Extracted from:

Developing for Apple Watch, Second Edition

Create Native watchOS 2 Apps with the WatchKit SDK

The Pragmatic Bookshelf

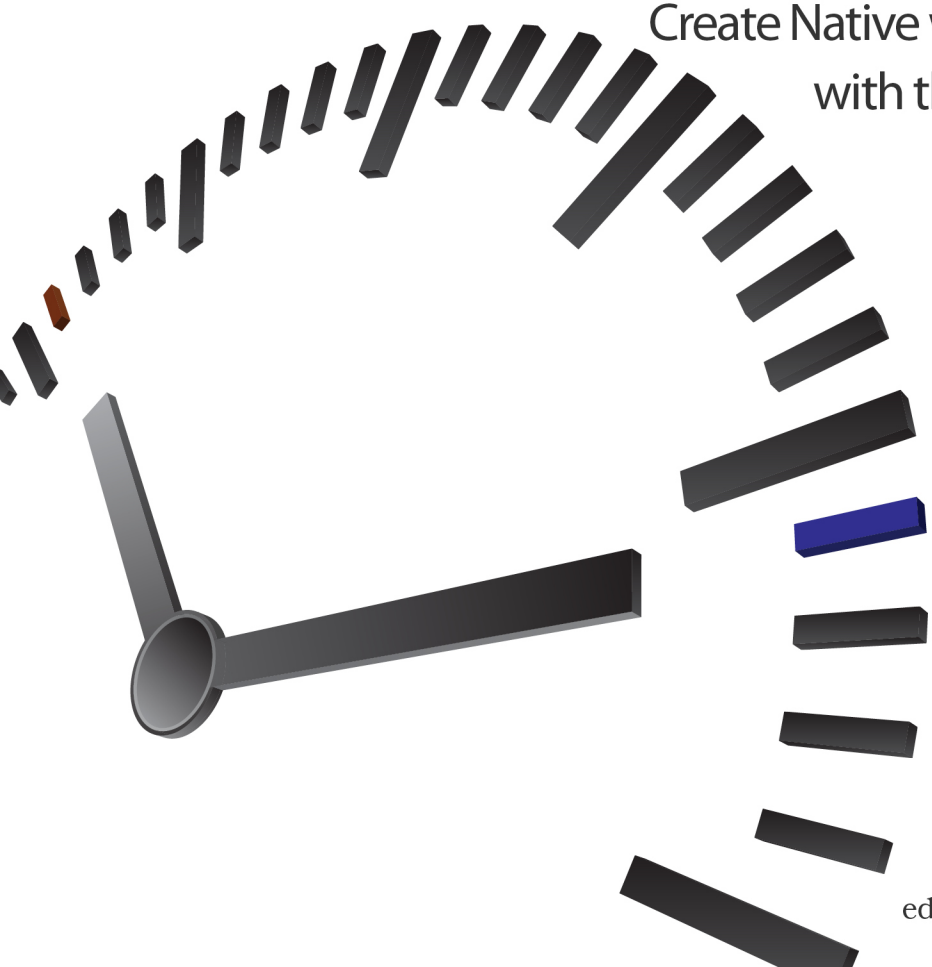
Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Second Edition

Developing for Apple Watch

Create Native watchOS 2 Apps with the WatchKit SDK



Jeff Kelley

edited by Rebecca Gulick

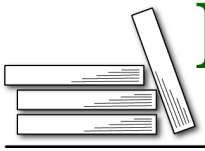
Developing for Apple Watch, Second Edition

Create Native watchOS 2 Apps with the WatchKit SDK

Jeff Kelley

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *https://pragprog.com*.

The team that produced this book includes:

Rebecca Gulick (editor)
Potomac Indexing, LLC (index)
Linda Recktenwald (copyedit)
Gilson Graphics (layout)
Janet Furlow (producer)

For customer support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

Printed in the United States of America.
ISBN-13: 978-1-68050-133-9
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—May 2016

CHAPTER 1

An Overview of Apple Watch

Why a whole chapter on the watch itself? I'm not trying to sell you one. But by understanding more about the device, you'll be able to make smart decisions about the types of apps you should be writing in the first place. In this chapter you'll learn about the fundamental design elements of apps that run on the watch and what your users expect of them. Maybe you're coming to this book thinking you want to make an app for the watch but aren't sure what kind; this chapter will help you brainstorm your app idea. By the end you'll be able to clearly define what the watch can do and, more importantly, what your app inside the watch *should* do.

Apple Watch Basics

Every Apple Watch *must* be paired with an iPhone 5 or newer in order for you to install apps. While a cynic might see this as a sneaky way for Apple to double up on device sales, there's a very good reason: every watchOS app needs to be bundled inside an iOS app that's installed on the iPhone. With the limited screen real estate of the watch, Apple can't exactly have an independent App Store on the device; instead, you'll use the iPhone's Apple Watch app to install new apps.

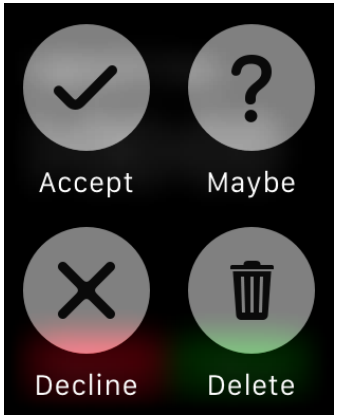
To know what kinds of apps you can make, it's important to understand the design language of Apple Watch apps. The building blocks you'll use to make the apps are extremely different from the components available to you on iOS. You'll find an entire chapter devoted to the interface components you can use, but a brief overview of what watch apps *look like* is a good place to start brainstorming ideas. Let's begin there.

Apple Watch App-Design Concepts

Much like an iPhone app, you can think of the content of your Apple Watch app one screen at a time, but, of course, the watch screen is much smaller. Here you see an example of a screen you might design for a stopwatch feature.

Every screen on Apple Watch automatically scrolls vertically to fit content that exceeds the screen size; the user can either pan with a finger or turn the *Digital Crown*, a grooved button on the side of the watch that rotates to accommodate scrolling and other user input, much like a crown on a traditional wristwatch. Each screen can have a title at the top, tinted to match your app's main color. A stark black background is common, though you *can* change it. Then there are two labels, followed by some buttons. Because of the small screen size, buttons take up a large portion of the screen. You may notice that they go edge to edge, and that's intentional. The bezel of the device adds a black border to every screen, and the margin on the edges is built in, so your designs should stretch as far as possible. The background color of the buttons helps them stand out—bold colors look great on the watch screen.

This is just one screen, and your app will have many. Organizing them is simple, as we'll discuss in a later chapter about transitioning between them. The most common way to have many screens in one app is to use a paged layout. Page-indicator dots at the bottom will inform your users that they can swipe between screens, giving them quick access to all of your features.



When the user presses the screen a bit harder than with a normal tap (called a *Force Touch* in Apple parlance), you can display a contextual menu with additional actions. This is a good place for less-common actions that you want to make available in your app but that you don't necessarily want to take up screen space for. They look like the screen shown here.

One thing you'll notice is that the colors of the app below show through. This transparency (a key element of iOS 7's redesign) is apparent in

Apple Watch apps as well, giving the user a sense of spatial awareness as your app moves from screen to screen.

As you'll see, you can implement many user-interface elements (including images) to create amazing interfaces on these small screens.

From iPhone App to Apple Watch App

When the iPhone SDK came out in 2008, developers struggled to find the right balance. They had Mac apps with tons of great features, but not all of those features made sense on iPhones of the time. Apple gave some guidance, suggesting to developers that they aggressively pare down their apps' feature sets in the transition from OS X to iOS. As time went on, iOS apps became more full-featured than those initial apps, especially with the introduction of the iPad, but OS X apps are still, by nature, more complex.

The Apple Watch and its WatchKit SDK bring us to another such point in time. Not every single feature makes sense to bring from your iPhone app to its WatchKit extension, and some just flat out don't belong. If your app is an RSS reader, for instance, your users aren't going to be reading long-form think pieces on their wrists during their morning commute; they'll just pull out their iPhones for that. That doesn't mean that an RSS reader shouldn't have a WatchKit extension, though—users might want to skim headlines on their watches, saving interesting ones to their reading lists or marking uninteresting articles as read. Other app types might want to bring every feature to the watch if every feature makes sense. So the question is how you pare down the features of an existing iPhone app to those that make sense on the watch.

Finding the Right Features

Think about your app in the context of the watch. Do you have features that would excel on a smaller screen? Quick, easy-to-navigate pieces of information are key here. If your app does light messaging, an inbox is a perfect addition to it. If your app is an RSS reader, on the other hand, full-length feeds aren't as compelling. Focus your effort on the parts of your app that can save your users time—that's why they bought the watch in the first place.

Often, it's a good idea to incorporate only the best-fitting part of your app's features in the watch app. If your app allows its users to change the temperature in their homes as well as schedule recurring temperature changes, consider showing only the controls for adjusting the current temperature.

Since you know users can do more advanced changes on their iPhones, it's a great way to focus and make a more streamlined interface.

After the watch's announcement, you may have started to consider using the device's hardware capabilities. It has a microphone for Siri, an accelerometer, and a heart rate sensor. While the first version of WatchKit didn't include access to any of this hardware, apps built for watchOS 2 can take full advantage of these hardware features and use them to create more engaging apps. You're not limited to simple taps and finger gestures on the watch, so get creative!