Migrating to Swift from Flash and ActionScript

Radoslava Leseva Adams
Hristo Lesev

Apress®

Contents at a Glance

Contents

About the Authors

Radoslava Leseva Adams is a software developer and programming book author. Her affair with programming languages began in the early 1990s, when her father handed her a book on Basic as a form of summer holiday entertainment. Since then she has built a career out of freely jumping between different languages and platforms, including C, C++, Delphi, Java, ActionScript, Objective-C, and most recently Swift. She passionately hates wordy manuals and having to click more than once to do a build. Radoslava and her brother Hristo run EasyNativeExtensions.com and DiaDraw.com, where they help ActionScript developers do cross-platform programming with AIR Native Extensions.

Dr. Hristo Lesev is a software developer at heart, passionate speaker, educator, and entrepreneur. Having had long experience with C++, C#, and ActionScript for desktop and mobile platforms, lately he can be heard more and more often advocating for Swift as the latest and greatest. When not busy developing mobile apps, Hristo enjoys teaching other developers as an assistant professor at Plovdiv University, Bulgaria. He is obsessed with computer graphics and can often be found coding 3D stuff late at night.

About the Technical Reviewer

Robert Otani grew up in Los Angeles helping to repair cars in the family business, where got away with playing with welding torches and dangerous chemicals. He earned a B.S. in Physics from California State Polytechnic University and then entered the Ph.D. program at Arizona State University. He dropped out to pursue fortunes in the San Francisco Bay Area during the great Internet boom. Since then, he's worked as a designer and engineer for Sony Entertainment, Vitria, AvantGo (acquired by Sybase/SAP), Yahoo!, and virtual world startup IMVU. Most recently, he was iOS Lead at Mix.com (an Expa portfolio company) and is now an Engineering Lead for a yet-to-be named brand out of Silicon Valley. He's married to an amazing wife, with whom he's raising two children and a strange dog. He plans to update his site, otanistudio.com, sometime within the next decade.

Acknowledgments

Deciding to write a book while running a business full time and with one of us eight months pregnant was what a lot of our friends politely called "adventurous." They were right; working on this book was an adventure. We are indebted to our families for their support throughout this project and especially to Radoslava's husband, Steve, for reading all chapter drafts and offering helpful advice while being on full-time dad duty every weekend.

This project would not have been possible without the hard work of the Apress editorial team. We would like to thank Steve Anglin for the idea for this book and Ben Renow Clarke and Nancy Chen for keeping us on track.

Special thanks go to Robert Otani and Chris Nelson. Robert's constructive input ensured technical accuracy and helped us stay ahead of changes in the rapidly evolving language that is Swift and Chris burned the midnight oil to help us improve the text.

Some of our most indispensable resources have been Apple's book *The Swift Programming Language* (`https://goo.gl/1GRVRM`) and the online project "Swift Programming Language Evolution" (`https://github.com/apple/swift-evolution`).

Last, but not least, we are grateful to all contributors on `stackoverflow.com` who tirelessly shine light in the darker corners of Swift and Xcode.

Introduction

The Swift programming language has eased the learning curve for iOS development, compared with the early days when one had to become familiar with Objective-C. A lot of the Swift philosophy and syntax will be familiar to an ActionScript developer and will allow a rapid transition. This book offers the quickest way not just to learn a new programming language but also to migrate your whole workflow to a new platform.

Who Is This Book For?

Migrating to Swift from Flash and ActionScript is for developers who are transitioning to Swift for iOS. In particular, it has been written with the Adobe AIR community in mind to help bridge the gap between ActionScript and Swift for mobile devices.

You do not need background in ActionScript in order to benefit from this book, however. Basic experience with any object-oriented language would ensure that you adapt to Swift in no time.

Personally we tend to learn much quicker by following screenshots and diagrams and getting our hands dirty with code, rather than from reading pages and pages of text. So we have prepared examples and tutorials for you to do the same.

How to Use This Book

There is a lot to learn when you migrate your development process to a new platform: besides a new language, you have to become familiar with new tools and a new operating system, change your workflow, and learn what the best practices are in specific situations. This book is really four books in one, each part addressing an aspect of the migration process.

You don't need to read the material from cover to cover before you start coding with Swift. When you set your teeth into making native apps, we want this book to be your companion and provide guidance by walking you through a tutorial or two or by being a quick reference.

The book shows you how to make 16 different apps. Each chapter in the first three parts of the book offers a self-contained tutorial, so you are not dependent on having read and implemented the tutorials that come before it. The chapters on debugging and releasing your app are an exception and use code you have written in previous chapters.

Here is how the book is organized:

- **Part I: Tool Migration**. We recommend that you start here and go through the four chapters of **Part I** in order. We have intentionally kept this part brief. It will help you configure your environment and development devices, so that you are on your way to making your first app with Swift.
- **Part II: Workflow Migration**. This part walks you through the main parts of the programming workflow and shows you how to structure your user interface and use Xcode's help with layout, how to take advantage of concurrency, and how to use the debugger and automated testing tools. Apart from **Chapter 8**, which builds on the example of the preceding chapter to demonstrate debugging and testing techniques, each of the rest of the chapters comes with its own example. This means that you can go through this part in the order you find you need each topic as you migrate your workflow. Where there are new Swift concepts in the examples we have included pointers to the language reference part of this book, **Part IV**, so you can quickly find details on syntax or language idioms.
- **Part III: Making Apps with Swift—Applied Examples**. This is probably the most fun part of the book. It offers a series of tutorials that cover a lot of common scenarios you may want to include in your apps. Here you build 12 self-contained practical apps and learn how to
  - send e-mail, SMS, and make phone calls from your application.
  - post to social networks.
  - use the motion sensors and show a user's location on a map.
  - take photos, manipulate them, and communicate with the photo gallery.
  - work with local data and iCloud.
  - connect to and communicate with network services.
  - monetize your apps with advertisements and stay in touch with your users through push notifications.
  - build 2D and 3D games with iOS SDK's graphics frameworks.
- **Part IV: Language Migration**. This part was the most fun for us to write. In it we have tried to distill the main ideas that underpin the Swift programming language: it encourages you to be concise and at the same time forces you to be explicit and take maximum advantage of the compiler in order to ensure correct code. This is not meant to be a comprehensive Swift manual but to help you hit the ground running when it comes to language specifics. We recommend that you read the introductory **Chapter 17** first and then use the rest as a reference, which you can come back to whenever you need a Swift concept explained. There are no apps to build in this part of the book. Instead, we help you set up a Swift playground, where you can experiment with individual pieces of code. We have tried to make the explanations of the language concepts simple, so that our baba Ani can understand them too.[1]

[1]Baba (Bulgarian for "grandmother") Ani doesn't have ActionScript experience and is not even a programmer. She is one smart cookie, though.

- **Bonus Chapter: Publishing your app in the app store.** The point of making an app is to share it with the world and allow millions of users to enjoy your creation. Apple's process for releasing apps in the App Store, however, is far from intuitive. We thought that a book that that shows you how to create apps for iOS devices would not be complete without a walk through this process and some tips on how to keep it as smooth as possible.

A Note on Swift Versions

To say that Swift evolves quickly would be an understatement. A lot has changed in the language itself and in the iOS SDK since the first version of Swift was released in 2014. The examples in this book are consistent with version 2.2, which is the state of the art as we are finishing the last chapter. However, with Swift 3 just around the corner, we want you to be ahead of the wave of changes, so we have added notes and extra examples where code will be affected.

Note that some of the application programming interfaces (APIs) in the iOS SDK may be renamed when Swift 3 is released. We maintain a list of changes and source code for download for each of the tutorials in this book at www.apress.com/9781484216675. If you would like to be notified when updates are made, we encourage you to join our mailing list at http://diadraw.com/migrating-swift-flash-actionscript/.

Now let us get on with some work, shall we? We will see you in **Chapter 1**.

PART I

Tool Migration

CHAPTER 1

Setting Up Your Environment

Imagine a craftsman's workbench with tools nicely laid out and labeled, a cup of steaming coffee at one end . . . ideally the kind of image that gives you an itch to start a fresh project you can pour your heart into. This chapter is all about setting up that workbench by sourcing, installing, and configuring the tools that you will need for native iOS development.

In this chapter you will do the following:

- Learn about what Xcode can offer you as an iOS developer.
- Download and install Xcode.
- Set up Xcode for use with your Apple ID.

When you are done, you will have a fully set up IDE for developing iOS applications with Swift.

What Is Xcode?

Xcode is an integrated development environment (IDE). It is free and is developed by Apple. With it comes the development toolset for making apps for Apple devices: Mac, iPhone, iPod, iPad, Apple Watch, Apple TV.

FLASH ANALOGY

You have probably used one or more of the following IDEs to create Flash or AIR applications. They come with a source code editor, user interface editor, and integrated debugger and some even have a profiler, which you can use to measure your apps' performance.

- Adobe Flash Professional CC
- Adobe Flash Builder
- FlashDevelop

Electronic supplementary material The online version of this chapter (doi:10.1007/978-1-4842-1666-8_1) contains supplementary material, which is available to authorized users.

R. L. Adams and H. Lesev, *Migrating to Swift from Flash and ActionScript*,
DOI 10.1007/978-1-4842-1666-8_1

Xcode provides the usual features you would expect from an IDE and more. If you have developed iOS applications with AIR, you will find that Xcode improves and speeds up your workflow considerably. It offers all the tools you will need along the way: from rapid prototyping using Swift Playgrounds, through managing devices and debugging your app on them, to automated testing and even submitting your finished app to Apple's App Store. Xcode's profiling instruments could have a whole new book dedicated to them and will leave no doubt about your apps' performance by measuring speed, memory, energy and network usage, GPU and CPU utilization, file activity and lots more.

Before You Begin

To download, install, and run Xcode you will need the following:

- An **Apple ID**. If you don't have one, you can register at `https://appleid.apple.com/account`. This ID identifies you as an Apple user, just like an Adobe ID identifies you as an Adobe user. It will stand you in good stead for developing and testing simple apps and you won't need to enroll in any paid development program until you decide to use advanced SDK features or to publish your apps in the App Store—more on that in **Chapter 2**.
- A **Mac computer**. At the time of this writing Xcode 7.3 is the current release and requires that you run OS X 10.11 or later.
- At least **10 GB of free space** on your hard disk.

Step 1: Download Xcode

You have several choices for how to get Xcode, depending on whether you want an official release or a pre-release version. When new versions of Apple software are made available, they typically go through three stages: beta, seed, and official release. The main differences between versions in different stages are how stable they are, who can use a given version, and whether it can be used to build apps for release in Apple's App Store.

- **Beta**. This is a pre-release version, which is still under development and is available for download to anyone who has an Apple ID. Using a beta version puts you ahead of changes, as it allows you to update and test your apps with the latest tools and SDKs before they are officially available. There are a couple of drawbacks, however. Although they are close to the final thing, beta versions are by definition not as polished as official releases, so you may encounter bugs or inconsistencies. Another drawback is that apps that were built using beta tools are not accepted in Apple's App Store: to be able to release your app in the store, you will need to use either an official Xcode and iOS SDK release or a GM seed (see the next bullet).
- **Seed**. A seed version also comes out before the official release but, unlike a beta, is available only to participants in Apple's testing and feedback program, *AppleSeed*. Taking part in the program is voluntary and by invitation only. As a participant you are in effect taking part in shaping Apple's software, so you are expected to give active feedback. In fact, failure to do so may get you excluded from the program. Similarly to betas, you cannot release an app built with a seed, unless it has been labeled Gold Master (GM).

- **Official release**. An official release is made after Apple's new or updated development tools have undergone rigorous testing and customer feedback as betas and as seeds, so you can expect it to be as stable and as polished as it gets. The release version is the one you are expected to build your apps with before you release them in Apple's App Store. Note that to publish your apps in the store you will need to be a member of the Apple Developer Program, which requires paid membership.

We will go through the steps for obtaining an official Xcode release and a beta.

Option A: Get the Official Release

If you want to build apps for the app store, you will need the latest official version of Xcode and the iOS SDK. Open the **Store** app on your Mac and do a search for Xcode. Then click **Install (**Figure 1-1**)**. This will take care of the download and the installation in one go.

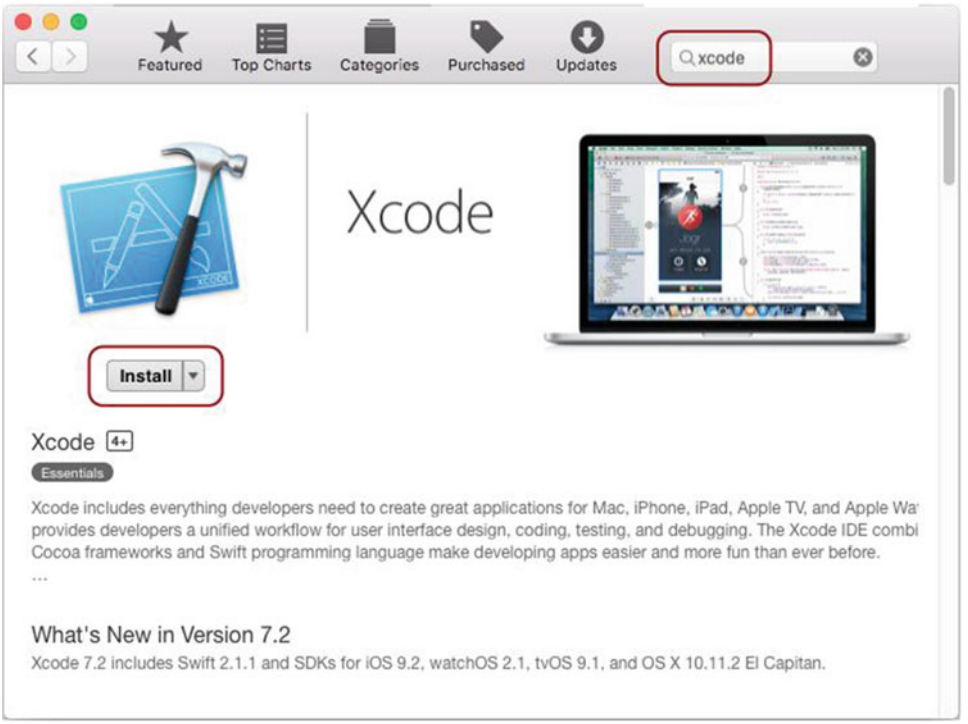

Figure 1-1. *Option 1: Download Xcode from Apple's App Store*

Option B: Get the Latest Beta

If the timeline for releasing your app is further in the future, you might prefer an even newer version of Xcode with the latest additions to Apple's SDKs, You can go for an early beta, usually available from Apple's web site at `https://developer.apple.com/xcode/download/` (Figure 1-2). This will download a `.dmg` file, which you can double-click to start the installation.

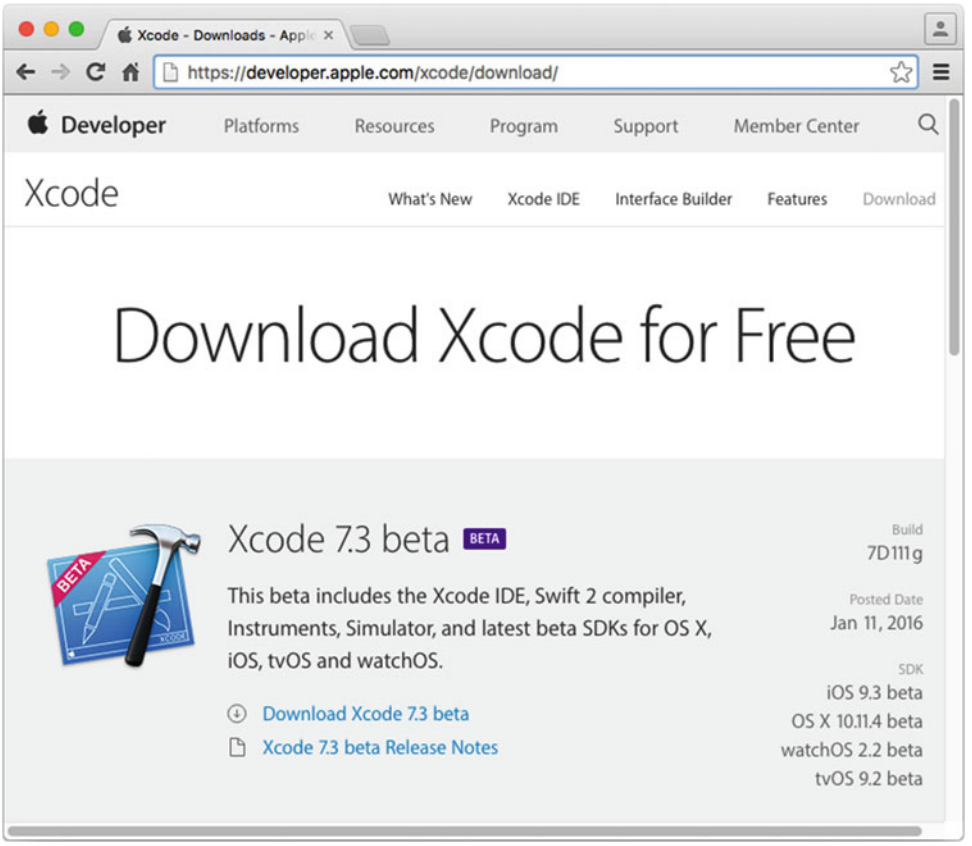

Figure 1-2. *Option 2: Download an Xcode beta from Apple's web site*

Step 2: Run Xcode

Now that you have downloaded and installed Xcode, let us run it and see what it looks like out of the box. We will also lift the curtain a bit and have a look at where the SDK files are, as well as other locations that will come in handy when you start developing apps.

Running Xcode for the First Time

The very first thing you see when you run Xcode after its installation is a **License Agreement**, which starts with the preemptive "Please scroll down and read all of the following terms and conditions carefully. . ." (Figure 1-3).

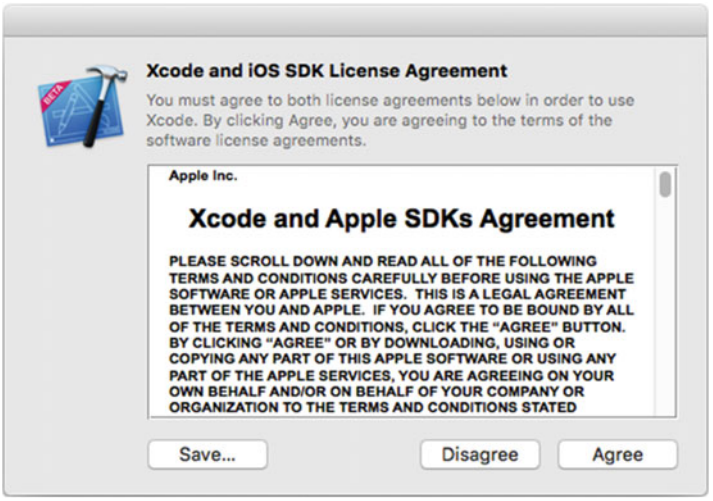

Figure 1-3. *You need to agree with Xcode's License Agreement before you can use it*

After scrolling down, reading carefully for about a whole of five lines, then clicking **Agree** anyway, Xcode's welcome screen appears (Figure 1-4). If you are anxious to begin development, jump straight to **Chapter 2**, which shows you how to make and run your first iOS application.

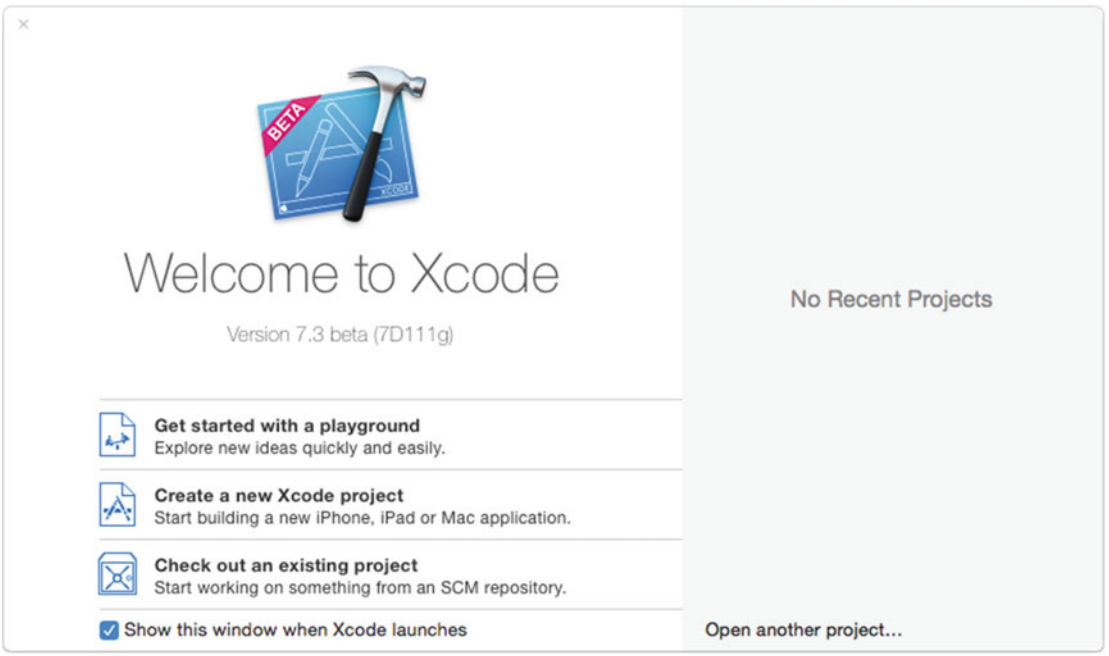

Figure 1-4. *Xcode's welcome screen*

Where Does It All Go?

If you like to be in control of your machine and development environment, take a detour with me and let us see where Xcode installs things. This is useful for when you want to do more advanced maintenance of your tools: check what's in the SDK, find logs when Xcode crashes (yes, it does . . .); know where it places your projects' temporary files among other things.

The Xcode Installation Folder

To find where Xcode was installed, open **Terminal** on your Mac and run the following command:

```
xcode-select --print-path.
```

This will give you the path to Xcode's installation folder, typically `/Applications/Xcode.app/Contents/Developer/`. To explore it in **Finder**, navigate to **`Xcode.app`**, right-click on it, select **Show Package Contents** and open the **Contents** folder.

The iOS SDK

You can find the iOS SDK that comes with Xcode in the `Developer/Platforms/iPhoneOS.platform/Developer/SDKs` folder inside the Xcode installation.

Location Settings

Xcode gives you control over where certain things should be found or stored. From Xcode's main menu select **Xcode ➤ Preferences ➤ Locations** to see what you can set (Figure 1-5).

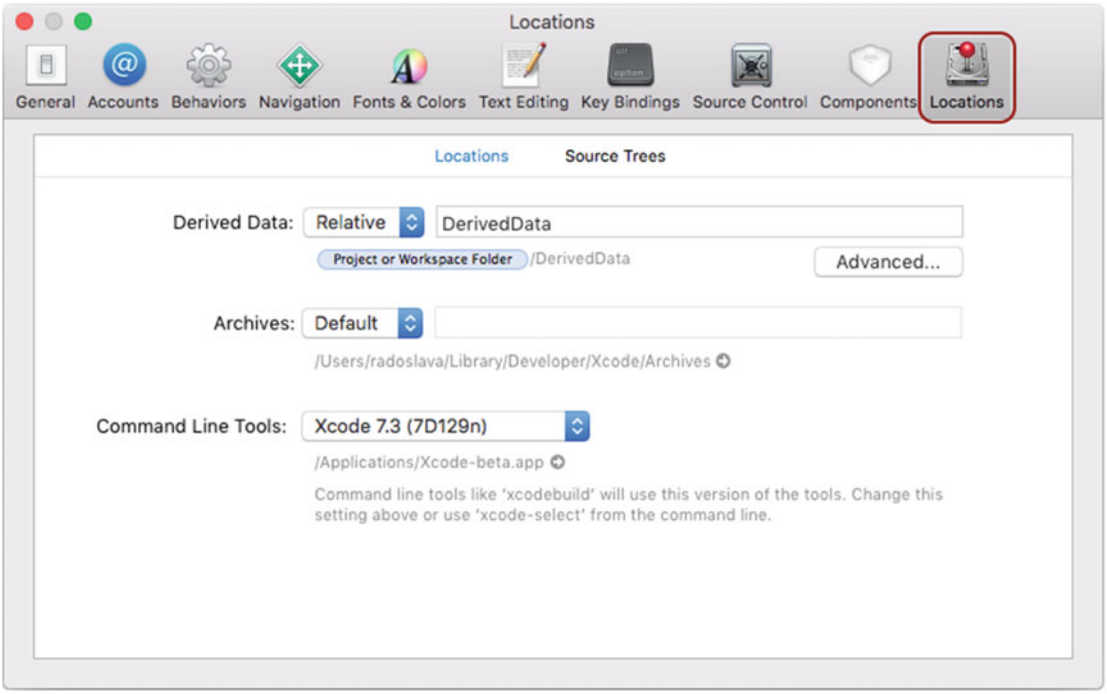

Figure 1-5. *Open Xcode ➤ Preferences ➤ Locations to see paths to important folders for your projects*

Following is a brief description of each of the locations you can control:

- **Derived Data.** Xcode uses two folders for putting temporary files when building a project: one is the **build** folder, which is always located in your project's directory; the other one is called **Derived Data** and you have a choice of where this goes: in the same location as the project or in the folder shared by all of your projects. It is good to know the location of this folder for dealing with compilation problems where a simple project clean doesn't seem to do the job. Deleting a project's **Derived Data** and **build** folders is equivalent to doing a manual clean.
- **Archives.** This is the location of the **.xcarchive** files Xcode creates for your projects: these contain your app executable and a .DSYM file—a file with debug information, which allows you to symbolicate a crash log, for example. You may have also used this to debug iOS native extensions for AIR.
- **Command-Line Tools.** Here you can choose the path to Xcode's command-line tools, in case you have more than one Xcode installation, thus more than one toolchain.

Documentation Settings

One thing you will not find in your fresh Xcode installation is documentation, as Apple provides comprehensive guides and manuals online. Sometimes, however, it is useful to be able to look things up when you are not connected to the Internet (long plane rides and family holidays in remote locations come to mind). To make any part of the documentation available offline open **Xcode ➤ Preferences ➤ Components ➤ Documentation** and click the arrow next to a document to download it (Figure 1-6). Documentation is stored in `~/Library/Developer/Shared/Documentation/DocSets`.

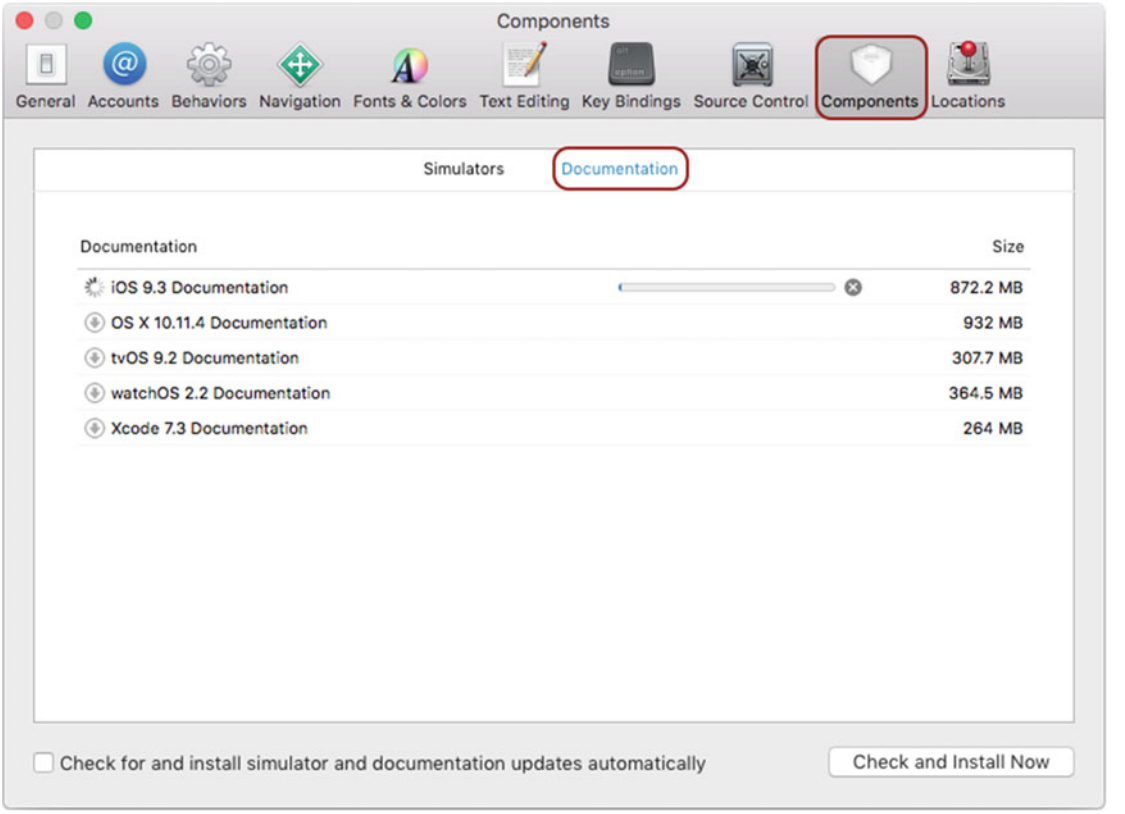

Figure 1-6. *Make documentation available for offline reading from Xcode ➤ Preferences ➤ Components*

Use the same pane to keep your offline documentation up to date: click the **Check and Install Now** button to get an update if one is available.

Step 3: Tell Xcode Who You Are

"Are we nearly there yet?" I can almost hear you say . . . I can't blame you: I am a coder at heart and to say that I don't enjoy following lengthy setup procedures would be putting it mildly. If you want to run applications on a physical device, there is one more thing left to do: Xcode needs to know about you. This section explains why and shows you how to finish the setup.

If, on the other hand, you would like to jump straight to creating your first native iOS app and see it run in a simulator, you can go to **Chapter 2** and come back to this step when you need to.

Before you can install and test your apps on physical devices the apps will need to be cryptographically signed. A code signature uses a *signing identity* to ascertain that an app was developed and built by you and a *certificate*, created specifically for that app or for a group of apps, which allows the group members to use particular services. A signing identity is based on your Apple ID: Xcode can create one for you and install it in **Keychain Access** automatically.

To help Xcode create a signing identity, you need to let it know your Apple ID: from Xcode's main menu select **Xcode ➤ Preferences** and in the dialog that appears open the **Accounts** tab (Figure 1-6). Click the + button to add a new account and enter your Apple ID and password.

After your account has been added and appears in the **Apple ID** column, select it, then select the relevant **Team** in the right part of the dialog shown in Figure 1-7—this could be a development team your Apple ID is part of or just your Apple ID. Then click the **View Details...** button. This will open another dialog with a list of signing identities (Figure 1-8).

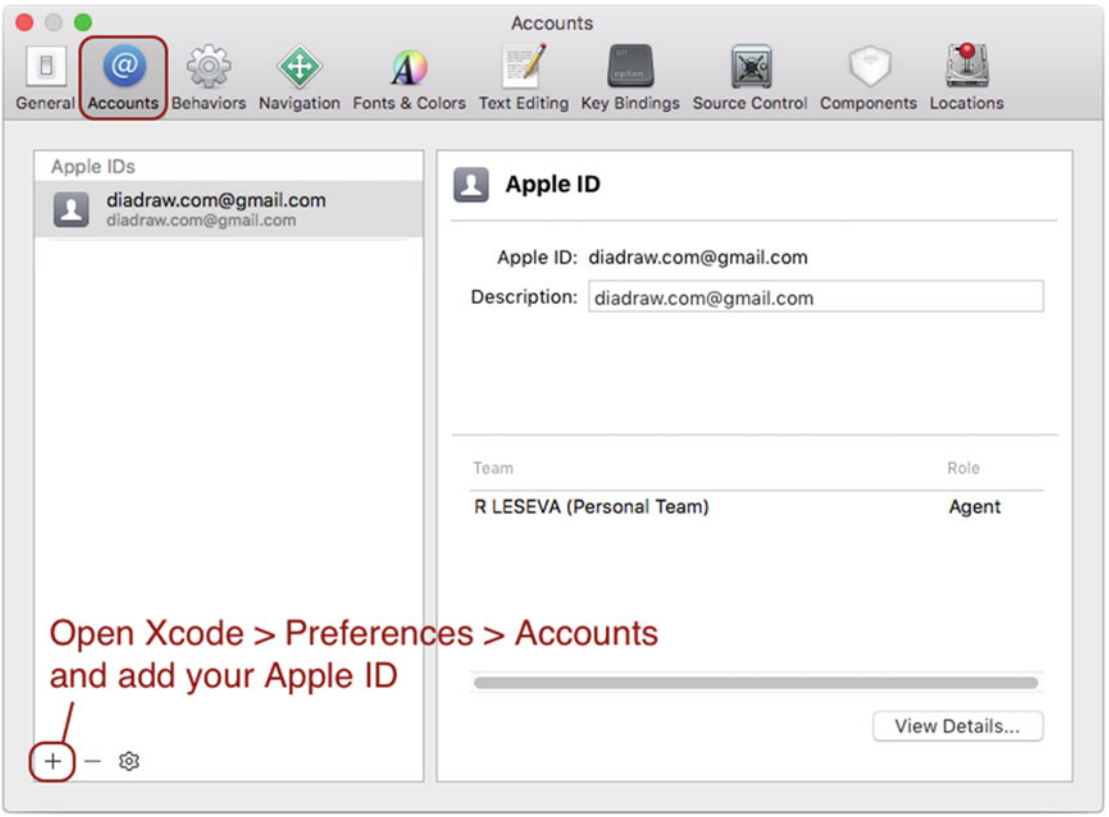

Figure 1-7. *Let Xcode know who you are by adding your Apple ID in Xcode ➤ Preferences ➤ Accounts*

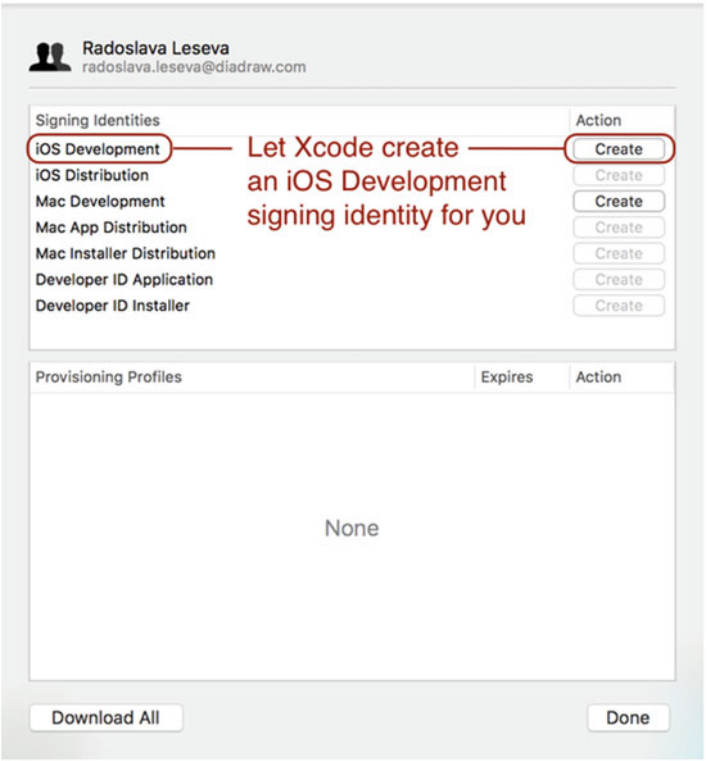

Figure 1-8. *Xcode 7 can create a signing identity for you*

A signing identity is effectively a certificate that will be installed in the keychain of your development machine and is used to sign the apps you build. For the purposes of the tutorials in this book we will need an iOS Development signing identity. Select it from the list and click the **Create** button.

To see what Xcode has created, launch the **Keychain Access** app on your Mac. Under **Keychains** select **login** and under **Category** select **My Certificates**. Your signing identity should appear in the list on the right as iPhone Developer: <The name you registered your Apple ID with> and should say certificate in the **Kind** column (Figure 1-9).

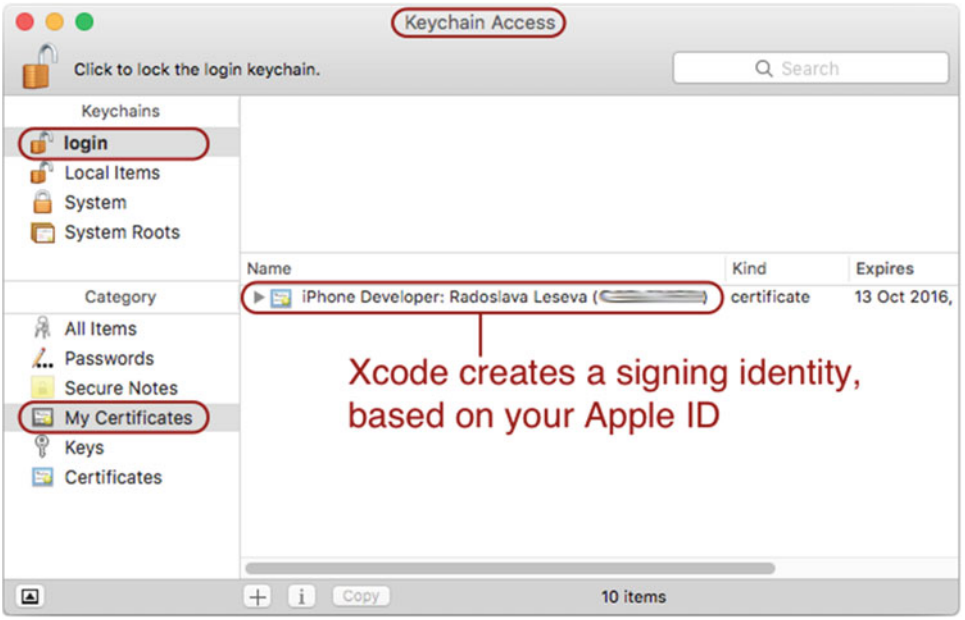

Figure 1-9. *Check what Xcode has created in Keychain Access ➤ login ➤ My Certificates*

Summary

Remember the workbench image, with which we opened this chapter? I hope that this is how your development machine looks and feels now: the tools for native iOS development laid out and ready to start crafting beautiful code. I'm afraid you will have to make the cup of coffee yourself, though.

With your choice of motivational drink ready, let us get on with making your first application in **Chapter 2**.