Learn how to make Watch apps with WatchKit and Swift

Learn WatchKit for iOS

Kim Topley

Apress®

Contents at a Glance

Chapter 1

Welcome to the Apple Watch

When I learned to program computers more years ago than I care to remember, the hardware that I used literally filled a room. Ten years later, it was possible to build a computer that was small enough and cheap enough to have in your home or on your desk at work. Today, we think nothing of carrying in our pockets computers that are more powerful than the ones that were used back in the "good old days" to run a business or navigate a spacecraft to the moon and back. We use them to schedule our lives, read books, listen to music, send and receive e-mails, and even make phone calls. Over the last 15 or so years, several companies have experimented with the idea of making it possible to wear your phone or personal computing device instead of carrying it in your pocket. By 2014, Samsung, Sony, Motorola, and others had taken this idea to its logical conclusion by developing and marketing a range of *smartwatches*—devices that are basically wearable computers packaged as wristwatches—with varying degrees of success. In 2015, Apple released its own wearable computing device called the Apple Watch. It remains to be seen whether it will be as popular as the company's other consumer products. If it is, then it represents a major new opportunity for iOS developers to profit by extending their existing applications to work with the Apple Watch and to write new applications that make use of its unique features.

As you'll see in the first few chapters of this book, an Apple Watch application is really just an extension of an application written to run on an iPhone, so you'll need to know how to write iPhone applications. This book does not teach iPhone programming from scratch—I assume you already have some experience of developing for iOS using the Swift

programming language. If you don't, then you'll need to first learn the basics by reading an introductory iOS programming book, such as *Beginning iPhone Development with Swift: Exploring the iOS SDK* by David Mark, Jack Nutting, Kim Topley, Fredrik Olsson, and Jeff LaMarche (Apress, 2014—see www.apress.com/9781484204108).

Your First Watch Application

Let's dive straight in and write our first watch application. Like iPhone and iPad applications, Apple Watch applications are developed using Xcode and can be debugged using both the iOS simulator and a real device. At the time of writing (May 2015), the simulator does not let you test the complete lifecycle of a watch application—for example, the simulator does not show the home screen of the watch, so you can't see your application's icon and launch it from there. You also can't configure glances or test how your application handles local notifications (glances and notifications are covered in Chapters 8 and 9, respectively). Although the simulator is useful for debugging, testing on a real device is important before releasing your application to the App Store.

The classes that you'll use to build Watch applications are included in the *WatchKit framework*, which is included in the iOS SDK starting with version 8.2. You can build Apple Watch applications with version 6.2 or higher of Xcode. The examples in this book use version 1.2 of the Swift language (which was introduced with Xcode 6.3 and is not completely compatible with the version of the language supported by Xcode 6.2) and were tested with Xcode 6.3 and 6.4. I assume that you are already an iOS developer, so you should have Xcode installed. If you don't have the most recent version, you can get it from Apple's developer web site at https://developer.apple.com/xcode/downloads.

EXAMPLE SOURCE CODE

The source code for the examples in this book can be downloaded from the book's page on the Apress website at www.apress.com/9781484210260. Once you have downloaded the source code archive and unpacked it, you'll see that each example has a separate folder with a name that includes the number of the chapter to which it relates. These folders contain the completed application. You'll also find folders that contain images and other resources that you'll need when following the step-by-step instructions to construct each example from scratch.

Creating a WatchKit Application

Fire up Xcode and let's create our first WatchKit application. As an experienced iOS developer, you already know how to create a project for a new application—just go to the Xcode menu bar and open the New Project dialog by selecting **File ➤ New ➤ Project...** and then choose the appropriate template from the iOS section. Try that now. Unfortunately, you'll find that there is no template for a WatchKit application. That's because you can't just build a WatchKit application and release it to the App Store. An application for the Apple Watch is supposed to be an add-on to an existing iOS application, so you first have to create an iOS application and then add a WatchKit application to it. This makes a lot of sense—the Apple Watch is smaller and less powerful and has a much smaller screen than the iPhone to which it is paired. It can't really support full-fledged applications (at least not yet). Instead, you'll need to decide which parts of your application's functionality, if any, make sense in the context of a watch and implement those features using WatchKit. Later in this book, you'll see a practical example of this—we'll take an iPhone weather application and add the ability for the user to view weather forecasts on Apple Watch. However, in most of the examples in this book, the iOS application does nothing.

So, how do we start writing our add-on WatchKit application? It's easy. First we need to create the hosting iOS app. Select **File ➤ New ➤ Project...** and then choose **Single View Application** from the **iOS Application** section of the template chooser (see Figure 1-1).

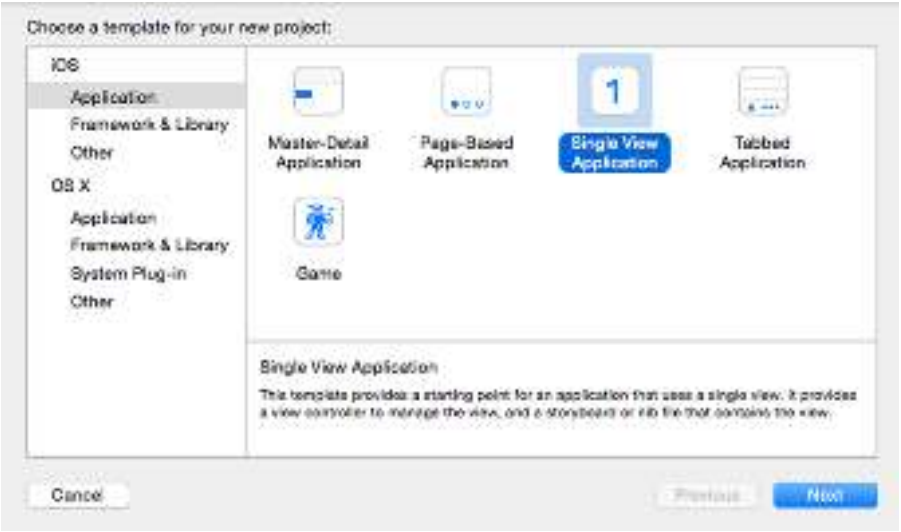

Figure 1-1. *Creating a host application for a WatchKit app. Notice there is no WatchKit app project template*

Click **Next** and then enter `HelloWatch` in the **Product Name** field. Enter the same values that you would use for other iOS projects in the **Organization Name** and **Organization Identifer** fields. Select `Swift` as the **Language**, `Universal` for **Devices**, and leave the **Core Data** check box unchecked. Press **Next** and choose the save location for the project.

> **Note** You can use Objective-C to write a WatchKit application if you prefer. All the source code in this book is written in Swift.

To add a WatchKit application to the project, you need to add a new target. From the menu bar, select **File ➤ New ➤ Target…**. In the template chooser that appears, select **WatchKit App** from the **iOS Apple Watch** section, as shown in Figure 1-2, then click **Next**.

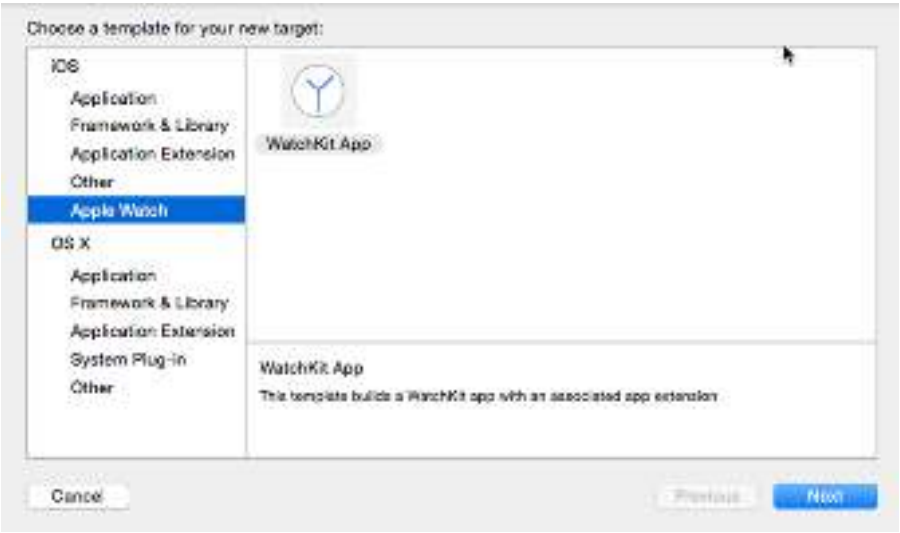

Figure 1-2. Adding a WatchKit app target to an iOS application

The next page of the chooser contains various options, most of which have fixed values derived from the name and identifier of the host application (see Figure 1-3). Uncheck **Include Notification Scene** and **Include Glance Scene**, leave everything else on this page as it is, and click **Finish** to add the WatchKit target.

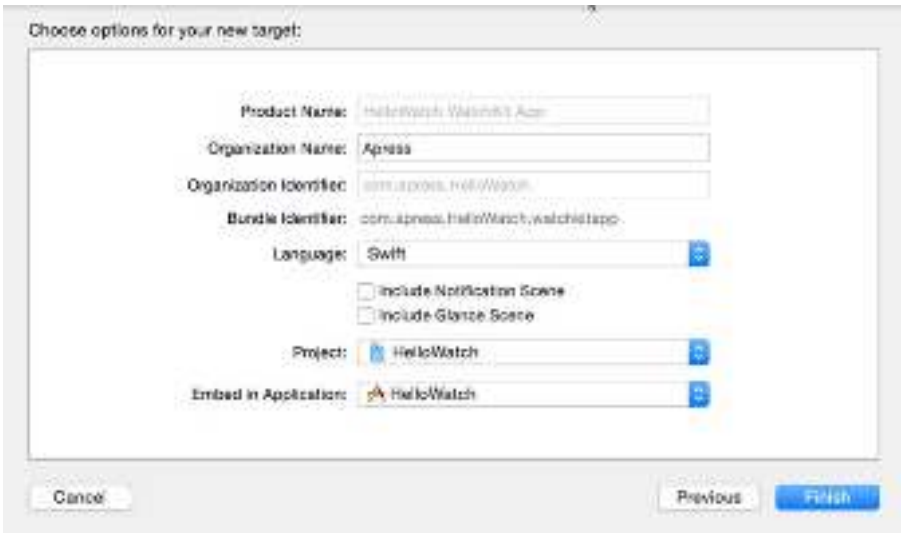

***Figure 1-3.** Completing the addition of the WatchKit app target*

Xcode prompts you to confirm that you want a scheme that will allow you to run your WatchKit application to be activated. Check the **Do not show this message again** checkbox and then click **Activate** to activate the scheme.

In addition to this new scheme, Xcode adds two new groups and two new targets to your project. Let's take a look at what's in the new groups. Select the Project Navigator tab if it's not already selected (a quick way to do this is to press ⌘**1**) and you'll see that the new groups are called `HelloWatch WatchKit Extension` and `HelloWatch WatchKit App` (see Figure 1-4).

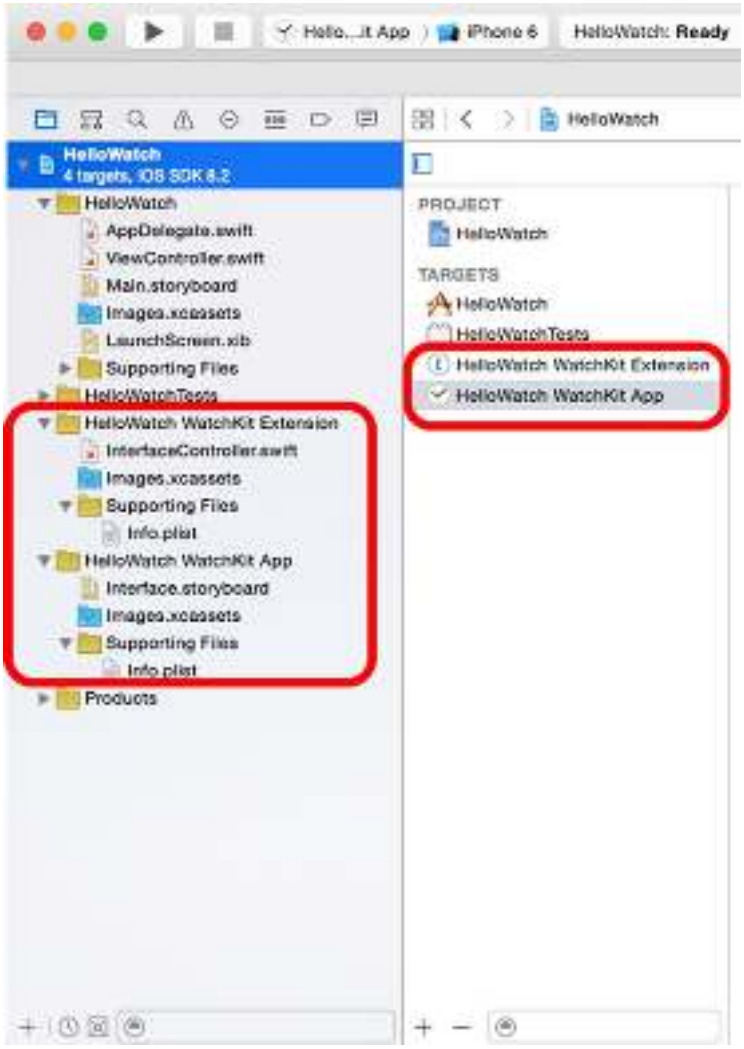

Figure 1-4. *The WatchKit Extension and WatchKit App groups and targets in Xcode*

The `HelloWatch Watchkit App` group contains a storyboard and an asset catalog that you'll use to create the user interface for your WatchKit application. You'll notice that it does not contain any `.swift` files—that's because all the code for the application is in the WatchKit Extension, not in the WatchKit application itself.

Now let's look at the `HelloWorld WatchKit Extension` group. As you can see in Figure 1-4, Xcode created a file called `InterfaceController.swift` in this group. Select this file in the Project Navigator so that you can see what it contains (Figure 1-5).

Figure 1-5. *The skeleton interface controller for a WatchKit application*

As you can see, `InterfaceController` is a subclass of the WatchKit class `WKInterfaceController` (the names of all of the WatchKit classes begin with the letters `WK`). `WKInterfaceController` is WatchKit's version of the `UIViewController` class in UIKit—it contains the controller code for your application's user interface. The Xcode template gives you just a single controller class that manages the screen that the user sees when your application launches. For some applications, that's all you need, but for more complex applications, WatchKit provides mechanisms that allow you to navigate from screen to screen, including segues that are similar to the ones that you are familiar with from UIKit. There is one `WKInterfaceController` for each screen in your application, so if your application requires more than one screen, you'll need to have more than one `WKInterfaceController`. Chapter 5 talks more about interface controllers, including how to write applications with more than one interface controller.

WATCHKIT DOCUMENTATION AND RESOURCES

There is plenty of documentation available for WatchKit. You can get to the reference page for any WatchKit class by searching for it in the Xcode documentation window (**Window ➤ Documentation and API Reference**, or press ⇧⌘0). A quicker way do the same thing if you have the class name in some code is to hover the mouse over the class name and then press the ⌥ (option) key. This opens a pop-up window with brief documentation of the class, including a link to the full documentation page. The same technique works for method and property names.

You can find reference information for all of the WatchKit classes online at https://developer.apple.com/library/ios/documentation/WatchKit/Reference/WatchKit_framework/index.html and a full list of all of the available documentation and resources at https://developer.apple.com/watchkit. You should pay special attention to the *Apple Watch Human Interface Guidelines*, which you can find at https://developer.apple.com/watch/human-interface-guidelines. Finally, there is a lot of useful discussion on the WatchKit forum at https://forums.developer.apple.com/community/app-frameworks/watchkit.

It is important to note that the code for a WatchKit application runs in the WatchKit app extension on the iPhone, not on the Apple Watch itself. That's so important that I'll say it again: **None of your WatchKit application's code runs on the Apple Watch—it executes in an extension on the paired iPhone**. That means you don't have direct access to the Apple Watch hardware. Instead, you have to rely on the WatchKit classes to draw your user interface on the screen, react to user input, tap the user's wrist when required, and so on. You can find out more about what your WatchKit application can and can't do in the section "Some Things That a WatchKit Application Can and Can't Do" at the end of this chapter.

All of your application's code files will be in the WatchKit Extension group and will be assigned to the WatchKit Extension target. The exception to this rule is any code that you need to share between the WatchKit app extension and the iOS application, which you should put into a shared framework. I discuss this in detail in Chapter 7.

> **Caution** Do not add any code to the `Hello WatchKit App` group or target—all the code for the WatchKit application *must* be in the group and target that Xcode created for your WatchKit app's extension.

EXTENSION PROGRAMMING

Don't be concerned if you're not familiar with writing iOS extensions. For the most part, coding an extension is the same as coding an application. The main thing you need to be aware of is that some APIs are not available to code running in an extension. I'll point out some of the differences as we progress through the book. Usually, you can work around the limitations by delegating work to your iOS application and you'll see exactly how to do that in Chapter 7. You can read about extensions in the *App Extension Programming Guide*, which you'll find at `https://developer.apple.com/library/ios/documentation/General/Conceptual/ExtensibilityPG/index.html`.

Building the User Interface

There are three ways to construct the user interface for an iOS application: you can create it entirely in code, you can build it from nib files, or you can use a storyboard. By contrast, there is only one way to build the user interface for a WatchKit application—you have to use the storyboard that Xcode created when you added the WatchKit application target. Select the `Interface.storyboard` file in the WatchKit App target to open it in the editor area. Right now, the storyboard contains a single empty screen, as shown in Figure 1-6.

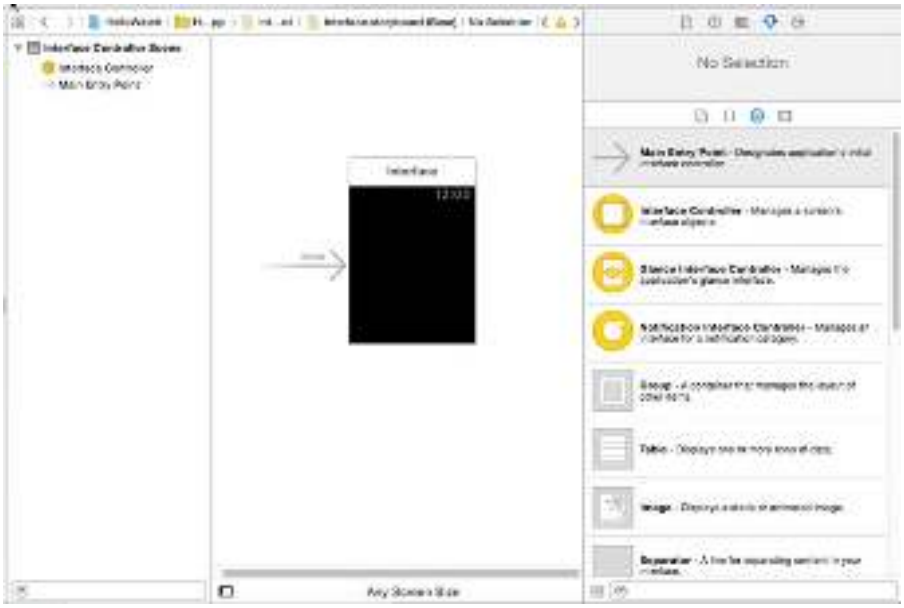

Figure 1-6. *The WatchKit application's storyboard and the Object Library*

To the right of the editor area, you'll find the Object Library (use **View ➤ Utilities ➤ Show Object Library** to make it visible if necessary). While you are editing a WatchKit application storyboard, the Object Library contains WatchKit *user interface objects*, which are roughly equivalent to views in UIKit. As you can see, there are far fewer of these than there are UIKit views. Take the opportunity to scroll through the list to see what's available—Chapters 3 through 6 talk in detail about them all.

Our first WatchKit application is very simple, consisting of just a label and an image. Let's start with the label. Locate a `Label` object in the Object Library and drag it over the watch screen in the storyboard. As you do so, the cursor changes, and a blue outline appears on the watch screen, as shown on the left in Figure 1-7.

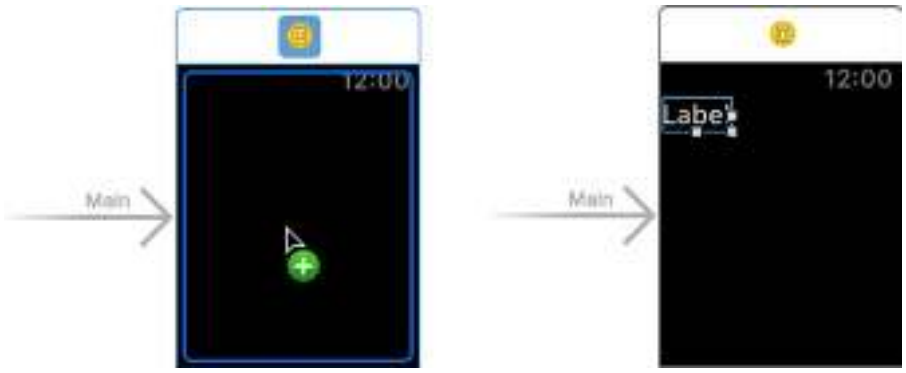

***Figure 1-7.** Adding a label to the user interface*

The outline indicates the area in which you can drop the label. In this case, because the screen is currently empty, Xcode indicates that you can drop it anywhere. With the label positioned anywhere over the watch screen, release the mouse to drop it and you'll see that it instantly snaps to the top left corner (see the right image in Figure 1-7). Unlike UIKit, WatchKit does not have a sophisticated layout mechanism like Auto Layout. In fact, pretty much all you can do is arrange your user interface objects in a row or a column. Objects that are added directly to the watch screen (which are actually being added to the screen's interface controller) are always arranged vertically, one above the other. That's why the label was placed right at the top of the screen.

Make sure the label is selected, which is indicated by the blue outline and the white resize handles shown in Figure 1-7. If it's not currently selected, just click on it. Now open the Attributes Inspector (**View ➤ Utilities ➤ Show Attributes Inspector**) to see the attributes of the label that you can set (see Figure 1-8).

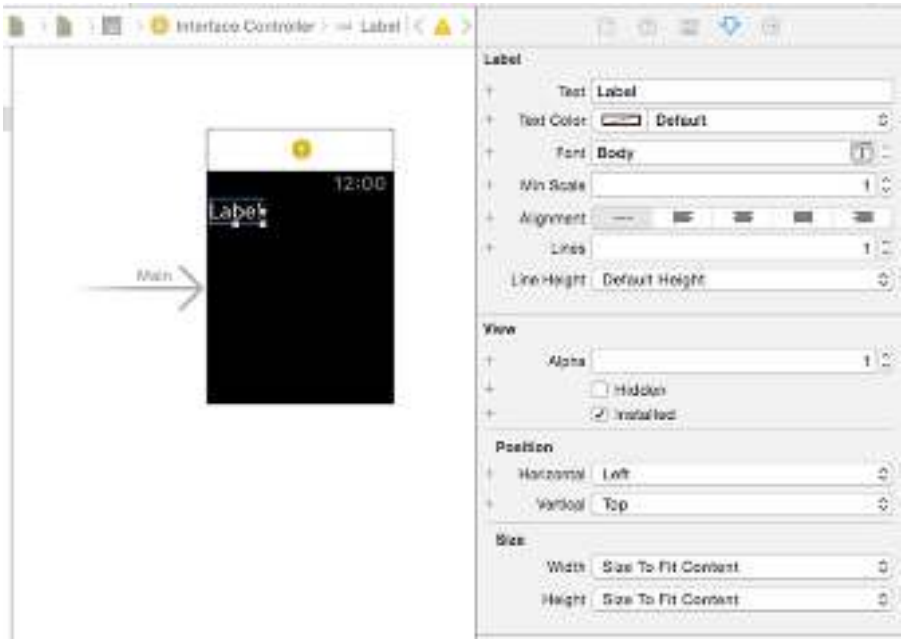

Figure 1-8. *Inspecting the attributes of the* `Label` *object*

Use the Attributes Inspector to change the label text from `Label` to `Hello Watch`. Next, use the `Font` control to change the text font from `Body` to `Headline`. As you can see, the label resizes automatically to fit its new content. Let's also horizontally center the text on the screen. A little below the `Text` field is a segmented control labeled `Alignment`, which sets the position of the text within the label itself. You can click the third segment from the left to center the text within the label itself, but that doesn't do what we want—we actually need to reposition the label relative to the interface controller. For that, you need to use the controls in the `Position` section of the Attributes Inspector. Click the `Horizontal` selector and choose `Center`. You'll see that the label repositions itself horizontally while remaining at the top of the screen, as shown in Figure 1-9.

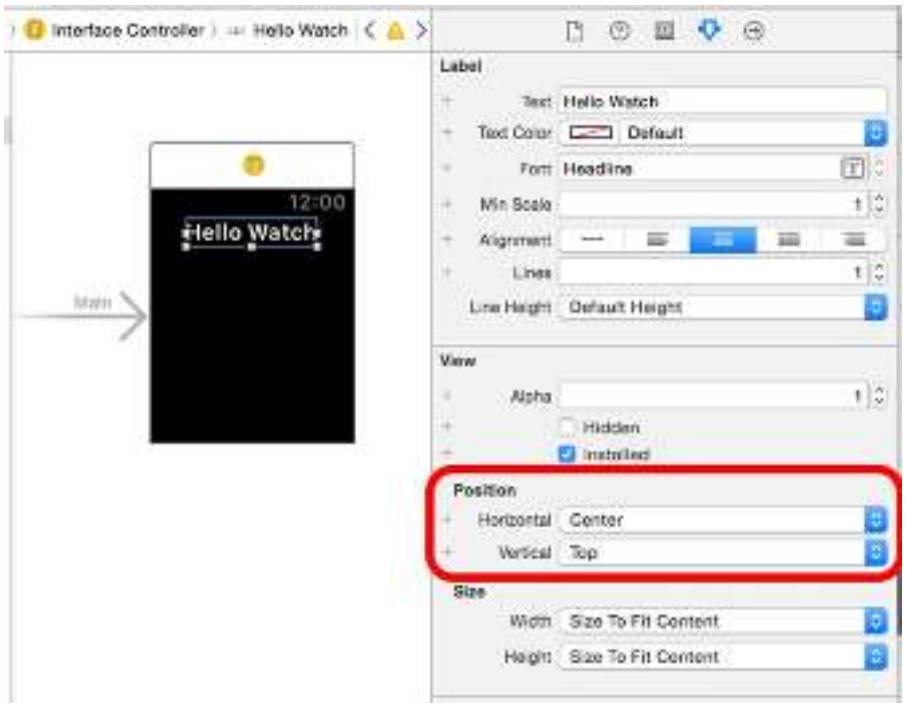

Figure 1-9. Horizontally centering the label

Now let's add an image below the label. Drag an `Image` object from the Object Library and drop it onto the storyboard. It doesn't matter where you release the mouse as long as it's over the watch screen and somewhere below the label—the image will be positioned underneath the label, up against the left edge of the screen.

The `Image` object is empty until you give it something to display. In the `1 - HelloWatch Icon` folder of the book's example source code archive, you'll find a suitable image. In the Project Navigator, select `Image.xcassets` from the HelloWatch WatchKit App group to open the WatchKit app's asset catalog and then drag the image into the editor area and drop it (see Figure 1-10).

Figure 1-10. Adding an image to the WatchKit application

> **Note** The images that you plan to use in the WatchKit application should be added to the asset catalog of the WatchKit application itself. As you'll see later, it is possible to programmatically install images from WatchKit extension's asset catalog, but images configured in the storyboard *must* come from the WatchKit application's asset catalog.

The image came from a file called `SmileyFace@2x.png` and it was added to the `2x` section of the entry in the asset catalog. The Apple Watch has a Retina screen, so all images must be prepared at double the required point size. In this case, the image is a 206-pixel square, which maps to a 103-point square from the point of view of the WatchKit software.

Back in the storyboard, make sure the Attributes Inspector is open, select the `Image` object, and then choose `SmileyFace` from the `Image` selector. The smiley face image appears in the storyboard, but it's still aligned to the left of the screen. You can center the image using the `Horizontal` selector from the `Position` section, as you did with the label.

Before running the application, let's give it a title. To do this, select the interface controller by clicking it in the Document Outline and then enter `Hello` in the `Title` field in the Attributes Inspector. Press the Return button, and the title appears at the top left of the watch screen (see Figure 1-11).

Figure 1-11. Setting the interface controller's title

Running the Application on the Simulator

When you added the WatchKit application target to your project, Xcode also added a scheme that you can use to run it on the simulator (or a real Apple Watch). Open the scheme selector and you'll see that there are two schemes for `HelloWatch` (shown in Figure 1-12).

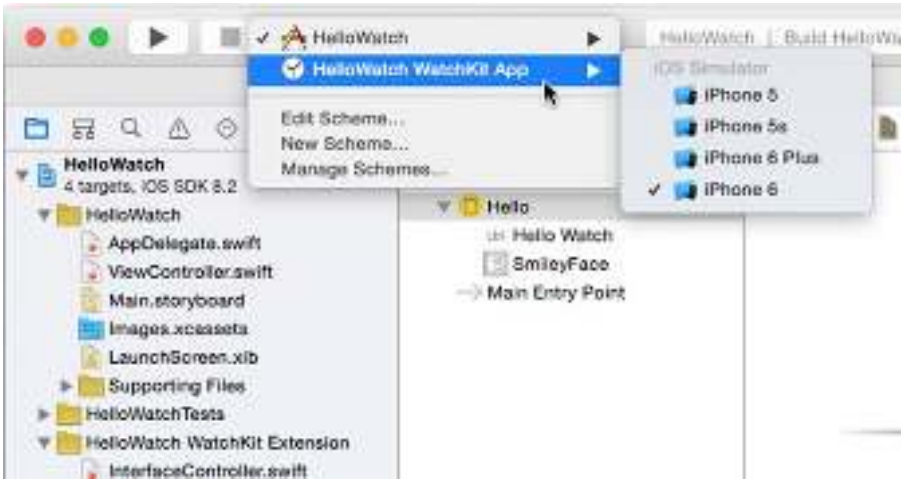

Figure 1-12. Selecting the scheme to run the WatchKit application

The scheme labeled `HelloWatch` runs the HelloWatch iOS application, which is just an empty shell because we didn't add anything to it. The second scheme, labeled `HelloWatch WatchKit App`, is the one that you need to launch the WatchKit application. Select this scheme and then choose one

of the iPhone simulators. You need to select an iPhone simulator because when you start your WatchKit application, the code in the HelloWatch WatchKit extension is actually run on the iPhone simulator. In a real application, the code in the extension would perform initialization and prepare the interface controller to be displayed. For this simple example, we don't need any initialization, so the extension doesn't do anything useful.

The user interface of the WatchKit application is not displayed on the iPhone simulator—instead, it's shown on an external display. To make this possible, you need to select an appropriate simulated external display. To do that, with the `HelloWatch WatchKit App` scheme selected, build and run the WatchKit application in the usual way using **Product ➤ Run** from the menu bar or the keyboard shortcut ⌘**R**. When the iOS simulator starts, select **Hardware ➤ External Displays** from its menu bar and you'll see a menu of external displays, as shown in Figure 1-13.

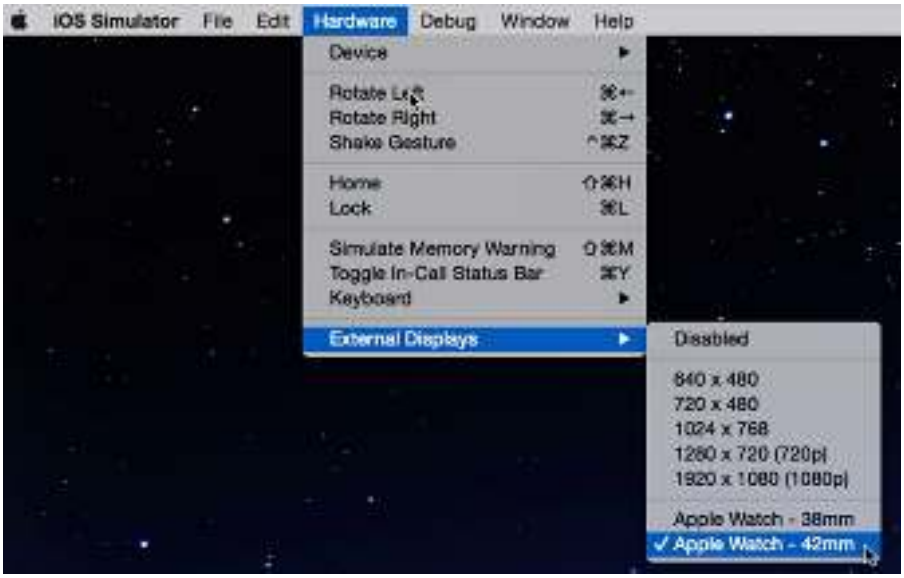

***Figure 1-13.** Choosing an external display for the Apple Watch*

Select the 38mm Apple Watch screen and a separate empty window will open. Now stop and rerun the WatchKit application and you should see the user interface from your storyboard in this window. Next, select the 42mm screen and run the application again to see how it looks on the larger screen. The results are shown in Figure 1-14.

Figure 1-14. *Running the HelloWatch WatchKit app on the 38mm (left) and 42mm (right) screens*

As you can see, this simple application works well on both sizes of screen, although you might prefer to add more vertical spacing to get a more balanced view on the 42mm screen. In Chapter 2, you'll see that it is possible to do that by creating a single design in the storyboard that has spacing and other attribute values that depend on the size of the screen of the watch on which the application is run.

Running the Application on an Apple Watch

To run an application on a real Apple Watch, the watch must be paired to your iPhone and the iPhone must be connected to your computer. You'll also need to be a paid-up member of Apple's iOS Developer program because Xcode will need to create a provisioning profile that includes the watch.

Before we run our example on the Apple Watch, let's give it a home screen icon. You need to provide several icons for a production WatchKit application. You can see what they are by selecting `Images.xcassets` in the HelloWatch WatchKit App group in the Project Navigator and selecting the `AppIcon` image set (see Figure 1-15).

Figure 1-15. The full set of application icons for a WatchKit icon

For most of the examples in this book, we won't supply any of these icons, and for this example, we're only going to include the home screen icon. You should, of course, design an appropriate set of icons for any application that you intend to release to the App Store.

> **Note** You'll see warnings in Xcode if you don't supply all the icons that Xcode thinks you need to include. Click on the warning indicator (the yellow triangle in the Activity View at the top of the Xcode window) to see which icons are missing.

Below each group of icons, you'll see a description of what each icon is used for and its size. The slot for the Home Screen icon, which is in the middle in Figure 1-15, shows that the icon needs to be a 40 pt. square (all Apple Watch icons are square). Because the Apple Watch screen is a 2x Retina display (which means that 1 point is equivalent to 2 pixels), you actually need to create an image that's 80 pixels square. Some icons need to be sized differently for the 38mm and 42mm watches. For example, the Short Look Notification icon (you'll find out what this is used for in Chapter 9) is 86 points on the 38mm device and 98 points on the 42mm device.

> **Tip** Apple provides design guidelines for icons (and for all aspects of building Apple Watch applications) in its document *Apple Watch Human Interface Guidelines*, where you'll find links to a set of resource files that include Photoshop templates that you can use as the basis for your icon designs.

You'll find a Home Screen icon for our example in the `1 - HelloWatch Home Screen Icon` folder of the example source code archive. Drop the icon onto the Home Screen slot of `Images.xcassets`, and you're almost ready to run the application on the watch.

If you haven't already done so, connect your iPhone to your computer and make sure that the iPhone is paired with your Apple Watch. You can check that Xcode has recognized your watch by opening the Devices window (**Window ➤ Devices** in the Xcode menu). Select your iPhone in the device list on the left of the window and you should also see the watch, as shown in Figure 1-16.

Figure 1-16. The Apple Watch in the Xcode Devices window

To run the application, go the scheme selector in Xcode, select the HelloWatch WatchKit App scheme, and choose your iPhone as the target device (see Figure 1-17). Then click the `Run` button.

Figure 1-17. Choosing the scheme and target to run the example on your watch

The first time you do this, you'll probably see a dialog box reporting that Xcode failed to sign the WatchKit application. That's because you need to add the watch to your account at the Developer Portal and create a provisioning profile that includes it. You can do this manually, if you prefer (you'll find the identifier for the watch in the Devices screen, as shown in Figure 1-16), but it's much easier to allow Xcode to do it for you. Click the **Fix Issue** button in the dialog, and Xcode will try again. The HelloWatch application will be installed on your iPhone, and the watch application will be sent to the watch—if you have your watch screen unlocked and are showing the home screen, you'll see this happen. The application's icon should appear on the watch home screen. You can see the application's icon at the bottom right in Figure 1-18.

Figure 1-18. *The HelloWatch application installed on the Apple Watch*

> **Tip** To take a screenshot of the watch, press and hold the side button and then click the digital crown. The screen should flash, and the screenshot will be stored in the Camera Roll on the paired iPhone.

Even though you clicked the Run button in Xcode, the application doesn't automatically run on the watch—you need to tap on its icon to launch it. When you do that, you'll see a launch screen with the application's name and a progress indicator for a short while, and then the application's main screen appears (see Figure 1-19).

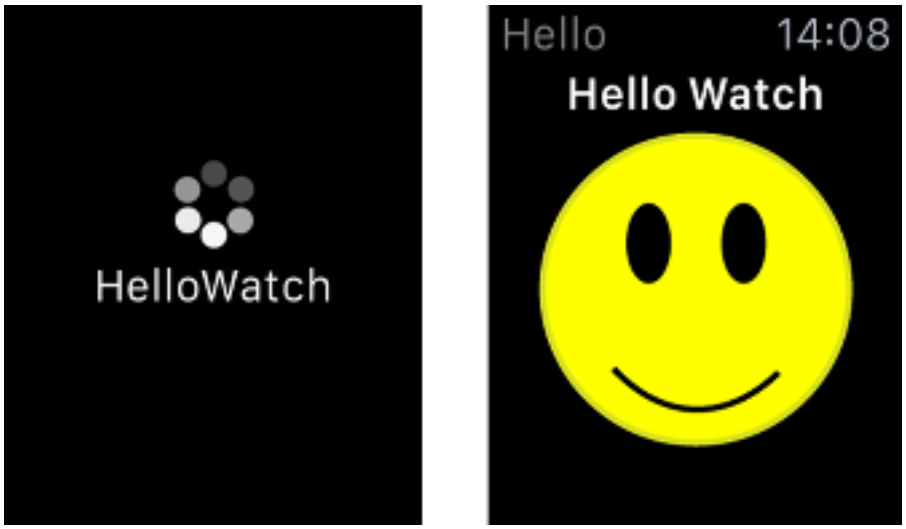

Figure 1-19. *Launching the HelloWatch application on the Apple Watch*

> **Note** Sometimes launching the application for the first time appears to stall. Wait for a few seconds. If it still doesn't work, click the digital crown to return to the home screen and try again.

The application remains installed until you do one of three things:

- Delete it on the watch by pressing and holding your finger on the screen and clicking the delete icon that appears, just as you would to delete an application from your iPhone. This is not recommended because the application does not get reinstalled if you run it again from Xcode. To fix this, delete the application from the iPhone (or the iPhone simulator) and run it again.
- Open the Apple Watch application on your iPhone, find the entry for HelloWatch, and switch off the `Show App on Apple Watch` setting. You can reinstall the application just by toggling the switch back to the on position.
- Delete the HelloWatch application from your iPhone. When you do this, the watch application is uninstalled from the watch.

For normal development, you don't need to manually delete the application from either the watch or the iPhone—if you make changes to the application and run it again from Xcode, it gets reinstalled automatically.

Congratulations! You just successfully built and ran your first Apple Watch application. Now let's take a quick look at what you can and can't do with the WatchKit framework.

Some Things That a WatchKit Application Can and Can't Do

WatchKit lets you build simple applications for the Apple Watch. You can create very useful extensions for your iPhone applications, like the weather application that you'll build in Chapter 7 of this book. However, you can't do everything that the Apple's own native Apple Watch applications can do. Here's a list of some of the things that you can and can't do with WatchKit:

- Your application can include a glance, which is a single screen that you can use to present useful information relating to your application. The user can configure which glances are available when by swiping up from the bottom of the Apple Watch screen. Chapter 8 adds a glance to the weather application from Chapter 7.
- You can handle local and remote notifications from your iPhone application on the watch, if the user allows it. You can choose to let the system display the notification using a default presentation, or you can build one of your own. We'll talk more about that in Chapter 9.