Building Apple Watch Projects

With Apple's eagerly anticipated entry into the wearable arena, the field is wide open for a new era of app development. The Apple Watch is one of the most important technologies of our times.

This easy-to-understand book takes beginners on a delightful journey on which they will discover the features available to the developer, right up to the completion of medium-level projects ready for App Store submission. It provides the fastest way to develop real-world apps for the Apple Watch by teaching you the concepts of Watch UI, visual haptic and audio, message and data exchange between watch and phone, web communication, and finally visual, haptic, and audio feedback for users.

By the end of this book, you will have developed at least four fully functioning apps for deployment on watchOS 2.

Who this book is written for

If you have some basic knowledge of programming in Swift and are looking for the best way to get started with Apple Watch development, this book is the right one for you!

What you will learn from this book

- Understand the concept of the Apple Watch as an autonomous device as well as it being paired with the iPhone
- Get your app up and running
- Design exciting, inspiring, and attractive layouts for your apps
- Make your user interface more engaging using images and animation
- Enable your Watch and iPhone apps to transport and share data
- Leverage the feature-rich set of WatchKit technologies provided by Apple
- Connect your apps to the Internet
- Submit your app to the App Store

Who this book is written for

If you have some basic knowledge of programming in Swift and are looking for the best way to get started with Apple Watch development, this book is the right one for you!

What you will learn from this book

- Understand the concept of the Apple Watch as an autonomous device as well as it being paired with the iPhone
- Get your app up and running
- Design exciting, inspiring, and attractive layouts for your apps
- Make your user interface more engaging using images and animation
- Enable your Watch and iPhone apps to transport and share data
- Leverage the feature-rich set of WatchKit technologies provided by Apple
- Connect your apps to the Internet
- Submit your app to the App Store


In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Exploring the New Platform'
- A synopsis of the book’s content
- More information on Building Apple Watch Projects
Stuart Grimshaw has programmed for Apple computers since the days before OS X and has been involved with developing for Apple Watch since its release. Born in the UK and having lived in Germany and the Netherlands, he is currently an iOS developer in Auckland, New Zealand, where he works on some of Australia–New Zealand's largest video and TV delivery apps and heads the research and development of both watchOS and tvOS applications. He is passionate about the potential of the Apple Watch and Apple TV as well as Apple's Swift programming language and is a keen proponent of beach coding.
Building Apple Watch Projects provides you with end-to-end guidance on creating a range of apps for the Apple Watch, covering such essential topics as location frameworks, data storage, communication with the iPhone, and animation, to name just a few. It applies the reader's basic Swift knowledge to real-world programming challenges in an easy, step-by-step manner, starting with a simple animated version of the ubiquitous Hello World app, progressing to apps that are Internet connected, location-aware, and fascinating to use, spanning the genres of productivity, games, and lifestyle apps.

The book contains many tips around making the best use of your coding skills, the tools that surround app development, and the many resources and utilities that exist to make your progress as a developer as smooth and enjoyable as it can be.

By the end of this book, you will have taken apps from the earliest conceptual stages right up to the Store submission.

What this book covers

Chapter 1, Exploring the New Platform, introduces the reader to the Apple Watch itself and the many design features that set the context in which the rest of the book is presented.

Chapter 2, Hello Watch, covers the setup of a new Xcode project and the creation of an uncomplicated but attractive take on the Hello World theme, including a little animation eye candy.

Chapter 3, C-Quence – A Memory Game, presents a minimalist version of a common memory game and covers the planning and design of an app that will involve more than one screen.
Preface

Chapter 4, Expanding on C-Quence, builds on the preparations of Chapter 3 to complete a colorful and functional working app, which will also be able to communicate with its paired iPhone to gather textual input from the user.

Chapter 5, On Q – A Productivity App, makes use of image assets to add a little sophistication to the user interface of a cue-card app, introduces an interactive menu screen, and makes use of the Taptic Engine to provide feedback to the user.

Chapter 6, Watching the Weather, adds Internet connectivity into the mix to create a weather app that fetches its live data from the Web and presents it using a table-based interface. This chapter also introduces the Glance screen, making a portion of the data available to the user without launching the app.

Chapter 7, Plot Buddy – All about Location, presents a location-aware app, with which the user can store sets of location data, with or without its paired iPhone. We also see the introduction of Swift protocols and custom initialization methods.

Chapter 8, Images, Animation, and Sound, adds icons to the app as well as introducing sequential animations and audio/video media playback. We also look at configuring an app’s UI almost purely in code, in addition to class extensions and Xcode asset catalogs.

Chapter 9, Wear It, Test It, Tweak It, Ship It, is all about the steps to be taken after coding is finished, including installation and testing on a physical device as well as preparation of everything that is necessary for submission to the App Store.

Chapter 10, This Is Only the Beginning, takes a look at some more advanced techniques to make your apps stand out from the crowd, introduces some techniques for improving your workflow, and covers a range of topics that any watchOS/iOS developer will want to add to his or her programming skills, including a number of peripheral tools essential to professional work in a team environment, in order to set the stage for the reader's progress beyond the ground covered by this book.
Exploring the New Platform

There couldn't have been a more exciting time to be a software developer. A bold claim, undeniably, but if we take a look at the seemingly limitless range of contexts in which we have come to use smart mobile technology for both work and play, it is hard to imagine a more fertile environment in which employ both our creative and technical skills in shaping the next generation of mobile devices.

Wide open future

It is a rare moment indeed in which developers have the opportunity to use a new programming language, for developing on a new platform, for a new genre of device. Whether relatively new to programming, or with decades of experience across a multitude of platforms and languages, we are, in a sense, all very much beginners, and it is this that many will find the most thrilling part of engaging with the Apple Watch as a developer. We are all in at the ground floor, so to speak, and none of us knows where it will lead us, what users will expect from wearable devices as they become established as mainstream products, what previously unimagined uses will evolve, and what challenges we will face as developers.

As someone who is developing for the Apple Watch, you are truly at the center of this digital revolution. The company that revolutionized our attitudes to computing in general, and mobile devices in particular, is revolutionizing both its hardware and the ways in which developers are able to engage their users, offering them an ever more immersive experience, while the boundaries between hardware and software become increasingly blurred. More than any device before it, the Apple Watch blends into this new landscape, at once a small part in the larger context of mobile computing, and a radical step forward, into a realm of wearable devices that accompany us throughout the day in closest possible proximity.
So before we get down to any coding, we will take a brief look over the concepts, an important undertaking, as we need to understand what users will expect from a device that in some respects resembles others they know already, in order to delight them with things they have never experienced.

In this chapter, we will cover the following

- A closer look at the watch
- A look under the hood
- One app, four interfaces
- User input hardware
- The Watch as extension of the iPhone

A closer look at the watch

Just as the iPhone presented a revolutionary step in user interface design, so the Watch brings with it a way of thinking about interacting with the user that goes beyond the simple shrinking down of screen content to fit the new device and instead gives us a reimagining of how we both read and input information, while effectively leveraging our previous experience and expectations of using a touchscreen.

The following image shows the Watch's home screen and its older sibling on the iPhone (not to scale):

![Watch's home screen and iPhone home screen](image-url)
And as the platform matures there will be endless opportunities, even necessity, for developers to think further out of the box and offer users new experiences and ways to respond to apps that they are just beginning to discover.

**Building on success**

The arrival of the Apple Watch took its time, but in 2015 it was time for us to become familiar with a new kind of device, for which the user experience had once again made a complete rethink of the interface necessary. The screen was too small for conventional icons above conventional labels, too small for any kind of menus accessible from a status bar, and generally too small for anything approaching a simple migration of UI elements from the touchscreen of the iPhone. In terms of size, the closest we had was the iPod Nano, but even that 2.5 inch screen allowed a much more conventional interaction with the user than was the case with the Watch.

And once again, the Cupertino design gurus did not disappoint. The home screen, while being recognizably the offspring of the iPhone home Screen, had morphed into a rearrangeable matrix (or *Carousel*) of circular icons that resized as they approached the center of the screen, icons that for the first time had to say all they had to say without the use of a text label, or, it should be noted, without recourse to text within the icon itself — the calendar icon's weekday label really is at the outer edge of readability and should be viewed more as a very neat bit of decoration rather than an attempt to add meaningful textual information to the icon.

A look at the calendar icon on this home screen will, perhaps, make the point better than words:

![Calendar Icon](image)

In addition to the usual gestures of iOS touchscreens, the Watch's touchscreen gained what Apple dubbed the **Force Touch**, adding a layer of input over the familiar one without adding any extra visual elements, the use of which is accompanied by a satisfying tapping sensation produced by the **Taptic Engine**.
Ah yes, the Taptic Engine. Mobile phones have been buzzing away in our pockets and on our table tops for a long time now, with varying degrees of urgency (or sometimes irritation), and other companies' watch offerings have adopted that particular method of notification, as one would expect they would, though with little change in approach. But the Apple Watch—and this really isn't sales hyperbole—takes haptic feedback to a whole new level. The variety, subtlety, and outright aesthetic quality of the Taptic Engine, as Apple has called its tiny stroke of genius, literally has to be experienced to be understood.

So one begins to get an idea of how much we, as Apple Watch developers, have our work cut out for us in terms of rethinking how we will both present information to, and get information from our users, and how their experience in working (and playing) with our apps will be very different to what they, and we, are accustomed to.

A look under the hood

It is about time we started getting a little technical. We are, after all software developers, and tempting as it may be to look at the many aesthetic qualities of the Watch, we are here to roll up our sleeves and get our hands dirty, and devote ourselves to some technical specifications of the device that will have a direct impact on our work. And the web really has no shortage of images, and articles about, the various and sometimes very luxurious materials and styling that, whether we like to admit it among ourselves or not, are as important to potential early adopting watch owners as all the hard work we do as developers.

So let's start with answering a few questions around the environment in which we will (very soon) be creating our software.

• What hardware are we dealing with, exactly?
• What is watchOS 2?
• What is WatchKit?

We'll start by taking a look some features of the hardware that are relevant to the developer.

Download the example code

You can download the example code files for all Packt books you have purchased from your account at http://www.packtpub.com. If you purchased this book elsewhere, you can visit http://www.packtpub.com/support and register to have the files e-mailed directly to you.
Vital statistics

The Apple Watch comes in two sizes, referred to by the height of the casing, 38mm and 42mm.

The 38mm model’s casing is 33.3mm wide, its height is actually 38.6mm, and houses a display measuring 21.11mm by 26.52mm.

The 42mm model’s width is 35.9mm, and has a display measuring 24.34mm by 30.42mm.

Both models’ casings are 10.5mm deep.

Onto this tiny canvas, Apple have managed to squeeze an impressively sharp and vibrantly colored, pressure sensitive touchscreen, with resolutions of 272 × 340 px on the smaller model, and 312 × 390 px on the larger, giving 290 pixels per inch (PPI)—also often referred to as dots per inch (DPI)—enough resolution to satisfy even the most fastidious graphics geek.

The screens of the least expensive model, the Sport model, are, according to Apple:

…protected by a lightweight aluminosilicate glass that’s especially resistant to scratches and impact.

Hands up anyone who can tell us about aluminosilicate glass?

All other models are equipped with Sapphire screens; this sounds a little less technical but it is the second hardest transparent material on earth after diamonds. So don’t store one in your pocket together with any diamond jewelry.

The 18 hours of battery life claimed by Apple has proved to be a perfectly reasonable estimate, assuming the pattern of engagement that most of us are anticipating at the moment, and even under heavy use and testing during development, the watch is able to go a full working day without a recharge. The user can, when the remaining battery charge becomes perilously low, activate a low power mode that will restrict the watch to telling the time, and your app stays shut out in the cold. We will, of course, later have a look at what power conservation considerations you will want to be including when designing your own apps to avoid this situation.

Apple does the heavy lifting around designing and coding for a single device, one that just happens to come in two sizes, so you will only be working on a single UI design, and only writing one set of source code.

Just to round off the vital statistics, the Watch has 512MB of RAM and 8GB of storage, but the companion app on the iPhone (and there always is one) will store most of your larger resources, images and video, for example, if you have any.
watchOS

Based on and in many respects similar to iOS, watchOS is the operating system that powers the Watch. It is orientated towards conserving power and the economic use of resources in memory, while offering a rich and engaging experience for the user, and most developers will find leveraging their previous iOS programming experience easy and intuitive, while adapting to the optimizations of many common programming tools and techniques. You will grow accustomed to alternate ways of providing information to the user, for example using tables to display lists, which, while visually similar to those on an iPhone, are handled quite differently at a code level to tables in iOS.

watchOS was revealed to the world along with the Watch in April 2015, and was upgraded to watchOS 2 in the following October. From our perspective as developers, version 2.0 was a huge leap forward, not only as the Watch gained the ability to function away from the iPhone, significantly increasing its scope of use, but also because, just as critically, we gained access to many of the features of watchOS and its hardware that were hitherto reserved for Apple’s own apps, most notably the Taptic Engine, Digital Crown, accelerometer, heart rate sensor, speaker, and microphone, and the addition of custom Complications (some of which we will be looking at shortly).

WatchKit

WatchKit is the Apple framework that will provide you with the technologies that you will use to create your apps. Think of WatchKit as a vast library of code that all developers will need, code that has been written, tested, and optimized for you by the very people that built both the hardware and its operating system, ensuring a maximum level of uniformity, compatibility, code safety, security, and convenience for the developer. There is, to take one example, a massive amount of work for your app involved in redrawing a button when its title changes, work that will be almost identical every time that button changes its appearance in any way, but you as a developer will only write one line of code to get all that boiler plate set up.

This you will likely know already from UIKit, with which iOS apps spring to life.
One App, four interfaces

Your users have a number of options when engaging with your app, or better said, your app and the user have a number of ways to engage with each other, both parties being able to initiate that engagement:

- The main app
- Glances
- Complications
- Notifications

The Main app

The main app is the only part of this quartet that is always present in the bundle that gets installed on the watch by the phone app, the other three are optional, and whether to include them or not will depend entirely on what makes sense for your app. This itself adds a new dimension to your app design, as information can be spread across these various channels, partly duplicated, or some combination of both. While the scope and contents of the main app are directly comparable to the equivalent iOS apps, it is worth taking a brief look at what the other three do, how they do it, and what likely use cases they will be associated with.

Glances

As the name might suggest, the Glance is meant as a brief, and one-way, encounter. The user is presented with a single screen of information, and is offered no interaction other than a single tap, which will take her to the main app. It is an interaction initiated by the user, swiping up from the clock screen. In the images below, you can see that the Glance (left) presents information in a format that is easy to read quickly, containing only essential information, while the main app presents more than just the most immediate data, but requires more than just a casual glance (hence the name) from the user:
Depending on the nature of your app, you may find your users consulting this screen more often than your main app, so it pays to regard the Glance as more than just a stripped down extension of your main app and to give it a level of attention commensurate with its importance. Those who enjoy rising to the challenge of doing a lot with limited resources will likely find this a most pleasing area of design and implementation.

**Correction**

It might seem strange to give the name Complication to a UI element, particularly given the emphasis Apple give to the simplicity of using the Watch, but the term actually stems from the tradition of watch-making, and refers to any information, typically dates and weekdays, presented by a watch beyond simply the time of day.

We can see in these images how some watch faces offer different numbers of Complications:

![Complications](image)

A Complication, as implemented on the Apple Watch, is your chance to present a very small amount of information on the watch face itself, if the user has selected your app from a list of those that offer information in this format. If implemented by yourself and activated by the user, this is also the most direct route to your app, since it involves just a solitary tap on the complication to launch your main app.

**Notifications**

This is the one interface that can initiate an exchange between the user and your app. Modelled on and similar in many ways to what we already know from iOS Notifications, your app can draw attention to itself and present to the users a number of options, including launching your main app, or choosing from a set of actions to be undertaken.
The Notification view (left) can be a scrollable view (center) and may offer the user the chance to choose actions, tapping the app icon launches the main app (right).

Once again, depending on your app's functionality, this may be the interface that your user sees and touches the most. Apple's own Messages app is such a case, in which a user may read all of his messages as they come in, but only react to a limited number of them.

**User Input Hardware**

Almost every user will be familiar with swiping and tapping a touch screen as ways of interacting with smartphones and tablets, and the Apple Watch inherits those features in full, but as we have seen already, beginning with watchOS 2 the developer now has access to a number of hardware features built into the device. This is made all the more exciting by the nature of some of these new features, which don't really have any precedent among the devices that preceded the Watch. They include:

- Digital Crown
- Force Touch
- Taptic Engine
- Audio In Out
- Accelerometer

We will not use all of the Watch's hardware features in this book, but in designing an app it makes sense for us to be aware of all the tools at our disposal.
The reader is encouraged to read through, at least once, the Apple Watch Programming Guide, to gain a basic overview of the options available for engaging in a meaningful, original, stylish, and fun way with the user. https://developer.apple.com/library/ios/documentation/General/Conceptual/WatchKitProgrammingGuide/ Much of this stuff really is too good to miss.

**Digital Crown**
Possibly more thanks to its snappy name than anything else, one of the most widely reported hardware features of the Watch has been the Digital Crown, which functions both as a push button, comparable in use to the iPhone's *Home* button, and as a method for selecting or entering data into your app. While not exactly a radical development, the royal scroll wheel really does make entering data, selecting options, and navigating around an interface very much quicker and more precise than tapping and swiping what are frequently extremely small UI elements on the screen and also avoids covering a large part of the screen content while doing so; this is the inevitable result of using finger touch gestures.

And if the battery gets a bit low, you could try using it to wind up the watch. Just kidding.

Arousing much less interest than the Crown, and slightly more conventional, is the humble Side Button located below it, which is, well, a button. Everyone needs a button, right? Perhaps to bring up the Friends Screen, with which to send messages and initiate phone calls. It's a fine button.

Oh, and your app can't use it.

**Force Touch**
In addition to the standard taps (or touches), and double taps, with which to select options, activate buttons, and so on, in much the same way as she would on any other iOS touchscreen, the user can also deploy the Force Touch, a harder and slightly longer press on the screen, which is acknowledged by watchOS with a satisfying pulse (see Taptic Haptic, below), and can be interpreted by an app in whichever way is appropriate.
Chapter 1

Taptic Engine

Wikipedia offers the definition of haptic technology as

…technology which recreates the sense of touch by applying forces, or motions to the user

Haptic feedback has long been a part of gaming controllers and other devices, including smartphones, made by many manufacturers. Apple’s take on this, the Taptic Engine, is definitely one of the industry’s classiest implementations of this technology, and employs a linear actuator (look it up, if you like) to produce a quite stunning range of buzzing and tapping sensations on the wrist, that users will have available to them all day, no matter what they are wearing (dresses and smartphones, no pockets, now what?), and no matter where they are, making the Watch into a device that can communicate meaningfully with the user with minimal distraction, maximum discretion (you can’t hear the Taptic Engine next to you at all), and one that can initiate that communication (beyond just beeping) in a way that is simply not possible on a phone, which may be on a table in front of the user, in a jacket pocket on a coat hook somewhere, or on a table somewhere quite different.

WatchKit provides a number of preset taptics, which the developer is encouraged to use according to the conventions suggested by the naming of each preset, so that users will, over time, become accustomed to each haptic notification and know intuitively and without needing to look at the watch, what kind of information the watch is offering to convey.

Whether a subtle and gentle tap on the wrist, or an insistent, repeated buzzing sensation, as inaudible as it is invisible, the Taptic Engine represents not only state of the art haptics, but also points the way to a whole new genre of communication from a device to the user. To a great extent, what becomes of this is down to what designers and developers can dream up.

Audio in out

While the audio output of the Watch is limited to a tiny but impressively loud and clear speaker, one which is loud enough to conduct a telephone conversation in an averagely loud city street (so no headphone output), the audio input of the device is both surprisingly high quality (call recipients are unaware that a conversation is being had from the watch) and pleasingly flexible, in that it gets piped into Apple’s voice-controlled assistant Siri (yes she lives on the Watch too) where it can be used to dictate text and save audio files, among many other things.

Alternatively, the user can connect to a Bluetooth headset, which is a more suitable arrangement for that jog around the park.
Exploring the New Platform

♥ It

The Heartrate Sensor, in enabling users to monitor one of the most important aspects of their physical health, is undoubtedly an important feature of the Apple Watch. The ability to share heartbeats with another Watch owner perhaps less so.

Extension of the iPhone, but more

To some extent, users are, by complementing an iPhone with an Apple Watch, extending the reach of the device they already know, and widening the range of ways they have to interact with much of the software they already use. Indeed, at the time of writing, Apple does not accept submissions of apps that provide functionality exclusively for the Watch, demonstrating quite clearly their current view of how the Watch fits into the ecosystem. In this model, the watch is very much an extra level of engagement with the phone, and introduces what may well become a dominant feature of wearables development, namely that the physical separation of two devices does not need to be a fundamental part of the relationship between them. In a sense, it is like taking a small bit of screen and a button or two from the watch, and locating it somewhere more convenient, for use in appropriate situations. Apple’s emphasis on fitness tracking is a great example, but should be viewed more as some smart reading of the nascent market for wrist-borne computers, than as an indication of some imagined limit on the scope of the Watch's utility. These are very early days, and it would be a foolhardy author who risks a prediction of the ways in which our use of wearables will evolve in the next few years, that is still very much in the stars. But evolve it will, of that we can be sure.

Opportunity knocks

And it is because of this wide open scope for development and yet-to-be-discovered potential that we also need to be able to view the Apple Watch as not simply an extension of the iPhone (though it certainly is that), but also as an autonomous device in its own right, one that can collect data from a number of sources (whether the user, the phone or the web), process that data natively, present it visually or otherwise, at its own initiative, and even store that data onboard or online. Since the release of watchOS 2, the limitations of what we can do independently of the phone are defined more by our imagination than by actual barriers inherent in the technology, and the Watch fits in perfectly with the tendency of apps to be increasingly spread across devices and platforms. Most users of social networks, for example, will regularly use more than one platform to catch up on what’s going on and update their status, typically a web browser on a laptop and a phone app, but this trend is being broadened across a whole range of software categories, as users increasingly expect to be able to use apps across several devices according to whatever is appropriate to their situation.
And so we have an operating system, watchOS, that is both a part of its larger sibling, iOS, and a separate OS in its own right, that perfectly mirrors the relationship between the physical watch, and its larger counterpart in our pockets. The relationship between WatchKit and UIKit is the same. Our success in designing and developing for this platform will likely depend to a significant extent on our ability to embrace this dual personality.

If that sounds like an exciting challenge, and a stimulating environment in which to expand your skills as a developer, then this book is for you.

Summary

In this chapter, we have gained an overview of the Apple Watch — the physical device itself, the operating system that powers it, and the wider context in which you will be creating WatchKit apps that make the platform come alive. We have covered details of the hardware, the context in which the Watch idiom is being developed, watchOS, Apple's framework for developing for watchOS (WatchKit), the various interfaces available to your app, the user interface hardware, and some UI objects in Interface Builder.

In the next chapter, we will create an app that will run on the Watch Simulator (which is a part of Xcode).

You are, as designer and developer, and as such, pioneer of the new platform, in a unique position to contribute to its growth and success, leveraging its almost limitless potential to transform how we engage with the technology we carry with us (almost) everywhere we go. So let's do it.
Where to buy this book

You can buy Building Apple Watch Projects from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.