FREE eBook

LEARNING watchkit

Free unaffiliated eBook created from
Stack Overflow contributors.

#watchkit

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: watchkit

It is an unofficial and free watchkit ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official watchkit.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with watchkit

Remarks

- Apple Documentation on WatchKit (through API Reference): https://developer.apple.com/reference/watchkit
- Getting familiar with Xcode: Xcode Interface

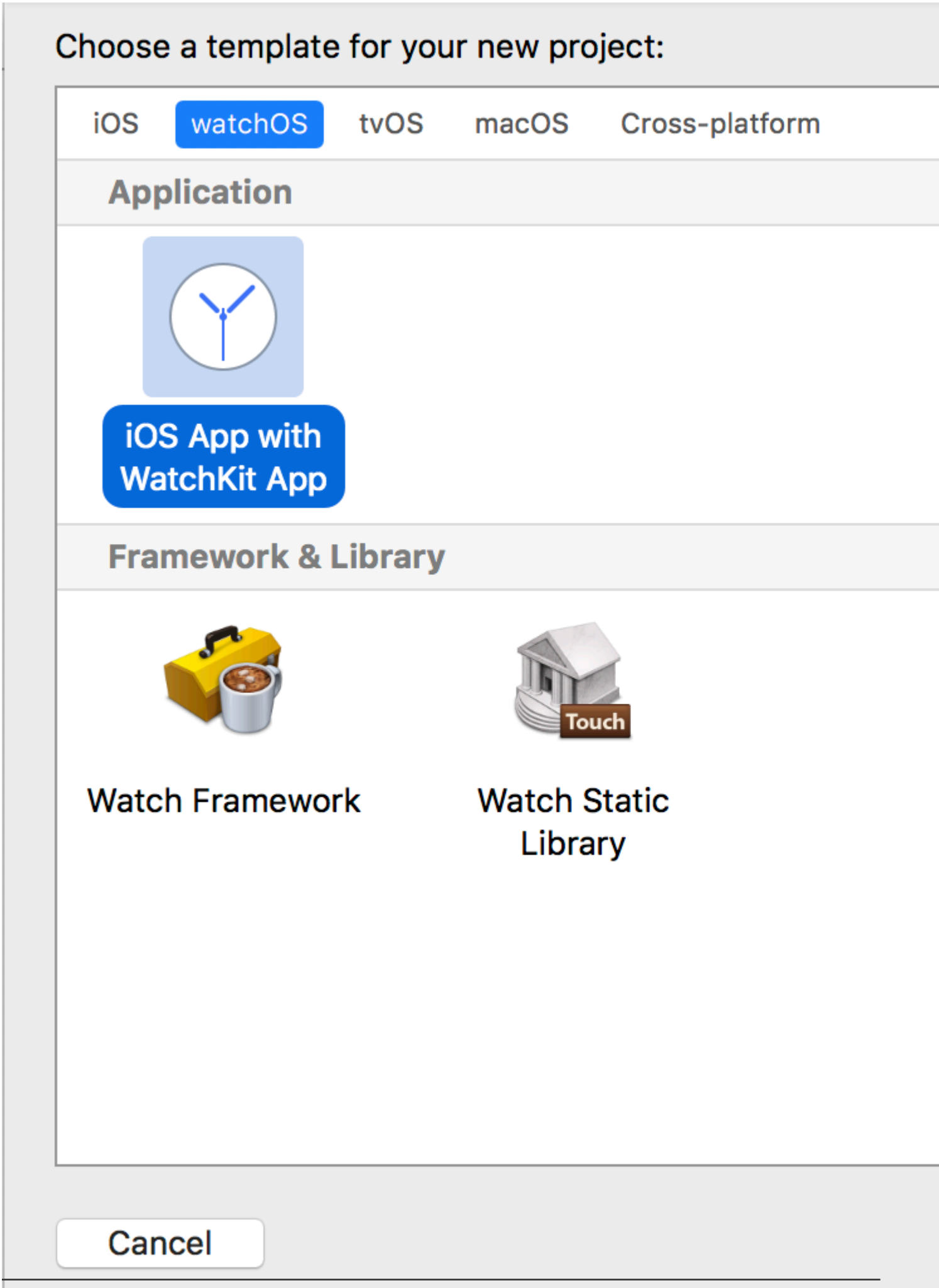
Examples

Creating a new watchOS project

To develop an application for watchOS, you should start with Xcode. Xcode only runs on macOS. At the time of writing, the latest version is Xcode 8.3.

If you want to start a new project from scratch:

1. Boot up your Mac and install Xcode from the App Store if it's not already installed.
2. Choose to create a new project.
3. In templates, choose watchOS and then "iOS App with WatchKit App".

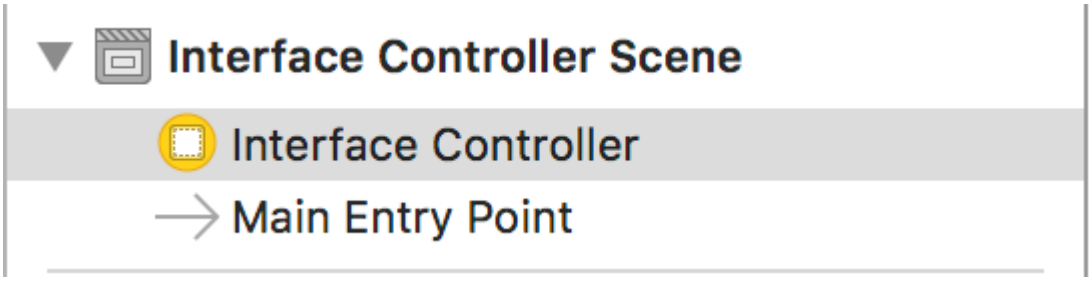

Choose a template for your new project:
iOS
watchOS
tvOS
macOS
Cross-platform
Application
iOS App with WatchKit App
Framework & Library
Touch
Watch Framework
Watch Static Library
Cancel

`Assets.xcassets` file to put your assets in.

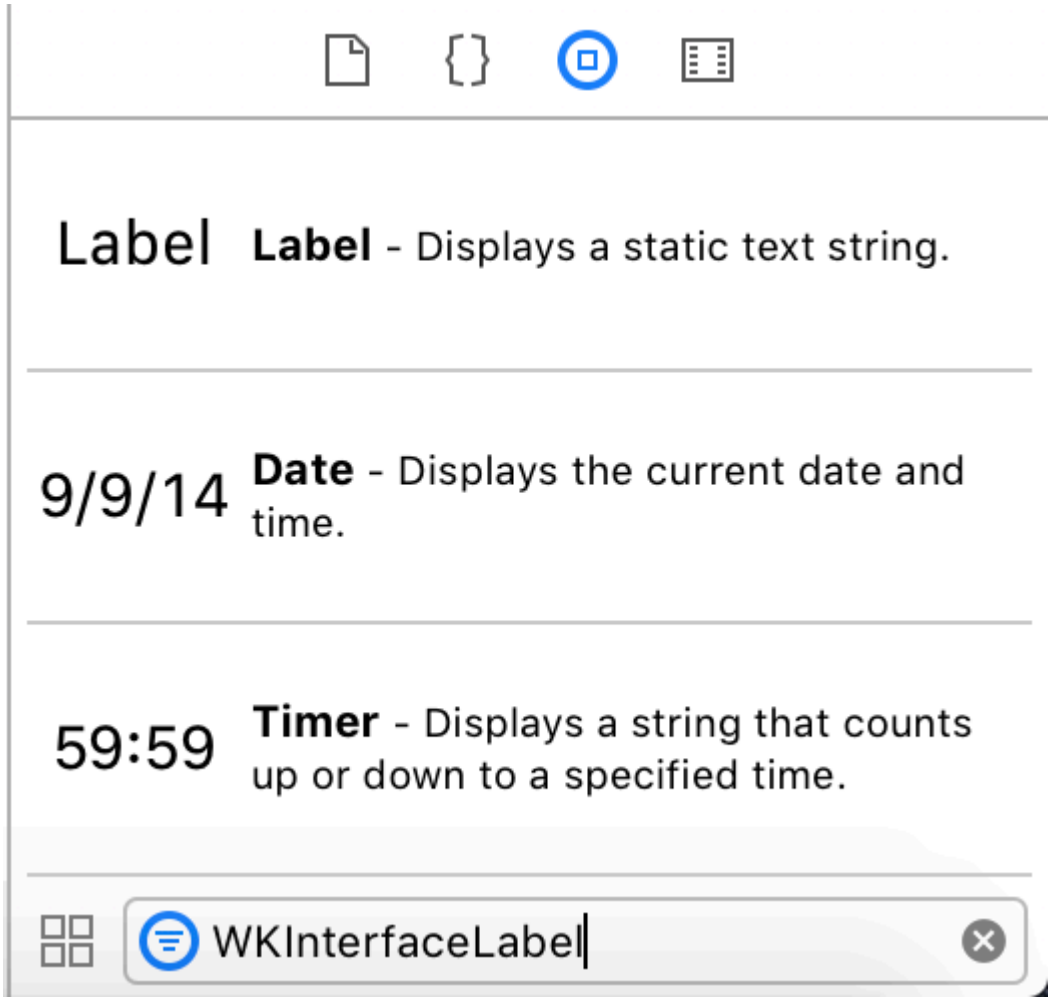
Each WatchKit Extension has a `InterfaceController.swift` file (actually a `WKInterfaceController` subclass) which is similar to the `ViewController` file in iOS.

To make a Hello World app:

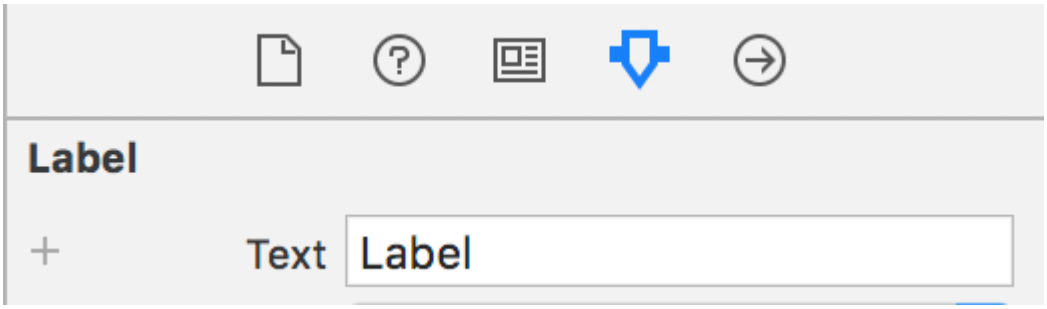
1. Open up the `Interface.storyboard`.
2. Locate the main `InterfaceController`.



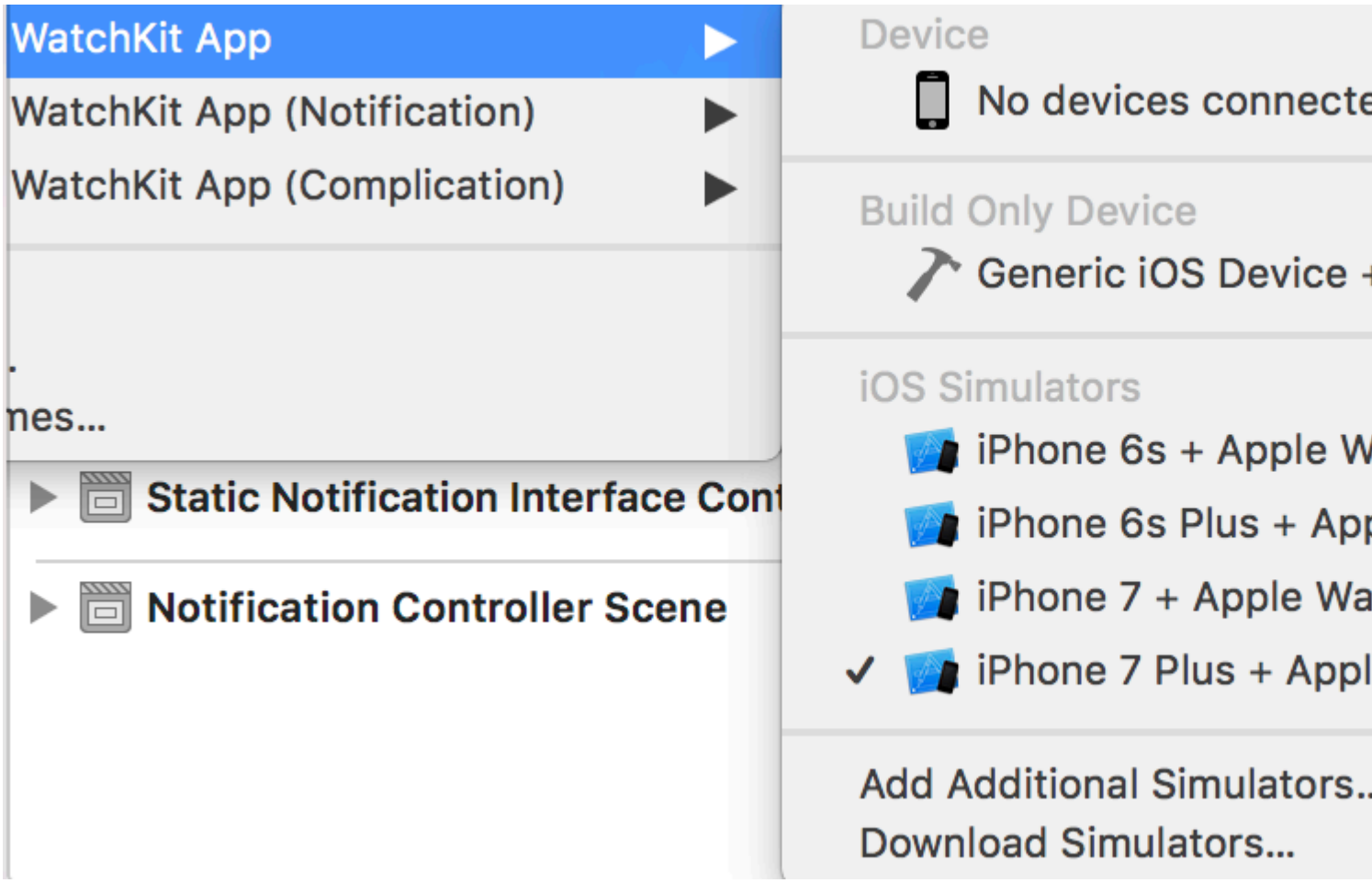
3. From the library in the right pane, add a `WKInterfaceLabel`.

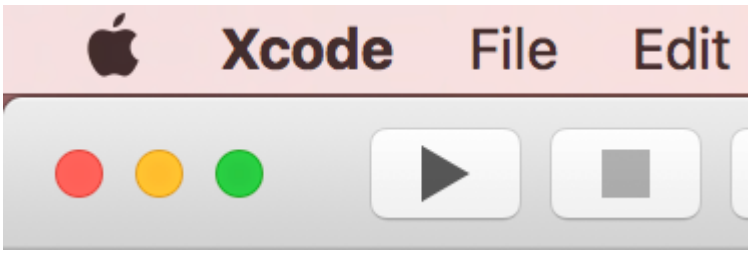


4. Drag the label and set its text in the right pane to "Hello, World!".



5. Select the correct scheme (according to the next picture), then run the project by either tapping the run button in the top bar, using Product menu, pressing Cmd-R or tapping run in the Touch Bar.





Apple Watch simulator will eventually show up with your app running.

Connecting the code with the UI

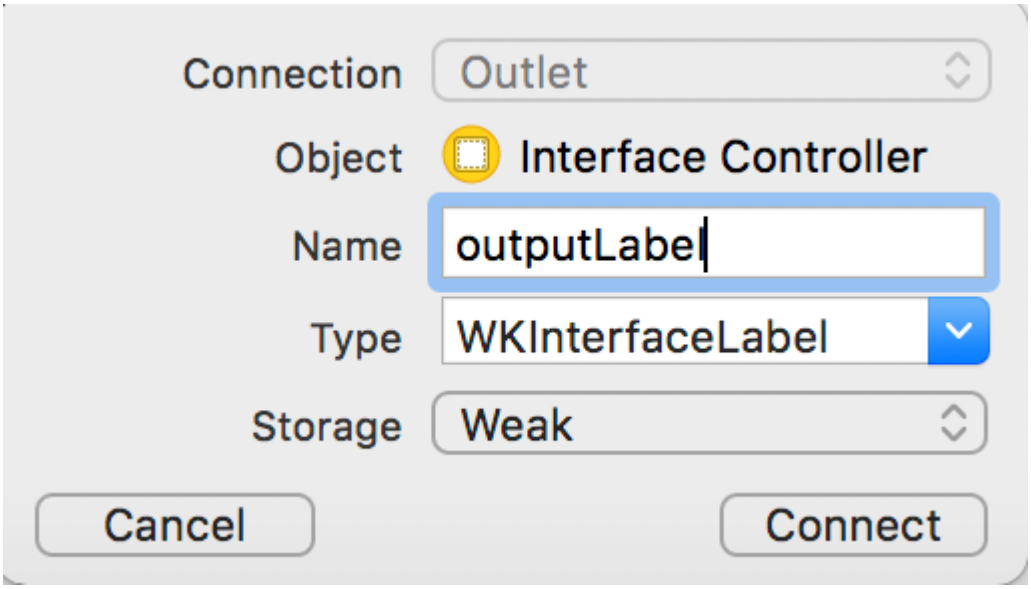
Like iOS where you use `@IBOutlet` and `@IBAction`, here you could use them too.

Let's say we have a button which when clicked changes the label's text to something else.

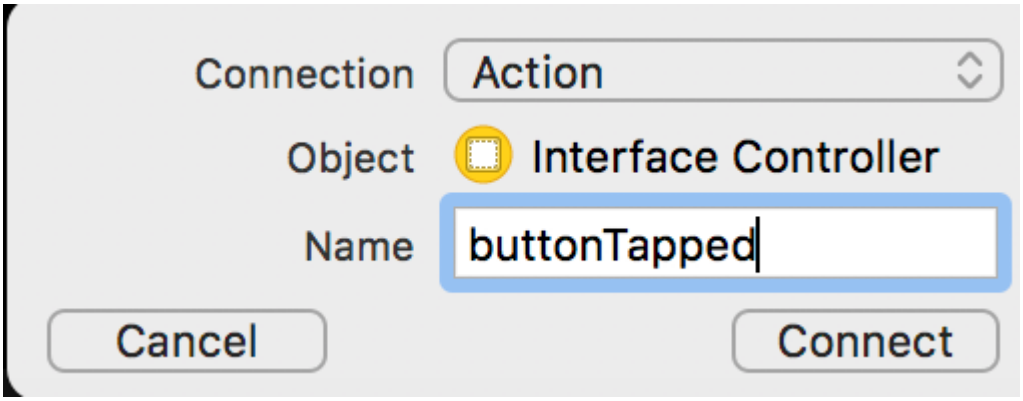
To get started:

1. Add a `WKInterfaceLabel` and a `WKInterfaceLabel` to the `InterfaceController`.

2. Ctrl-Drag from the `WKInterfaceLabel` to `InterfaceController.swift` and enter the details as shown in the following picture to add an outlet property:



3. Ctrl-Drag from the `WKInterfaceButton` to `InterfaceController.swift` and enter the details as shown in the following picture to add an action method:



5. Fill the action method:

Swift

```
outputLabel.setText("Button Tapped!")
```

Objective-C

```
[[self outputLabel] setText:@"Button Tapped!"]
```

6. Run the program and tap the button to see the result.

Read Getting started with watchkit online: https://riptutorial.com/watchkit/topic/9568/getting-started-with-watchkit

Chapter 2: Navigation

Remarks

Important Note

> Apple has highly discouraged use of both navigation styles in one controller, and this may result in an app rejection.

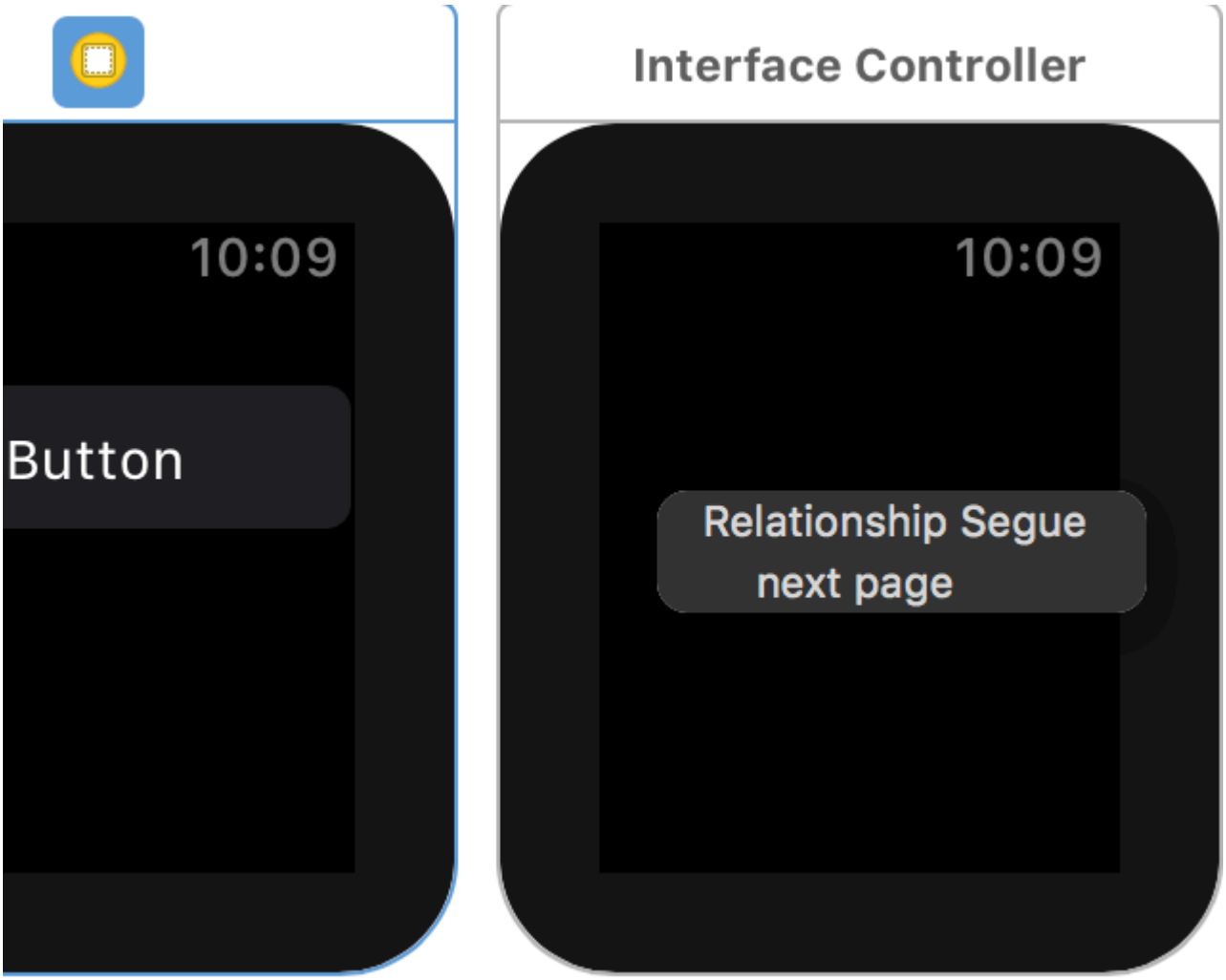
Currently, the preferred way is to use hierarchical style rather than page based, as used in many more Apple apps than before.

Examples

Page-based navigation

Many watchOS apps (like Activity) have several pages which you could simply scroll between them, which is a very good way to use Apple Watch.

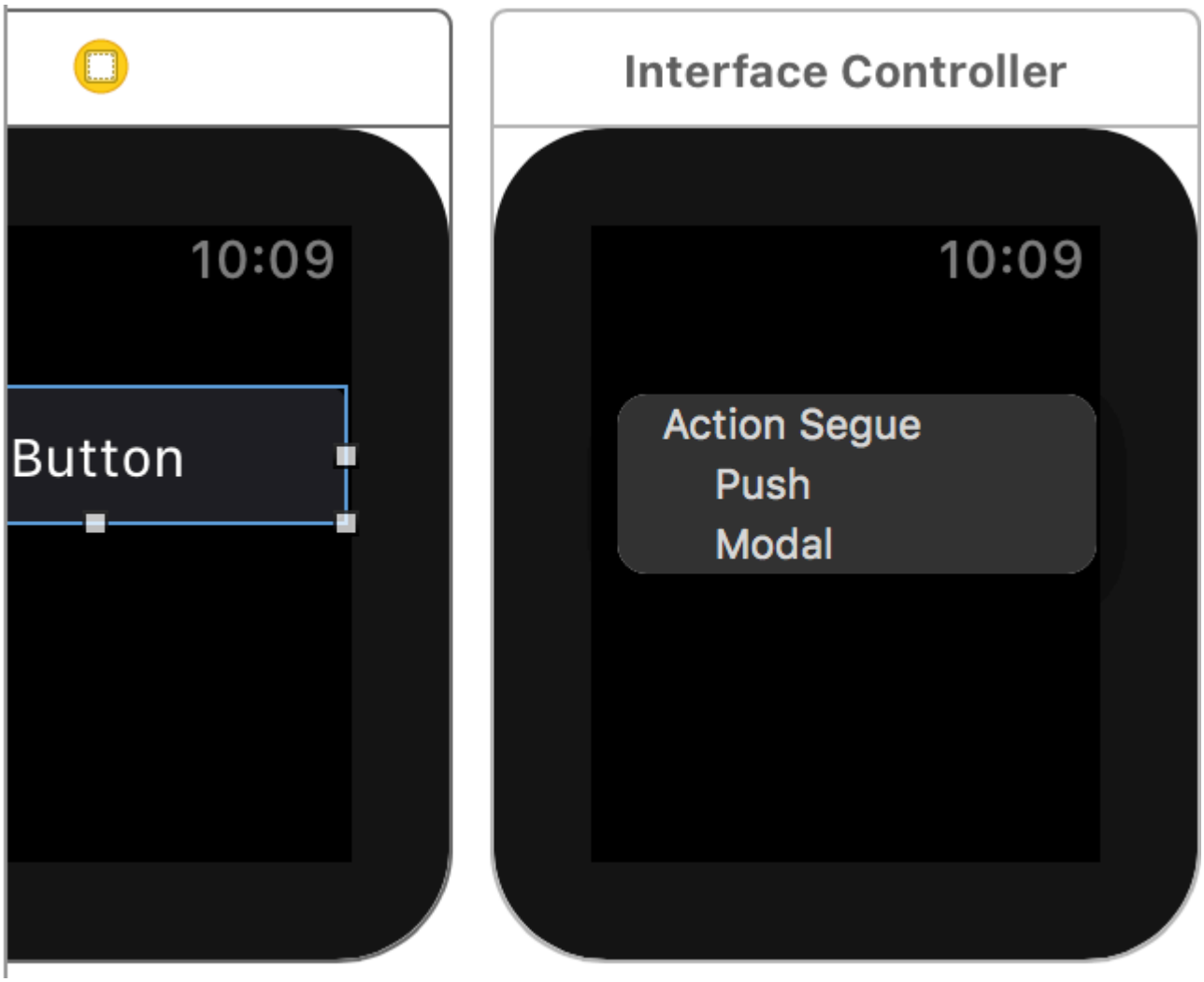
To create a page based navigation, Ctrl-Drag from one controller to another, and select "next page", as shown in the following picture:



Hierarchical view structure

Many watchOS apps (like Workout, Weather, Music, etc) have a main `WKInterfaceTable` or a set of buttons which are hooked up to another controller, similar to the navigation on iOS. This is called hierarchical view structure.

To connect a button, Ctrl-Drag from the button to a controller, and select "push" or "modal" based on your need, as shown in the following picture:



Read Navigation online: https://riptutorial.com/watchkit/topic/9599/navigation

Chapter 3: WatchConnectivity

Introduction

Connecting your WatchOS application to your iOS application can be a task to complete when you have never done it before. This tutorial will show you the basic fundamentals in order to accomplish this very important task.

Examples

iOS Configuration

iPhone App

1. import WatchConnectivity and conform to WCSessionDelegate.
2. use the static session delegate via `WCSession.default()`.
3. Send data to Watch app using:

```
WCSession.default().sendMessage(message, replyHandler:_ errorHandler:_)
```

4. The message object should be a dictionary of type [String:Any]
5. If you are looking for data to be returned from the Watch app, provide the logic in a closure defined in the replyHandler; otherwise, pass in nil.
6. To respond to messages sent from the Watch app, you will use the WCSessionDelegate callback method

```
func session(_ session: WCSession, didReceiveMessage message: [String : Any],
replyHandler: @escaping ([String : Any]) -> Swift.Void){...}
```

You need to import the WatchConnectivity framework into your *view controller* file in the iOS application. This gives you access to the class objects that are designed to communicate with the Watch companion application. You will need to import this on the Watch app side as well. The only real difference with the Watch app is that you will not have a *view controller* file, but an *interface controller* file.

```
import WatchConnectivity
```

Next, you will need to make sure that your application can support a session for WatchConnectivity. If it can, then you must set the view controller as its delegate and activate the default session. You will get an error here. You need to conform to WCSessionDelegate and implement a few methods before the IDE begins to calm down.

```
// MARK: - View Life Cycle Callbacks
override func viewDidLoad() {
    super.viewDidLoad()
```

```
    automaticallyAdjustsScrollViewInsets = false
    if WCSession.isSupported() {
        WCSession.default().delegate = self
        WCSession.default().activate()
    }
    else {
        print("\nViewController: connectionManager is nil\n")
    }
}
```

To conform to WCSessionDelegate, let's add an extension to the bottom of the view controller. Some people hate this approach. I have my way, but for tutorial purposes, I will follow the *RayWenderlich* approach. I am a huge fan of just getting code directly from COMMAND + CLICKING on the delegates and grabbing ALL OF THE METHODS out of the specification and begin to manipulate it and understand how things work. In this extension, I am providing you will all of the methods. They are already marked so that you will see when each of them fire in the console. If you feel frisky, delete some of the ones that are marked OPTIONAL to make your code file look spiffy.

```
extension ViewController : WCSessionDelegate {

    func session(_ session: WCSession, activationDidCompleteWith activationState:
WCSessionActivationState, error: Error?) {
        print("0. ViewController: ", "activationDidCompleteWith activationState")
    }


    /** ------------------------ iOS App State For Watch ----------------------- */

    func sessionDidBecomeInactive(_ session: WCSession) {
        print("1. ViewController: ", "sessionDidBecomeInactive")
    }


    func sessionDidDeactivate(_ session: WCSession) {
        print("2. ViewController: ", "sessionDidDeactivate")
    }


    func sessionWatchStateDidChange(_ session: WCSession) {
        print("3. ViewController: ", "sessionDidDeactivate")
    }


    /** ------------------------ Interactive Messaging ------------------------ */

    func sessionReachabilityDidChange(_ session: WCSession) {
        print("4. ViewController: ", "sessionReachabilityDidChange")
    }


    func session(_ session: WCSession, didReceiveMessage message: [String : Any]) {
        print("5. ViewController: ", "didReceiveMessage")
    }
```

```
    func session(_ session: WCSession, didReceiveMessage message: [String : Any],
replyHandler: @escaping ([String : Any]) -> Swift.Void) {
        print("6. ViewController: ", "didReceiveMessage")
        // This is where you handle any requests coming from your Watch App
    }

    func session(_ session: WCSession, didReceiveMessageData messageData: Data) {
        print("7. ViewController: ", "didReceiveMessageData")
    }

    func session(_ session: WCSession, didReceiveMessageData messageData: Data, replyHandler:
@escaping (Data) -> Swift.Void) {
        print("8. ViewController: ", "didReceiveMessageData")
    }

    /** ------------------------- Background Transfers ------------------------ */

    func session(_ session: WCSession, didReceiveApplicationContext applicationContext:
[String : Any]) {
        print("9. ViewController: ", "didReceiveApplicationContext")
    }

    func session(_ session: WCSession, didFinish userInfoTransfer: WCSessionUserInfoTransfer,
error: Error?) {
        print("10. ViewController: ", "didFinish userInfoTransfer")
    }

    func session(_ session: WCSession, didReceiveUserInfo userInfo: [String : Any] = [:]) {
        print("11. ViewController: ", "didReceiveUserInfo")
    }

    func session(_ session: WCSession, didFinish fileTransfer: WCSessionFileTransfer, error:
Error?) {
        print("12. ViewController: ", "didFinish fileTransfer")
    }

    func session(_ session: WCSession, didReceive file: WCSessionFile) {
        print("13. ViewController: ", "didReceive file")
    }
}
```

To send data to your Watch App, once the Watch App has requested it, you will handle this in the method...

```
func session(_ session: WCSession, didReceiveMessage message: [String : Any], replyHandler:
@escaping ([String : Any]) -> Swift.Void) {
    print("6. ViewController: ", "didReceiveMessage")
    // build out your response message using a Dictionary
    let returnMessage: [String : Any] = [
       "key1" : value1,
       "key2" : value2,
       "key3" : value3
    ]
    // return your data in this manner
    replyHandler(returnMessage)
    // WARNING
    // You must call the replyHandler before the method ends, otherwise, your app will crash.
}
```

This is only one half of the transaction! You must configure your Watch app to connect to the iPhone app and handle any returned messages!

Watch Extension Configuration

WatchKit App

1. import WatchConnectivity and conform to WCSessionDelegate.
2. use the static session delegate via `WCSession.default().`
3. Send data to the iPhone app using:

```
WCSession.default().sendMessage(message, replyHandler:_ errorHandler:_)
```

4. The message object should be a dictionary of type [String:Any]
5. If you are looking for data to be returned from the Watch app, provide the logic in a closure defined in the replyHandler; otherwise, pass in nil.
6. To respond to messages sent from the iPhone app, you will use the WCSessionDelegate callback method

```
func session(_ session: WCSession, didReceiveMessage message: [String : Any],
replyHandler: @escaping ([String : Any]) -> Swift.Void){...}
```

7. These methods will not be required in your Watch app to properly conform to the WCSessionDelegate:

```
func sessionDidBecomeInactive(_ session: WCSession)
func sessionDidDeactivate(_ session: WCSession)
func sessionWatchStateDidChange(_ session: WCSession)
```

And finally, it is generally best practice to store any common images, the images and assets that will continually be used for the Watch app to be placed in the Watch's xcassets folder. Okay, now that you are confused, let's get to the details!

It might be a surprise to you, but you need to import WatchConnectivity again.

```
import WatchConnectivity
```

Next, you need to verify that the session is even possible.

```
// MARK: - View Life Cycle Callbacks
override func awake(withContext context: Any?) {
    super.awake(withContext: context)

    // Configure interface objects here.
    if WCSession.isSupported() {
        WCSession.default().delegate = self
        WCSession.default().activate()
        print("InterfaceController: Session Activated")

        // Request Data from iPhone App
```

```
        let requestMessage = ["message":"get-data"]

        WCSession.default().sendMessage(requestMessage, replyHandler: { (replyMessage) in
            print("Got a reply from the phone: \(replyMessage)")

            // handle reply message here

        }, errorHandler: { (error) in
            print("Got an error sending to the phone: \(error)")
        })
    }
    else {
        print("\nViewController: connectionManager is nil\n")
    }
}
```

But none of this will work, unless you implement methods required for the WCSessionDelegate.

```
extension InterfaceController : WCSessionDelegate {

    func session(_ session: WCSession, activationDidCompleteWith activationState:
WCSessionActivationState, error: Error?) {
        print("1. InterfaceController: ", "activationDidCompleteWith activationState") //
first
    }

    /** ------------------------- Interactive Messaging ------------------------- */

    func sessionReachabilityDidChange(_ session: WCSession) {
        print("2. InterfaceController: ", "sessionReachabilityDidChange") // second
    }

    func session(_ session: WCSession, didReceiveMessage message: [String : Any]) {
        print("3. InterfaceController: ", "didReceiveMessage")
    }

    func session(_ session: WCSession, didReceiveMessage message: [String : Any],
replyHandler: @escaping ([String : Any]) -> Swift.Void) { // third
        print("4. InterfaceController: ", "didReceiveMessage")
        //print("Message Contents: ", message["message"]!)
    }

    func session(_ session: WCSession, didReceiveMessageData messageData: Data) {
        print("5. InterfaceController: ", "didReceiveMessageData")
    }

    func session(_ session: WCSession, didReceiveMessageData messageData: Data, replyHandler:
@escaping (Data) -> Swift.Void) {
        print("6. InterfaceController: ", "didReceiveMessageData")
    }

    /** ------------------------- Background Transfers ------------------------- */

    func session(_ session: WCSession, didReceiveApplicationContext applicationContext:
[String : Any]) {
        print("7. InterfaceController: ", "didReceiveApplicationContext")
    }

    func session(_ session: WCSession, didFinish userInfoTransfer: WCSessionUserInfoTransfer,
error: Error?) {
```

```
        print("8. InterfaceController: ", "didFinish userInfoTransfer")
    }

    func session(_ session: WCSession, didReceiveUserInfo userInfo: [String : Any] = [:]) {
        print("9. InterfaceController: ", "didReceiveUserInfo")
    }

    func session(_ session: WCSession, didFinish fileTransfer: WCSessionFileTransfer, error:
Error?) {
        print("10. InterfaceController: ", "didFinish fileTransfer")
    }

    func session(_ session: WCSession, didReceive file: WCSessionFile) {
        print("11. InterfaceController: ", "didReceive file")
    }
}
```

Sending Data

Sending data to your Watch app from your iPhone or from your iPhone to your Watch App after triggering some event is very simple, though the code can look a little complex at first.

```
let message = ["key":"value-to-send"]

WCSession.default().sendMessage(message, replyHandler: { (replyMessage) in
    print("Got a reply from the phone: \(replyMessage)")

    if let returnedValues = replyMessage["returned-value"] as? NSArray {
        for val in returnedValues {
            // do something here with the data
            // Dispatch to Main Thread if affecting UI
        }
    }
}, errorHandler: { (error) in
    print("Got an error sending to the phone: \(error)")
})
```

Read WatchConnectivity online: https://riptutorial.com/watchkit/topic/10733/watchconnectivity

Credits

S. No	Chapters	Contributors
1	Getting started with watchkit	Community, Seyyed Parsa Neshaei
2	Navigation	Seyyed Parsa Neshaei
3	WatchConnectivity	mrfilter