tests

Figure 1-2. *Enter your app's product name and other basic information*

4. Enter a name and location on disk for your project. In this case, the project is named SimpleApp (you can use that name if you want to follow along).

5. Click Next, and the project is created for you. You may have to open folders in the project navigator at the left of the window to see your project files, as you see in Figure 1-3.



Identity
Display Name: SimpleApp
Bundle Identifier: com.champlainarts.SimpleApp
Version: 1.0
Build: 1

Signing
✓ Automatically manage signing
Xcode will create and update profiles, app IDs, and certificates.
Team: None
Provisioning Profile: Xcode Managed Profile
Signing Certificate: iOS Developer
Status: Signing for "SimpleApp" requires a development team.
Select a development team in the project editor.

Deployment Info
Deployment Target: 10.1
Devices: Universal
Main Interface: Main
Device Orientation: ✓ Portrait, ☐ Upside Down, ✓ Landscape Left, ✓ Landscape Right
Status Bar Style: Default
☐ Hide status bar
☐ Requires full screen

App Icons and Launch Images

Embedded Binaries
Add embedded binaries here

Linked Frameworks and Libraries
Name Status
Add frameworks & libraries here

Identity and Type
Name: SimpleApp
Location: Absolute
SimpleApp.xcodeproj
Full Path: /Users/jessefeller/Desktop/SimpleApp/SimpleApp.xcodeproj

Project Document
Project Format: Xcode 3.2-compatible
Organization: Jesse Feller
Class Prefix:

Text Settings
Indent Using: Spaces
Widths: 2 (Tab) 2 (Indent)
✓ Wrap lines

Figure 1-3. View the project navigator and the target in the main view

6. Select the SimpleApp project itself (the blue icon at the top of the project navigator). You'll see the default settings, as shown in Figure 1-3.

7. If you see a status warning for code signing, you can safely ignore it for now.

8. Choose a device simulator for the project from the top of the window. iPhone 7 Plus is chosen in Figure 1-3. Click the triangle to build and run the app.

9. The app is built and runs in the iOS Simulator for iPhone 7 Plus, as shown in Figure 1-4. There's not much to see, but the app is running.

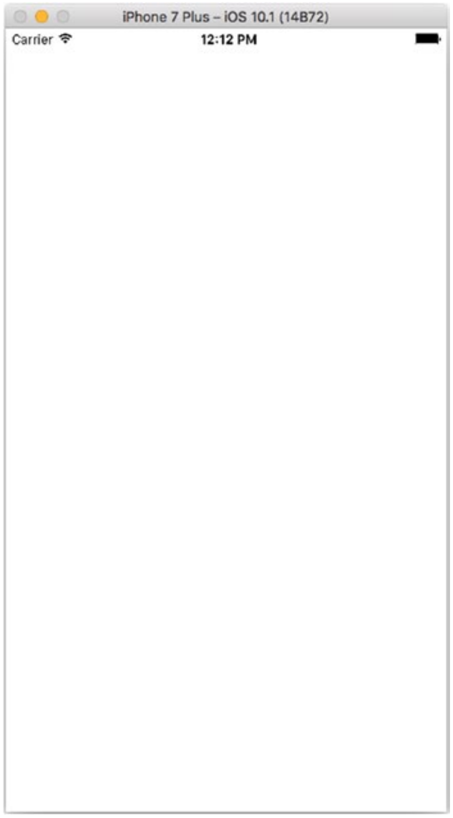
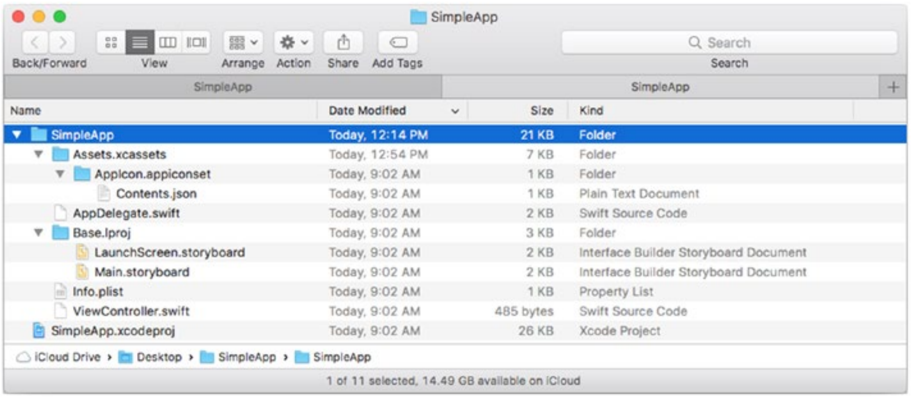


Figure 1-4. *A basic app just runs until you create its user interface*

Exploring the Single View Application

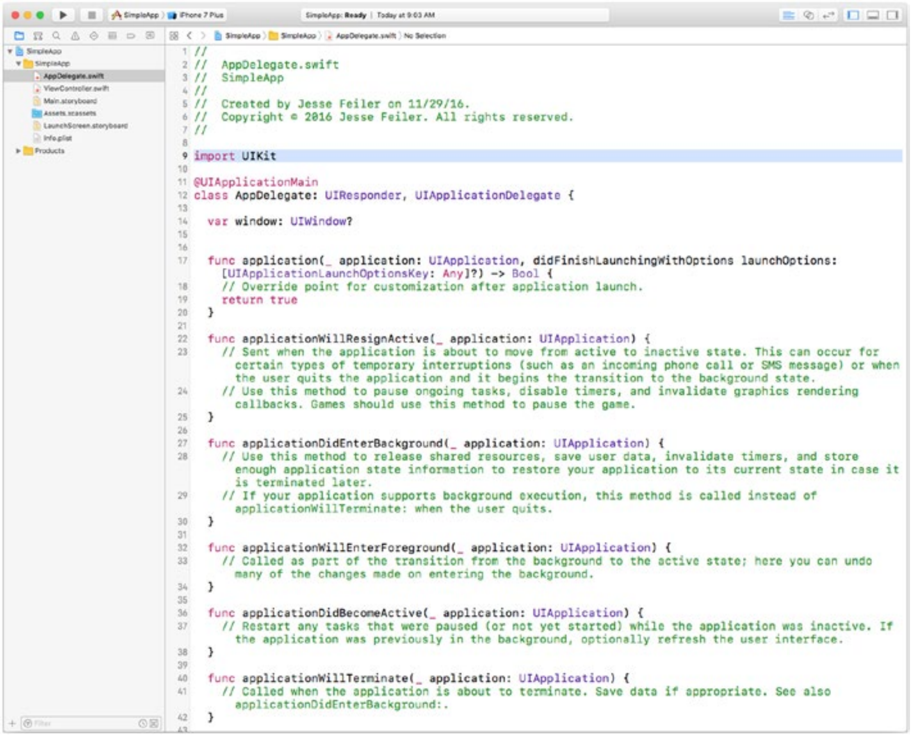
You can explore the files that are automatically created for you. They're shown in Figure 1-5. What Xcode gives you is the ability to create all of those files and a runnable app with only a few keystrokes.


SimpleApp
Back/Forward
View
Arrange
Action
Share
Add Tags
Search
Search
SimpleApp
SimpleApp
iCloud Drive › Desktop › SimpleApp › SimpleApp
1 of 11 selected, 14.49 GB available on iCloud

Name	Date Modified	Size	Kind
▼ SimpleApp	Today, 12:14 PM	21 KB	Folder
▼ Assets.xcassets	Today, 12:54 PM	7 KB	Folder
▼ AppIcon.appiconset	Today, 9:02 AM	1 KB	Folder
Contents.json	Today, 9:02 AM	1 KB	Plain Text Document
AppDelegate.swift	Today, 9:02 AM	2 KB	Swift Source Code
▼ Base.lproj	Today, 9:02 AM	3 KB	Folder
LaunchScreen.storyboard	Today, 9:02 AM	2 KB	Interface Builder Storyboard Document
Main.storyboard	Today, 9:02 AM	2 KB	Interface Builder Storyboard Document
Info.plist	Today, 9:02 AM	1 KB	Property List
ViewController.swift	Today, 9:02 AM	485 bytes	Swift Source Code
SimpleApp.xcodeproj	Today, 9:02 AM	26 KB	Xcode Project

Figure 1-5. *The app's files are created inside the app's folder*

These files are just the tip of the iceberg. If you look inside AppDelegate.swift (one of the main files of the project), you'll see the code shown in Figure 1-6 at the top of the file.



```
//
//  AppDelegate.swift
//  SimpleApp
//
//  Created by Jesse Feiler on 11/29/16.
//  Copyright © 2016 Jesse Feiler. All rights reserved.
//

import UIKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

  var window: UIWindow?


  func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
    [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
    // Override point for customization after application launch.
    return true
  }

  func applicationWillResignActive(_ application: UIApplication) {
    // Sent when the application is about to move from active to inactive state. This can occur for
      certain types of temporary interruptions (such as an incoming phone call or SMS message) or when
      the user quits the application and it begins the transition to the background state.
    // Use this method to pause ongoing tasks, disable timers, and invalidate graphics rendering
      callbacks. Games should use this method to pause the game.
  }

  func applicationDidEnterBackground(_ application: UIApplication) {
    // Use this method to release shared resources, save user data, invalidate timers, and store
      enough application state information to restore your application to its current state in case it
      is terminated later.
    // If your application supports background execution, this method is called instead of
      applicationWillTerminate: when the user quits.
  }

  func applicationWillEnterForeground(_ application: UIApplication) {
    // Called as part of the transition from the background to the active state; here you can undo
      many of the changes made on entering the background.
  }

  func applicationDidBecomeActive(_ application: UIApplication) {
    // Restart any tasks that were paused (or not yet started) while the application was inactive. If
      the application was previously in the background, optionally refresh the user interface.
  }

  func applicationWillTerminate(_ application: UIApplication) {
    // Called when the application is about to terminate. Save data if appropriate. See also
      applicationDidEnterBackground:.
  }
```

Figure 1-6. *Basic app code is placed in the files for you*

Looking into the Frameworks

Most of the code in Figure 1-6 consists of comments and stubs for functions. At the top of the file, you'll notice a line of Swift code to include the UIKit framework:

```
import UIKit
```

UIKit is the framework that contains the classes to support windows, views, view controllers, and most of the user interface in iOS, tvOS, and watchOS. (AppKit is the comparable framework for macOS). You use UIKit in any app you write that has a user interface, and Xcode puts it in place for you, so you may not even think about it as you develop your app. Other frameworks need to be added for specific functionalities, such as frameworks for system configuration, web services, Core Data, and many more. This integration of frameworks with your code is a key component of Xcode. At the bottom of Figure 1-3 you can see the Xcode interface that lets you add other frameworks.

In short, Xcode provides a simple and almost effortless way of integrating thousands of lines of code in the various frameworks into your app.

Swift Playgrounds

Playgrounds in their basic form won't help you create full-fledged apps. But you can build a functioning playground for testing code and learning how to use the APIs. If you want to build something extremely simple such as the traditional Hello World app that is one line of C code, Xcode and UIKit are overkill.

Building the Classic Hello World App

As a point of reference, the classic Hello World code in C is the following (or some variation):

```
#include <stdio.h>

main( )
{
  printf("hello, world\n");
}
```

(This code is from *Programming in C: A Tutorial* by Brian Kernighan, www.lysator.liu.se/c/bwk-tutor.html).

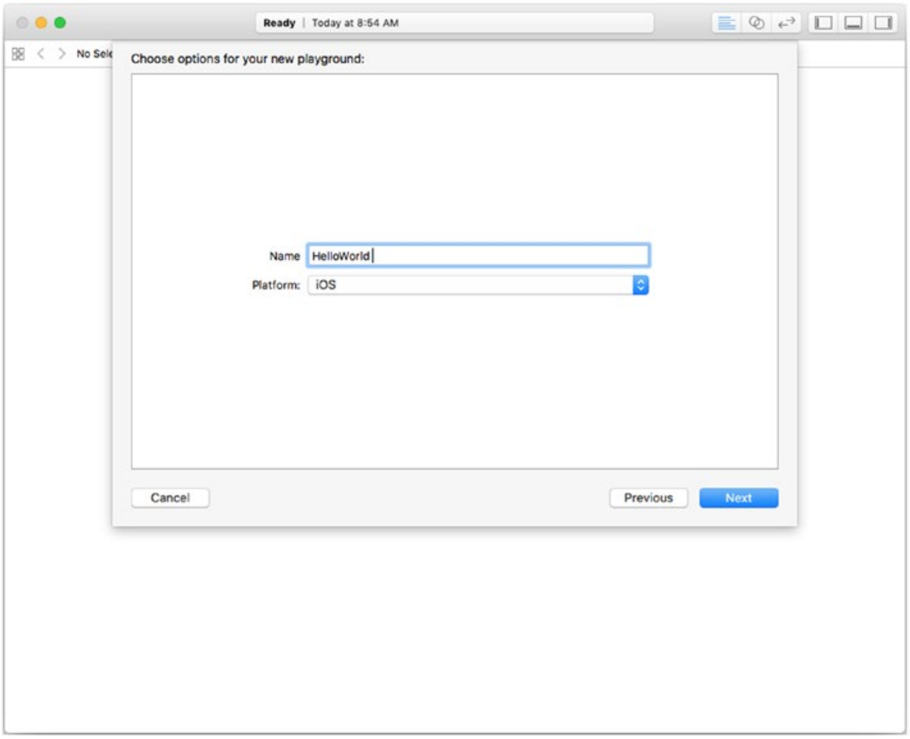
The heart of the Hello World code is the `printf` line: the rest is the environment that makes it run. Depending on the spacing, this basic program can be anywhere from one to six lines of code. Certainly, that's simpler than the steps to create even the basic Single View Application in Xcode.

Building a Hello Playground

To build a comparable playground, follow these steps (you may want to compare them with the Xcode steps earlier in this chapter):

1. Launch Xcode and choose New ➤ Playground.

2. Set the options for your playground: the main one is the name. By default, you will probably be using iOS. If that is not the choice for platform, change it. The options are shown in Figure 1-7 (they're much simpler than the full app options shown in Figure 1-2).



Choose options for your new playground:

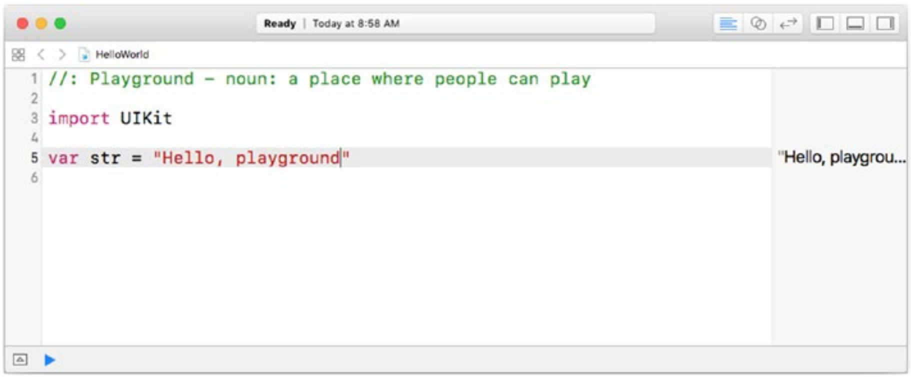
Name: HelloWorld

Platform: iOS

Figure 1-7. Set options for a playground

3. Click Next and choose the location on disk for the playground.

4. The playground you created is shown, as you see in Figure 1-8.



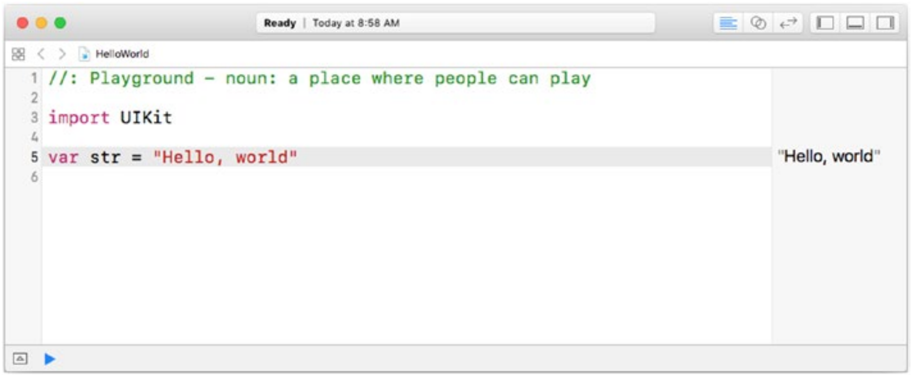
```
//: Playground - noun: a place where people can play

import UIKit

var str = "Hello, playground"
```

"Hello, playgrou...

Figure 1-8. *A basic playground is created*

5. You may have to wait a moment for the text in the sidebar to appear. The playground is running, and it needs to make the connection to the interface. Be patient if you don't see it immediately.

6. To convert this to a Hello World app, edit the word `playground` in the code to `world`, as shown in Figure 1-9.



```
//: Playground - noun: a place where people can play

import UIKit

var str = "Hello, world"
```

"Hello, world"

Figure 1-9. *Turn "Hello, playground" into "Hello, world"*

There's no build process, and there's no iOS Simulator—the playground executes in its own window.

Playgrounds and Xcode apps are similar in many respects, but different in many others. You need the overhead, power, and complexity of Xcode to build an app for an iOS device, but you can build code in a playground without any of that. If your objective at the moment is to build and test some code, a playground may be the best choice. Once you have tested your code, you can copy and paste it into an Xcode project.