LEARNING WatchKit PROGRAMMING

A Hands-On Guide to Creating Apple Watch Applications

WEI-MENG LEE

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Learning WatchKit Programming

Addison-Wesley Learning Series

Visit **informit.com/learningseries** for a complete list of available publications.

The **Addison-Wesley Learning Series** is a collection of hands-on programming guides that help you quickly learn a new technology or language so you can apply what you've learned right away.

Each title comes with sample code for the application or applications built in the text. This code is fully annotated and can be reused in your own projects with no strings attached. Many chapters end with a series of exercises to encourage you to reexamine what you have just learned, and to tweak or adjust the code as a way of learning.

Titles in this series take a simple approach: they get you going right away and leave you with the ability to walk off and build your own application and apply the language or technology to whatever you are working on.

Learning WatchKit Programming

A Hands-On Guide to Creating Apple Watch Applications

Wei-Meng Lee

Addison-Wesley

New York • Boston • Indianapolis • San Francisco
Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2015940909

ISBN-13: 978-0-13-419544-5
ISBN-10: 0-13-419544-2

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing, June 2015

Editor-in-Chief
Mark L. Taub

Senior Acquisitions Editor
Trina MacDonald

Development Editor
Sheri Cain

Managing Editor
John Fuller

Full-Service Production Manager
Julie B. Nahil

Copy Editor
Stephanie Geels

Indexer
Jack Lewis

Proofreader
Anna Popick

Technical Reviewers
Mark H. Granoff
Chaim Krause
Niklas Saers

Editorial Assistant
Olivia Basegio

Cover Designer
Chuti Prasertsith

Compositor
CIP Group

❖

I dedicate this book with love to my family, and to my dearest wife, who has had to endure my irregular work schedule and take care of things while I was trying to meet writing deadlines!

❖

This page intentionally left blank

Contents at a Glance

Preface **xiii**
Acknowledgments **xvii**
About the Author **xix**

1 Getting Started with WatchKit Programming **1**

2 Apple Watch Interface Navigation **17**

3 Apple Watch User Interface **45**

4 Interfacing with iOS Apps **99**

5 Displaying Notifications **149**

6 Displaying Glances **179**

Index **195**

This page intentionally left blank

Contents

Preface **xiii**
Acknowledgments **xvii**
About the Author **xix**

1 Getting Started with WatchKit Programming 1
Specifications of the Apple Watch **1**
Getting the Tools for Development **2**
Understanding the WatchKit App Architecture **3**
Deploying Apple Watch Apps **4**
Interaction between the Apple Watch and iPhone **4**
Communicating with the Containing iOS App **5**
Types of Apple Watch Applications **6**
Hello, World! **6**
Creating an iPhone Project **6**
Adding a WatchKit App Target **8**
Examining the Storyboard **11**
WatchKit App Lifecycle **12**
Modifying the Interface Controller **13**
Running the Application on the Simulator **14**
Summary **16**

2 Apple Watch Interface Navigation 17
Interface Controllers and Storyboard **17**
Lifecycle of an Interface Controller **19**
Navigating between Interface Controllers **22**
Hierarchical Navigation **23**
Page-Based Navigation **27**
Passing Data between Interface Controllers **28**
Customizing the Title of the Chevron or Cancel Button **34**
Navigating Using Code **35**
Presenting a Series of Pages **38**
Changing the Current Page to Display **40**
Summary **43**

3 Apple Watch User Interface 45
Responding to User Interactions **45**
Button **46**
Switch **59**
Slider **62**
Displaying Information **65**
Labels **65**
Images **65**
Table **71**
Gathering Information **82**
Getting Text Inputs **82**
Getting Emojis **85**
Laying Out the Controls **86**
Force Touch **91**
Displaying a Context Menu **91**
Adding Menu Items Programmatically **97**
Summary **98**

4 Interfacing with iOS Apps 99
Localization **99**
Localizing the User Interface **102**
Creating Localizable Strings **106**
Using the Date Control **112**
Communicating between the WatchKit App and the Extension **113**
Location Data **114**
Displaying Maps **123**
Accessing Web Services **126**
Sharing Data **130**
Summary **148**

5 Displaying Notifications 149
What Is a Notification? **149**
Types of Notifications on the Apple Watch **152**
Implementing the Short-Look Interface **153**
Implementing the Long-Look Interface **167**
Summary **178**

6 Displaying Glances 179
What Is a Glance? **179**
Implementing Glances **180**
Customizing the Glance **182**
Testing the Glance **186**
Making the App Useful **186**
Creating a Shared App Group **187**
Implementing Background Fetch **188**
Updating the Glance **192**
Summary **194**

Index 195

This page intentionally left blank

Preface

Welcome to *Learning WatchKit Programming*!

This is an exciting time to be a programmer, as we are witnessing a new era of wearables. While the Apple Watch is not the first wearable device in the market, its launch signified the intention of Apple to enter the wearable market in a big way. After successfully changing various industries—music, computer, phone, and mobile computing—Apple looks set to change the wearable industry. And nobody is taking this lightly.

As with the iPhone, much of the usefulness and functionality of the Apple Watch device actually come from the creativity of the third-party developers. In the early days of the iPhone, Apple restricted all third-party apps to web applications, as they wanted to retain the monopoly on developing natively for the device. However, due to the overwhelming protests of developers, Apple finally relented by releasing an SDK to support third-party apps. It was this decision that changed the fate of the iPhone; the iPhone would never have been so successful without the ability to support third-party apps.

When the Apple Watch was announced, Apple was quick to learn its lesson and realized that the success of the Apple Watch largely depends on the availability of apps that support it. Hence, before the release of the Apple Watch, the SDK was made available to developers to have a hand in developing Apple Watch apps.

The book you are holding in your hands right now (or reading on your phone or tablet) is a collection of tutorials that help you navigate the jungle of Apple Watch programming. This book contains all of the fundamental topics that you need to get started in Apple Watch programming. As this is a book on Apple Watch programming, I am going to make a couple of assumptions about you, the reader:

- You should already be familiar with the basics of developing an iOS application. In particular, concepts like outlets and actions should not be new to you.
- You should be comfortable with the Swift programming language. See the next section on how to get started with Swift if you are new to it.

What You'll Need

To get the most out of this book:

- You need a Mac, together with **Xcode**.
- Your Mac should be running at least **Mac OS X Yosemite (v10.10)**, or later.

- You can download the latest version of Xcode from the Mac App Store. All the code samples for this book are tested against Xcode 6.3.
- If you plan to test your apps on a real device, you need to register to become a paid iOS developer (https://developer.apple.com/programs/ios/). The program costs $99/year for individuals. Once registered, you can request a certificate to sign your apps so that they can be deployed onto your devices. To install your apps onto your devices, you also need to create provisioning profiles for your devices. Obviously, you also need an Apple Watch, which should be paired to your iPhone. The Apple Watch can only work with iPhone 5, iPhone 5c, iPhone 5s, iPhone 6, and iPhone 6 Plus.
- All code samples in this book can be tested and run on the iPhone Simulator without the need for a real device or Apple Watch. However, for some code examples, you need access to the iOS Developer Program and a valid provisioning profile in your applications before they can work. Hence, even if you do not have an Apple Watch and you do not intend to test the apps on a real device, you still need to have access to a paid iOS developer account to test some of the examples in this book.
- A number of examples in this book require an Internet connection in order to work, so ensure that you have an Internet connection when trying out the examples.
- All of the examples in this book are written in Swift. If you are not familiar with Swift, you can refer to my book *Beginning Swift Programming* (Wrox, 2014) for a jumpstart, or download my Swift Cheat Sheets at http://weimenglee.blogspot.sg/2014/11/swift-cheat-sheets-download-today.html.

How This Book Is Organized

This book is styled as a tutorial. You will be trying out the examples as I explain the concepts. This is a proven way to learn a new technology, and I strongly encourage you to type in the code as you work on the examples.

- **Chapter 1, Getting Started with WatchKit Programming**: In this chapter, you learn about the architecture of Apple Watch applications and how they tie in with your iOS apps. Most importantly, you get your chance to write a simple Apple Watch app and deploy it onto the simulator.
- **Chapter 2, Apple Watch Interface Navigation**: In this chapter, you dive deeper into how your Apple Watch application navigates between multiple screens. You get to see how data is passed between screens and how to customize the look and feel of each screen.
- **Chapter 3, Apple Watch User Interface**: Designing the user interface (UI) for your Apple Watch application is similar to designing for iPhone apps.

However, space is at a premium on the Apple Watch, and every millimeter on the screen must be put to good use in order to convey the exact intention of your app. In this chapter, you learn how to use the various UI controls in the Apple Watch to build your application.

- **Chapter 4, Interfacing with iOS Apps**: This chapter shows all the exciting features that you can add to your Apple Watch applications. You learn how to localize your apps, how to communicate between the watch app and the containing iOS app, how to call web services, and more!
- **Chapter 5, Displaying Notifications**: In this chapter, you learn how to display notifications on your Apple Watch. Notifications received by the iPhone are sent to the Apple Watch, and you have the chance to customize the notifications so that you can display the essence of the notifications quickly to the user.
- **Chapter 6, Displaying Glances**: Glances on the Apple Watch provide the user a quick way to gather information from apps. For example, Instagram's glance on the Apple Watch may show the most recently shared photo, while Twitter may show the latest trending tweets. In this chapter, you learn how to implement glances for your own apps.

About the Sample Code

The code samples in this book are written to provide the simplest way to understand core concepts without being bogged down with details like beautifying the UI or detailed error checking. The philosophy is to convey key ideas in the simplest manner possible. In real-life apps, you are expected to perform detailed error handling and to create a user-friendly UI for your apps. Although I do provide several scenarios in which a certain concept is useful, it is ultimately up to you, the reader, to exercise your creativity to put the concepts to work, and perhaps create the next killer app.

Getting the Sample Code

To download the sample code used in this book, visit the book's web page on Informit.com at informit.com/title/9780134195445 and click the **Extras** tab.

Contacting the Author

If you have any comments or questions about this book, please drop me an email at weimenglee@learn2develop.net, or stop by my web site at learn2develop.net.

This page intentionally left blank

Acknowledgments

Writing a book on emerging technology is always an exciting and perilous journey. On one end, you are dealing with the latest developments, going where not many have ventured, and on the other end you are dealing with many unknowns. To endure this journey you need a lot of help and family support. And I would like to take this opportunity to thank the people who make all this happen.

I am indebted to Trina MacDonald, senior acquisitions editor at Addison-Wesley/Pearson Education, for giving me the chance to work on this book. She has always been supportive of my proposals for new titles, and I am really glad that we have the chance to work together on this project. Thank you very much for the opportunity and guidance, Trina! I hope I did not disappoint you.

I would like to thank the many heroes working behind the scene: copy editor Stephanie Geels, production editor Julie Nahil, and technical reviewers Mark H. Granoff, Chaim Krause, and Niklas Saers for turning the manuscript into a book that I am proud of!

Last but not least, I want to thank my family for all the support that they have always given me. Without their encouragement, this book would never have been possible.

This page intentionally left blank

About the Author

Wei-Meng Lee is a technologist and founder of Developer Learning Solutions (learn2develop.net), a technology company specializing in hands-on training on the latest web and mobile technologies. Wei-Meng speaks regularly at international conferences and has authored and coauthored numerous books on .NET, XML, Android, and iOS technologies. He writes extensively for informIT.com and mobiForge.com.

This page intentionally left blank

2

Apple Watch Interface Navigation

It's really hard to design products by focus groups. A lot of times, people don't know what they want until you show it to them.
Steve Jobs

In Chapter 1, "Getting Started with WatchKit Programming," you learned about the various specifications and features of the Apple Watch. You also had the chance to use Xcode to create a simple iPhone project that supports the Apple Watch. You then used the Apple Watch Simulator to test the application. In this chapter, you dive into how your Apple Watch application navigates between multiple screens.

Interface Controllers and Storyboard

As you learned in Chapter 1, the user interface of your Apple Watch application is encapsulated in a storyboard file. Within the storyboard file, you have an Interface Controller that represents a screen on the Apple Watch. In this section, let's create a project so that we can examine the storyboard in more detail:

1. Using Xcode, create a Single View Application project and name it **LifeCycle**.
2. Add the WatchKit App target to the project. Uncheck the option Include Notification Scene so that we can keep the WatchKit project to a bare minimum.

> **Note**
>
> If you are not sure how to add the WatchKit App target to the existing project, refer to Chapter 1.

3. Once the target is added to the project, select the Interface.storyboard file located within the LifeCycle WatchKit App group (see Figure 2.1). This opens the file using the Storyboard Editor.

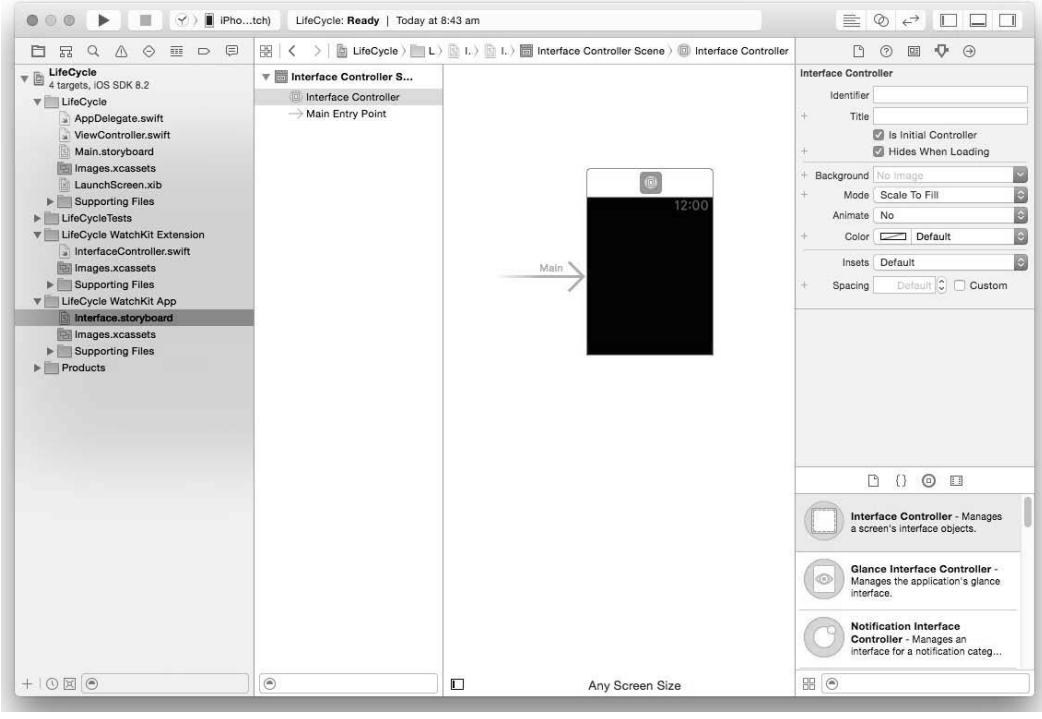

Figure 2.1 Editing the storyboard file

4. Select the Interface Controller and view its Identity Inspector window (see Figure 2.2). The Class is set to `InterfaceController`, which means that it is represented by a Swift class named `InterfaceController`.

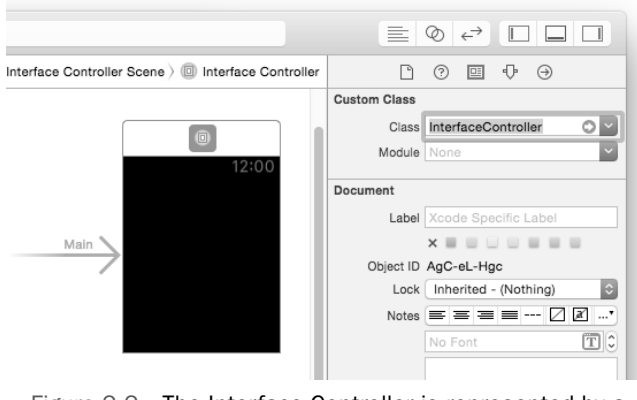

Figure 2.2 The Interface Controller is represented by a Swift class named InterfaceController

5. View its Attributes Inspector window and observe that the Is Initial Controller attribute is checked (see Figure 2.3). This attribute indicates that, when the application is loaded, this is the default Interface Controller that will be displayed.

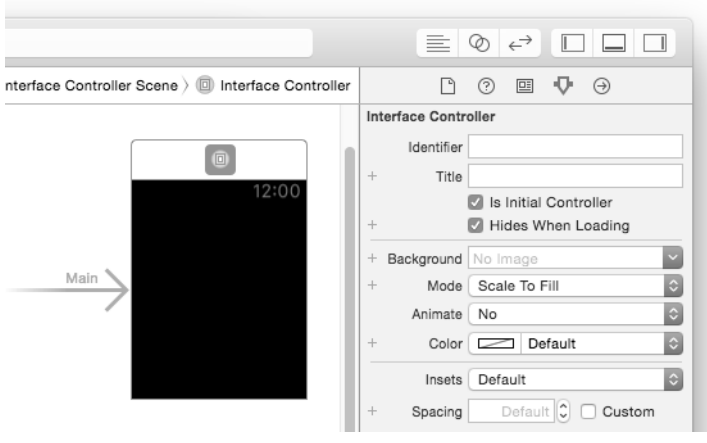

Figure 2.3 The Is Initial Controller attribute indicates that the current Interface Controller will be displayed when the application loads

Lifecycle of an Interface Controller

As you have seen in the previous section and in Chapter 1, an Interface Controller is connected to a Swift class located in the WatchKit Extension group of the project. In this example, this Swift class is named InterfaceController.swift. It has the following content:

```
import WatchKit
import Foundation

class InterfaceController: WKInterfaceController {

    override func awakeWithContext(context: AnyObject?) {
        super.awakeWithContext(context)

        // Configure interface objects here.
    }

    override func willActivate() {
        // This method is called when watch view controller is about to
        // be visible to user
        super.willActivate()
    }
```

```
    override func didDeactivate() {
        // This method is called when watch view controller is no longer visible
        super.didDeactivate()
    }
}
```

Specifically, it has three key methods:

- **awakeWithContext:**—The system calls this method at initialization time, passing it any contextual data from a previous Interface Controller. You should use this method to initialize and to prepare your UI for display, as well as to obtain any data that is passed to it from another Interface Controller (you will learn how this is done in the later section on passing data).
- **willActivate**—This method is called by the system when the Interface Controller is about to be displayed. You should use this method to make some last-minute changes to your UI and to refrain from performing any tasks that initialize the UI—these should be done in the `awakeWithContext` method.
- **didDeactivate**—This method is called when the Interface Controller is no longer onscreen. You should use this method to perform cleanup operations on your Interface Controller, such as invalidating timers or saving state-related information.

Besides the three methods just discussed, you can also add an initializer to the Interface Controller class:

```
override init() {
    super.init()
}
```

You can also perform initialization for your Interface Controller in this initializer, but you should leave the bulk of the UI initialization to the `awakeWithContext` method.

Let's try an example to better understand the use of the various methods:

1. Add the following statements in bold to the InterfaceController.swift file:

```
import WatchKit
import Foundation

class InterfaceController: WKInterfaceController {

    override init() {
        super.init()
        println("In the init initializer")
    }

    override func awakeWithContext(context: AnyObject?) {
        super.awakeWithContext(context)
```

```
        // Configure interface objects here.
        println("In the awakeWithContext event")
    }

    override func willActivate() {
        // This method is called when watch view controller is about to be
        // visible to user
        super.willActivate()
        println("In the willActivate event")
    }

    override func didDeactivate() {
        // This method is called when watch view controller is no longer
        // visible
        super.didDeactivate()
        println("In the didDeactivate event")
    }

}
```

2. Run the application on the iPhone 6 Simulator. When the application is loaded onto the Apple Watch Simulator, you should see the statements printed out in the Output Window in Xcode, as shown in Figure 2.4. Observe that the `init`, `awakeWithContext:`, and `willActivate` methods are fired when the Interface Controller is loaded.

> **Note**
>
> If you are not able to see the Output Window, press Command-Shift-C in Xcode.

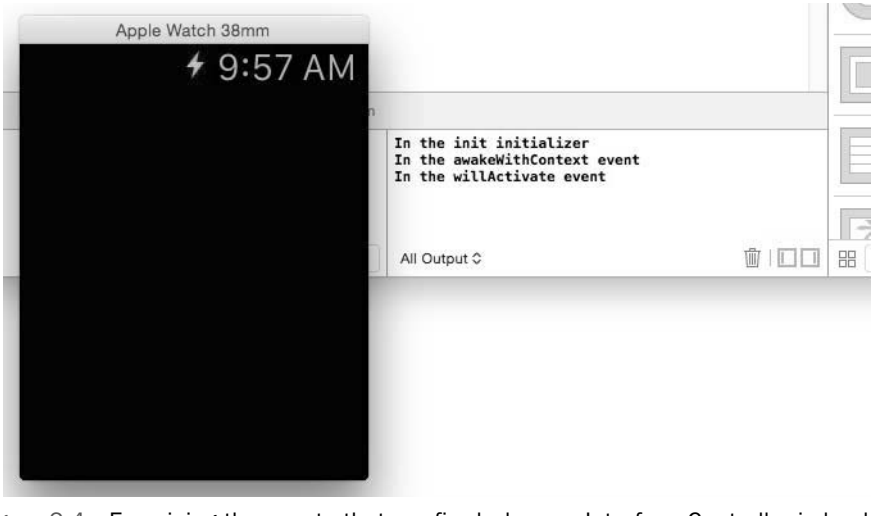

Figure 2.4 Examining the events that are fired when an Interface Controller is loaded

3. With the Apple Watch Simulator selected, select **Hardware** | **Lock** to lock the Apple Watch. Observe the output in the Output window (see Figure 2.5). Observe that the `didDeactivate` method is now executed.

> **Note**
>
> The `didDeactivate` method will also be fired when an Interface Controller transits to another Interface Controller.

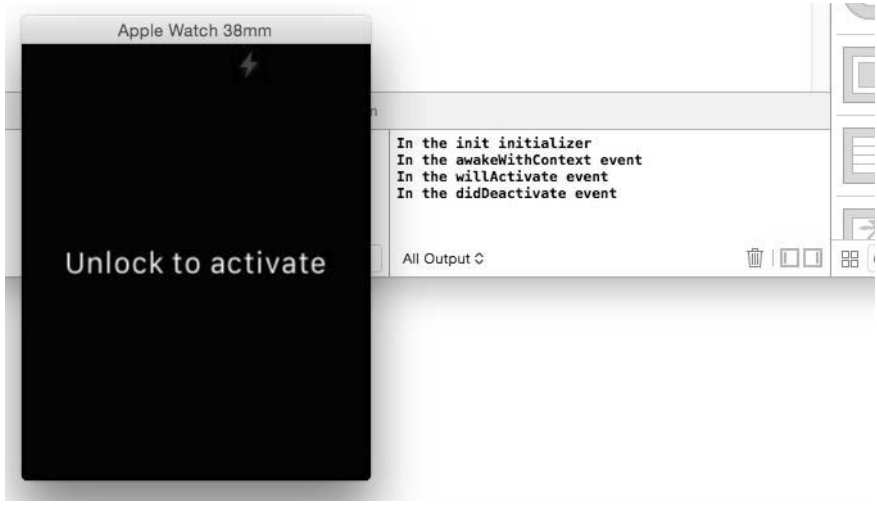

Figure 2.5 Examining the event that is fired when an Interface Controller is deactivated

> **Note**
>
> To unlock the Apple Watch Simulator, unlock the iPhone Simulator by selecting **Hardware | Home**, and then by swiping from left to right.

Navigating between Interface Controllers

The basic unit of display for an Apple Watch app is represented by an Interface Controller (of type `WKInterfaceController`). Depending on the type of application you are working on, there are times where you need to spread your UI across multiple Interface Controllers. In Apple Watch, there are two ways to navigate between Interface Controllers:

- **Hierarchical**: Pushes another Interface Controller on the screen. This model is usually used when you want the user to follow a series of related steps in order to perform a particular action.
- **Page-based**: Displays another Interface Controller on top of the current Interface Controller. This model is usually used if the information displayed on each

Interface Controller is not closely related to other Interface Controller. You can also use this model to display a series of Interface Controllers, which the user can select by swiping the screen.

> **Similarities to iPhone Development**
>
> The page-based navigation method is similar to presenting a modal View Controller in iPhone, whereas the hierarchical navigation method is similar to using a navigation controller in iPhone.

Hierarchical Navigation

A hierarchical interface always starts with a root Interface Controller. It then pushes additional Interface Controllers when a button or a control in a screen is tapped.

1. Using Xcode, create a Single View Application project and name it `UINavigation`.
2. Add a WatchKit App target to the project. Uncheck the option Include Notification Scene so that we can keep the WatchKit project to a bare minimum.
3. In the UINavigation WatchKit App group, select the Interface.storyboard file to edit it in the Storyboard Editor.
4. Drag and drop another Interface Controller object onto the editor, as shown in Figure 2.6. You should now have two Interface Controllers.

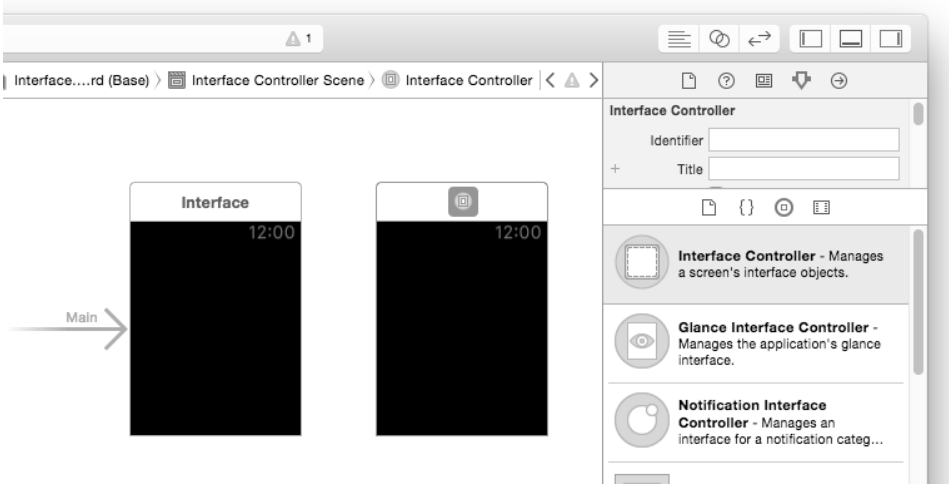

Figure 2.6 Adding another Interface Controller to the storyboard

5. In the original Interface Controller, add a Button control (see Figure 2.7) and change its title (by double-clicking it) to **Next Screen**.

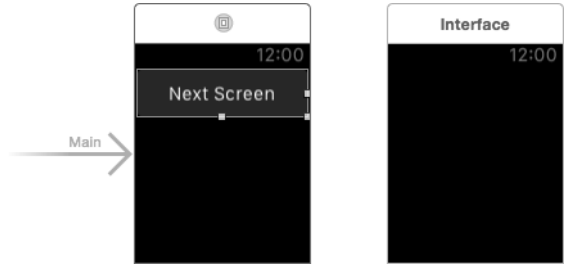

Figure 2.7 Adding a Button control to the first Interface Controller

6. Control-click the **Next Screen** button and drag and drop it over the second Interface Controller (see Figure 2.8).

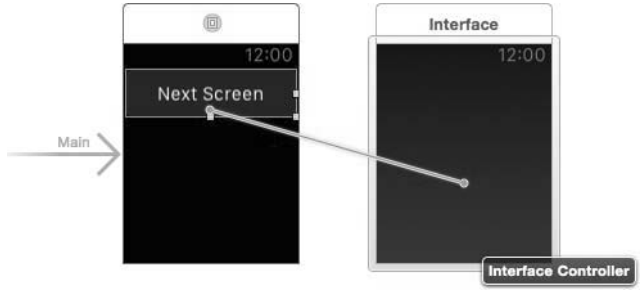

Figure 2.8 Control-click the Button control and drag and drop it over the second Interface Controller

7. You will see a popup called Action Segue. Select **push** (see Figure 2.9).

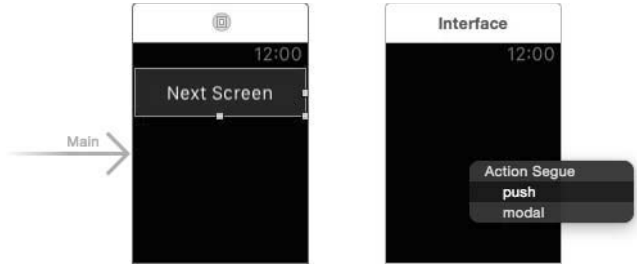

Figure 2.9 Creating a push segue

8. A segue will now be created (see Figure 2.10), linking the first Interface Controller to the second.

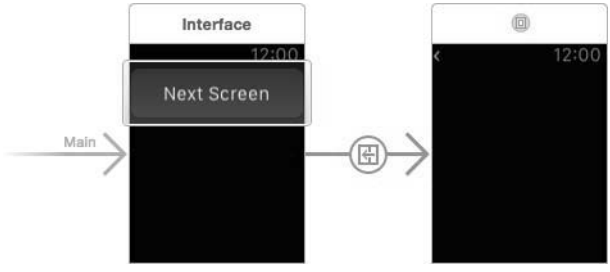

Figure 2.10 The segue that is created after performing the action

9. Select the segue and set its Identifier to **hierarchical** in the Attributes Inspector window (see Figure 2.11). This identifier allows us to identify it programmatically in our code later.

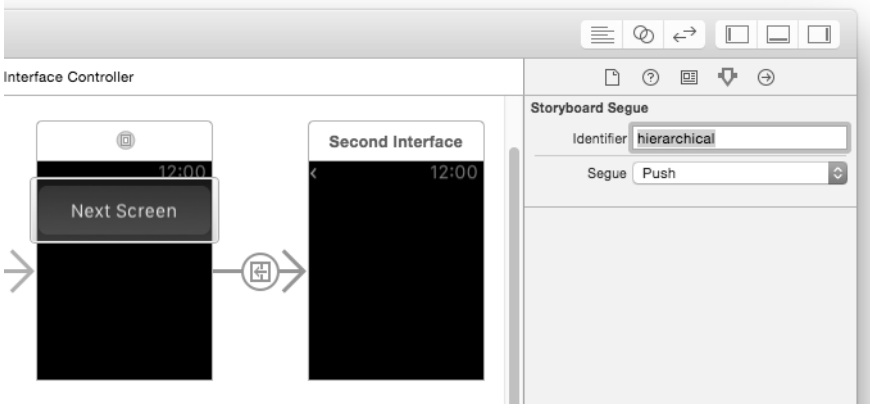

Figure 2.11 Naming the Identifier for the segue

10. Add a Label control to the second Interface Controller, as shown in Figure 2.12. Set the Lines attribute of the Label control to **0** in the Attributes Inspector window so that the Label can wrap around long text (used later in this chapter).

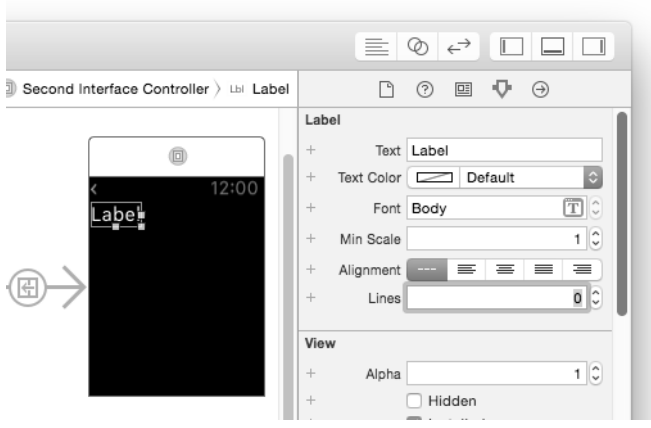

Figure 2.12 Adding a Label control to the second Interface Controller

11. You are now ready to test the application. Run the application on the iPhone 6 Simulator and, in the Apple Watch Simulator, click the **Next Screen** button and observe that the application navigates to the second Interface Controller containing the Label control (see Figure 2.13). Also, observe that the second Interface Controller has a < icon (known as a *chevron*) displayed in the top-left corner. Clicking it returns the application to the first Interface Controller.

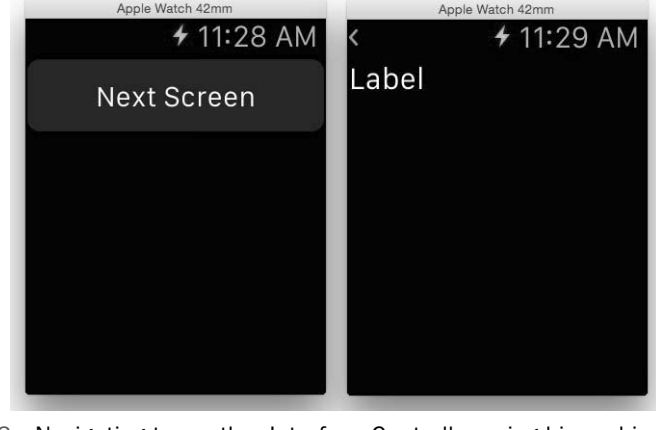

Figure 2.13 Navigating to another Interface Controller using hierarchical navigation

Note

At this point, the Label control on the second Interface Controller is still displaying the default text "Label." In later sections in this chapter, you learn how to pass data from the first Interface Controller to the second and then how to display the data in the Label control.

Page-Based Navigation

You can also display an Interface Controller modally. This is useful if you want to obtain some information from the user or get the user to confirm an action.

1. Using the same project created in the previous section, add another Button control to the first Interface Controller, as shown in Figure 2.14. Change the title of the Button to **Display Screen**.

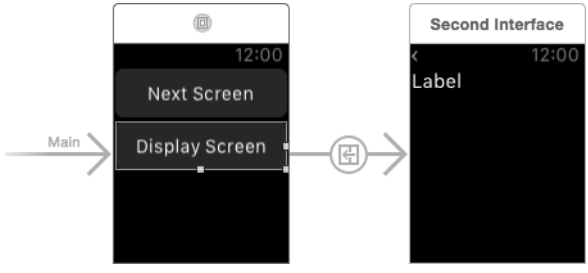

Figure 2.14 Adding another Button control to the first Interface Controller

2. Create a segue connecting the Display Screen button to the second Interface Controller. In the Action Segue popup that appears, select **modal**. Set the Identifier of the newly created segue to **pagebased** (see Figure 2.15).

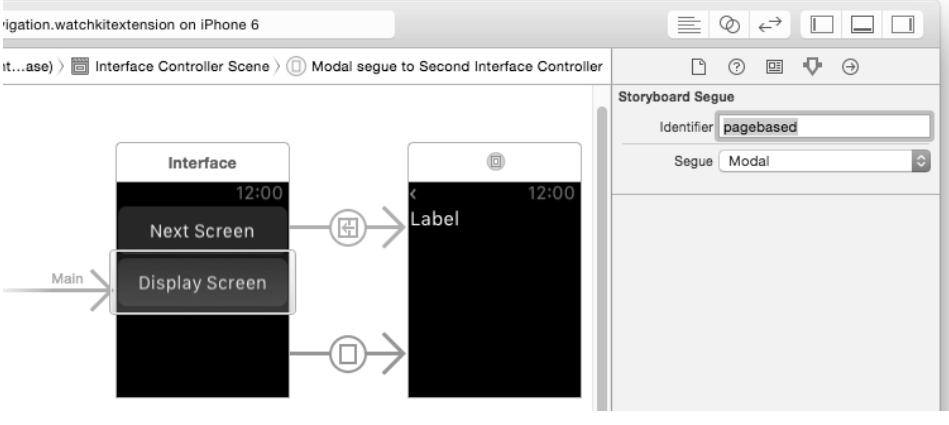

Figure 2.15 Creating a modal segue connecting the two Interface Controllers

3. Run the application on the iPhone 6 Simulator and, in the Apple Watch Simulator, click the **Display Screen** button and observe that the second Interface Controller appears from the bottom of the screen. Also, observe that the second Interface Controller now has a Cancel button displayed in the top-left corner (see Figure 2.16). Clicking it hides the second Interface Controller.

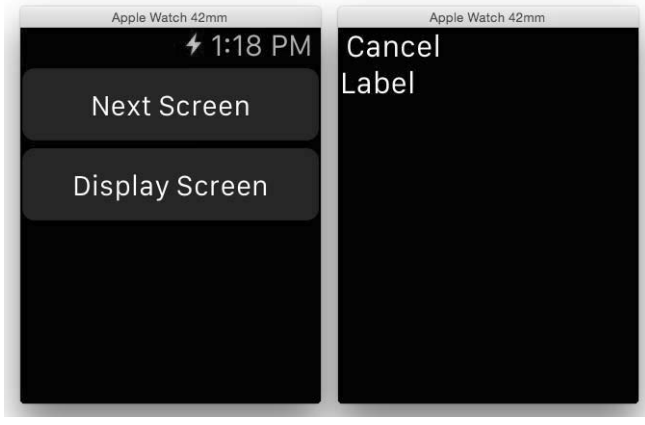

Figure 2.16 Displaying another Interface Controller modally

Passing Data between Interface Controllers

In the previous sections, you saw how to make your Apple Watch application transit from one Interface Controller to another, using either the hierarchical or page-based navigation method. One commonly performed task is to pass data from one Interface Controller to another. In this section, you do just that.

1. Using the UINavigation project that you used in the previous section, right-click the **UINavigation WatchKit Extension** group and select **New File…** (see Figure 2.17).

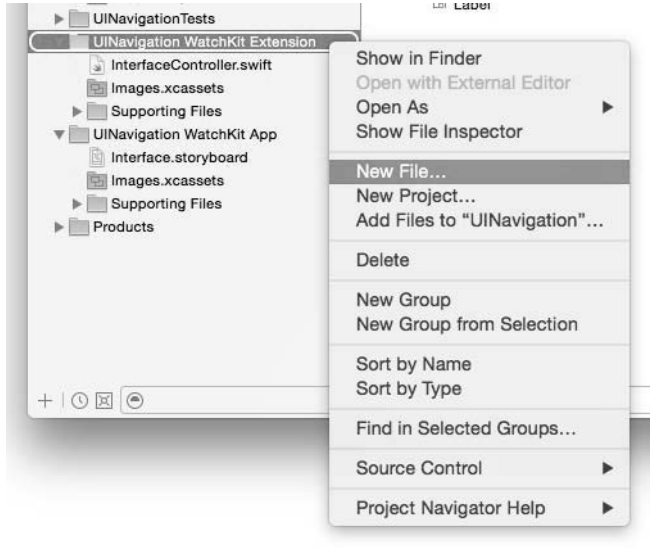

Figure 2.17 Adding a new file to the project

2. Select the **Cocoa Touch Class** (see Figure 2.18) template and click **Next**.

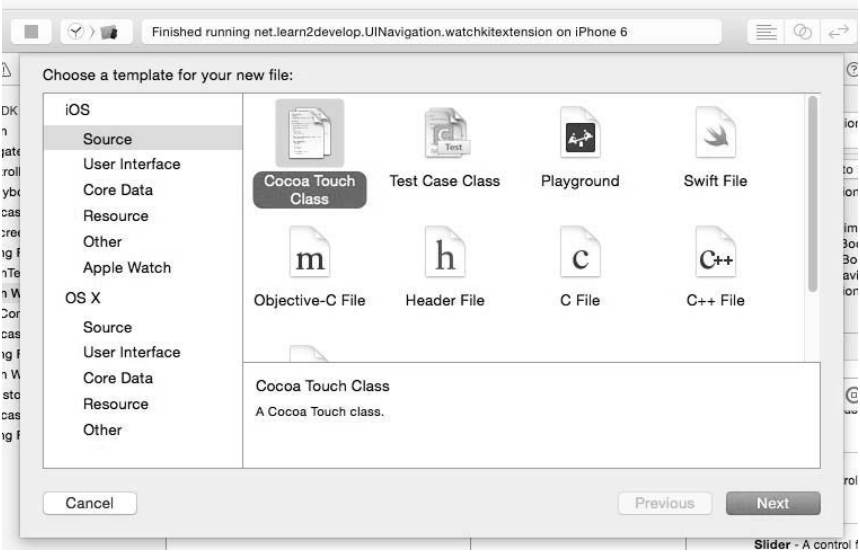

Figure 2.18 Selecting the Cocoa Touch Class template

3. Name the Class `SecondInterfaceController` and make it a subclass of `WKInterfaceController` (see Figure 2.19). Click **Next**.

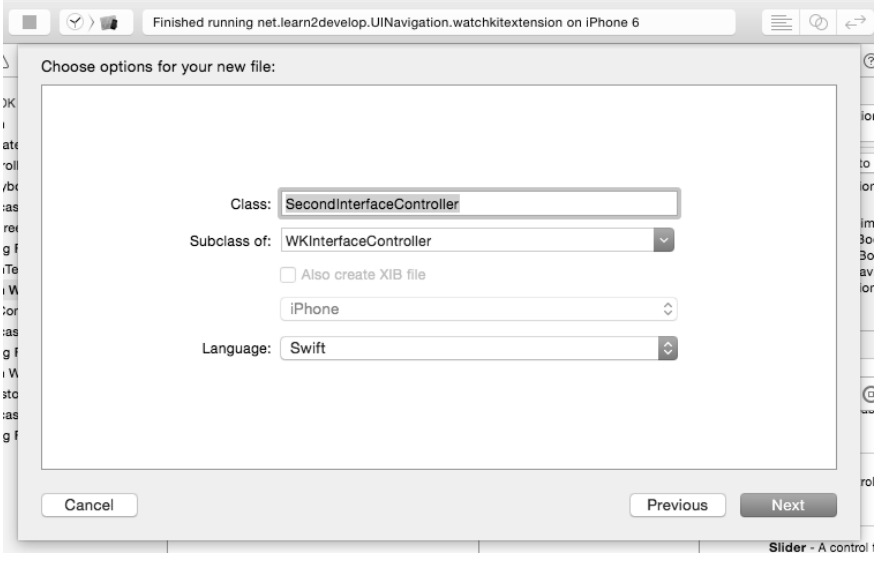

Figure 2.19 Naming the newly added class

4. A file named SecondInterfaceController.swift will now be added to the UINavigation WatchKit Extension group of your project.
5. Back in the Storyboard Editor, select the second Interface Controller and set its Class (in the Identity Inspector window) to `SecondInterfaceController` (see Figure 2.20).

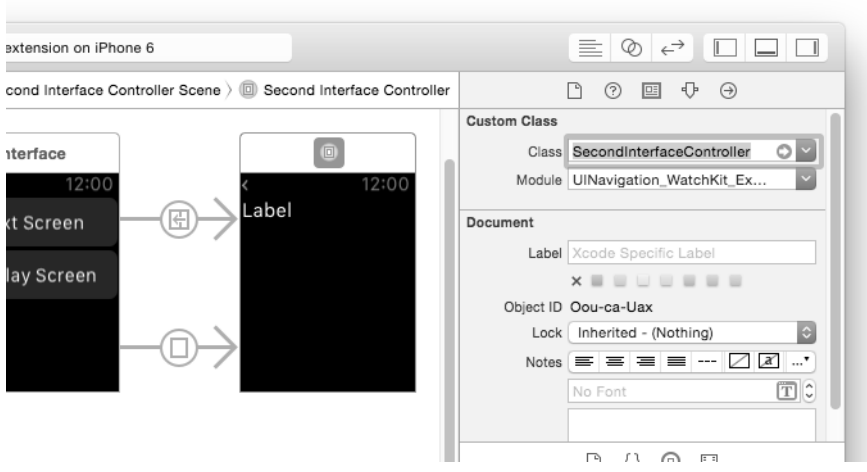

Figure 2.20 Setting the class of the second Interface Controller

6. Select the **View** | **Assistant Editor** | **Show Assistant Editor** menu item to show the Assistant Editor. Control-click the **Label** control and drag and drop it onto the Code Editor (as shown in Figure 2.21).

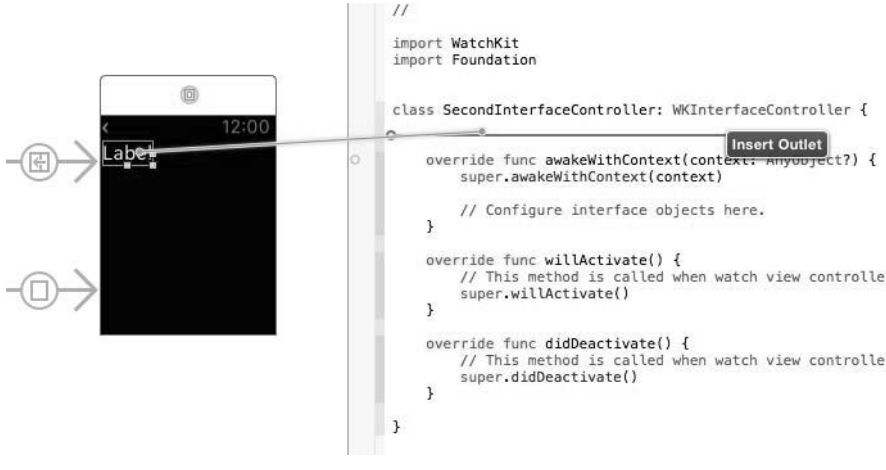

Figure 2.21 Creating an outlet for the Label control

7. Create an outlet and name it `label` (see Figure 2.22).

Figure 2.22 Naming the outlet for the Label control

8. An outlet is now added to the code:

```
import WatchKit
import Foundation

class SecondInterfaceController: WKInterfaceController {

    @IBOutlet weak var label: WKInterfaceLabel!

    override func awakeWithContext(context: AnyObject?) {
        super.awakeWithContext(context)

        // Configure interface objects here.
    }

    override func willActivate() {
        // This method is called when watch view controller is about to be
        // visible to user
        super.willActivate()
    }

    override func didDeactivate() {
        // This method is called when watch view controller is no longer
        // visible
        super.didDeactivate()
    }
}
```

9. Add the following statements in bold to the InterfaceController.swift file:

```
import WatchKit
import Foundation

class InterfaceController: WKInterfaceController {

    override func awakeWithContext(context: AnyObject?) {
        super.awakeWithContext(context)

        // Configure interface objects here.
    }

    override func willActivate() {
        // This method is called when watch view controller is about to be
        // visible to user
        super.willActivate()
    }

    override func didDeactivate() {
        // This method is called when watch view controller is no longer
        // visible
        super.didDeactivate()
    }

    override func contextForSegueWithIdentifier(segueIdentifier: String) ->
        AnyObject? {
            if segueIdentifier == "hierarchical" {
                return ["segue": "hierarchical",
                    "data":"Passed through hierarchical navigation"]
            } else if segueIdentifier == "pagebased" {
                return ["segue": "pagebased",
                    "data": "Passed through page-based navigation"]
            } else {
                return ["segue": "", "data": ""]
            }
    }

}
```

The `contextForSegueWithIdentifier:` method is fired before any of the segues fire (when the user taps on one of the Button controls). Here, you check the identifier of the segue (through the `segueIdentifier` argument). Specifically, you return a dictionary containing two keys: `segue` and `data`.

10. Add the following statements in bold to the SecondInterfaceController.swift file:

```
import WatchKit
import Foundation
```

```
class SecondInterfaceController: WKInterfaceController {

    @IBOutlet weak var label: WKInterfaceLabel!

    override func awakeWithContext(context: AnyObject?) {
        super.awakeWithContext(context)

        // Configure interface objects here.
        var dict = context as? NSDictionary
        if dict != nil {
            var segue = dict!["segue"] as! String
            var data = dict!["data"] as! String
            self.label.setText(data)
        }
    }
```

When the second Interface Controller is loaded, you retrieve the data that is passed into it in the `awakeWithContext:` method through the `context` argument. Since the first Interface Controller passes in a dictionary, you can typecast it into an `NSDictionary` object and then retrieve the value of the `segue` and `data` keys. The value of the `data` key is then displayed in the Label control.

11. Run the application on the iPhone 6 Simulator and in the Apple Watch Simulator, click the **Next Screen** button, and observe the string displayed in the second Interface Controller (see Figure 2.23).

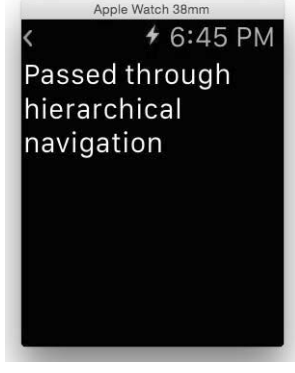

Figure 2.23 Displaying the data passed through the hierarchical navigation

12. Click the < chevron to return to the first Interface Controller and click the **Display Screen** button. Observe the string displayed in the second Interface Controller (see Figure 2.24).

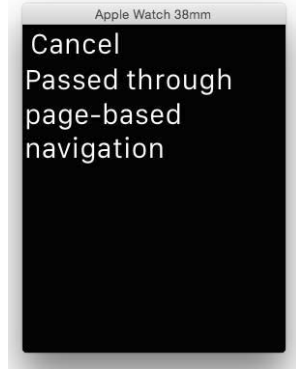

Figure 2.24 Displaying the data passed through the page-based navigation

Customizing the Title of the Chevron or Cancel Button

As you have seen in the previous section, a chevron is displayed when you push an Interface Controller using the hierarchical navigation method. A default Cancel button is displayed when you display an Interface Controller modally. However, the chevron or Cancel button can be customized.

1. Add the following statements in bold to the SecondInterfaceController.swift file:

```
import WatchKit
import Foundation

class SecondInterfaceController: WKInterfaceController {

    @IBOutlet weak var label: WKInterfaceLabel!

    override func awakeWithContext(context: AnyObject?) {
        super.awakeWithContext(context)

        // Configure interface objects here.
        var dict = context as? NSDictionary
        if dict != nil {
            var segue = dict!["segue"] as! String
            var data = dict!["data"] as! String
            self.label.setText(data)
            if segue == "pagebased" {
                self.setTitle("Close")
            } else {
                self.setTitle("Back")
            }
        }
    }
```

2. Run the application on the iPhone 6 Simulator and in the Apple Watch Simulator, click the **Next Screen** button, and observe the string displayed next to the chevron (see Figure 2.25).

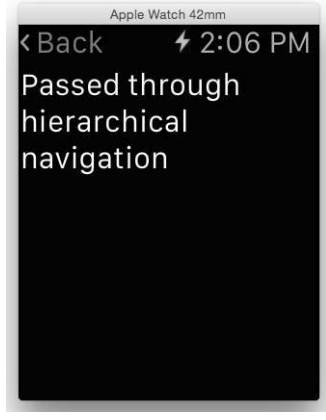

Figure 2.25 Displaying a string next to the chevron

3. Click the **<Back** chevron to return to the first Interface Controller and click the **Display Screen** button. Observe that the Cancel button is now displayed as Close (see Figure 2.26).

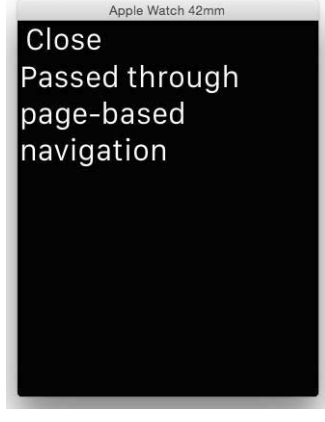

Figure 2.26 Modifying the button for a modal Interface Controller

Navigating Using Code

Although you can link up Interface Controllers by creating segues in your storyboard, it is not versatile. In a real-life application, the flow of your application may depend on

certain conditions being met, and hence, you need to be able to decide during runtime which Interface Controller to navigate to (or display modally).

1. Using Xcode, create a new Single View Application project and name it **`NavigateUsingCode`**.
2. Add a WatchKit App target to the project. Uncheck the option Include Notification Scene so that we can keep the WatchKit project to a bare minimum.
3. Click the **Interface.storyboard** file located in the NavigateUsingCode WatchKit App group in your project to edit it using the Storyboard Editor.
4. Add two Button controls to the first Interface Controller and then add another Interface Controller to the storyboard. In the second Interface Controller, add a Label control, as shown in Figure 2.27.

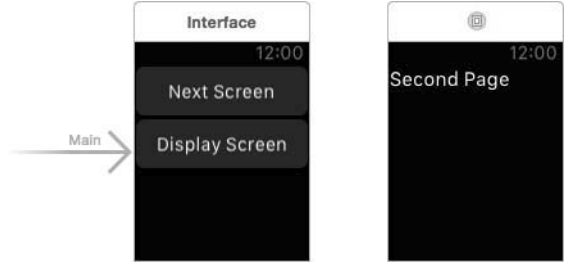

Figure 2.27 Populating the two Interface Controllers

5. Select the second Interface Controller and set its Identifier attribute (in the Attributes Inspector window) to **secondpage**, as shown in Figure 2.28.

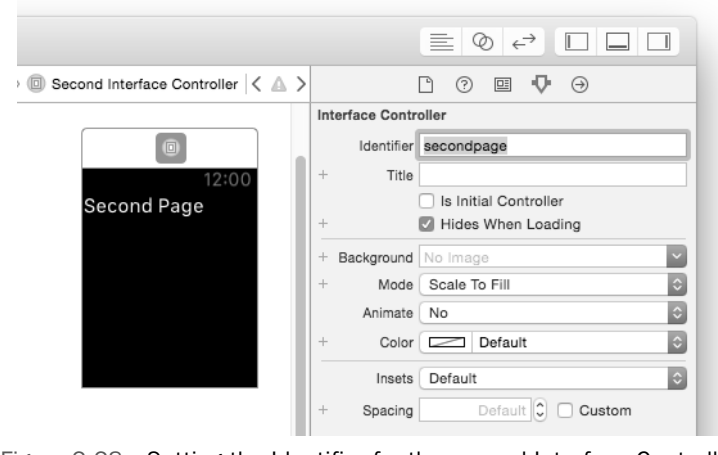

Figure 2.28 Setting the Identifier for the second Interface Controller

6. In the first Interface Controller, create two actions (one for each button) and name them as shown here in the InterfaceController.swift file. You should create the actions by control-dragging them from the storyboard onto the Code Editor:

```
import WatchKit
import Foundation

class InterfaceController: WKInterfaceController {

    @IBAction func btnNextScreen() {
    }

    @IBAction func btnDisplayScreen() {
    }
```

7. Add the following statements to the two actions in the InterfaceController.swift file:

```
import WatchKit
import Foundation

class InterfaceController: WKInterfaceController {

    @IBAction func btnNextScreen() {
        pushControllerWithName("secondpage", context: nil)
    }

    @IBAction func btnDisplayScreen() {
        presentControllerWithName("secondpage", context: nil)
    }
```

   Observe that the first button uses the `pushControllerWithName:context:` method to perform a hierarchical navigation. The first argument to this method takes in the identifier of the Interface Controller to navigate to (which we had earlier set in Step 5). The `context` argument allows you to pass data to the target Interface Controller, which in this case we simply set to `nil`. For the second button, we use the `presentControllerWithName:context:` method to perform a page-based navigation. Like the `pushControllerWithName:context:` method, the first argument is the identifier of the Interface Controller to display, whereas the second argument allows you to pass data to the target Interface Controller.

8. Run the application on the iPhone 6 Simulator. Clicking either button brings you to the second Interface Controller (see Figure 2.29).

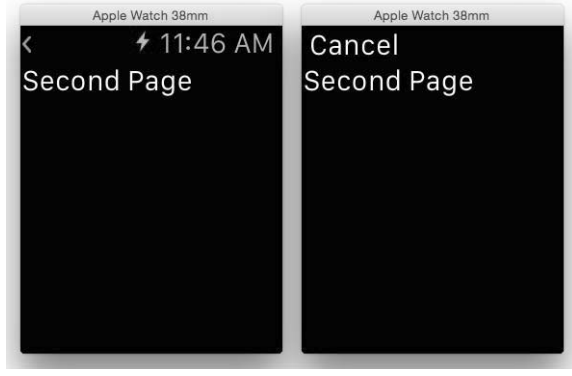

Figure 2.29 Navigating the Interface Controllers programmatically

> **Returning to the Previous Screen**
>
> Although you can return to the previous screen by tapping either the chevron or the **Cancel** button, you can also programmatically return to the previous screen. If you navigate to an Interface Controller using the `pushControllerWithName:context:` method, you can programmatically return to the Interface Controller using the corresponding `popController` method. If you display an Interface Controller using the `presentControllerWithName:context:` method, you can dismiss the current Interface Controller using the corresponding `dismissController` method.

Presenting a Series of Pages

For page-based applications, you can display more than one single Interface Controller modally—you can display a series of them.

1. Using the same project created in the previous section, add a third Interface Controller to the storyboard and add a Label control to it. Set the Label text to **Third Page** (see Figure 2.30).
2. Set the Identifier attribute of the third Interface Controller to **thirdpage** in the Attributes Inspector window (see Figure 2.31).
3. Add the following statements in bold to the InterfaceController.swift file:

```
@IBAction func btnDisplayScreen() {
    //presentControllerWithName("secondpage", context: nil)
    presentControllerWithNames(["secondpage", "thirdpage"], contexts: nil)
}
```

   Instead of using the `presentControllerWithName:context:` method, we now use the `presentControllerWithNames:context:` method. The only difference between the two methods is that the latter takes in an array of string in the first argument. This array of string contains the identifiers of Interface Controllers that you want to display.

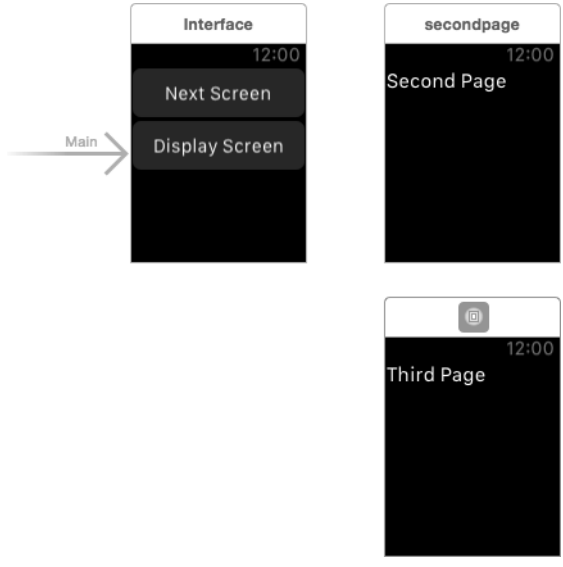

Figure 2.30 Adding the third Interface Controller

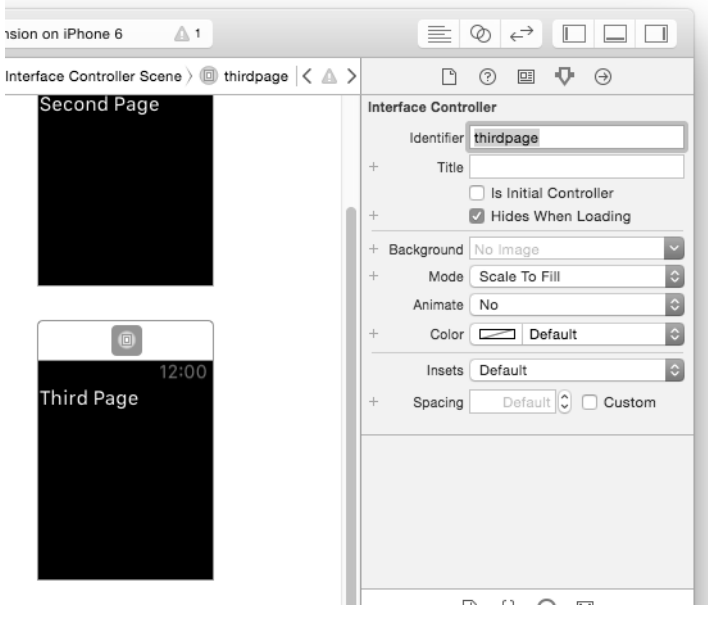

Figure 2.31 Setting the Identifier for the third Interface Controller

4. Run the application on the iPhone 6 Simulator and click the **Display Screen** button on the Apple Watch simulator. This time, you see that the second

Interface Controller is displayed with two dots at the bottom of the screen. Swiping from right to left reveals the third Interface Controller (see Figure 2.32).

Figure 2.32 The user can slide between the two Interface Controllers

Changing the Current Page to Display

In the previous section, you saw that you could display a series of Interface Controllers that the user can swipe through. What if you want to programmatically jump to a particular page? In this case, what if you want to display the Third Page instead of the Second Page? Let's see how this can be done.

1. Add two `WKInterfaceController` classes to the NavigateUsingCode WatchKit Extension group of the project and name them **SecondInterfaceController.swift** and **ThirdInterfaceController.swift**, respectively. Figure 2.33 shows the location of the files.

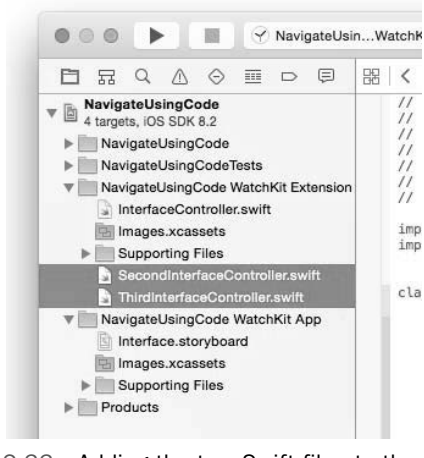

Figure 2.33 Adding the two Swift files to the project

2. Populate the SecondInterfaceController.swift file as follows:

```
import WatchKit
import Foundation

class SecondInterfaceController: WKInterfaceController {

    override func awakeWithContext(context: AnyObject?) {
        super.awakeWithContext(context)

        // Configure interface objects here.
        println("SecondInterfaceController - awakeWithContext")
    }

    override func willActivate() {
        // This method is called when watch view controller is about to be
        // visible to user
        super.willActivate()
        println("SecondInterfaceController - willActivate")
    }

    override func didDeactivate() {
        // This method is called when watch view controller is no longer
        // visible
        super.didDeactivate()
        println("SecondInterfaceController - didDeactivate")
    }

}
```

3. Populate the ThirdInterfaceController.swift file as follows:

```
import WatchKit
import Foundation

class ThirdInterfaceController: WKInterfaceController {

    override func awakeWithContext(context: AnyObject?) {
        super.awakeWithContext(context)

        // Configure interface objects here.
        println("ThirdInterfaceController - awakeWithContext")
    }

    override func willActivate() {
        // This method is called when watch view controller is about to be
        // visible to user
        super.willActivate()
```

```
        println("ThirdInterfaceController - willActivate")
    }

    override func didDeactivate() {
        // This method is called when watch view controller is no longer
        // visible
        super.didDeactivate()
        println("ThirdInterfaceController - didDeactivate")
    }

}
```

4. In the Interface.storyboard file, set the Class property of the second Interface Controller to **`SecondInterfaceController`** (see Figure 2.34). Likewise, set the Class property of the third Interface Controller to **`ThirdInterfaceController`**.

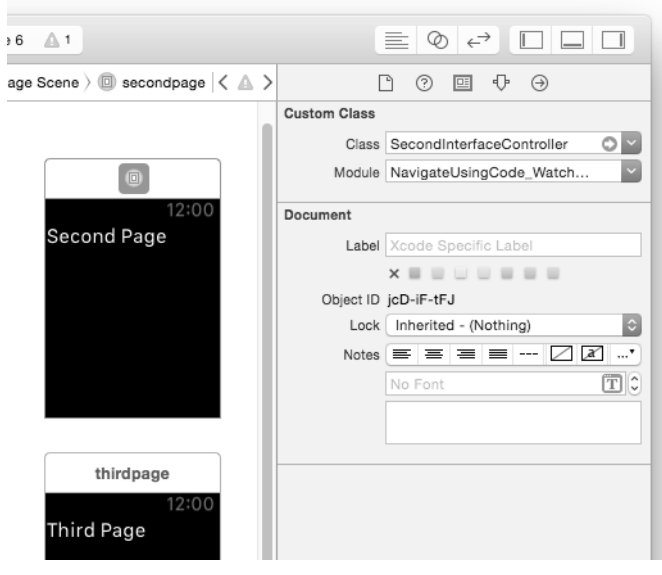

Figure 2.34 Setting the class for the second Interface Controller

5. Run the application on the iPhone 6 Simulator and click the **Display Screen** button on the Apple Watch simulator. Observe the statements printed in the Output window (see Figure 2.35). As you can see, the `awakeWithContext` method is fired for both the second and third Interface Controllers, even though only the second Interface Controller is visible initially.

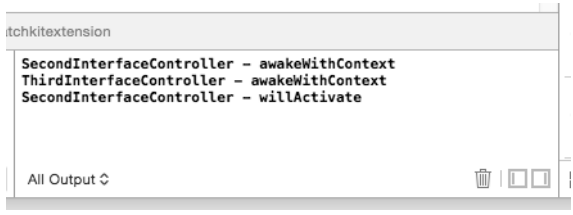

Figure 2.35 Both Interface Controllers fire the awakeWithContext method

6. If you want the third Interface Controller to load instead of the second, you can use the `becomeCurrentPage` method. Calling this method in an Interface Controller brings it into view. Because both the second and third Interface Controllers fire the `awakeWithContext` method when you click the **Display Screen** button, you can call the `becomeCurrentPage` method in the `awakeWithContext` method. Hence, add the following statement in bold to the ThirdInterfaceController.swift file:

```
override func awakeWithContext(context: AnyObject?) {
    super.awakeWithContext(context)

    // Configure interface objects here.
    becomeCurrentPage()
    println("ThirdInterfaceController - awakeWithContext")
}
```

7. Run the application on the iPhone 6 Simulator and click the **Display Screen** button on the Apple Watch simulator. This time, you see that after the second Interface Controller is displayed, it will automatically scroll to the third one.

Summary

In this chapter, you delved deeper into how Interface Controllers work in your Apple Watch application. You learned

- The lifecycle of an Interface Controller
- How to navigate between Interface Controllers
- The different methods of displaying an Interface Controller
- How to programmatically display an Interface Controller
- How to display a series of Interface Controllers

This page intentionally left blank

Index

Symbols

– (minus) button, on Slider control, 62, 64–65
+ (plus) button, on Slider control, 62, 64–65
< (chevron)
  customizing title of, 34–35
  in hierarchical navigation, 26

A

Accessing web services, 126–130
Action buttons
  destructive, 151–152, 163
  displaying multiple, 161–163
  handling, 163–167
  for notifications, 150–152
  types of, 150, 163
Action Segue
  modal selection, 27
  push selection, 24
Animation, performing, 69–71
Apple Developer Program, 131–132
Apple Watch apps. *See also* Application(s)
  icons for, 159–160
  localization of. *See* Localization
  modifying display name of, 158
  sharing files with iOS app, 143–148
  testing, 14–15
  tools for, 2
  types of, 6
Apple Watch Simulator
  app tested on, 14
  Button tested on, 47
  dictation and, 85
  emojis and, 86
  glance displayed on, 186
  location data displayed on, 123
  notification displayed on, 156
  temperature displayed on, 130
  unlocking, 22
Apple Watch specifications, 1–2
ApplicationGroupContainerIdentifier key, 142
`application:handleActionWithIdentifier: forLocal-Notification:` method, 163–167
`application:handleActionWithIdentifier: forRemote-Notification:` method, 163–167
Application(s)
  adding target to, 8–11
  Apple Watch. *See* Apple Watch apps
  creating iPhone, 6–8
Archive button, for notifications, 150–151
Attributed strings
  customizing fonts with, 52–55
  displaying, 51
Attribute(s)
  Background, 59
  for customizing glances, 185
  Identifier, 25, 27, 36–39, 75
  Image control, 79, 95, 145–146
  Label control, 73
  Lines, 156–157
  Menu Item control, 93–94
  Mode, 70
  Selectable, 75, 81
  Slider control, 63–64
  Steps, 64–65
  Vertical, 73
Attributes Inspector window
  Background attribute in, 59
  changing Button title in, 46–47
  changing sash/title color in, 177
  Glance Interface Controller in, 182–183
  hierarchical setting in, 25
  of Interface Controller, 18
`awakeWithContext` method
  changing page displayed, 41–43
  initializing Interface Controller, 13, 20–22
  passing/retrieving data, 32–33

B

Background action button
  function of, 163
  for notifications, 150
Background fetch, implementing, 188–192
Background image
  Button control, 56–59
  setting Interface Controller, 65–67
  on Static Interface Controller, 160–161
`becomeCurrentPage` method, for changing display page, 43
`body` key, 174
Button control
  adding to Interface Controller, 46–47, 83, 86–87
  attributed strings and, 51
  changing background image, 56–59
  changing title dynamically, 50
  creating outlet for, 49–50
  creating/naming action for, 47–49, 114, 126
  custom fonts and, 52–55
  duplicating, 88
  features of, 46
  hierarchical navigation and, 23–24
  localization and, 100–101
  moving into Group control, 88–90
  in navigating using code, 36–37
  page-based navigation and, 27
Buttons project, 46–47

C

Cancel button
  customizing title of, 34–35
  page-based navigation and, 27–28
Chevron (<)
  customizing title of, 34–35
  in hierarchical navigation, 26
Color change for sash/title, 177
Controls (views)
  Button. *See* Button control
  Date, 112–113
  Group, 86–91
  Image. *See* Image control
  Label. *See* Label control
  Map, 123–125
  Menu, 91–92
  Menu Item, 91, 93–94, 97–98
  Slider, 62–65
  Switch, 59–61
  Table. *See* Table control
CoreLocation.framework, for location data, 115–116
`currentDateToString` method, in background fetch, 189, 191
Custom fonts
  getting names of, 55–56
  using, 52–55
Customization
  chevron/cancel button, 34–35
  Date control, 112–113
  font, 52–55
  glance, 182–185

D

Data
  passing between controllers, 28–33
  retrieving, 138–139
  saving in shared app group, 135–138
`dataToPhone` dictionary, passing data and, 127
`dataToWatch` dictionary
  accessing web services, 129–130
  for current location data, 119–120
Date control
  customizing, 112–113
  in different languages, 113
`dateStringToDate:` method of accepting information, 193–194
Device-specific images, 56–57
Dictation, inputs via, 84–85
Dictionary
  accessing web services, 127–130
  location data via, 115, 119–120
`didDeactivate` method
  changing page displayed, 41–42
  initializing Interface Controller, 13, 20–22
  passing data to controllers, 32
`didReceiveLocalNotification:withCompletion:` method, for long-look interface, 172
`didReceiveRemoteNotification:withCompletion:` method, for long-look interface, 172
Digital Crown, 2
Dismiss button, for Static Interface Controller, 154, 156, 157
Displaying information
  Image control for. *See* Image control
  Label control for, 65
  Table control for. *See* Table control
DisplayingGlances project, 180–194
`downloadImage:` method, in sharing files, 144–145
Dynamic Interface Controller
  changing sash/title color for, 177
  for long-look interface, 154, 168–173
  setting/displaying icons, 159–160

showing new notifications, 176
simulating delays in displaying, 178

E

Edit Scheme… menu item, 175
Emoji inputs, 85–86

F

Files, selecting/creating in localization, 103–104
First Button item
on Apple Watch notification, 156
multiple action buttons and, 161–162
First Interface Controller
Button control added to, 23–24, 27
passing data from, 28–33
returning to, 26
segue connecting, 25, 27
Fonts
customizing, 52–55
getting names of, 55–56
ForceTouch project, 91–97
Force Touch
adding images to project, 95
adding Label control with, 96
adding Menu control with, 91–92
definition of, 2
displaying context menu, 94, 97
Image control added with, 95
setting Menu item attributes with, 93–94
Foreground action button
function of, 163
for notifications, 150

G

Gathering information
dictation for, 84–85
emojis for, 85–86
text inputs for, 82–84
`GetCurrentLocation` class, 117–120
GetLocation project, 114–123
`getLocationWithCompletion:` method, for location data, 119–120
`GlanceController` class, 182
Glance Interface Controller
Attributes Inspector window of, 182–183
displaying information in, 192–194
implementing glances, 180–182
Glances
customizing, 182–185
implementing, 180–182
modifying for usefulness, 187–192
overview of, 6, 179
testing, 186
updating, 192–194
Gmail notifications, 150–152
Group control
adding/modifying Button for, 86–87
centralizing, 90
duplicating Button for, 88
implementing, 90–91
moving Buttons into, 88–89

H

`handleActionWithIdentifier: forLocalNotification:` method, 163–166
`handleActionWithIdentifier: forRemoteNotification:` method, 163–166
HelloAppleWatch project, 6–14
Hierarchical navigation
customizing chevron in, 34–35
displaying data passed via, 33
between Interface Controllers, 22, 23–26
Horizontal attribute, for Label control, 73

I

Icons, for Apple Watch apps, 159–160
Identifier attribute
of Interface Controller, 25, 27, 36–39
of Table Row Controller, 75
Identity Inspector window
Class attribute in, 12, 30, 169
of Interface Controller, 18
Image control
adding to Interface Controller, 67, 145–146
adding to Table control, 78
connecting outlet to, 79–80
creating outlet for, 146
for long-look interface, 168–169
performing animations via, 69–71
programmatically setting, 68–69
setting attributes for, 79
setting background for, 65–67
setting/testing, 68
uses of, 65
Images
adding to WatchKit app, 169
animation, 69–71
changing background, 56–59, 65–67
project, 65–71
setting background, 160–161
Table control displaying, 78–81
Impact font, 52–55

Include Glance Scene option, 180
Include Notification Scene option, in short-look interface, 153
Info.plist file, adding key to, 120
Information inputs
  dictation, 84–85
  emojis, 85–86
  text inputs for, 82–84
Initialization methods for Interface Controller, 13, 19–22
Interactive notifications, 150–151
`InterfaceController` class
  content of, 12–13
  selecting, 11–12
Interface Controller(s)
  action button launching, 166
  of Apple Watch app, 11–12
  Attributes Inspector window, 18
  Button control added to, 46–47, 83, 86–87
  changing background of, 65–67
  changing page displayed, 40–43
  connected to Swift class, 18–19
  Date control added to, 112–113
  deactivating, 22
  displaying series of, 37–38
  Glance Interface Controller with, 180–181
  Group control added to, 88–90
  hierarchical navigation, 23–26
  Image control added to, 67–69, 145–146
  initialization methods for, 19–20
  Label control added to, 13, 60, 63, 83, 96
  loading, 20–21
  Map control added to, 124
  Menu control added to, 91–92
  navigating using code, 35–38
  navigation between, 22–23
  page-based navigation, 27–28
  passing data between, 28–33
  Slider control added to, 62–65
  Switch control added to, 59
  Table control added to, 72
iOS app
  adding WatchKit app and, 9–11
  bundle, 4
  communicating with, 5–6
  consuming web services on, 126–130
  getting user location, 115–120, 122
  interfacing with. *See* Localization
  performing background fetch in, 188–192
  shared app groups and. *See* Shared app group
  sharing files with watch app, 143–148
iOS notifications, 149–152. *See also* Notifications
iPhone
  adding target to app, 8–11
  Apple Watch interaction with, 4–5
  creating app for, 6–8
iPhone Simulator
  Apple Watch app on, 142
  changing language on, 105
  displaying downloaded image, 147–148
  resetting to English, 112
  selecting country on, 137–138
  testing app on, 14–15
  unlocking, 22

L

Label control
  adding to Interface Controller, 13, 60, 63, 83, 96
  adding to Table control, 72
  connecting outlet to, 75
  creating outlet for, 30, 114, 126, 164
  features of, 65
  in hierarchical navigation, 25–26
  Lines attribute of, 156–157
  localizing, 102–106
  for long-look interface, 168–169
  naming outlet for, 31
  in navigating using code, 36
  setting attributes for, 73
  for Static Interface Controller, 154
  typing text into, 14
Languages
  Date control and, 112–113
  localization and, 102–106, 109–111
Layouts project, 86–91
Lifecycle of Interface Controller, 19–22
LifeCycle project, 17–19
Lines attribute, setting, 156–157
Local notifications, 149
Localization
  adding string file, 107
  changing language and, 105
  displaying title in, 106
  file selection/addition for, 103–104
  language selection for, 102, 109
  for multiple languages, 99–101
  naming string file and, 108
  project, 100–113
  of string files, 110–112
  string literals used in, 104
Location data
  adding Button/Label controls for, 114

adding new key for, 120
adding Swift file for, 117–118
displaying maps with, 123–125
displaying on Apple Watch, 123
displaying on Label control, 121–122
implementing code for, 118–119, 121
obtaining permission to access, 122
`openParentApplication:` method in, 115
preparing/adding new framework, 115–116
Long-look interface for notifications
features of, 167
implementing, 168–173
Lower group selections, in customizing glances, 184–185

M

Map control, for location data, 123–125
Menu control, 91–92
Menu Item controls
adding programmatically, 97–98
displaying image, 91, 93, 97
setting attributes for, 93–94
Minus (–) button on Slider control, 62, 64–65
Mode attribute, of Image control, 70

N

NavigateUsingCode project, 36–43
Navigation, of Interface Controller
hierarchical, 23–26
overview of, 22–23
page-based, 27–28
using code, 35–38
`NotificationController` class, 169–172
Notification Simulation File, 173
Notifications
action buttons for, 163–167
on Apple Watch, 152–153
customizing, 156–157
definition of, 149
long-look interface for, 167–172
other payloads for simulating, 173–176
overview of, 6
overview of iOS, 150–152
project, 153–178
setting background image for, 160–161
short-look interface for, 153–156
types of, 149–150
NSLocationAlways-UsageDescription key, 120–122
`NSURLConnection` class, downloading images and, 144–145
`NSURLSession` class, connecting to web service, 129–130
`NSUserDefaults` setting
in background fetch, 187, 189–191
saving data and, 135–137, 193

O

`openParentApplication:` method, passing data to iOS app, 115, 120, 127–129
Options button, for notifications, 151–152

P

Page-based navigation
changing page displayed and, 40–43
customizing Cancel button in, 34–35
displaying data passed via, 33–34
displaying series of pages and, 38–40
between Interface Controllers, 22–23, 27–28
`parseJSONData:` method
connecting to web service, 129–130
extracting data, 189–190
Picker view, saving data and, 135–137
Plus (+) button on Slider control, 62, 64–65
`presentControllerWithName:context:` method
displaying series of pages, 38–40
in page-based navigation, 37–38
`presentTextInputControllerWithSuggestions:` method
for emojis, 85–86
for text inputs, 82–84
Push notifications, 149
`pushControllerWithName:context:` method, in hierarchical navigation, 37–38
PushNotificationPayload.apns file, 154–156, 161–163
PushNotificationPayload-delayed.apns file, 173–176

R

Remote notifications
definition of, 149
with multiple action buttons, 161–163
Reply button, for notifications, 150–151
`replyDataFromPhone` dictionary, 127–128
Resolutions, of Apple Watch sizes, 1–2
Root.plist file, 141–142

S

Sash color, changing, 177
Second Interface Controller
Cancel button on, 27–28
Label control added to, 25
passing data to, 28–33
Segue, in hierarchical navigation, 24–25, 32–33
Selectable attribute, of Row controller, 75, 81

`setImageNamed:` method, 68–69
`setMinimumBackgroundFetchInterval` method, 190–191
Shared app group
  adding to WatchKit Extension, 187–188
  creating/adding to iOS project, 187
  development team/app group for, 133
  enrolling in Apple Developer Program, 131–132
  entering Apple ID/password, 132
  for extension target, 134–135
  naming new container for, 133–134
  retrieving data from, 138–139
  saving data in, 135–138
  sharing files and, 143–148
  turning on Capabilities feature, 131
  viewing newly created app group, 134
Short-look interface for notifications
  implementing, 153–156
  with multiple action buttons, 161–163
Single View Application, creating, 6–7
Slider control
  adding/testing, 62
  creating outlet for, 63
  setting attributes for, 63–64
  Steps attribute and, 64–65
Sliders project, 62–65
Specifications, Apple Watch, 1–2
Static Interface Controller
  changing sash/title color for, 177
  customizing notifications on, 156–157
  displaying action buttons, 161–163
  modifying display name on, 158
  reverting back to, 178
  setting background image for, 160–161
  setting/displaying icons, 159–160
  for short-look interface, 154
Steps attribute, for Slider control, 64–65
Stock prices
  background fetch of, 188–190
  retrieving, 193
Storyboard Editor, examining, 11–12
Storyboard file
  adding Interface Controllers to, 23, 154
  background image in, 59
  drag/drop Button onto, 46
  editing, 18
  selecting, 17
String files
  adding, 107
  language selection for, 109
  localization of, 110–112
  naming, 108
Swift class
  adding to project, 73
  assigning Table control to, 74
  for current location data, 117–118
Switch control
  adding to Interface Controller, 59
  changing title of, 59–60
  creating outlet for, 60
  testing, 61
Switches project, 59–62

T

Table control
  adding to Interface Controller, 71–72
  adding/assigning to Swift class, 73–74
  connecting image outlet in, 79–80
  creating outlet for, 76
  displaying images in rows, 81
  displaying list of items, 77
  features of, 71
  Image control added to, 78
  Image control attributes and, 79
  Label control added to, 72
  selecting items via, 81–82
  setting Table Row Controller Identifier, 75
Table Row Controller
  adding Image control to, 78
  Identifier attribute for, 75
  selecting, 74
`table:didSelectRowAtIndex:` method, 81–82
Tables project, 71–82
Taptic Engine, 2
TextInputs project, 83–85
Text inputs, 82–85
`timeIntervalSinceDate:` method, of retrieving information, 193–194
Title color, changing, 177

U

UI (user interface) controls
  Button. *See* Button control
  Date, 112–113
  Group, 86–91
  Image. *See* Image control
  Label, 65
  obtaining inputs and, 82–86
  overview of, 45
  Slider, 62–65
  Switch, 59–61
  Table. *See* Table control
UI (user interface) localizations, 102–106

UINavigation project, 23–34
Upper group selections, in customized glances, 183
User interaction response controls
  Button. *See* Button control
  overview of, 45
  Slider, 62–65
  Switch, 59–61
`UserInfo` argument
  in accessing web services, 128, 130
  passing data to iOS app, 118, 120

V

Vertical attribute, for Label control, 73
View Controller, saving data and, 135–137
Views. *See* Controls (views)

W

WatchKit app
  adding images to, 169
  adding to iPhone app, 8–11
  adding/naming font file in, 52–54
  deploying, 4
  function of, 3–4
  interaction with WatchKit Extension, 3–4
  lifecycle of, 12–13
  modifying name of, 158
  overview of, 6
  Settings app for, 140–143
WatchKit Extension
  adding shared app group to, 187–188
  adding to iPhone app, 10–11
  adding/naming font file in, 52–53
  function of, 3–4
  interaction with WatchKit app, 3–4
WatchKit framework
  types of applications, 6
  understanding, 3–6
WatchKit Settings Bundle
  adding items to, 142
  naming/viewing file in, 141
  selecting, 140
Weather information access, 126–130, 138–139
Web service access, 126–130
WebServices project, 126–148
`willActivate` method
  changing page displayed, 41–42
  initializing Interface Controller, 13, 20–22
  passing data to controllers, 32
  updating glances, 182
`WKInterfaceController` class
  `GlanceController` class extending, 182
  naming subclass of, 29
  subclassing, 12–13

X

Xcode
  for Apple Watch apps, 2
  background fetch and, 191, 192
  creating iPhone app in, 6–8
  Output Window in, 21–22
  in testing app, 14

Y

Yahoo web service connection, 188–190

This page intentionally left blank