# Developing for Apple watchOS

## Overview

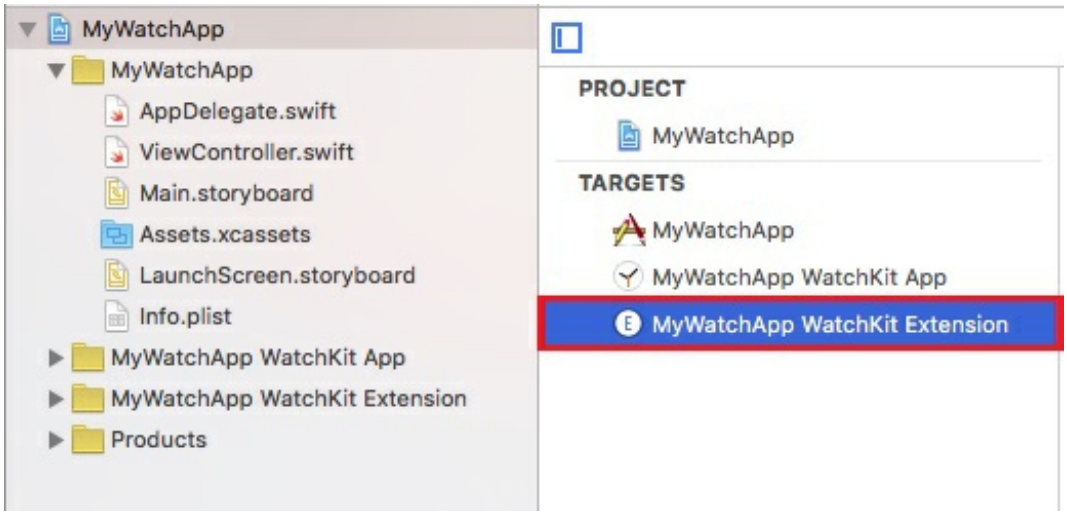
In this example, you will learn how to set up the development environment for watchOS 2 and later using MobileFirst framework. The example is created and described using watchOS 2. It also works fine on watchOS 3.

## Setup

To set up the development environment for watchOS , create the Xcode project, add the watchOS framework, and set up the necessary targets.

1. Create a watchOS 2 app in Xcode.
    - Choose the **File → New → Project** option; the **Choose a template for your new project** dialog appears.
    - Choose the **watchOS2/Application ** option, click **Next**.
    - Name the project and click **Next**.
    - From the navigation dialog, choose the project folder.

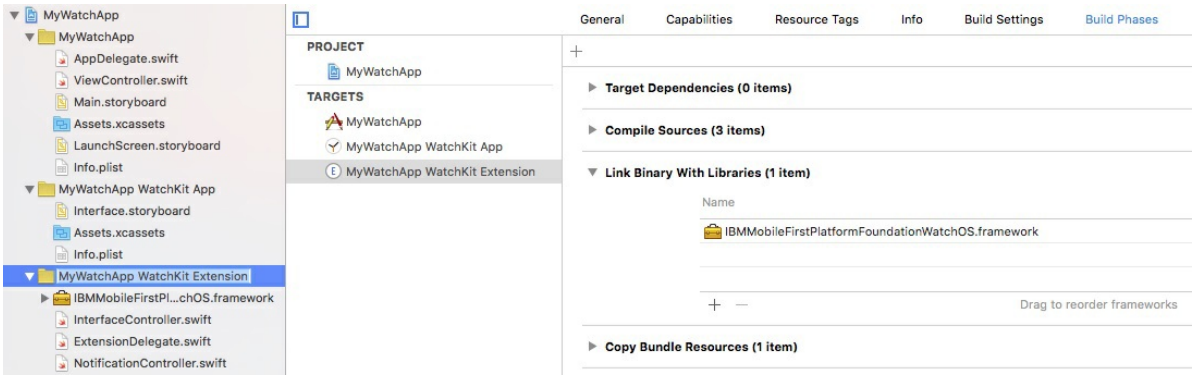
    The project navigation tree now contains a main app folder and a **[project name] WatchKit Extension** folder and target.




2. Add the MobileFirst watchOS framework.
    - To install the necessary frameworks with CocoaPods, see Adding the MobileFirst Native SDK (../../application-development/sdk/ios/#adding-support-for-apple-watchos) tutorial.
    - To install the necessary frameworks manually:
        - Obtain the watchOS framework from MobileFirst Operations Console's Download Center.
        - Select the **[project name] WatchKit Extension** folder in the left navigation pane.
        - From the **File** menu, choose **Add Files**.
        - Click the **Options** button and select the following:
            - **Copy items if needed** and **Create groups** options.
            - **[project name] WatchKit Extension** in the **Add to targets** section.

- Click **Add**.

Now when you select the **[project name] WatchKit Extension** in the **Targets** section: * The framework path appears in the **Framework Search Paths** setting in the **Search Paths** section of the **Build Settings** tab. * The **Link Binary With Libraries** section of the **Build Phases** tab lists the **IBMMobileFirstPlatformFoundationWatchOS.framework** file:




> **Note:** WatchOS 2 requires bitcode. From Xcode 7 the **Build Options** is set to **Enable Bitcode Yes** (**Build Settings** tab, **Build Options** section).

3. Register both the main app and the WatchKit extension on the server. Run `mfpdev app register` for each Bundle ID (or register from the MobileFirst Operations Console):
    - com.worklight.[project_name]
    - com.worklight.[project_name].watchkitextension
4. In Xcode, from the File->Add File menu, navigate to the mfpclient.plist file created by mfpdev and add it to the project.
    - Select the file to display the **Target Membership** box. Select the **WatchOSDemoApp WatchKit Extension** target in addition to the **WatchOSDemoApp**.

The Xcode project now contains a main app and a watchOS 2 app, each can be developed independently. For Swift, the entry point for the watchOS 2 app is the **InterfaceController.swift** file in the **[project name] watchKit Extension** folder. For Objective-C the entry point is the **ViewController.m** file.

# Setting up MobileFirst security for the iPhone app and the watchOS app

The Apple Watch and iPhone devices differs physically. Therefore the security checks for each must be appropriate for the available input devices. For example, the Apple Watch is limited to a number pad and does not allow the usual username/password security check. Therefore access to protected resources on the server could be enabled using a pin code. Because of these and similar differences, it is necessary to apply different security checks for each target.

Below is one example of creating an app with both an iPhone and an Apple Watch target. This architecture allows each to have its own security check. The differing security checks are just examples of how you can design features for each target. Additional security checks might be available.

1. Determine the scope and security checks defined by the protected resource.
2. In the IBM MobileFirst Platform Operations Console:
    - Ensure that both apps are registered on the server:
        - com.worklight.[project_name]

        - com.worklight.[project_name].watchkitextension
    - Map the scopeName to the defined security checks:
        - For com.worklight.[project_name] map it to the username/password check.
        - For com.worklight.[project_name].watchkitapp.watchkitextension map it to the pin code security check.

# WatchOS Limitation

The optional frameworks that add features to the MobileFirst app are not provided for watchOS development. Some other features are limited by constraints imposed by the watchOS or Apple Watch device.

| Feature | Limitation |
|---|---|
| openSSL | Not supported |
| JSONStore | Not supported |
| Notifications | Not supported |
| Message alerts displayed by the MobileFirst code | Not supported |
| Application authenticity validation | Not compatible with bitcode, and therefore not supported |
| Remote disable/notify | Requires customization (see below) |
| Usernames/password security check | use the CredentialsValidation security check |

## Remote Disable/Notify

With the IBM MobileFirst Platform Operations Console, you can configure the IBM MobileFirst Platform Server to disable access (and return a message) to client applications based on the version they are running (see Remotely disabling application access to protected resources (../../administering-apps/using-console/#remotely-disabling-application-access-to-protected-resources). Two options provide default UI alerts:

- when the app is active but a messages is sent: **Active and Notifying**
- when the app is outdated and access is denied: **Access Denied**

For watchOS:

- To see messages where the app is set to **Active and Notifying,** a custom remote disable challenge handler must be implemented and registered. The custom challenge handler must be initialized with the security check `wl_remoteDisableRealm`.
- In the case where the access is disabled (**Access Denied**) the client app receives an error message in the failure callback or request delegate handler. The developer can decide how to handle the error, either notifying the user through the UI or writing to the log. In addition the above method of creating a custom challenge handler can be used.

*Last modified on November 28, 2016*