Extracted from:

Developing for Apple Watch

Your App on Their Wrists

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Pragmatic exPress

Developing for Apple Watch

Your App on Their Wrists

Jeff Kelley

edited by Rebecca Gulick

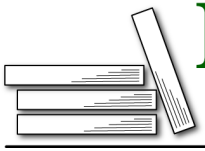
Developing for Apple Watch

Your App on Their Wrists

Jeff Kelley

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina



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)
Candace Cunningham (copyeditor)
Dave Thomas (typesetter)
Janet Furlow (producer)
Ellie Callahan (support)

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

Printed in the United States of America.
ISBN-13: 978-1-68050-068-4
Encoded using the finest acid-free high-entropy binary digits.
Book version: P2.0—July 2015

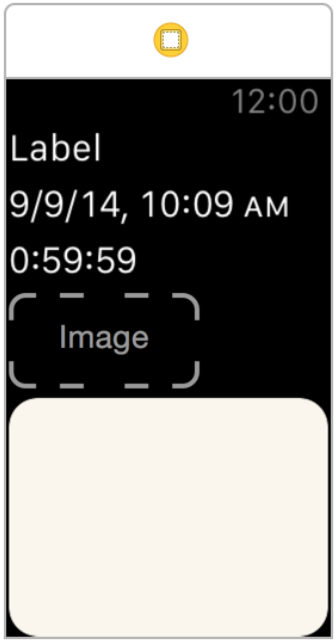
Meet the Interface Objects

If this were a book on iOS in general and this chapter were the chapter on UIView, it would be insane to begin with a list of all of the UIView subclasses. Lucky for us, this is a book on WatchKit, and there are only 11 subclasses of WKInterfaceObject to deal with. Expect this number to grow as WatchKit matures, much like the iPhone SDK matured as it became the iOS SDK. I'll divide the interface objects into three categories: objects that display data; objects that receive user input; and objects used for layout. Let's meet our interface objects!

Objects That Display Data

We've already seen one of these interface objects: WKInterfaceLabel, which we used to display "Hello, Watch!" to our users. Labels act much like UILabel in UIKit, and you can call setText() or setAttributedText() to change its contents, just like in UIKit. The other objects in this category include WKInterfaceTimer and WKInterfaceDate, which help your watch app tell time. A timer object counts down to a given moment in time, represented by an NSDate, using a format you specify.

The format must be specified in Interface Builder, but once you have created your timer, you can change it by calling setDate() on it. Using its start() and stop() methods, you can control whether your WKInterfaceTimer is updating, though it always counts down to the same date, regardless of you stopping it. Finally, WKInterfaceDate displays the current date and/or time to the user in a label. Like a timer, you must set up your date formatting in Interface Builder. Once you've created a date object, you can use setTimeZone() and setCalendar() to adjust the display of the date. There are two more interface objects in this category, and you can see them all in this image.



Another important object is WKInterfaceImage, which acts like UIImageView on iOS, displaying images to the user. Image objects support displaying static and animated images with setImage(), setImageData(), and setImageNamed()

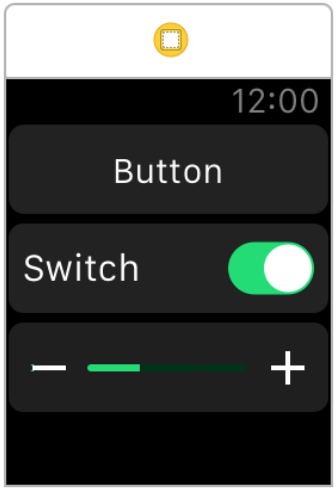
methods, and if you supply template images, you can provide a tint color using setTintColor(). Images in your WatchKit app can come from your watch app's asset catalog or your WatchKit extension, and there are significant performance differences between the two, which we'll discuss later in this book, on page ?.

The last interface object you can use to display data to the user is WKInterfaceMap. Like its MKMapView counterpart, the map object displays a map using Apple's mapping service. You can add your own pins to the map using its addAnnotation(_:withPinColor:) method, allowing you to show locations to your users. If you have a custom image to use instead of the default colored pins, you can use addAnnotation(_:withImage:centerOffset:) or addAnnotation(_:withImageNamed:centerOffset:), providing an offset in case you need to move the pin image relative to the map location (normally the pin image is centered over the location). You'll want to position the map in the proper location so users see your pins, which you'll do with setVisibleMapRect() or setRegion(). These methods take the same MapKit types as MKMapView.

Interacting with Maps

There's one caveat to using WKInterfaceMap: once you position the map, the user can't interact with it. There's no zooming, panning, or selecting of pins. Scrolling the watch's Digital Crown will only scroll your interface controller's content, if there's enough to scroll. When the user taps the map, the watch's Maps app opens. The center of the visible location of your map is a pin in the user's Maps app. This allows your users to get directions or anything else they'd be able to do in the Maps app normally. If you want to display a specific location to users, set that location as the center of your map; in the Maps app on the watch, the users will be able to get directions or anything else they'd normally do with a map location.

Objects That Receive User Input



On iOS a middleman—UIControl—sits between UIView and objects that are meant for user interaction. Although there's no equivalent intermediate class in WatchKit, some interface objects embody the spirit behind controls on iOS. We've already seen WKInterfaceButton, which allows us to call a method when the user taps the button. The method it calls, unlike UIButton, must be defined in our storyboard. Aside from buttons, there are switches and sliders: WKInterfaceSwitch and WKInterfaceSlider, respectively. They mostly act like UISwitch and UISlider, with a few differences. Visually, a WKInterfaceSwitch looks a lot like a WKInterfaceButton, with a title label in front of the background, but there's also a switch control on the right side. Sliders have a similar appearance, with incrementing and decrementing buttons on either side and the main slider in the middle. You can see all of these objects in this image.

Sliders and switches, just like buttons, require you to set up the method they call in your storyboard. The method you write for a switch must take a Bool parameter for the switch's value, and a slider's method must take a Float. Both are called every time the user adjusts the value. For sliders you have some additional control: in the storyboard you can set the minimum and maximum value for the slider, and either in the storyboard or by calling setNumberOfSteps() you can set the number of steps between those values. If you wanted to make a slider that could select any integral value between 0 and 5, you'd set the minimum to 0, the maximum to 5, and the number of steps to 6.

Objects Used for Layout

The final category of interface objects is an interesting one; these objects exist to help out with WatchKit's unique layout system. The most important of these is WKInterfaceGroup, which is so vital to the watch UI that it has its own chapter, on page ?. In short, a group can contain other objects, allowing you more freedom in how they're laid out. Groups can even contain other groups, allowing you to create complex hierarchies of layout. Along with groups comes WKInterfaceSeparator, which is by far the simplest of all interface objects—it's basically a line drawn between other objects to separate them, and it has but one method for you to call, setColor().

Our final interface object that helps with layout is WKInterfaceTable. Since every interface object you use must be created in your storyboard, using a table is one of the only ways to get dynamic content into your app. Each row in the table is actually a group, so you can lay out whatever objects you need in each row. We'll cover tables more in their own chapter, on page ?.