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.

Visit informit.com/learningseries for a complete list of available publications.
Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the United States, please contact international@pearsoned.com.

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2015952426

Copyright © 2016 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copy-right, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearsoned.com/permissions/.

Apple, the Apple logo, Apple TV, Apple Watch, Cocoa, Cocoa Touch, eMac, FaceTime, Finder, iBook, iBooks, iCal, Instruments, iPad, iPad Air, iPad mini, iPhone, iPhoto, iTunes, the iTunes logo, iWork, Keychain, Launchpad, Lightning, LocalTalk, Mac, the Mac logo, MacApp, MacBook, MacBook Air, MacBook Pro, MacDNS, Macintosh, Mac OS, Mac Pro, MacTCP, the Made for iPad logo, the Made for iPhone logo, the Made for iPod logo, Metal, the Metal logo, the Monaco computer font, MultiTouch, the New York computer font, Objective-C, OpenCL, OS X, Passbook, Pixlet, PowerBook, Power Mac, Quartz, QuickDraw, QuickTime, the QuickTime logo, Retina, Safari, the Sand computer font, Shake, Siri, the Skia computer font, Swift, the Swift logo, the Textile computer font, Touch ID, TrueType, WebObjects, WebScript, and Xcode are trademarks of Apple, Inc., registered in the United States and other countries. OpenGL and the logo are registered trademarks of Silicon Graphics, Inc. Intel, Intel Core, and Xeon are trademarks of Intel Corp. in the United States and other countries.


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

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>xiii</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>xvii</td>
</tr>
<tr>
<td>About the Author</td>
<td>xix</td>
</tr>
<tr>
<td>1 Getting Started with WatchKit Programming</td>
<td>1</td>
</tr>
<tr>
<td>2 Apple Watch Interface Navigation</td>
<td>17</td>
</tr>
<tr>
<td>3 Responding to User Actions</td>
<td>49</td>
</tr>
<tr>
<td>4 Displaying and Gathering Information</td>
<td>73</td>
</tr>
<tr>
<td>5 Accessing the Apple Watch Hardware</td>
<td>125</td>
</tr>
<tr>
<td>6 Programming Complications</td>
<td>137</td>
</tr>
<tr>
<td>7 Interfacing with iOS Apps</td>
<td>161</td>
</tr>
<tr>
<td>8 Displaying Notifications</td>
<td>205</td>
</tr>
<tr>
<td>9 Displaying Glances</td>
<td>237</td>
</tr>
<tr>
<td>Index</td>
<td>253</td>
</tr>
</tbody>
</table>
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 5  
   Types of Apple Watch Applications 6  
   Hello, World! 7  
      Creating an iPhone Project 7  
      Examining the Storyboard 10  
      WatchKit App Lifecycle 10  
      Modifying the Interface Controller 13  
      Running the Application on the Simulator 13  
   Summary 15  

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 26  
      Passing Data between Interface Controllers 27  
      Customizing the Title of the Chevron or Cancel Button 33  
      Navigating Using Code 34  
      Presenting a Series of Pages 37  
      Changing the Current Page to Display 39  
      Returning Data from an Interface Controller 42  
   Summary 48
Building a Movie Showtime Complication Example 141
Creating the Project 141
Selecting Complication Families Support 143
Creating the Complication Placeholder Template 143
Setting Privacy Behavior 149
Populating the Complications with Real Data 150
Time Travel 154
Setting the Refresh Frequency 159
Summary 159

7 Interfacing with iOS Apps 161
Introducing the Watch Connectivity Framework 161
Types of Communication 162
Using the Watch Connectivity Framework 165
Comparing the Different Modes 183
Connecting to the Outside World 185
Getting Location Data 185
Display Map 192
Accessing Web Services 194
Saving Data 198
Creating the Project 198
Writing to Files 199
Using NSUserDefaults 202
Summary 203

8 Displaying Notifications 205
What Is a Notification? 205
Types of Notifications on the Apple Watch 208
Implementing the Short-Look Interface 209
Implementing the Long-Look Interface 225
Summary 235

9 Displaying Glances 237
What Is a Glance? 237
Implementing Glances 238
Customizing the Glance 240
Testing the Glance 244
Welcome to *Learning WatchKit Programming, Second Edition*!

This is an exciting time to be a programmer, as we are witnessing a new era of wearables. Although 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 it 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.

 Barely two months after the Apple Watch was made available for sale, Apple announced the second version of the Apple Watch OS, aptly named watchOS 2. Unsurprisingly, watchOS 2 now supports native apps and comes with a slew of new features.

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 the fundamental topics that you need to get started in Apple Watch programming. In particular, this second edition has been fully updated to cover watchOS 2 programming.

Because this is a book on Apple Watch programming, I 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, but 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, note the following:

- 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 of the code samples for this book have been tested against Xcode 7.
- If you plan to test your apps on a real device, you need to register to become a paying Apple developer (https://developer.apple.com/programs/). The program costs $99 per year for individuals. Once registered, you can register your Apple Watch’s UDID with Apple (necessary for testing on Apple Watch). The Apple Watch works only with iPhone 5, iPhone 5c, iPhone 5s, iPhone 6, iPhone 6 Plus, iPhone 6s, and iPhone 6s Plus (or newer versions of the iPhones).
- Most of the 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 a real Apple Watch (for example, to access the hardware features like accelerometer, microphone, etc.).
- 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 2.0. If you are not familiar with Swift, you can refer to Apple’s web page on Swift at https://developer.apple.com/swift/resources/.

How This Book Is Organized

This book is styled as a tutorial. You try 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 it ties in with your iOS apps. Most importantly, you get your chance to write a simple Apple Watch app and deploy it onto the Apple Watch 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, “Responding to User Actions”: 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. You will start off with the controls with which the user interacts.

Chapter 4, “Displaying and Gathering Information”: While Chapter 3 covers the various controls with which the user interacts through the tap gesture, this chapter continues to explore the various controls available in the WatchKit framework, focusing on controls that display information, as well as controls that gather information.

Chapter 5, “Accessing the Apple Watch Hardware”: In watchOS 1, Apple did not provide third-party developers access to the various hardware features of the Apple Watch, such as accelerometer, microphone, and Taptic Engine. However, in watchOS 2, Apple has exposed some of these features to developers so that they can create more exciting watch apps. In this chapter, you learn how to access some of these hardware features and see how they can be useful to the apps you are building.

Chapter 6, “Programming Complications”: A complication is a function on a timepiece that does more than just tell the time. Complications on a timepiece include alarms, tachymeters, chronographs, calendars, and so on. In watchOS 2, third-party apps can now also display data in watch face complications. In this chapter, you learn the process of creating an application that displays complication data.

Chapter 7, “Interfacing with iOS Apps”: This chapter discusses the Watch Connectivity Framework, a set of APIs that allow the containing iOS app to communicate with the watch app (and vice versa). In addition to discussing how apps intercommunicate, this chapter also discusses how to use location services in your watch app, as well as how to consume web services. Last, but not least, this chapter ends with a discussion on persisting data on your watch.

Chapter 8, “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 their essence quickly to the user.

Chapter 9, “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, and 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 getting 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 http://informit.com/title/9780134398983, click the Extras tab, and register your book.

Contacting the Author

If you have any comments or questions about this book, drop me an email at weimenglee@learn2develop.net, or stop by my web site at http://learn2develop.net.
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. I want to take this opportunity to thank the people who make all this happen.

I am indebted to Trina MacDonald, senior acquisitions editor at Pearson, 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 want to thank the many heroes working behind the scenes—copy editor Barbara Wood; 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.
About the Author

Wei-Meng Lee is a technologist and founder of Developer Learning Solutions (http://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
Responding to User Actions

If you haven’t found it yet, keep looking. Don’t settle. As with all matters of the heart, you’ll know when you find it. And like any great relationship, it just gets better and better as the years roll on.

Steve Jobs

Designing the user interface (UI) for your Apple Watch application is similar to designing for the iPhone. 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.

The UI of an Apple Watch application is represented by various controls (commonly known as *views* in iOS programming), and they are divided into two main categories:

- **Responding to user actions**: Users directly interact with these controls to perform some actions. Examples of such controls are Button, Switch, Slider, Picker, and Table.

- **Displaying information**: These controls mainly display information to the user. Examples of such controls are Label, Image, and Table.

In this and the next chapter, you learn how to use these various controls to build the UI of your application.

Using the Tap Gesture to Interact with Controls

One key way to interact with the Apple Watch is to use the tap gesture. You can tap the following controls:

- Button
- Switch
- Slider
- Table
Chapter 3  Responding to User Actions

Let’s take a more detailed look at these objects!

**Note**
I cover the Table control in the next chapter where we discuss controls that display information.

**Button**
The Button control is the most direct way of interacting with an Apple Watch application. A button can display text as well as a background image. Tapping a button triggers an action on the Interface Controller where you can write the code to perform the appropriate action.

**Adding a Button to an Interface Controller**
In this section, you create a project that uses a Button control. Subsequent sections show you how to customize the button by creating an action for it and then displaying its title using custom fonts.

1. Using Xcode, create a new iOS App with WatchKit App project and name it **Buttons**. Uncheck the option Include Notification Scene so that we can keep the WatchKit project to a bare minimum.
2. Select the Interface.storyboard file to edit it in the Storyboard Editor.
3. Drag and drop a Button control onto the storyboard, as shown in Figure 3.1.

![Figure 3.1 Adding a Button control to the Interface Controller](image)

4. In the Attributes Inspector window, set the Title attribute to **Play** (see Figure 3.2).
Using the Tap Gesture to Interact with Controls

5. Select the WatchKit App scheme and run the project on the Apple Watch Simulator. You should see the button on the Apple Watch Simulator (see Figure 3.3). You can click it (or tap it on a real Apple Watch).

Creating an Action for a Button

For the Button control to do anything useful, you need to create an action for it so that when the user taps it, your application performs some actions. To create this action, follow these steps:

1. In the Storyboard Editor, select the View | Assistant Editor | Show Assistant Editor menu item to show the InterfaceController.swift file.

2. Control-click the Button control in the Interface Controller and drag it over the InterfaceController class (see Figure 3.4).
Chapter 3  Responding to User Actions

3. Create an action for the button and name it **btnPlay** (see Figure 3.5). Click **Connect**.

4. You now see the action created in the InterfaceController.swift file:

```swift
import WatchKit
import Foundation

class InterfaceController: WKInterfaceController {

    @IBAction func btnPlay() {
    }
}
```

5. Add the following statement in bold to the InterfaceController.swift file:

```swift
@IBAction func btnPlay() {
    print("The button was tapped!")
}
```

6. Select the WatchKit App scheme and run the project on the Apple Watch Simulator. Click the **Play** button and observe the statement printed in the Output window (see Figure 3.6).
Creating an Outlet for a Button

You can also programmatically change the title of the Button control during runtime. To do so, you need to create an outlet for the button:

1. With the Assistant Editor shown, control-click the button and drag it over the InterfaceController.swift file. Name the outlet `button1` (see Figure 3.7) and click Connect.

2. This creates an outlet in the InterfaceController.swift file:

```swift
import WatchKit
import Foundation

class InterfaceController: WKInterfaceController {

    @IBOutlet var button1: WKInterfaceButton!

    @IBAction func btnPlay() {
        print("The button was tapped!")
    }
}
```

Figure 3.6 Clicking the button fires the action

Figure 3.7 Creating an outlet for the button
3. Add the following statements in bold to the InterfaceController.swift file:

```swift
override func awakeWithContext(context: AnyObject?) {
    super.awakeWithContext(context)
    // Configure interface objects here.
    button1.setTitle("Play Video")
}
```

**Note**

Observe that, while you can change the title of a button, you cannot get the title of the button programmatically.

4. Select the WatchKit App scheme and run the project on the Apple Watch Simulator. You should now see the title of the button changed to “Play Video” (see Figure 3.8).

![Figure 3.8 Changing the title of the button dynamically](image)

**Displaying Attributed Strings**

The Button control supports attributed strings. Attributed strings allow you to specify different attributes (such as color, font, size, etc.) for different parts of a string. In the following steps, you display the title of the button using different colors:

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

```swift
override func awakeWithContext(context: AnyObject?) {
    super.awakeWithContext(context)
    // Configure interface objects here.
    // button1.setTitle("Play Video")
    let str = NSMutableAttributedString(string: "Hello, Apple Watch!")
    button1.setTitle(str)
}
```
Using the Tap Gesture to Interact with Controls

```objective-c
//-----display the Hello in yellow---
str.addAttribute(NSForegroundColorAttributeName,
    value: UIColor.yellowColor(),
    range: NSMakeRange(0, 5))

//---display the , in red---
str.addAttribute(NSForegroundColorAttributeName,
    value: UIColor.redColor(),
    range: NSMakeRange(5, 1))

//---display Apple Watch! in green---
str.addAttribute(NSForegroundColorAttributeName,
    value: UIColor.greenColor(),
    range: NSMakeRange(7, 12))

button1.setAttributedTitle(str)
```

2. Select the WatchKit App scheme and run the project on the Apple Watch Simulator. You should see the title of the button displayed in multiple colors, as shown in Figure 3.9 (readers of the print book will not see the colors in the figure).

![Figure 3.9 Displaying the button title with mixed colors](image)

**Using Custom Fonts**

Using attributed strings, you can also use different fonts for parts of a string. To illustrate this, let’s modify the example in the previous section to display part of the button’s title using a custom font.

For this example, use the Impact font that is installed on your Mac. The Impact font is represented using the Impact.ttf file located in the /Library/Fonts/ folder.

1. Drag and drop a copy of the Impact.ttf file onto the Extension project in Xcode.
2. You are asked to choose a few options. Select the options shown in Figure 3.10. This adds the Impact.ttf file onto the Extension and WatchKit App projects.
Chapter 3  Responding to User Actions

Figure 3.10  Adding the font file to the Extension and the WatchKit App

Note

Remember to add the font file to both the WatchKit Extension and WatchKit App. Also, be aware that adding custom fonts to the project adds considerable size and memory usage to your watch app. So, try to use the system font unless you have a very good reason not to.

3. Figure 3.11 shows the Impact.ttf file in the project.

Figure 3.11  The font file in the project
Using the Tap Gesture to Interact with Controls

4. Add a new key named `UIAppFonts` to the Info.plist file located in the Extension and set its Item 0 to `Impact.ttf` (see Figure 3.12).

**Note**
If your Info.plist file does not show the items as shown in Figure 3.12, simply right-click any of the items in it and select **Show Raw Keys/Values**.

![Figure 3.12 Specifying the font filename in the Extension project](image1.png)

5. Likewise, add a new key named `UIAppFonts` to the Info.plist file located in the WatchKit App and set its Item 0 to `Impact.ttf` (see Figure 3.13).

![Figure 3.13 Specifying the font filename in the WatchKit app project](image2.png)
6. Add the following statements in bold to the InterfaceController.swift file:

```swift
override func awakeWithContext(context: AnyObject?) {
    super.awakeWithContext(context)
    // Configure interface objects here.
    // button1.setTitle("Play Video")
    let str = NSMutableAttributedString(
        string: "Hello, Apple Watch!",
        attributes: [NSForegroundColorAttributeName: UIColor.yellowColor(),
                     NSFontAttributeName: UIFont(name: "Impact", size: 22.0)!,
                     NSForegroundColorAttributeName: UIColor.redColor()])
    //---display the Hello in yellow---
    str.addAttribute(NSForegroundColorAttributeName,
                     value: UIColor.yellowColor(),
                     range: NSMakeRange(0, 5))
    //---display Hello using the Impact font, size 22---
    str.addAttribute(NSFontAttributeName,
                     value: UIFont(name: "Impact", size: 22.0)!,
                     range: NSMakeRange(0, 5))
    //---display the , in red---
    str.addAttribute(NSForegroundColorAttributeName,
                     value: UIColor.redColor(),
                     range: NSMakeRange(5, 1))
    //---display Apple Watch! in green---
    str.addAttribute(NSForegroundColorAttributeName,
                     value: UIColor.greenColor(),
                     range: NSMakeRange(7, 12))
    button1.setAttributedTitle(str)
}
```

7. Select the WatchKit App scheme and run the project on the Apple Watch Simulator. You should now see “Hello” displayed using the Impact font (see Figure 3.14).

![Figure 3.14 Displaying “Hello” using a custom font](image)
Using the Tap Gesture to Interact with Controls

Note
Once you have added a custom font to your project, you can use the font directly in Interface Builder by setting the Font attribute of a control to Custom and then selecting the font that you want to use in the Family attribute.

Getting the Font Name
One common problem in dealing with fonts is that the filename of the custom font that you are using is not always the same as the font name. The following code snippet allows you to print out the name of each font family and its corresponding font name:

```swift
for family in UIFont.familyNames() {
    print(family)
    for name in UIFont.fontNamesForFamilyName(family as String) {
        print("--\($(name)"")
    }
}
```

This code snippet prints the output as shown in Figure 3.15. For example, if you want to use the Helvetica Neue font, you have to specify in your code one of the font names printed: HelveticaNeue-Italic, HelveticaNeue-Bold, etc.

![Figure 3.15 Printing out the font families and their associated font names](image-url)
Changing the Background Image of Button

Besides displaying text, the Button control can also display a background image. The following exercise shows you how to add an image to the project and use it as the background of a button:

1. Drag and drop the image named play.png onto the Assets.xcassets item in the WatchKit App (see Figure 3.16).

   ![Figure 3.16 Adding an image to the project](image)

   **Figure 3.16 Adding an image to the project**

   2. In the Attributes Inspector window for the play.png image, check the watchOS checkbox (see Figure 3.17, right). Then, move the play.png into the box labeled

   ![Figure 3.17 Specifying device-specific images to use](image)

   **Figure 3.17 Specifying device-specific images to use**
2× (see Figure 3.17, middle). This signifies that this image will be displayed for all sizes of Apple Watch. If you want to use different images for the 38mm Apple Watch and the 42mm Apple Watch, you can drag and drop different images onto the boxes labeled “38 mm 2×” and “42 mm 2×.” For this example, you will use the same image for the two different watch sizes.

3. In the InterfaceController.swift file, add the following statements in bold:

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

    // Configure interface objects here.
    // button1.setTitle("Play Video")

    /*
    let str = NSMutableAttributedString(
        string: "Hello, Apple Watch!"
    )

    //---display the Hello in yellow---
    str.addAttribute(NSForegroundColorAttributeName, 
        value: UIColor.yellowColor(),
        range: NSMakeRange(0, 5))

    //---display Hello using the Impact font, size 22---
    str.addAttribute(NSFontAttributeName, 
        value: UIFont(name: "Impact", size: 22.0)!,
        range: NSMakeRange(0, 5))

    //---display the , in red---
    str.addAttribute(NSForegroundColorAttributeName, 
        value: UIColor.redColor(),
        range: NSMakeRange(5, 1))

    //---display Apple Watch! in green---
    str.addAttribute(NSForegroundColorAttributeName, 
        value: UIColor.greenColor(),
        range: NSMakeRange(7, 12))
    button1.setTitle(str)
    */

    button1.setBackgroundImageNamed("play")
}
```

4. Select the WatchKit App scheme and run the project on the Apple Watch Simulator. You should now see the image on the button (see Figure 3.18).
Do not use the `setBackgroundImage:` method by passing it a `UIImage` instance, like this:

```swift
button1.setBackgroundImage(UIImage(named: "play"))
```

This is because the `UIImage` class looks for the specified image ("play") in the main bundle (the Extension). And because the play.png file is in the Watch-Kit App, the image cannot be found and, therefore, the image will not be set successfully.

5. You can also set the background image of the button in the storyboard via the Background attribute in the Attributes Inspector window.

**Switch**

The Switch control allows the user to toggle between the ON and OFF states. It is commonly used in cases where you allow users to enable or disable a particular setting. In the following example, you will create a project and see how the Switch control works:

1. Using Xcode, create a new iOS App with WatchKit App project and name it **Switches**. Uncheck the option Include Notification Scene so that we can keep the WatchKit project to a bare minimum.
2. Select the Interface.storyboard file to edit it in the Storyboard Editor.
3. Drag and drop a Switch control onto the default Interface Controller (see Figure 3.19).
4. In the Attributes Inspector window, set the Title attribute of the Switch control to **Aircon** (see Figure 3.20).
5. Add a Label control to the Interface Controller (see Figure 3.21).
6. Create an outlet for the Switch control and name it **switch**. Likewise, create an outlet for the Label control and name it **label**. Then, create an action for the Switch control and name it **switchAction**. The InterfaceController.swift file should now look like this:

```swift
import WatchKit
import Foundation

class InterfaceController: WKInterfaceController {
    @IBOutlet var `switch`: WKInterfaceSwitch!
    @IBOutlet var label: WKInterfaceLabel!
    @IBAction func switchAction(value: Bool) {
    }
}
```

**Note**
Because **switch** is a reserved word in the Swift programming language, if you try to use it as the name of an outlet, you have to enclose it with a pair of back quotes (` `).

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

```swift
@IBAction func switchAction(value: Bool) {
    value ? label.setText("Aircon is on") :
    label.setText("Aircon is off")
}

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

    // Configure interface objects here.
    `switch`.setOn(false)
    label.setText("")
}
```

**Note**
You can programmatically set the value of a Switch control, but you will not be able to get its value. To know its value, you need to implement the action of the Switch control and save its value whenever its state changes.

9. Select the WatchKit App scheme and run the project on the Apple Watch Simulator. On the Apple Watch Simulator, click the Switch control to turn it on and off and observe the message printed in the Label control (see Figure 3.22).
Slider

The Slider control is a visual control with two buttons (− and +) that allow the user to decrement or increment a floating-point value. It is usually used in situations where you want the user to select from a range of values, such as the temperature settings in a thermostat or the volume of the iPhone.

1. Using Xcode, create a new iOS App with WatchKit App project and name it Sliders. Uncheck the option Include Notification Scene so that we can keep the WatchKit project to a bare minimum.
2. Select the Interface.storyboard file to edit it in the Storyboard Editor.
3. Drag and drop a Slider control onto the default Interface Controller (see Figure 3.23).
4. Select the WatchKit App scheme and run the project on the Apple Watch Simulator. On the Apple Watch Simulator, click the + and – buttons (see Figure 3.24) and observe the slider.

![Slider Test](image)

Figure 3.24 Testing the slider

5. Add a Label control to the Interface Controller (see Figure 3.25).

![Label Added](image)

Figure 3.25 Adding a label to the Interface Controller

6. Create an outlet for the Slider control and name it `slider`. Likewise, create an outlet for the Label control and name it `label`. Then, create an action for the Slider control and name it `sliderAction`. The InterfaceController.swift file should now look like this:

```swift
import WatchKit
import Foundation
```
Using the Tap Gesture to Interact with Controls

67

class InterfaceController: WKInterfaceController {

    @IBOutlet var slider: WKInterfaceSlider!
    @IBOutlet var label: WKInterfaceLabel!

    @IBAction func sliderAction(value: Float) {
    }

7. Set the attributes for the Slider control as follows (see Figure 3.26):
    Maximum: 10
    Steps: 5

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

    @IBAction func sliderAction(value: Float) {
        label.setText(String(value))
    }

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

        // Configure interface objects here.
        slider.setValue(0.0)
        label.setText("0.0")
    }

    Note
You can programmatically set the value of a Slider control, but you will not be able to get its value. To know its value, you need to implement the action of the Slider control and save its value whenever the value changes.
Chapter 3  Responding to User Actions

9. Select the WatchKit App scheme and run the project on the Apple Watch Simulator. Click the – and + buttons and observe the value printed on the Label control (see Figure 3.27).

![Testing the slider](image)

Figure 3.27  Testing the slider

The Steps attribute specifies how many times you can click the slider to reach its maximum value. The increment or decrement value of the slider at any point is dependent on the length of the slider (Maximum value minus Minimum value) divided by the value of Steps. In this example, the length of the slider is 10 (maximum of 10 minus minimum of 0) and the value of Steps is 5; hence, the slider increments or decrements by 2 whenever the + or – button is clicked.

Alerts and Action Sheets

In watchOS 2, Apple now allows developers to display alerts and actions just like they did in iPhone and iPad:

1. Using Xcode, create a new iOS App with WatchKit App project and name it UsingAlerts. Uncheck the option Include Notification Scene so that we can keep the WatchKit project to a bare minimum.
2. Select the Interface.storyboard file to edit it in the Storyboard Editor.
3. Drag and drop a Button control onto the default Interface Controller (see Figure 3.28) and set its title to Show Alerts.
4. Create an action for the Button control and name it btnShowAlerts. The InterfaceController.swift file should now look like this:

```swift
import WatchKit
import Foundation

class InterfaceController: WKInterfaceController {

    @IBAction func btnShowAlerts() {
    }
}
```
Using the Tap Gesture to Interact with Controls

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

```swift
import WatchKit
import Foundation

class InterfaceController: WKInterfaceController {

    func performAction(actionStyle: WKAlertActionStyle) {
        switch actionStyle {
        case .Default:
            print("OK")
        case .Cancel:
            print("Cancel")
        case .Destructive:
            print("Destructive")
        }
    }

    @IBAction func btnShowAlerts() {
        let okAction = WKAlertAction(title: "OK",
                                     style: WKAlertActionStyle.Default) { () -> Void in
                                      self.performAction(WKAlertActionStyle.Default)
        }

        let cancelAction = WKAlertAction(title: "Cancel",
                                          style: WKAlertActionStyle.Cancel) { () -> Void in
                                      self.performAction(WKAlertActionStyle.Cancel)
        }

        let abortAction = WKAlertAction(title: "Abort",
                                         style: WKAlertActionStyle.Destructive) { () -> Void in
                                      self.performAction(WKAlertActionStyle.Destructive)
        }
    }
```
Chapter 3  Responding to User Actions

```swift
presentAlertControllerWithTitle("Title",
message: "Message",
preferredStyle: WKAlertControllerStyle.Alert,
actions: [okAction, cancelAction, abortAction])
}
```

Here, you first defined a function named `performAction:` that prints out a message depending on the style that is passed in as the argument. Next, in the `btnShowAlerts` action, you created three `WKAlertAction` instances, each with a specific style (Default, Cancel, and Destructive). Within each instance, you have a closure that is fired when the user clicks on the action buttons. When each button is clicked, you simply call the `performAction:` function to print out a message so that you know which button was clicked. Finally, you called the `presentAlertControllerWithTitle:message:preferredStyle:actions:` method to display an alert, together with the three action buttons.

6. Select the WatchKit App scheme and run the project on the Apple Watch Simulator. Clicking the button displays an alert (see Figure 3.29).

![Figure 3.29 Displaying an alert in the Apple Watch](image)

**Note**

Note that the Abort button in the alert is displayed in red as its style is set to Destructive.

7. Modify the `presentAlertControllerWithTitle:message:preferredStyle:actions:` method, as follows:

```swift
//---SideBySideButtonsAlert supports exactly two actions---
presentAlertControllerWithTitle("Title",
message: "Message",
preferredStyle: WKAlertControllerStyle.SideBySideButtonsAlert,
actions: [okAction, cancelAction])
```
8. Select the WatchKit App scheme and run the project on the Apple Watch Simulator. Clicking the button displays an alert with the two buttons displayed side by side (see Figure 3.30).

![Figure 3.30 Displaying an alert with two buttons side by side in the Apple Watch](image)

**Note**
For the SideBySideButtonsAlert style, you need to specify exactly two action buttons.

9. Modify the `presentAlertControllerWithTitle:message:preferredStyle:actions:` method as follows:

```swift
presentAlertControllerWithTitle("Title",
message: "Message",
preferredStyle: WKAlertControllerStyle.ActionSheet,
actions: [okAction, cancelAction, abortAction])
```

10. Select the WatchKit App scheme and run the project on the Apple Watch Simulator. Clicking the button displays an alert, as shown in Figure 3.31.

**Note**
When using the ActionSheet style, the action button that is set to the Cancel style is displayed at the top-left corner of the screen. Even if you do not specify the cancel action button, a default Cancel button is still displayed to close the action sheet (though in this case you cannot handle the event that is fired when the user taps the Cancel button).
Summary

In this chapter, you looked at the various controls that you can use to build the UI of your Apple Watch application. In particular, you saw the various controls that you can interact with by using the tap gesture, such as the Button, Switch, and Slider controls. In addition, you learned about the new alerts and action sheets that you can use to display information in watchOS 2. In the next chapter, you learn more about the other controls that primarily display information to the user.
Symbols and Numbers
< (chevron), 25, 33–34
38mm (small) watch, 1–2
42mm (large) watch, 1–2

A
Accelerometer, 131–134
Action buttons
  in email, 206–208
  in notifications, generally, 219–220
  in notifications, handling, 220–225
  types of controls, 51–53
Actions
  segues in, 24
  sheets for, 68–72
Alerts. See also Notifications, 68–72
Apple Watch
  Application Context for, 162–163
  creating apps in, generally, 7
  creating iPhone apps in, 7–9
  deploying apps in, 4–5
  development tools for, 2
  File Transfer for, 164
  hardware in. See Hardware, Apple Watch
  Hello, World! app in. See Hello, World! app
  Interface Controller in. See Interface Controller
    introduction to, 1–2
  iPhones interacting with, 5–6
  lifecycle of apps in, 10–13
  live communications in, 165
  Send Message mode in, 165
  Simulator in. See Apple Watch Simulator
    specifications of, 1–2
    storyboards in, 10–11
    types of apps in, 6
    User Info for, 163–164
    user interfaces in. See User interfaces (UI)
    WatchKit architecture and, 3–4
Apple Watch Simulator
  adding images in, 76–78
  animation in, 80
  background images in, 74
  didActivate method in, 21–22
  display of Current page in, 41–42
  emojis in, 110
  Force Touch in, 118–121
  Glance scenes in, 244
  hierarchical navigation in, 25–26, 32
  images next to text in, 89
  interactive messaging in, 182–183
  introduction to, 13–15
  location data in, 188–191
  Map controls in, 193–194
  menu items in, adding programmatically, 122–123
  Movie controls in, 104–106
  navigation using code in, 36, 38
  notifications in, 210
  page-based navigation in, 26–27, 33
  passing data between Interface Controllers in, 32–33
  Picker controls in, 92, 94–97, 100
  selecting items in tables in, 90
  Table controls in, 86
  text input in, 108
  Watch Connectivity Framework in, 171, 174
  writing data to files in, 201–202
Application Context
background transfers in, 162–163
comparing to other modes, 184
introduction to, 162
sending data in iOS app, 169–170
sending data in WatchKit Extension, 167–169
on Simulators, 171
testing applications in, 171
Application tests. See Testing applications
Architecture, 3–4
Assets.xcassets files
animation in, 78
background images in, 60, 74–75
context menus in, 116–118
control knobs in, 99
images next to text in, 86
Picker displaying images in, 93
playing movies in, 104
poster images in, 105
Attributed strings, 54–55
Attributes Inspector, 24–25, 240–243
Audio recordings, 127–130
awakeWithContext method
in display of Current page, 41–42
for Glance scenes, 240
in Interface Controller, 20–21
in Label controls, 30–32
in ReturningValues project, 45–47
B
Background action buttons, 206
Background fetch, 244–249
Background images
in Button controls, 60–62, 74
in Interface Controller, 74–75
in notifications, 217–218
Background transfers
in Application Context, 162–163
comparison of communication modes for, 184
in File Transfer, 164
in User Info, 163–164
in Watch Connectivity Framework, 162–164
Bluetooth LE (Low Energy) connections, 3
Bundles, 3
Button controls
action sheets in, 68–72
adding to Interface Controller, 50–51
alerts in, 68–72
for audio recordings, 127–130
for background actions, 206
background images and, 60–62, 74
for cancellations, 33–34
chevron, 33–34
creating actions for, 51–53
creating outlets for, 53–54
custom fonts in, 55–59
displaying attributed strings in, 54–55
for displaying screens, 26–27
for email actions, 206–208
for foreground actions, 206
Group controls and, 111–115
introduction to, 50
for making phone calls, 125–127
for notification actions, 219–225
for open actions, 207–208
for reply actions, 206–208
for saving data, 198–199
for sending messages, 125–127
Slider controls in, 65–68
Switch controls in, 62–65
in Taptic Engine, 134–136
text input and, 107–108
C
Cancellations
in Interface Controller, 33–34
in Watch Connectivity Framework, 179–180
Chevron (<), 25, 33–34
Circular Small complications
introduction to, 139–140
in Movie Showtime project, 153, 156, 158
support for, 143
CLKComplicationDataSource protocol, 138–139, 143
ClockKit Framework
introduction to, 138–139
placement of complications in, 139–140
template classes in, 140
Code Editor
- Label controls and, 29–30
- navigation using, 35
- returning data with, 44

Communications
- Application Context mode of. See Application Context
  - comparison of modes of, 184
- File Transfer mode of. See File Transfer
  - live, 165
- User Info mode of. See User Info
  - in Watch Connectivity Framework, generally, 162–165, 184
  - in WatchKit Extension, 175

Complications
- ClockKit Framework for, 138–140
- families of, 143
- introduction to, 6, 137–138
- Movie Showtime project example of. See Movie Showtime project
  - placeholder templates for, 143–149
  - placement of, 139–140
  - populating with data, 150–153
  - privacy behavior in, 149–150
  - refresh frequency in, 159
  - summary of, 159
  - template classes for, 140
- Time Travel for, 154–158

Connectivity
- location data in, 185–191
- Map display in, 192–194
- Web services, accessing, 194–198

Containing iOS Apps. See also iOS Apps, 3, 10

Controllers
- Dynamic Interface, 210, 216
- Glance Interface, 238–243
- Static Interface. See Static Interface Controller
  - View. See View Controller
  - WKInterfaceController class for, 12

Controls
- Button. See Button controls
- Group, 111–115
- Image. See Image controls
- Label. See Label controls

- layout of, 111–115
- Menu, 116–121
- Movie, 73
- Picker. See Picker controls
- Slider, 65–68
- Switch, 62–65
- Table, 73, 87–90

in user interfaces. See User interfaces (UI)

Cook, Tim, 1

Custom fonts, 55–59

D

Data
- passing between Interface Controllers, 27–33
- returning from Interface Controllers, 42–48
- saving, 198–202
- sending. See Sending data

Delegation design patterns, 42

Deploying Apple Watch apps, 4–5

Development tools, 2

didActivate method
  - in display of Current page, 40
  - for Glance scenes, 240
  - in Label controls, 31
  - in LifeCycle project, 20–22

Digital Crown
- Apple Watch hardware and, 130–131
- introduction to, 2
- Picker controls in, 90, 92, 97

dismissController method, 36

Displaying information. See also Displays
  - background images in, 74–75
  - Force Touch for, adding items
    - programatically, 121–123
  - Force Touch for, context menus in,
    - 115–121
  - Force Touch for, generally, 111–115
  - Image controls for, animations in, 78–80
  - Image controls for, generally, 74, 76–78
  - images next to text in, 86–89
  - introduction to, 73
  - Label controls for, generally, 73
  - laying out controls for, 111–115
  - Movie controls for, 103–106
Displaying information (continued)
movies, playing programatically, 101–103
Picker controls for, captions in, 96–97
Picker controls for, control knobs in, 97–100
Picker controls for, displaying images in, 93–94
Picker controls for, generally, 90
Picker controls for, lists of text in, 91–93
Picker controls for, scrolling style in, 94–96
playing media files, 100–106
summary of, 123
Table controls for, adding Image controls to, 87–89
Table controls for, generally, 80–86
Table controls for, selecting items in, 89–90
Displays
of Current page, 39–42
Display Screen buttons for, 26–27
of Glance scenes. See Glance scenes
of information. See Displaying information
of Interface Controller pages, 37–38
of maps, 192–194
Modal, 26–27
of notifications. See User Info
Dynamic Interface Controller, 210, 216

E
Email, 206–208
Emojis, 109–110

F
File Transfer
background transfers in, 164
canceling outstanding transfers in, 179–180
comparison to other communication modes, 184
introduction to, 162
sending data/files in iOS app, 177–179
sending data/files in WatchKit Extension, 175–177
testing applications in, 179
Fonts, 55–59
Force Touch
context menus, adding items programmatically, 121–123
context menus for, generally, 115–121
introduction to, 2, 111–115
Menu Item, 116–123
Foreground action buttons, 206

G
Gathering information
emojis, 109–110
introduction to, 73, 106
summary of, 123
text inputs, 106–108
Get Weather, 194–198
Glance Interface Controller, 238–243, 249–252
Glance scenes
background fetch in, 244–249
customizing, 240–253
implementing, 238–240
introduction to, 6, 237
summary of, 252
testing, 244
updating, 249–252
usefulness of, 244
Gmail, 206–208
Group controls, 111–115

H
Hardware, Apple Watch
accelerometer and, 131–134
for audio recording, 127–130
Digital Crown and, 130–131
introduction to, 125
for phone calls, 125–127
for sending messages, 125–127
Taptic Engine and, 134–136
Hello, World! app, 7–9
Hierarchical navigation
between Interface Controllers, 22–26
using code for, 36
Home Screen, 215–216

I
Icons, 215–217
Identifier attributes, 43
Image controls
animation with, 78–80
emojis in, 109–110
in Force Touch, 119–120
introduction to, 74, 76–78
saving data with, 198–199
in Table controls, 87–89
text next to images and, 86–89
ImageView, 177–178
Include Notification Scene, 209
Information
displaying. See Displaying information
gathering. See Gathering information
Interactive Messaging
comparison to other communication
modes, 184
in iOS Apps, 181–182
testing applications in, 182–183
in Watch Connectivity Framework, 180–183
in WatchKit Extension, 180–181
Interface Controller
accelerometers in, 131–134
animation in, 78–80
audio recordings in, 127–130
background images in, 74–75
Button controls in, 50–51, 91–93
Cancel buttons in, 33–34
designating specific pages to display in,
39–42
displaying series of pages in, 37–38
emojis in, 109–110
File Transfer in, 175–177
Force Touch in, 115–121
Get Weather in, 194–198
Glance scenes in, 238–240
Group controls in, 111–115
hierarchical navigation in, 22–26
Image controls in, 74–78
interactive messaging in, 180–181
introduction to, 10–13
Label controls in, 80–86
LifeCycle project in, 17–22
lifecycle of, 17–22
location data in, 186–188
Map controls in, 192–193
menu items in, adding programmatically,
121–123
modification of, 13
Movie controls in, 103–106
navigating between, generally, 22–27
navigating using code, 34–37
notifications in, 210
passing data between, 27–33
phone calls in, 126–127
Picker controls in. See Picker controls
returning data from, 42–48
saving data in, 198–202
selecting items in tables in, 89–90
sending messages in, 126–127
Storyboard Editor and. See Storyboard
Editor
summary of, 48
Swift class and, 17–22
Taptic Engine in, 134–136
text input and, 107–108
User Info in, 172–174
Watch Connectivity Framework in,
167–169
Interface.storyboard file, defined. See also
Storyboard Editor, 10
iOS Apps
Application Context in, 169–170
background fetch in, 245–249
containing, 3, 10
creating iPhone apps, 7–9
Interactive Messaging in, 181–182
interfacing with, generally. See Watch
Connectivity Framework
introduction to, 3–5
location data in, 185–191
Map controls in, 192–194
saving data in, 198–202
sending data in, 169–174, 177–179
User Info in, 173–174
Watch Connectivity Framework for. See
Watch Connectivity Framework
iPhone apps
Apple Watch interacting with, 5–6
Application Context for, 162–163
creation of, 7–9
File Transfer for, 164
live communications in, 165
Send Message mode in, 165
simulation of. See iPhone Simulator
User Info for, 163–164
View Controller for, 22
iPhone Simulator
DisplayingGlances project in, 249
interactive messaging in, 182–183
introduction to, 13–15
location data in, 188–191
notifications in, 211, 213–218
unlocking, 22
Watch Connectivity Framework in, 171, 174
writing data to files in, 201–202

Jobs, Steve
on change, 161
on creating for oneself, 237
on design as function, 1
on ease of use, 125
on innovation, 73
on leading via innovation, 73
on limitation of focus groups, 17
on mistakes, 137
on quality of choices, 49

Label controls
for accelerometers, 131–134
for displaying information, 73
Force Touch and, 120–121
navigation and, 25, 29–30
notifications in, 212–214
for saving data, 198–199
in Table controls, 80–86
text input and, 107–108

Large (42mm) watch, 1–2
Layout of controls, 111–115
LifeCycle project, 17–22
Lifecycles of apps, 10–13
Live communications, 165
Local notifications, 205–208, 220
Locations
getting data for, 185–191
Map controls for, 192–194
in ReturningValues project, 45–48
Long-look interfaces
handling notifications with, 225–230
introduction to, 209–210
naming notifications in, 215–216
Low Energy (Bluetooth LE) connections, 3
Lower group in Glance Interface Controller, 240–243

M
Map controls, 192–194
Media files, playing, 100–106
Menu controls, 116–121
Menu Item controls
adding programmatically, 121–123
Force Touch and, 116–121
Messages, sending. See Sending messages
Modal displays, 26–27
Modal segues, 22, 26–27
Modular Small/Large complications
changing complications to, 147–149
introduction to, 139–140
in Movie Showtime project, 152, 155–157
placeholder templates for, 144–145
support for, 143
Movie Showtime project
complication families supported by, 143
complication placeholder template in, 143–149
creating project for, 141–143
introduction to, 141
populating with data, 150–153
privacy behavior in, 149–150
refresh frequencies in, 159
Time Travel for, 154–158
Movies
controls for, 73, 103–106
Movie Showtime project for. See Movie
Showtime project
playing programmatically, 101–103
refreshing, 159
timelines of, 151–158

N
Naming notifications, 214–215
NavigateUsingCode project, 35
Navigation
hierarchical, 22–26, 36
between Interface Controllers, 22–27
Label controls and, 25
page-based, 22, 26, 36
UINavigations project, 23, 27
using code for, 34–37
Notification Center, 215–216
Notifications
    action buttons in, generally, 219–220
    action buttons in, handling, 220–225
    background images in, 217–218
    customizing, 212–214
    definition of, 205–208
    displaying, generally, 205
    icons for Apple Watch apps and, 215–217
    introduction to, 6
    long-look interface for, 225–230
    modifying WatchKit app names for, 214–215
    sash colors in, 234–235
    short-look interfaces for, 209–212
    simulation of payloads for, 230–234
    summary of, 235
    types of, generally, 208–209
NSNotificationCenter, 42
NSUserDefaults class
    Glance scenes and, 245
    saving data and, 198, 202
O
    Open action buttons, 207–208
Outlets, 53–54
P
Page-based navigation
    between Interface Controllers, 26
    introduction to, 22
    using code in, 36
Passing data between Interface Controllers, 27–33
Payloads, 230–234
Phone calls. See also iPhone apps, 125–127
Picker controls
    captions in, 96–97
    control knobs in, 97–100
    in Digital Crown, 131
    displaying images with, 93–94
    displaying information with, 73, 90
    lists of text in, 91–93
    returning data with, 43, 45–48
    scrolling style in, 94–96
    in Taptic Engine, 134–136
Playing media files, 100–106
presentControllerWithName:context: method, 36
Privacy behavior, 149–150
Push segues, 24
pushControllerWithName:context: method, 36
PushNotificationPayload.apns file, 210–213, 219
R
Refresh frequencies, 159
Remote notifications
    action buttons for, 220
    displaying, 210–211
    introduction to, 205–208
    Reply action buttons, 206–208
    requestLocation method, 185–188
Returning data from Interface Controllers, 42–48
ReturningValues project, 42
S
Sash colors, 234–235
Saving data
    with Image controls, 198–199
    NSUserDefaults class in, 202
    writing to files, 199–201
Scrolling, 130
SecondInterfaceController.swift file, 39–42, 43
Sending data
    in iOS Apps, 169–170, 173–174
    via Application Context, 167–171
    via File Transer, 174–179
    via User Info, 172–174
Sending messages
    in Apple Watch, 125–127, 165
    in iPhones, 165
    in Watch Connectivity Framework, 180–181
Short-look interfaces, 209–212, 215–216
Simulation
    in Apple Watch. See Apple Watch Simulator
    introduction to, 13–15
    in iPhones. See iPhone Simulator
    of payloads, 230–234
Slider controls, 65–68
Small (38mm) watch, 1–2
Smartwatches. See Apple Watch
Specifications of Apple Watch, 1–2
Static Interface Controller
  action buttons in, 220
  background images in, 217–218
  customizing notifications in, 212–214
  displaying notifications in, generally, 210–212
  icons for Apple Watch app in, 216–217
  naming notifications in, 214–215
Stock prices, 245–252
Storyboard Editor
  accelerometers in, 131
  action sheets in, 68
  actions for buttons in, 51–52
  alerts in, 68
  background images in, 75
  buttons in, 50–52
  context menus in, 116
  control layout in, 111
  hierarchical navigation in, 23
  Interface.storyboard file in, generally, 10
  introduction to, 10–11
  lifecycles in, 17–22
  lists of text in, 91
  navigation using code in, 35
  notifications in, 210
  passing data between Interface Controllers in, 29
  phone calls in, 125–126
  playing movies programatically, 101
  recording audio in, 127
  returning data in, 42–44
  saving data in, 198
  sending messages in, 125–126
  Slider controls in, 65
  Switch controls in, 62–63
  Table controls in, generally, 80
  Table Row Controller in, 83, 88
  Taptic Engine in, 134
  text inputs in, 106
  Web services in, 194
Strings
  displaying attributed, 54–55
  saving to files, 199
Swift class
  in Interface Controller, 17–22
  in Table controls, 82–83
  Switch controls, 62–65
T
Table controls
  Image controls in, 87–89
  introduction to, 73, 80–86
  selecting items in, 89–90
Taptic Engine
  Apple Watch hardware and, 134–136
  introduction to, 2
  Picker controls in, 134–136
Templates
  in ClockKit Framework, 140
  for complications, 140, 143–149
  WatchKit Class, 28
Testing applications
  in Application Context, 170–171
  in File Transfer, 179
  Glance scenes, 244
  for interactive messaging, 182–183
  in User Info, 174
Text
  images next to, 86–89
  input of, 106–108
  lists of, in Picker, 91–93
ThirdInterfaceController.swift file, 39–42
Time Travel, 139, 154–158
Timelines of movies, 151–158
U
UI (User interfaces). See User interfaces (UI)
UINavigation project, 23, 27
Updating Glance scenes, 249–252
Upper group in Glance Interface Controller, 240–242
User Info
  background transfers in, 163–164
  canceling outstanding transfers in, 179–180
  comparison to other communication modes, 184
  introduction to, 162
  sending data in iOS Apps, 173–174
  sending data in WatchKit Extension, 172–173
  testing applications in, 174
User interfaces (UI)
  action sheets in, 68–72
  adding Button controls to Interface Controller, 50–51
  alerts in, 68–72
  background images of, 60–62
  Button controls in, generally, 50
  creating actions for, 51–53
  creating outlets for, 53–54
  custom fonts in, 55–59
  displaying attributed strings in, 54–55
  introduction to, 49
  Slider controls in, 65–68
  Switch controls in, 62–65
  tap gestures and, generally, 49–50
Utilitarian Small/Large complications
  introduction to, 139–140
  in Movie Showtime project, 152–153, 156, 158
  support for, 143, 144–145

V
Vibrator function, 134

View Controller. See also Views
  Application Context in, 169–170
  interactive messaging in, 181–182
  on iPhones, generally, 22
  User information in, 166
  Watch Connectivity Framework in, 166, 169–170

Views
  defined, 49
  ImageView for, 177–178
  windows of, 177–178

W
Watch Connectivity Framework
  background transfers in, 162–164
  cancelling outstanding transfers in, 179–180
  communication types in, 162–165, 184
  comparison of modes in, 183–184
  Interactive Messaging in, 180–183
  introduction to, 5–6, 161–162
  live communications in, 165
  sending data/files via File Transfer, 174–179
  sending data via Application Context, 167–171
  sending data via User Info, 172–174
  testing applications in, 170–171, 174, 179
  using, generally, 165–166
WatchKit. See also specific functions
  apps in, defined, 3, 6
  architecture of, 3–4
  Class template, 28
  deploying Apple Watch apps and, 4–5
  displaying information in, generally, 73
  Extension. See WatchKit Extension
  Interface Controller in. See Interface Controller
  introduction to, 1, 2
  iOS Apps in. See iOS Apps
  iPhone apps in. See iPhone apps
  lifecycle of apps in, 10–13
  storyboards in. See Storyboard Editor
  summary of, 15
WatchKit Extension
  creating iPhone apps in, 8–9
  ExtensionDelegate class in, 12–13
  Interactive Messaging in, 180–181
  introduction to, 3–4
  UINavigations project in, 27–33
  WatchOS operating system, 1, 3–4
  Weather, 194–198
  Web Services, 194–198
  willActivate method
    in display of Current page, 40
    for Glance scenes, 240
    in Interface Controller, 20–21
    in Label controls, 31
  WKInterfaceController class, 12
  WKSession object, 179–180
  World Wide Developers Conference (WWDC2015), 1

Writing data to files, 199–201
WWDC2015 (World Wide Developers Conference), 1

X
Xcode 7
  AudioRecord project in, 127
  Buttons in, 50
  Communications in, 165, 171, 174
Xcode 7 (continued)
  complication placeholder template in, 145
  custom fonts in, 55
  DisplayingGlances project in, 238, 245, 249
  FileStorage project in, 198, 201–202
  ForceTouch project in, 115
  Images in, 74
  introduction to, 2
  launching, 7–9
  Layouts in, 111
  LifeCycle project in, 17, 21
  Movie Showtime project in, 141
  Movies in, 101
  NavigationUsingCode project in, 35
  Notifications in, 209, 211, 221
  phone calls in, 125
  ReturningValues project in, 42
  simulators and, generally, 13–14
  Sliders in, 65
  Switches in, 62
  Tables in, 80
  Taptic Engine in, 134
  TextInputs project in, 106
  UINavigations project in, 23
  UsingAlerts project in, 68
  UsingCoreMotion project in, 131
  UsingLocation project in, 185, 188, 193
  UsingPicker project in, 91
  WebServices project in, 194, 197