

watchOS 2 App Development Essentials

watchOS 2 App Development Essentials – First Edition

© 2015 Neil Smyth. All Rights Reserved.

This book is provided for personal use only. Unauthorized use, reproduction and/or distribution strictly prohibited. All rights reserved.

The content of this book is provided for informational purposes only. Neither the publisher nor the author offers any warranties or representation, express or implied, with regard to the accuracy of information contained in this book, nor do they accept any liability for any loss or damage arising from any errors or omissions.

This book contains trademarked terms that are used solely for editorial purposes and to the benefit of the respective trademark owner. The terms used within this book are not intended as infringement of any trademarks.

Rev 1.0



Table of Contents

- 1. Start Here..... 1**
 - 1.1 Source Code Download..... 1
 - 1.2 Download the eBook..... 1
 - 1.3 Feedback 2
 - 1.4 Errata 2

- 2. watchOS 2 Apps – An Overview 3**
 - 2.1 What is a watchOS App? 3
 - 2.2 WatchKit App or watchOS App? 3
 - 2.3 WatchKit Apps and iOS Apps 4
 - 2.4 The watchOS SDK Frameworks 4
 - 2.5 The Key Components of a WatchKit App 5
 - 2.6 Basic WatchKit App Structure 6
 - 2.7 WatchKit App Entry Points..... 6
 - 2.8 Summary 6

- 3. Building an Example WatchKit App 9**
 - 3.1 Creating the WatchKit App Project 9
 - 3.2 Designing the iOS App User Interface 10
 - 3.3 Designing the WatchKit App Storyboard 11
 - 3.4 Running the WatchKit App..... 13
 - 3.5 Running the App on a Physical Apple Watch Device 14
 - 3.6 Setting the Scene Title and Key Color 14
 - 3.7 Adding App Icons to the Project 16
 - 3.8 Summary 18

- 4. An Overview of the WatchKit App Architecture 19**
 - 4.1 Basic WatchKit App Architecture 19
 - 4.2 WatchKit Interface Controllers 19
 - 4.3 WatchKit Action Methods..... 20
 - 4.4 WatchKit Outlets..... 21
 - 4.5 WatchKit App State Transitions 21
 - 4.6 The WatchKit Extension Delegate..... 22
 - 4.7 The Lifecycle of an Interface Controller 22
 - 4.8 WatchKit Extension Guidelines 24
 - 4.9 Summary 24

5. An Example Interactive WatchKit App	27
5.1 About the Example App	27
5.2 Creating the TipCalcApp Project	27
5.3 Adding the WatchKit App Target	27
5.4 Designing the WatchKit App User Interface	28
5.5 Reviewing the Interface Controller Class	30
5.6 Establishing Outlet Connections	31
5.7 Establishing Action Connections	34
5.8 Implementing the sliderChange Action Method	35
5.9 Implementing the calculateTip Action Method	37
5.10 Hiding the Tip Label	37
5.11 Removing the WatchKit App	38
5.12 Summary	39
6. An Overview of WatchKit Tables	41
6.1 The WatchKit Table	41
6.2 Table Row Controller	41
6.3 Row Controller Type	42
6.4 Table Row Initialization	42
6.5 Implementing a Table in a WatchKit App Scene	42
6.6 Adding the Row Controller Class to the Extension	44
6.7 Associating a Row Controller with a Row Controller Class	44
6.8 Creating Table Rows at Runtime	45
6.9 Inserting Table Rows	46
6.10 Removing Table Rows	47
6.11 Scrolling to a Specific Table Row	47
6.12 Summary	47
7. A WatchKit Table Tutorial	49
7.1 About the Table Example	49
7.2 Creating the Table Project	49
7.3 Adding the Table to the Scene	49
7.4 Creating the Row Controller Class	51
7.5 Establishing the Outlets	52
7.6 Connecting the Table Outlet	53
7.7 Creating the Data	54
7.8 Adding the Image Files to the Project	55
7.9 Testing the WatchKit App	56
7.10 Adding a Title Row to the Table	57
7.11 Connecting the Outlet and Initializing the Second Table Row	58

7.12 Summary	59
8. Implementing WatchKit Table Navigation	61
8.1 Table Navigation in WatchKit Apps	61
8.2 Performing a Scene Transition	61
8.3 Extending the TableDemoApp Project	62
8.4 Adding the Detail Scene to the Storyboard	62
8.5 Adding the Detail Interface Controller	64
8.6 Adding the Detail Data Array	65
8.7 Implementing the didSelectRow Method	66
8.8 Modifying the awakeWithContext Method	66
8.9 Adjusting the Interface Controller Insets	67
8.10 Summary	68
9. WatchKit Page-based User Interfaces and Modal Interface Controllers	69
9.1 The Elements of a Page-based WatchKit Interface	69
9.2 Associating Page Scenes	70
9.3 Managing Pages at Runtime	71
9.4 Modal Presentation of Interface Controllers	71
9.5 Modal Presentation in Code	72
9.6 Modal Presentation using Storyboard Segues	72
9.7 Passing Context Data During a Modal Segue	73
9.8 Summary	74
10. A WatchKit Page-based Interface Tutorial	75
10.1 Creating the Page Example Project	75
10.2 Adding the Image Files to the Project	75
10.3 Designing the First Interface Controller Scene	76
10.4 Adding More Interface Controllers	77
10.5 Establishing the Segues	78
10.6 Assigning Interface Controllers	79
10.7 Adding the Timer Interface Controller	79
10.8 Adding the Modal Segues	82
10.9 Configuring the Context Data	83
10.10 Configuring the Timer	84
10.11 Playing the Haptic Effect	85
10.12 Summary	86
11. Handling User Input in a WatchKit App	87
11.1 Getting User Input	87
11.2 Displaying the Text Input Controller	88

11.3 Detecting if Input is a String or NSData Object	89
11.4 Direct Dictation Input.....	89
11.5 Creating the User Input Example Project.....	89
11.6 Designing the WatchKit App Main Scene.....	90
11.7 Getting the User Input	90
11.8 Testing the Application	91
11.9 Summary	91
12. An Introduction to Watch Connectivity in watchOS 2	93
12.1 Watch Connectivity Communication Options.....	93
12.1.1 <i>Application Context Mode</i>	93
12.1.2 <i>User Information Transfer Mode</i>	93
12.1.3 <i>File Transfer Mode</i>	94
12.1.4 <i>Interactive Messaging Mode</i>	94
12.2 WatchConnectivity Session Creation	94
12.3 Obtaining Session State Information	95
12.4 The watchDirectoryURL Property	96
12.5 Sending and Receiving Application Context Data	96
12.6 Sending and Receiving User Information Data	97
12.7 Transferring Files.....	98
12.8 Sending and Receiving Interactive Messages	99
12.9 Summary	100
13. A WatchConnectivity Messaging Tutorial	101
13.1 About the Project.....	101
13.2 Creating the Project	101
13.3 Enabling Audio Background Mode.....	101
13.4 Designing the iOS App User Interface	102
13.5 Establishing Outlets and Actions.....	103
13.6 Initializing Audio Playback.....	104
13.7 Implementing the Audio Control Methods.....	106
13.8 Initializing the iOS App Watch Connectivity Session.....	106
13.9 Designing the WatchKit App Scene	107
13.10 Initializing the WatchKit App Connectivity Session.....	109
13.11 Sending the Message to the iOS app	110
13.12 Handling the Message in the iOS app	111
13.13 Testing the Application	113
13.14 Summary	113
14. An Overview of WatchKit Glances	115
14.1 WatchKit Glances	115

14.2 The Architecture of a WatchKit Glance.....	115
14.3 Adding a Glance During WatchKit App Creation.....	116
14.4 Adding a Glance to an Existing WatchKit App.....	118
14.5 WatchKit Glance Scene Layout Templates.....	120
14.6 Passing Context Data to the WatchKit App.....	121
14.7 Summary.....	121
15. A WatchKit Glance Tutorial	123
15.1 About the Glance Scene.....	123
15.2 Adding the Glance to the Project.....	123
15.3 Designing the Glance Scene Layout	126
15.4 Establishing Outlet Connections	127
15.5 Adding Data to the Glance Interface Controller	128
15.6 Storing and Retrieving the Currently Selected Table Row	129
15.7 Passing Context Data to the WatchKit App.....	130
15.8 Summary.....	132
16. A WatchKit Context Menu Tutorial	133
16.1 An Overview of WatchKit Context Menus	133
16.2 Designing Menu Item Images	134
16.3 Creating a Context Menu in Interface Builder	135
16.4 Adding and Removing Menu Items in Code.....	137
16.5 Creating the Context Menu Example Project	138
16.6 Designing the WatchKit App User Interface	138
16.7 Designing the Context Menu	139
16.8 Establishing the Action Connections.....	140
16.9 Testing the Context Menu App.....	140
16.10 Summary.....	141
17. Working with Images in WatchKit	143
17.1 Displaying Images in WatchKit Apps.....	143
17.2 Images Originating in the WatchKit Extension.....	143
17.3 Understanding Named Images	144
17.4 Adding Images to a WatchKit App	144
17.5 Compressing Large Images	146
17.6 Specifying the WKInterfaceImage Object Dimensions in Code	147
17.7 Displaying Animated Images.....	148
17.8 Template Images and Tinting.....	149
17.9 Summary.....	151
18. A WatchKit Animated Image Tutorial	153

18.1	Creating the Animation Example Project	153
18.2	Designing the Main Scene Layout	153
18.3	Adding the Animation Sequence Images	154
18.4	Creating and Starting the Animated Image	155
18.5	Summary	156
19.	WatchKit Dynamic Layout Changes and Animation	157
19.1	Changing the Position of an Interface Object	157
19.2	Changing the Size of an Interface Object	158
19.3	Setting the Visibility of an Interface Object	158
19.4	Animating Interface Changes	159
19.5	Animation and Interface Controller Lifecycle Methods	159
19.6	An Animation Example.....	159
19.7	Designing the User Interface.....	160
19.8	Performing the Layout Changes and Animation	161
19.9	Testing the Animation.....	161
19.10	Summary	161
20.	Working with Fonts and Attributed Strings in WatchKit	163
20.1	Dynamic Text and Text Style Fonts	163
20.2	Using Text Style Fonts in Code	165
20.3	Understanding Attributed Strings	165
20.4	Using System Fonts	167
20.5	Summary	169
21.	A WatchKit App Custom Font Tutorial	171
21.1	Using Custom Fonts in WatchKit.....	171
21.2	Downloading a Custom Font.....	172
21.3	Creating the Custom Font Project.....	173
21.4	Designing the WatchKit App Scene	173
21.5	Adding the Custom Font to the Project	174
21.6	Selecting Custom Fonts in Interface Builder	175
21.7	Using Custom Fonts in Code	176
21.8	Summary	178
22.	An Introduction to the WatchKit WKInterfacePicker Object	179
22.1	An Overview of the WKInterfacePicker Object	179
22.2	Adding a Picker Object to a Storyboard Scene.....	179
22.3	Understanding Picker Object Attributes	180
22.4	Understanding Picker Object Styles	180
22.5	Creating Picker Item Objects.....	181

22.6 Setting the Currently Selected Item	182
22.7 Coordinating Animations	182
22.8 Requesting Focus for the Picker Object	182
22.9 Enabling and Disabling a Picker Object	182
22.10 Responding to Picker Changes	182
22.11 Summary	183
23. A WatchKit Picker Tutorial	185
23.1 Creating the Picker Project	185
23.2 Designing the WatchKit App Scene	185
23.3 Implementing the Picker List Items	186
23.4 Implementing the Action Method	188
23.5 Testing the App	188
23.6 Summary	189
24. A WatchKit WKInterfacePicker Coordinated Animation Example	191
24.1 About the Coordinated Image Picker Project	191
24.2 Generating Radial Animations	192
24.3 Creating the Example Project	192
24.4 Designing the WatchKit App User Interface	192
24.5 Adding the Picker Image Sequences to the Project	193
24.6 Adding the Group Background Image Sequences to the Project	194
24.7 Implementing the Picker Animation Sequence	194
24.8 Configuring the Group Background Animation	195
24.9 Testing the App	196
24.10 Summary	196
25. Sharing Media Files Using App Groups	197
25.1 Sandboxes, Containers and User Defaults	197
25.2 Sharing Data Using App Groups	198
25.3 Adding the WatchKit App and Extension to an App Group	198
25.4 App Group File Sharing	201
25.5 Summary	202
26. Playing Movies and Audio using the WKInterfaceMovie Class	203
26.1 An Introduction to WKInterfaceMovie	203
26.2 Configuring a WKInterfaceMovie Instance	204
26.3 Directly Playing Content	205
26.4 Creating the WKInterfaceMovie Example	206
26.5 Designing the WatchKit App Scene	206
26.6 Adding the Video File to the Project	208

26.7	Configuring the Movie URL	208
26.8	Testing the App	208
26.9	Summary	209
27.	Recording and Playing Audio in a WatchKit App	211
27.1	The Audio Recording Controller	211
27.2	Launching the Audio Recording Controller	212
27.3	Using App Groups to Share Media File Access	214
27.4	The Audio Recording and Playback Tutorial	214
27.5	Designing the Main Storyboard Scene	214
27.6	Creating and Joining the App Group	215
27.7	Constructing the Save File URL	216
27.8	Implementing the Recording Code	217
27.9	Implementing the Playback Code	217
27.10	Testing the WatchKit App	218
27.11	Summary	218
28.	An Overview of ClockKit and Apple Watch Complications	219
28.1	What is a Complication?	219
28.2	Complication Families and Templates	221
28.3	The Complication Data Source	221
28.4	Complication Timeline Entry Objects	222
28.5	Complication Template Objects	222
28.6	Text Provider Classes	222
28.7	Image Provider Class	223
28.8	Creating a Timeline Entry Object	223
28.9	The Complication Data Source Delegate Methods	224
28.9.1	<i>getPlaceholderTemplateForComplication:withHandler:</i>	224
28.9.2	<i>getSupportedTimeTravelDirectionsForComplication:withHandler:</i>	225
28.9.3	<i>getTimelineStartDateForComplication:withHandler:</i>	225
28.9.4	<i>getTimelineEndDateForComplication:withHandler:</i>	225
28.9.5	<i>getCurrentTimelineEntryForComplication:withHandler:</i>	225
28.9.6	<i>getTimelineEntriesForComplication:beforeDate:limit:withHandler:</i>	225
28.9.7	<i>getTimelineEntriesForComplication:afterDate:limit:withHandler:</i>	226
28.9.8	<i>getNextRequestedUpdateDateWithHandler:handler:</i>	226
28.9.9	<i>getPrivacyBehaviorForComplication:withHandler:</i>	226
28.10	Managing Complications with the CLKComplicationServer Object	226
28.11	Summary	227
29.	A watchOS 2 ClockKit Complication Tutorial	229
29.1	About the Complication Project	229

29.2 Creating the Complication Project	229
29.3 Configuring the Supported Complication Families	230
29.4 Adding the Data and Image to the Data Source	231
29.5 Implementing the Placeholder Delegate Method	232
29.6 Configuring Travel Directions.....	233
29.7 Adding a Timeline Entry Creation Method	233
29.8 Specifying the Timeline Start and End Dates	234
29.9 Providing the Current Timeline Entry	235
29.10 Providing the Remaining Timeline Entries	236
29.11 Adding the Complication to a Clock Face.....	237
29.12 Testing the Complication	238
29.13 Summary	239
30. Supporting Different Apple Watch Display Sizes	241
30.1 Screen Size Customization Attributes	241
30.2 Working with Screen Sizes in Interface Builder	243
30.3 Identifying the Screen Size at Runtime	245
30.4 Summary	245
31. A WatchKit Map Tutorial.....	247
31.1 Creating the Example Map Project	247
31.2 Designing the WatchKit App User Interface	247
31.3 Configuring the Containing iOS App	248
31.4 Getting the Current Location	250
31.5 Adding Zooming Support	252
31.6 Summary	253
32. An Overview of Notifications in WatchKit	255
32.1 Default WatchKit Notification Handling.....	255
32.2 Creating Notification Actions	256
32.3 Inline Text Replies	258
32.4 Handling Standard Notification Actions.....	259
32.5 Handling Inline Text Reply Actions.....	260
32.6 Responding to Notifications.....	260
32.7 Custom Notifications	261
32.8 Dynamic and Static Notifications	261
32.9 Adding a Custom Notification to a WatchKit App.....	262
32.10 Configuring the Notification Category	263
32.11 Updating the Dynamic Notification Scene	264
32.12 Summary	264
33. A WatchKit Notification Tutorial	265

Start Here

33.1 About the Example Project	265
33.2 Creating the Xcode Project	265
33.3 Designing the iOS App User Interface	265
33.4 Setting the Notification	267
33.5 Adding the Notification Action	268
33.6 Implementing the <code>handleActionWithIdentifier</code> Method.....	270
33.7 Adding Notification Icons to the WatchKit App	271
33.8 Testing the Notification on the Apple Watch	272
33.9 Summary	273
34. A WatchKit Custom Notification Tutorial	275
34.1 About the WatchKit Custom Notification Example.....	275
34.2 Creating the Custom Notification Project	275
34.3 Designing the iOS App User Interface	276
34.4 Registering and Setting the Notifications	276
34.5 Configuring the Custom Notification	278
34.6 Designing the Dynamic Notification Scene	279
34.7 Configuring the <code>didReceiveLocalNotification</code> method	280
34.8 Adding the Images to the WatchKit App Bundle	281
34.9 Testing the Custom Notification	282
34.10 Summary	283
35. A WatchKit Inline Notification Text Reply Tutorial	285
35.1 Adding the Inline Reply Action.....	285
35.2 Configuring Text Input Suggestions	286
35.3 Handling the Text Input Action	287
35.4 Testing the App	288
35.5 Summary	288
Index	289

1. Start Here

Announced in September 2014, the Apple Watch family of devices is Apple’s first foray into the market of wearable technology. The introduction of this new device category was accompanied by the release of the WatchKit framework designed specifically to allow developers to build Apple Watch app extensions to accompany iPhone-based iOS apps. In June of 2015, Apple announced the introduction of watchOS 2, the second version of the operating system that runs on the Apple Watch. This new release of watchOS introduced a number of improvements to the performance of the Apple Watch and provided a wider range of options for developers creating WatchKit apps.

WatchOS 2 App Development Essentials is intended for readers with some existing experience of iOS development using Xcode and the Swift programming language. Beginning with the basics, this book provides an introduction to WatchKit apps and the watchOS 2 app development architecture before covering topics such as tables, navigation, user input handling, working with images, maps and menus.

More advanced topics are also covered throughout the book, including communication and data sharing between a WatchKit app and the parent iOS app, working with custom fonts, user interface animation, clock face complications and the design and implementation of custom notifications.

As with all the books in the “Development Essentials” series, watchOS 2 App Development Essentials takes a modular approach to the subject of app development for the Apple Watch, with each chapter covering a self-contained topic area consisting of detailed explanations, examples and step-by-step tutorials. This makes the book both an easy to follow learning aid and an excellent reference resource.

1.1 Source Code Download

The source code and Xcode project files for the examples contained in this book are available for download at:

<http://www.ebookfrenzy.com/print/watchos2/>

1.2 Download the eBook

Thank you for purchasing the print edition of this book. If you would like to download the eBook version of this book, please email proof of purchase to feedback@ebookfrenzy.com and we will provide you with a download link for the book in PDF, ePub and MOBI formats.

Start Here

1.3 Feedback

We want you to be satisfied with your purchase of this book. If you find any errors in the book, or have any comments, questions or concerns please contact us at feedback@ebookfrenzy.com.

1.4 Errata

Whilst we make every effort to ensure the accuracy of the content of this book, it is inevitable that a book covering a subject area of this size and complexity may include some errors and oversights. Any known issues with the book will be outlined together with solutions at the following URL:

<http://www.ebookfrenzy.com/errata/watchos2.html>

In the event that you find an error not listed in the errata, please let us know by emailing our technical support team at feedback@ebookfrenzy.com.

2. watchOS 2 Apps – An Overview

Before embarking on the creation of a watchOS 2 app it is important to gain a basic understanding of what an Apple Watch app consists of and, more importantly, how it fits into the existing iOS application ecosystem. Within this chapter, a high level overview of watchOS 2 apps will be provided, together with an outline of how these apps are structured and delivered to the customer.

2.1 What is a watchOS App?

watchOS is the name given to the operating system that runs on the Apple Watch device. Prior to the introduction of the Apple Watch family of devices, it was only possible to develop mobile applications for iPhone, iPad and iPod Touch devices running the iOS operating system. With the introduction of the Apple Watch, however, it is now possible for iOS developers to also create apps that run on watchOS.

In simplistic terms, watchOS apps are launched on an Apple Watch device either as the result of an action by the user or in response to some form of local or remote notification. Once launched, the watchOS app presents a user interface on the watch screen displaying information and controls with which the user can interact to perform tasks.

2.2 WatchKit App or watchOS App?

iOS apps are developed using a variety of software development kit frameworks. Although a number of frameworks are also available for developing watchOS apps, the primary framework used on watchOS is the WatchKit framework.

So far in this chapter we have referred to apps running on an Apple Watch device as watchOS apps. In actual fact, these apps are more correctly referred to as *WatchKit apps*. That being said, there seems to be little consistency in the terms used to refer to an app that runs on an Apple Watch. When reading Apple's marketing and technical documentation it is not uncommon to find Apple Watch apps referred to as watch apps, WatchKit apps or watchOS apps. For the avoidance of confusion, apps designed to run on an Apple Watch will be referred to as WatchKit apps throughout the remainder of this book.

2.3 WatchKit Apps and iOS Apps

It is important to understand that WatchKit apps are not standalone entities. A WatchKit app can only be created as an *extension* to an existing iOS app. It is not, therefore, possible to create a WatchKit app that is not bundled as part of a new or existing iOS application.

Consider, for example, an iPhone iOS application designed to provide the user with detailed weather information. Prior to the introduction of the Apple Watch, the only way for the user to access the information provided by the app would have been to pick up the iPhone, unlock the device, launch the iOS app and view the information on the iPhone display. Now that information can be made available via the user's Apple Watch device.

In order to make the information provided by the iOS app available via the user's Apple Watch, the developer of the weather app would add a WatchKit app extension to the iOS app, design a suitable user interface to display the information on the watch display and implement the logic to display the appropriate weather information and respond to any user interaction. Instead of having to launch the iOS app from the iPhone device to check the weather, the user can now launch the WatchKit app from the Apple Watch and view and interact with the information.

Clearly, the display size of an Apple Watch is considerably smaller than that of even the smallest of iPhone models. As such, a WatchKit app will typically display only a subset of the content available on the larger iPhone screen. For more detailed information, the user would still need to make use of the iOS application.

2.4 The watchOS SDK Frameworks

Running directly on the hardware of the Apple Watch is the watchOS 2 operating system. Included with the operating system is a set of frameworks that combine to make up the watchOS SDK. WatchKit apps are developed by making use of the various frameworks contained within the watchOS SDK. This can best be presented visually as outlined in the diagram shown in Figure 2-1:



Figure 2-1

The key layer for the app developer is the watchOS SDK which contains a number of different frameworks. Figure 2-2 illustrates the frameworks contained within the watchOS 2 SDK that are available for use when developing apps for watchOS:

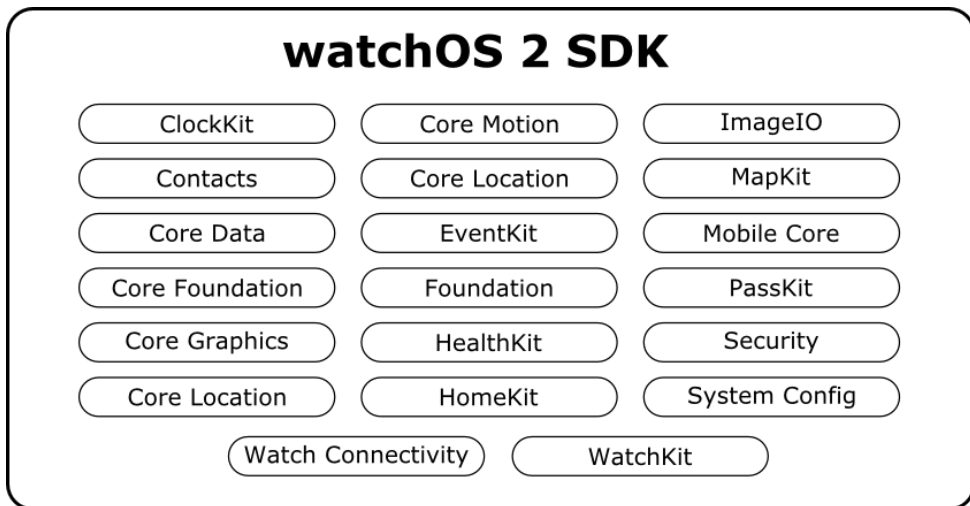


Figure 2-2

With the exception of the WatchKit and Watch Connectivity frameworks, many of these frameworks will be familiar to iOS developers, though it is important to be aware that not all of the features of a framework that are available on iOS are supported on watchOS.

2.5 The Key Components of a WatchKit App

A WatchKit app is comprised of the *Watchkit app* and a *WatchKit extension*.

Extensions are a feature introduced as part of the iOS 8 SDK release and were originally intended solely to allow certain capabilities of an application to be made available for use within other applications running on the same device. The developer of a photo editing application might, for example, have devised some unique image filtering capabilities and decide that those features would be particularly useful to users of the iOS Photos app. To achieve this, the developer would implement these features in a Photo Editing extension which would then appear as an option to users when editing an image within the Photos app. Other extension types are also available for performing document storage, creating custom keyboards and embedding information from an application into the iOS notification panel.

With the introduction of the Apple Watch and watchOS, however, the concept of extensions has now been extended to make the functionality of an iOS app available in the form of a WatchKit app.

Extensions are separate executable binaries that run independently of the corresponding iOS application. Although extensions take the form of an individual binary, they must be supplied and installed as part of an iOS

application bundle. The iOS application with which an extension is bundled is referred to as the *containing app* or *parent app*. The containing app must provide useful functionality and must not be an empty application provided solely for the purpose of delivering an extension to the user.

When an iOS application containing a WatchKit app has been installed on an iPhone device, both the WatchKit app and the corresponding WatchKit extension are subsequently transferred and installed onto the paired Apple Watch device. When the user launches a WatchKit app on a watch device, the WatchKit framework will launch the corresponding WatchKit extension before beginning the app initialization process.

2.6 Basic WatchKit App Structure

As previously outlined, the implementation of a WatchKit app is divided between the WatchKit app and WatchKit extension, both of which reside and execute on the Apple Watch device. This raises the question of how the responsibilities of providing the functionality of the WatchKit app are divided between these components. These responsibilities may be summarized as follows:

WatchKit app – Consists of the storyboard file containing the user interface and corresponding resources (such as image and configuration files).

WatchKit Extension – Contains all of the code required to provide the functionality of the WatchKit app and responding to user interaction. The extension may also contain resources such as images and media files.

2.7 WatchKit App Entry Points

There are number of different ways in which the user may enter a WatchKit app, each of which will be detailed in later chapters and can be summarized as follows:

- **Home Screen** – Once installed, the WatchKit app will be represented by an icon on the home screen of the Apple Watch display. When this icon is selected by the user the app will load and display the main user interface scene.
- **Glance** – When developing a WatchKit app, the option is available to add a *Glance* interface to the app. This is a single, non-scrollable, read-only scene that can be used to display a quick-look summary of the information normally presented by the full version of the app. Glances are accessed when the user performs an upward swiping motion on the watch display and, when tapped by the user, launch the corresponding WatchKit app.
- **Notifications** – When a notification for a WatchKit app appears on the Apple Watch device, the app will be launched when the notification is tapped.

2.8 Summary

A WatchKit app is an application designed to run on the Apple Watch family of devices. A WatchKit app cannot be a standalone application and must instead be created as an extension of an existing iOS application. The

WatchKit app is installed on the Apple Watch device and consists of a storyboard file containing the user interface of the app together with a set of resource files. The WatchKit extension is also installed on the Apple Watch device and contains all of the code logic required to implement the behavior of the WatchKit app.

3. Building an Example WatchKit App

Having outlined the basic architecture for a WatchKit app in the previous chapter, it is now time to start putting some of this knowledge to practical use through the creation of a simple example app.

The project created in this chapter will work through the creation of a basic WatchKit app that does nothing more than display a message and an image on an Apple Watch display.

3.1 Creating the WatchKit App Project

Start Xcode and, on the Welcome screen, select the *Create a new Xcode project* option. On the template screen choose the *Application* option located under *watchOS* in the left hand panel and select *iOS App with WatchKit App*. Click *Next*, set the product name to *WatchKitSample*, enter your organization name and identifier and make sure that the *Devices* menu is set to *Universal* so that the user interface will be suitable for deployment on all iPhone and iPad screen sizes. Before clicking *Next*, change the *Language* menu to *Swift* and turn off the *Include Notification Scene* option. On the final screen, choose a file system location in which to store the project files and click on the *Create* button to proceed to the main Xcode project window.

A review of the project files within the Project Navigator panel will reveal that, in addition to the iOS app target, new folders have been added for the WatchKit Extension and the WatchKit App (Figure 3-1) each of which contains the files that will need to be modified to implement the appearance and behavior of the WatchKit app:

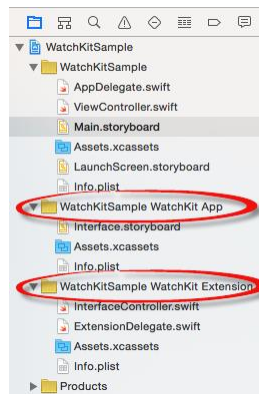


Figure 3-1

3.2 Designing the iOS App User Interface

The next step in the project is to design the user interface for the iOS app. This layout is contained within the *Main.storyboard* file and is listed in the Project Navigator panel on the left hand side of the main Xcode window. Locate and click on this file to load it into the Interface Builder environment. Once loaded, locate the Label view object in the Object Library panel and drag and drop it onto the storyboard scene. Double-click on the label and change the text so that it reads “Welcome to WatchKit” before positioning it so that it is centered in the layout canvas as illustrated in Figure 3-2:

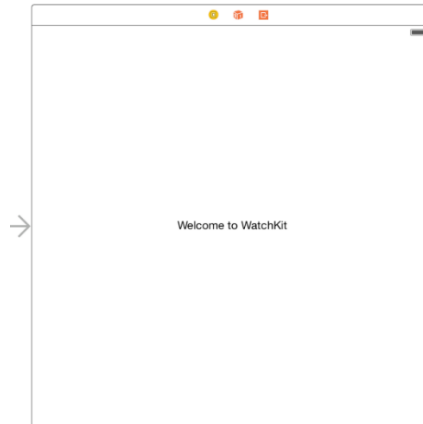


Figure 3-2

Select the new label in the layout canvas and display the *Resolve Auto Layout Issues* menu by clicking on the button in the lower right hand corner of the Interface Builder panel as indicated in Figure 3-3:

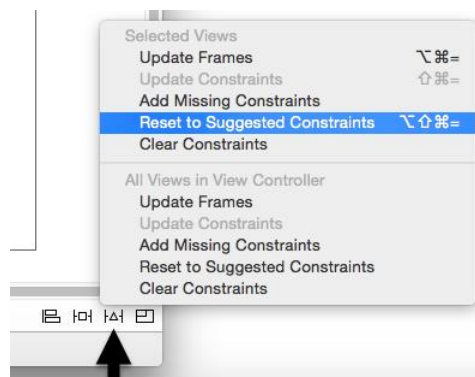


Figure 3-3

From the resulting menu, select the *Reset to Suggested Constraints* option. This will set up recommended layout constraints so that the label remains centered both horizontally and vertically within the screen regardless of whether the application is running on an iPhone or iPad display.

The user interface for the iOS application is now complete. Verify this by running the application on an iPhone device or iOS Simulator session before continuing.

3.3 Designing the WatchKit App Storyboard

The next step in the project is to design the user interface for the WatchKit app. This is contained within the *Interface.storyboard* file located under the *WatchKitSample WatchKit App* folder within the Project Navigator. Locate and select this file to load it into the Interface Builder tool where the scene will appear as illustrated in Figure 3-4:

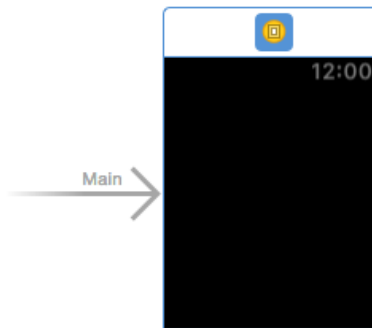


Figure 3-4

Designing the user interface for a WatchKit app involves dragging objects from the Object Library panel onto the layout canvas. When user interface objects are added to the layout canvas they are stacked vertically. These elements are then positioned at runtime by WatchKit based on the available display space combined with any sizing and positioning attributes declared during the storyboard design phase.

For the purposes of this example, the user interface will be required to display an image and a label. Locate the Image object in the Object Library panel and drag and drop it onto the scene layout. Repeat this step to position a Label object immediately beneath the Image object. Double click on the newly added Label object and change the text so that it reads “Hello WatchKit” such that the layout matches that of Figure 3-5:

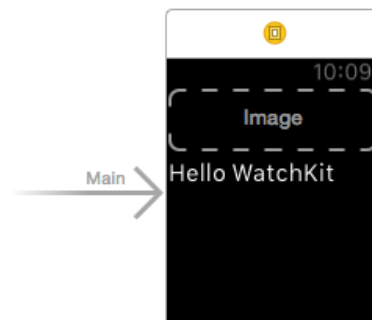


Figure 3-5

Building an Example WatchKit App

Before testing the app, some additional attributes need to be set on the objects in the user interface. The first step is to configure the Image object to display an image. Before this can be configured, however, the image file needs to be added to the project. The image file is named `watch_image@2x.png` and can be found in the `sample_images` folder of the sample code archive which can be downloaded from the following URL:

<http://www.ebookfrenzy.com/print/watchos2/index.php>

Within the Project Navigator panel, select the `Assets.xcassets` entry listed under `WatchKitSample WatchKit App` so that the asset catalog loads into the main panel. Locate the `watch_image@2x.png` image file in a Finder window and drag and drop it onto the left hand panel in the asset catalog as illustrated in Figure 3-6:

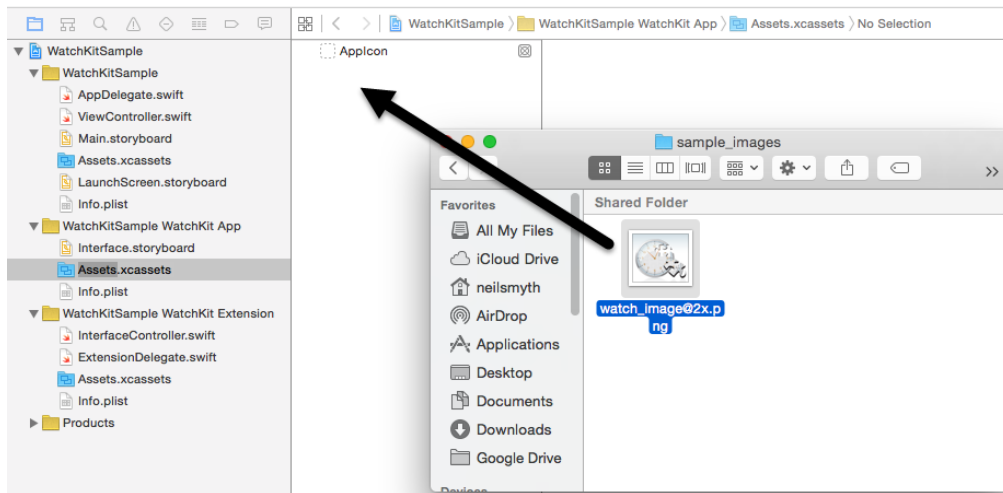


Figure 3-6

With the image file added to the project, the Image object needs to be configured to display the image when the app runs. Select the Image object in the storyboard scene and display the Attributes Inspector in the utilities panel (*View -> Utilities -> Show Attributes Inspector*). Within the inspector panel, use the drop down menu for the Image attribute to select the `watch_image` option:

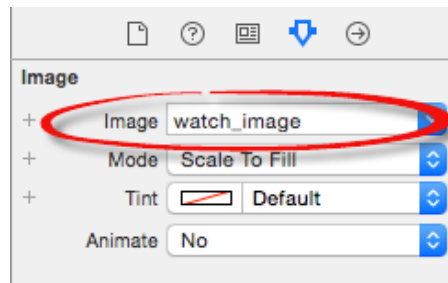


Figure 3-7

Finally, select the Label object in the scene and use the Attribute Inspector panel to change the *Alignment* attribute so that the text is centered within the label. Having set this attribute, a review of the scene will show that the text is still positioned on the left of the layout. The reason for this is that the text has been centered within the label but the Label object itself is still positioned on the left side of the display. To correct this, locate the *Alignment* section in the Attributes Inspector panel and change the *Horizontal* attribute from *Left* to *Center*. Figure 3-8 shows the Attributes Inspector panel with these attributes set:

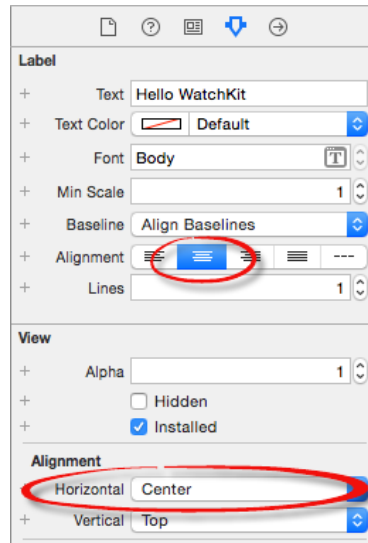


Figure 3-8

3.4 Running the WatchKit App

All that remains is to run the WatchKit app and make sure that it appears as expected. For the purposes of this example this will be performed using the simulator environment. In order to test the WatchKit app, the run target may need to be changed in the Xcode toolbar. Select the current scheme in the toolbar and use the drop down menu (Figure 3-9) to select the *WatchKitSample WatchKit App -> iPhone 6 + Apple Watch - 38mm* option:

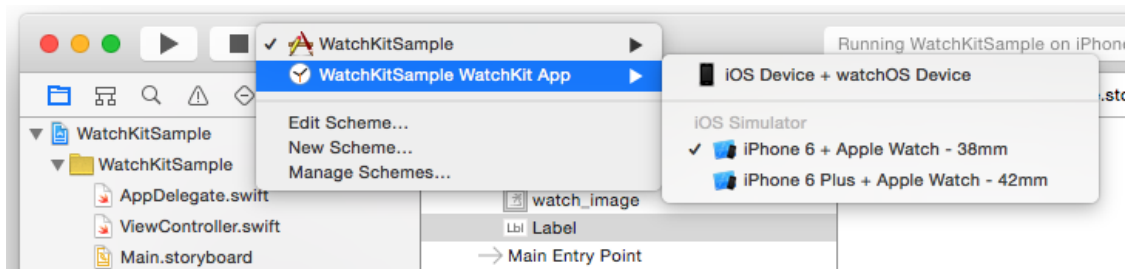


Figure 3-9

Building an Example WatchKit App

With the WatchKit app selected, click on the run button. Once the simulator has loaded, two windows should appear, one representing the iPhone 6 device and the other the Apple Watch device. After a short delay, the WatchKit app should appear on the watch simulator display as illustrated in Figure 3-10:



Figure 3-10

3.5 Running the App on a Physical Apple Watch Device

In order to test the app on a physical Apple Watch device, connect an iPhone with which an Apple Watch is paired to the development system and select it as the target device within the Xcode toolbar panel (Figure 3-11).

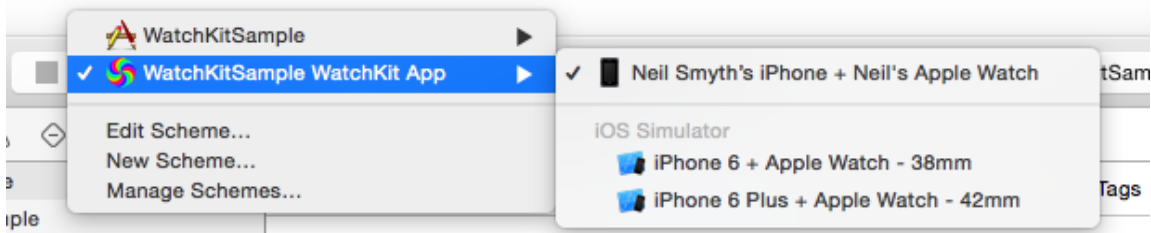


Figure 3-11

With *WatchKitSample WatchKit App* still selected as the run target, click on the run button and wait for the app icon to appear on the Apple Watch home screen and for the app to launch.

3.6 Setting the Scene Title and Key Color

The area at the top of the Apple Watch display containing the current time is the *status bar* and the area to the left of the time is available to display a title string. To set this property, click on the scene within the storyboard

so that it highlights in blue, display the Attributes Inspector panel and enter a title for the scene into the *Title* field:

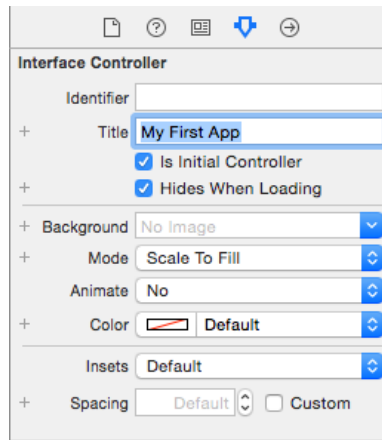


Figure 3-12

The foreground color of all of the scene titles in a WatchKit app may be configured by setting the *global tint* attribute for the storyboard file. To set this property, select the *Interface.storyboard* file in the Project Navigator panel and display the File Inspector panel (*View -> Utilities -> Show File Inspector*). Within the File Inspector panel change the color setting for the *Global Tint* attribute (Figure 3-13) to a different color.

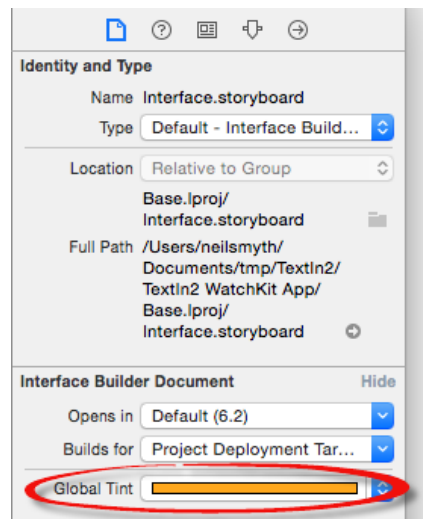


Figure 3-13

Next time the app runs, all of the titles in the scenes that make up the storyboard will be rendered using the selected foreground color.

The global tint color is also adopted by the app name when it is displayed in the short look notification panel, a topic area that will be covered in detail in the chapter entitled *An Overview of Notifications in WatchKit*.

3.7 Adding App Icons to the Project

Every WatchKit app must have associated with it an icon. This icon represents the app on the Apple Watch Home screen and identifies the app in notifications and within the iPhone-based Apple Watch app. A variety of icon sizes may need to be created depending on where the icon is displayed and the size of Apple Watch on which the app is running. The various icon size requirements are as outlined in Table 3-1:

Icon	38mm Watch	42mm Watch
Home Screen	80 x 80 pixels	80 x 80 pixels
Long Look Notification	80 x 80 pixels	88 x 88 pixels
Short Look Notification	172 x 172 pixels	196 x 196 pixels
Notification Center	48 x 48 pixels	55 x 55 pixels

Table 3-1

In addition to the icons in Table 3-1, icons are also required for the Apple Watch app on the paired iPhone device. Two versions of the icon are required for this purpose so that the icon can be represented on both iPhone (@2x) and iPhone Plus (@3x) size models:

Icon	iPhone @2x	iPhone Plus @3x
Apple Watch App	58 x 58 pixels	87 x 87 pixels

Table 3-2

Since the app created in this chapter does not make use of notifications, only Home Screen and Apple Watch app icons need to be added to the project. The topic of notification icons will be addressed in greater detail in the chapter entitled *A WatchKit Notification Tutorial*.

The home screen icon needs to be circular and 80x80 pixels in size with a 24-bit color depth. The image must be in PNG format with a file name ending with “@2x”, for example *homeicon@2x.png*.

Icons are stored in the asset catalog of the WatchKit app target. Access the image set in the asset catalog by selecting the *Assets.xcassets* file listed under the *WatchKitSample WatchKit App* folder in the project navigator panel. Within the asset catalog panel (Figure 3-14), select the *AppIcon* image set:

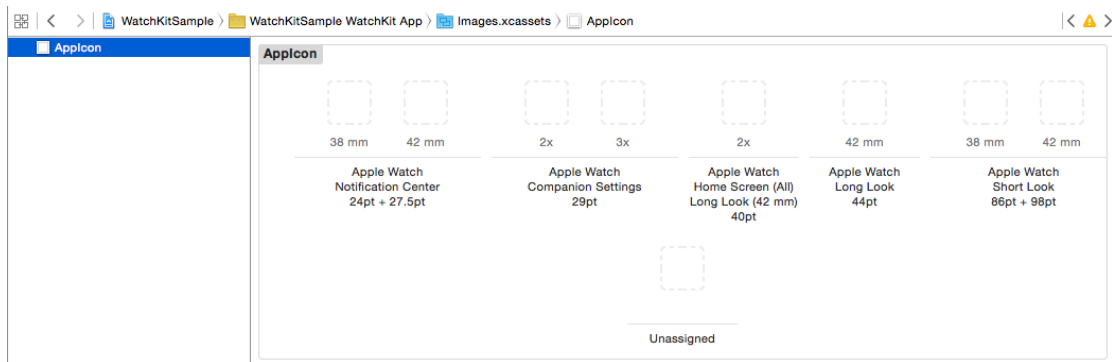


Figure 3-14

To add icons, locate them in a Finder window and drag and drop them onto the corresponding location within the image set. For the purposes of this example, app icons can be found in the *app_icons* folder of the sample code download.

Once the icons have been located, drag and drop the icon file named *Homelcon@2x.png* onto the *Apple Watch Home Screen (All)* image location within the image asset catalog as shown in Figure 3-15:

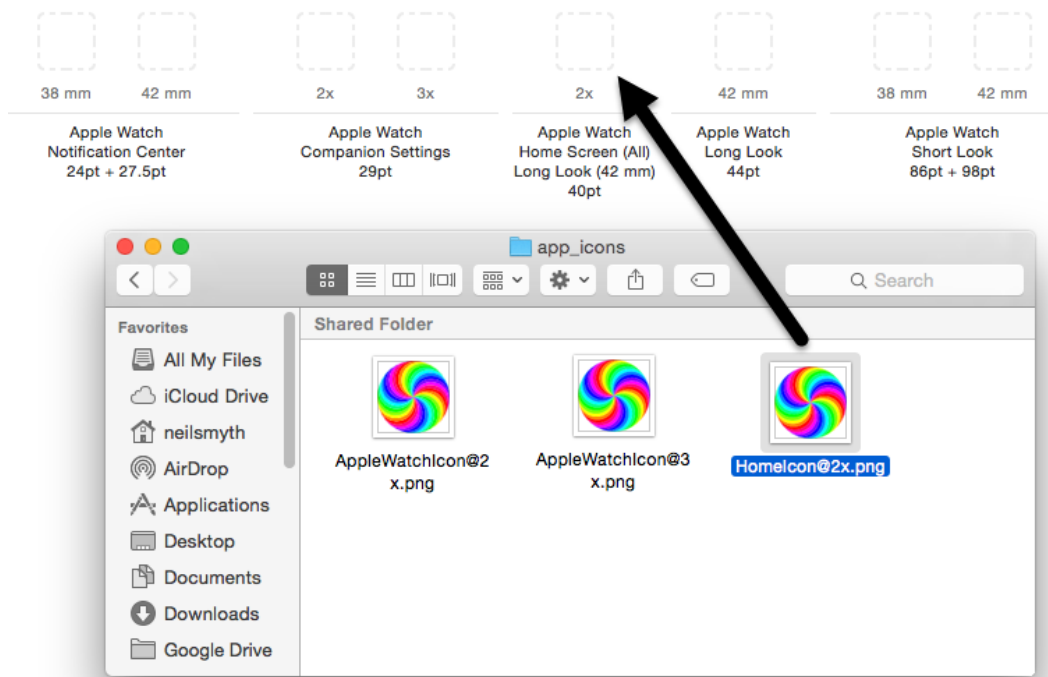


Figure 3-15

Building an Example WatchKit App

The two Apple Watch app icons are named *AppleWatchIcon@2x.png* and *AppleWatchIcon@3x.png* and should be placed in the *Apple Watch Companion Settings 2x* and *3x* image locations respectively. Once these icons have been added the three icon categories in the *AppIcon* image set should resemble Figure 3-16:



Figure 3-16

When the sample WatchKit app is now compiled and run on either a Watch simulator or a physical Apple Watch device the app will be represented on the device Home Screen by the provided icon (the home screen can be displayed on the Simulator by selecting the *Hardware -> Home* menu option).

3.8 Summary

This chapter has worked through the steps involved in creating a simple WatchKit app and running it within the simulator environment. A WatchKit app is added as a target to an existing iOS app project. When a WatchKit target is added, Xcode creates an initial storyboard for the WatchKit app user interface and the basic code for the WatchKit Extension template. The user interface for the WatchKit app is designed in the storyboard file by selecting and positioning UI objects in the Interface Builder environment and setting attributes where necessary to configure the appearance and position of the visual elements. In order to test run a WatchKit app, the appropriate run target must first be selected from the Xcode toolbar.

Before a WatchKit app can be published, app icons must be added to the image asset catalog of the WatchKit App target. These icons must meet strict requirements in terms of size and format, details of which have also been covered in this chapter.