

Glances, Notifications, and Handoff

CS193W - Spring 2016 - Lecture 4

Glances

- A single screen of content that provides an at-a-glance summary of your app



Adding Glances to your Project

- Easy Way: Add them when you create your WatchKit App Target

Choose options for your new target:

Product Name:

Organization Name:

Organization Identifier:

Bundle Identifier:

Language:

Include Notification Scene

Include Glance Scene

Include Complication

Project:

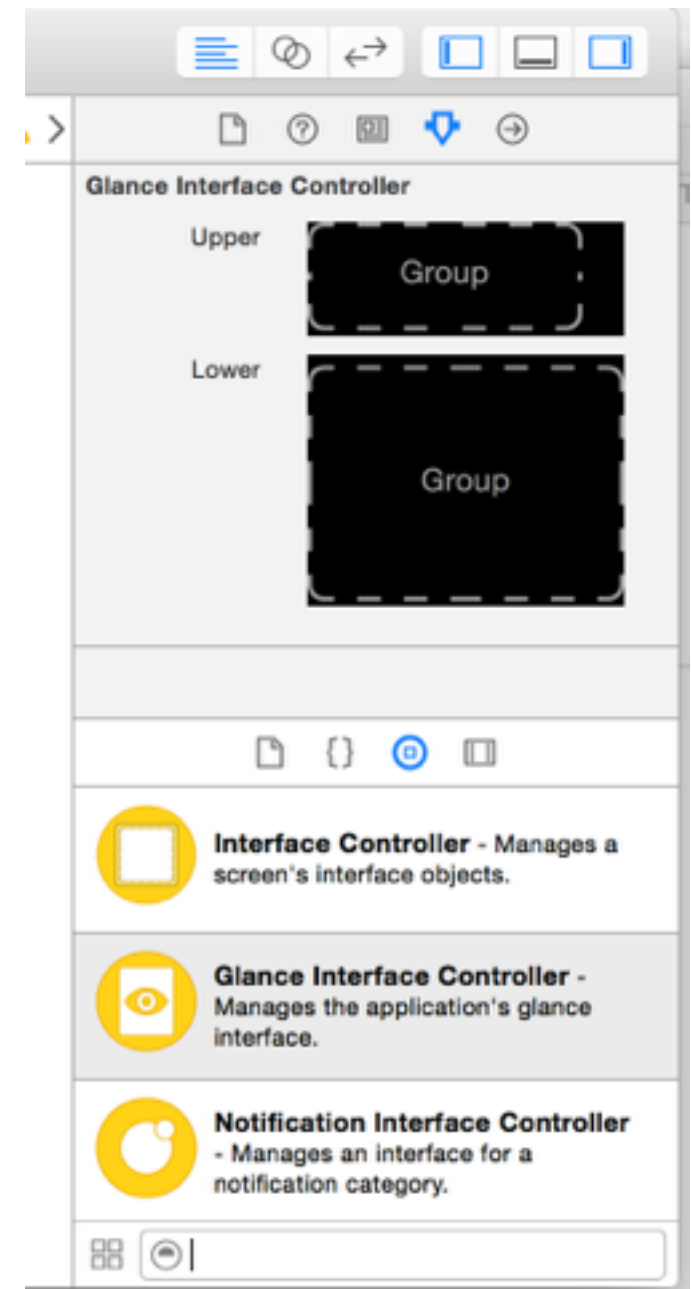
Embed in Companion Application:

Or Add it Later

1. Add a glance class to your WatchKit Extension target
2. Create a Glance interface controller in your WatchKit App Storyboard
3. Create a Glance Scheme (for debugging purposes)

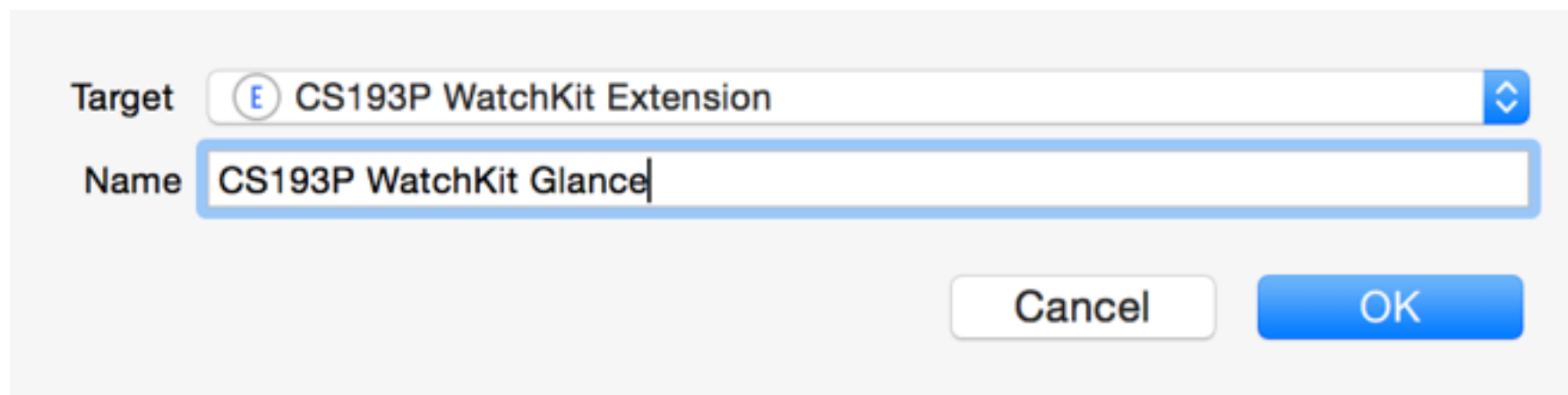
Code & Storyboard

- There is nothing special about the code of a glance. It is just a `WKInterfaceController`.
- You do use a special interface object however.



Adding a Glance Scheme

1. Go to Product -> Scheme -> New Scheme
2. Make the target your WatchKit Extension Scheme



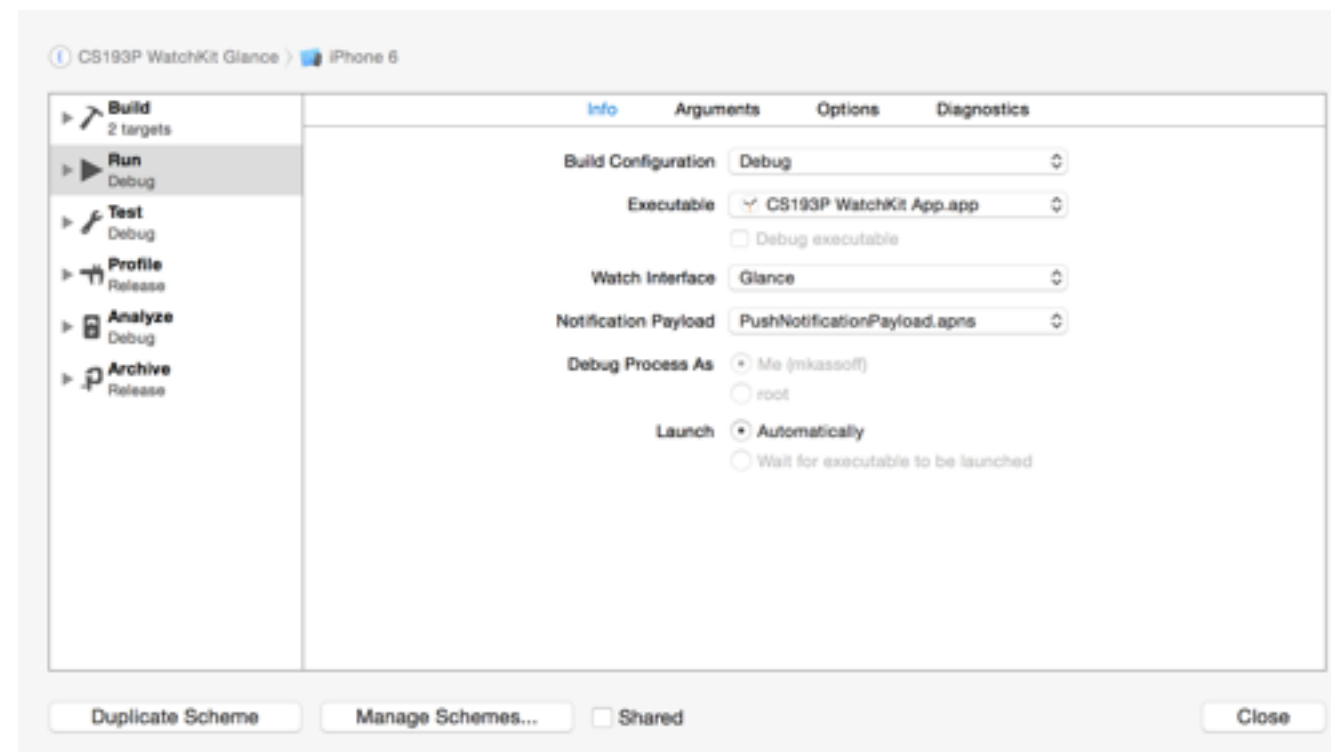
The image shows a screenshot of the 'New Scheme' dialog box in Xcode. The 'Target' dropdown menu is set to 'CS193P WatchKit Extension'. The 'Name' text field contains the text 'CS193P WatchKit Glance'. At the bottom right, there are two buttons: 'Cancel' and 'OK'.

Target	CS193P WatchKit Extension
Name	CS193P WatchKit Glance

Buttons: Cancel, OK

Adding a Glance Scene (cont'd)

3. Go to Product -> Scheme -> Edit Scheme
4. Choose your WatchKit App for Executable
5. Choose Glance for Watch Interface



Glance Interface Guidelines

- **Design your glance to convey information quickly.**
- **Focus on the most important data.**
- **Do not include interactive controls in your glance interface.**
- **Avoid tables and maps in your glance interface.** Although they are not prohibited, the limited space makes tables and maps less useful.
- **Be timely with the information you display.** Use all available resources, including time and location to provide information that matters to the user.
- **Use the system font for all text.** To use custom fonts in your glance, you must render that text into an image and display the image.

Tapping on a Glance

- Tapping anywhere on a glance should bring the user to a page in your app
- But, the Glance and the App run in separate sandboxes. Can't simply pass in the context.
- Use Handoff to communicate the context of the glance at the time of the tap

Handoff

- Handoff allows one executable to pass contextual info to another
- For example, a Glance passing context to a WatchKit App
- Handoff can be used to pass info from a WatchKit app to an iPhone app, an iPhone app to a iPad app, an iPad app to a Mac App, etc.

Handoff API (WKInterfaceController)

```
func updateUserActivity(_ type: String,  
                      userInfo userInfo: [NSObject : AnyObject]?,  
                      webpageURL webpageURL: NSURL?)
```

```
func handleUserActivity(_ userInfo: [NSObject : AnyObject]?)
```

```
func invalidateUserActivity()
```

In the Glance, call `updateUserActivity` to register the current activity. Call `updateUserActivity` again if the current activity changes, and call `invalidateUserActivity` to unregister the current activity.

`handleUserActivity` is called when entering the WatchKit app interface controller after tapping on the Glance.

updateUserActivity

For example, in your glance interface controller, call:

```
updateUserActivity("com.cs193w.flight-tracker.glance",  
userInfo: ["flight": "AA164"], webpageURL: nil)
```

type is a reverse-DNS-style string that represents the type of user activity. It is not really used for Glances; just choose something unique.

webpageURL is not relevant to glances since the Apple Watch doesn't have a browser. It can be used to open a webpage on the user's iPhone from an interface controller in the WatchKit app.

handleUserActivity

In your app, `handleUserActivity(userActivity: [AnyObject])` is called in your initial interface controller.

In a page-based interface, it is called on every interface controller that is part of your initial interface.

Do not call `super` in `handleUserActivity`.

Handing off to an iPhone

- To hand off to an iPhone, you must add an entry to the iPhone app's `Info.plist` with key `NSUserActivityTypes` that contains the activity types that the iPhone app supports.

Handoff Handlers in UIApplicationDelegate

When the user invokes Handoff, the system will call:

```
optional func application(_ application: UIApplication,  
willContinueUserActivityWithType userActivityType: String) -> Bool
```

followed by:

```
optional func application(_ application: UIApplication,  
    continueUserActivity userActivity: NSUserActivity,  
    restorationHandler restorationHandler: ([AnyObject]?) -> Void) -> Bool
```

In this method, call `restorationHandler` with an array of objects; the system will call `restoreUserActivityState:` on each of these objects.

Notifications on iOS

Notifications on iOS Devices

- Two types of notifications: *remote notifications* and *local notifications*
- Remote notifications are sent from a server
- Local notifications are sent from an iOS app

Three Aspects of Notifications

- Sounds/Vibrate: An audible alert plays.
- Alerts/Banners: An alert or banner appears on the screen.
- Badges: An image or number appears on the application icon.

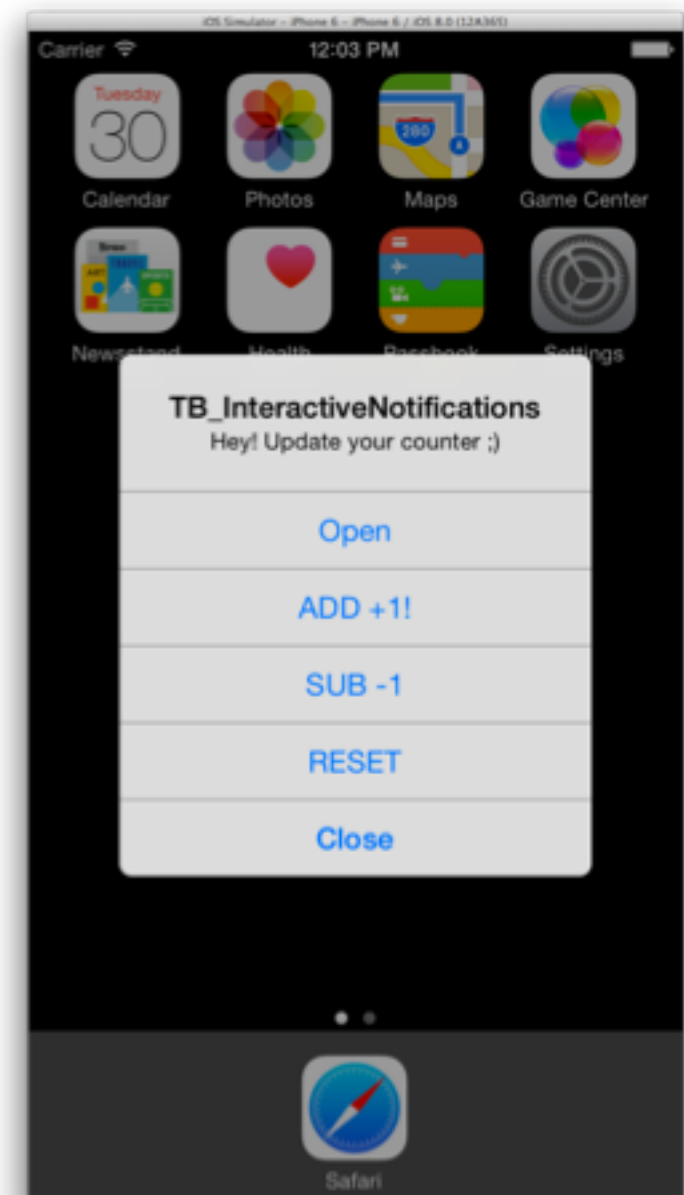
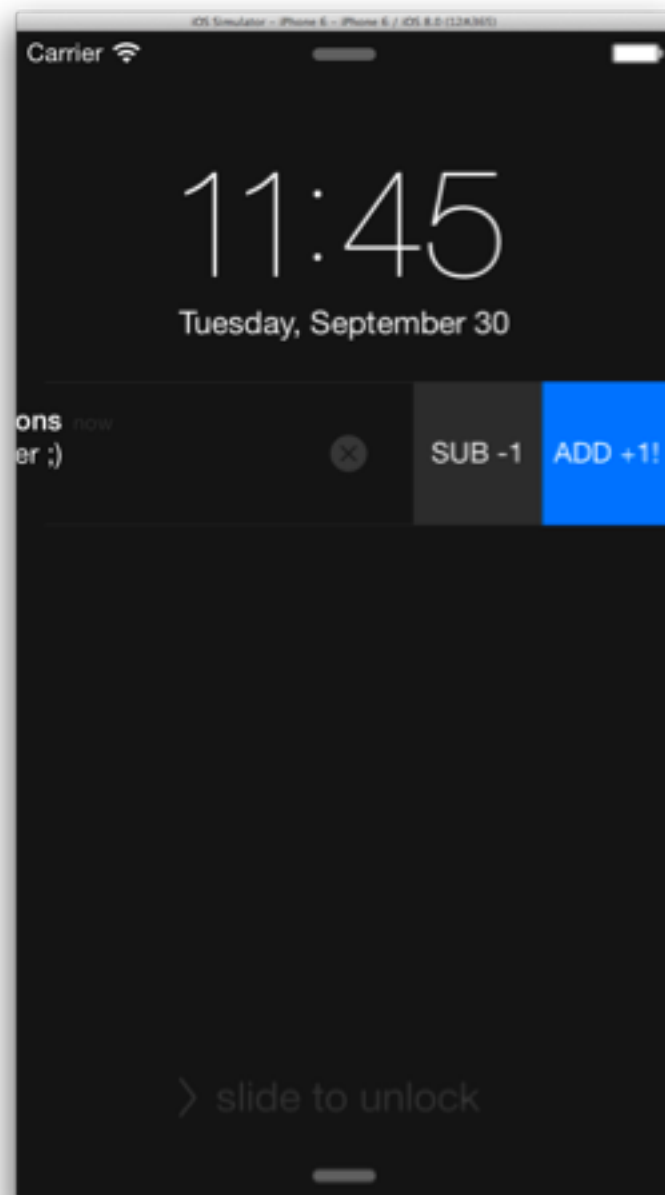
Registering for push notifications

- The user must explicitly opt in to push notifications (local and remote).

```
func application(application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [NSObject: AnyObject]?) -> Bool {
    // to register for notification types
    application.registerUserNotificationSettings(UIUserNotificationSettings(
forTypes: (.Badge
| .Sound | .Alert), categories: nil))
}
```

- The user will receive a prompt and can allow / deny push notifications for the app. He can later change this in the Settings app only.

Notification Actions



Notification Actions

- A *category* has a set of associated *actions*

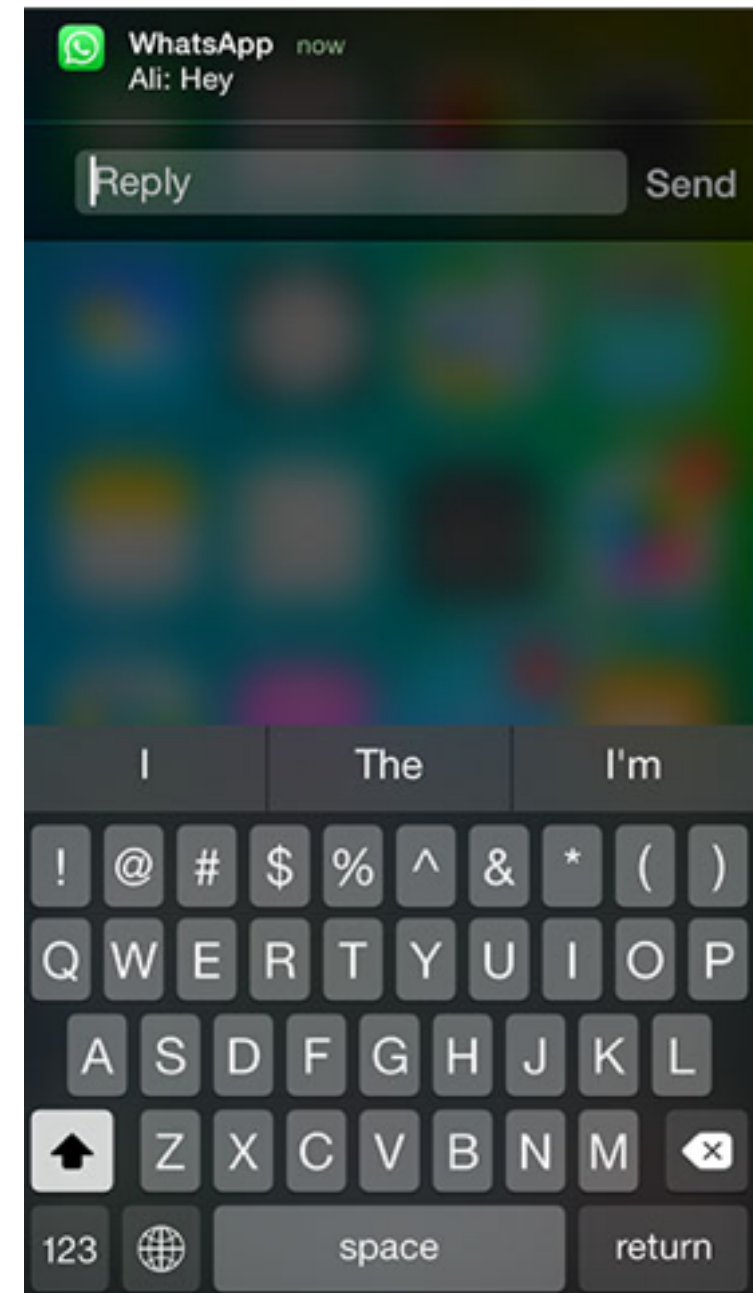
```
let acceptAction = UIMutableUserNotificationAction()  
acceptAction.identifier = "ACCEPT_ACTION"  
acceptAction.title = "Accept"  
acceptAction.activationMode = .Background  
acceptAction.destructive = false  
acceptAction.authenticationRequired = false
```

Registering a Category

```
let category = UIMutableUserNotificationCategory()  
category.identifier = "INVITE_CATEGORY"  
category.setActions([acceptAction, declineAction], forContext: .Minimal)  
category.setActions([acceptAction, declineAction, remindAction], forContext: .Default)  
  
application.registerUserNotificationSettings(UIUserNotificationSettings(forTypes: (.Badge  
| .Sound | .Alert), categories: [category]))
```

Inline Text Replies

```
let replyAction = UIMutableUserNotificationAction()  
replyAction.identifier = "REPLY_ACTION"  
replyAction.behavior = .TextInput
```



Scheduling a Local Notification

```
let notification = UILocalNotification()
notification.alertAction = "View"
notification.alertBody = "Bobby invited you to lunch."
notification.alertTitle = "Invite from Bobby" // Shown on Apple Watch
notification.fireDate = NSDate(timeIntervalSinceNow: 20)
notification.timeZone = NSTimeZone.localTimeZone()
notification.category = "INVITE_CATEGORY"

application.scheduleLocalNotification(notification)
```

Receiving a Local Notification

- App launched by notification:

```
optional func application(_ application: UIApplication,  
    willFinishLaunchingWithOptions launchOptions: [NSObject : AnyObject]?) -> Bool
```

```
optional func application(_ application: UIApplication,  
    didFinishLaunchingWithOptions launchOptions: [NSObject : AnyObject]?) -> Bool
```

launchOptions is a NSDictionary. Look at the
UIApplicationLaunchOptionsLocalNotificationKey

- App launched previously:

```
optional func application(_ application: UIApplication,  
    didReceiveLocalNotification notification: UILocalNotification)
```

Handling a notification action

```
optional func application(_ application: UIApplication,  
    handleActionWithIdentifier identifier: String?,  
    forLocalNotification notification: UILocalNotification,  
    completionHandler completionHandler: () -> Void)
```

Remote Notifications

- Similar to local notifications, but come from a server
- You register for remote notifications like so:

```
application.registerForRemoteNotifications()
```

You'll get a callback on one of:

```
optional func application(_ application: UIApplication,  
didRegisterForRemoteNotificationsWithDeviceToken deviceToken: NSData)
```

```
optional func application(_ application: UIApplication,  
didFailToRegisterForRemoteNotificationsWithError error: NSError)
```

Receiving Remote Notifications

```
optional func application(_ application: UIApplication,  
    didReceiveRemoteNotification userInfo: [NSObject : AnyObject],  
    fetchCompletionHandler handler: (UIBackgroundFetchResult) -> Void)
```

```
UIBackgroundFetchResult -> UIBackgroundFetchResultNoData,  
UIBackgroundFetchResultNewData, UIBackgroundFetchResultFailed
```

The application must call `handler()` within 30 seconds.

Or, look for `UIApplicationLaunchOptionsRemoteNotificationKey` in one of:

```
application:willFinishLaunchingWithOptions:  
application:didFinishLaunchingWithOptions:
```

Notifications on Apple Watch

3 Types of Notification Views on Apple Watch

1. Short look - shown briefly
2. Long look - show if user keeps looking at short look
 - a. Static long look
 - b. Dynamic long look

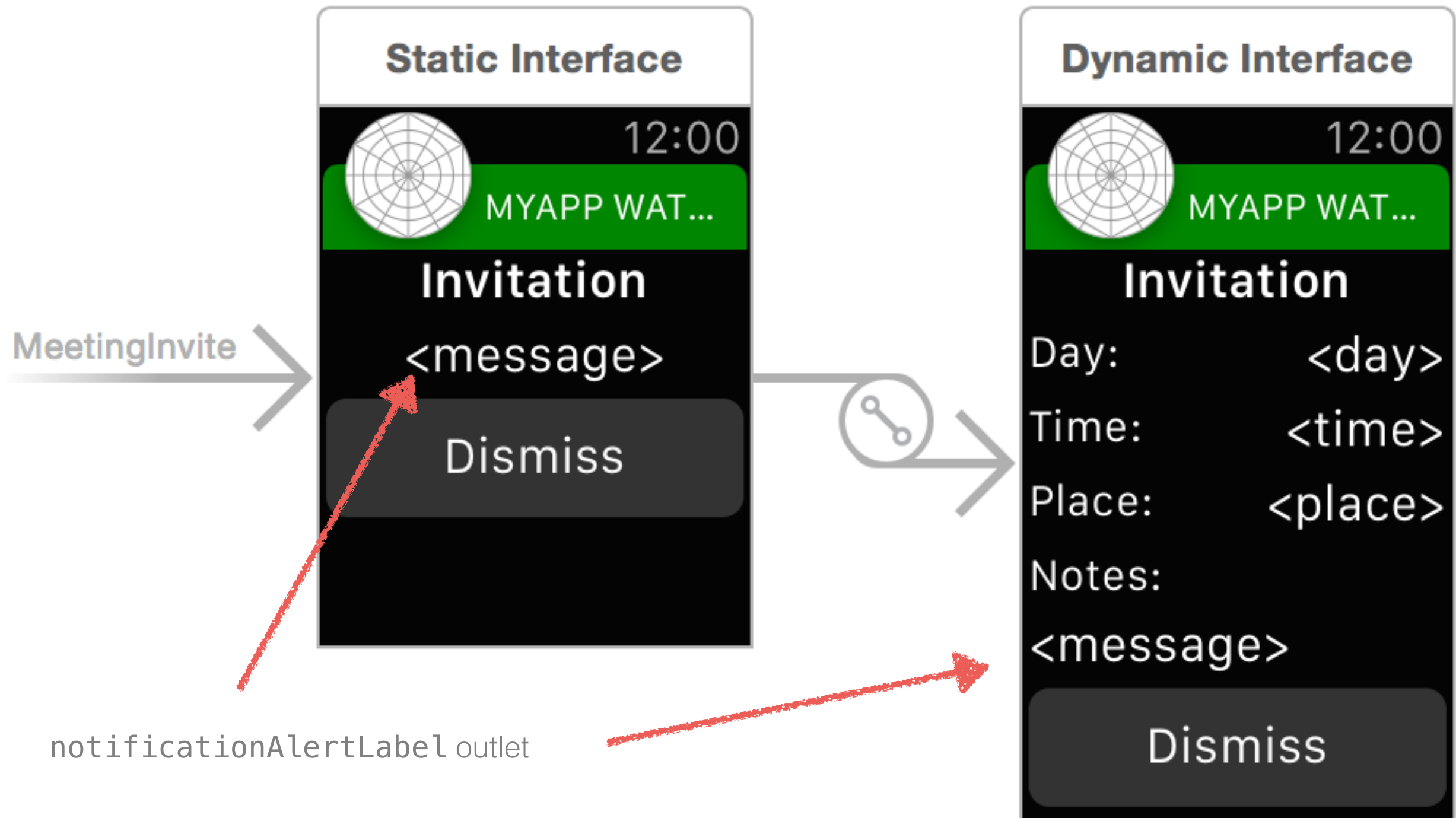
Short Look



Long Look



Static vs Dynamic Long Looks



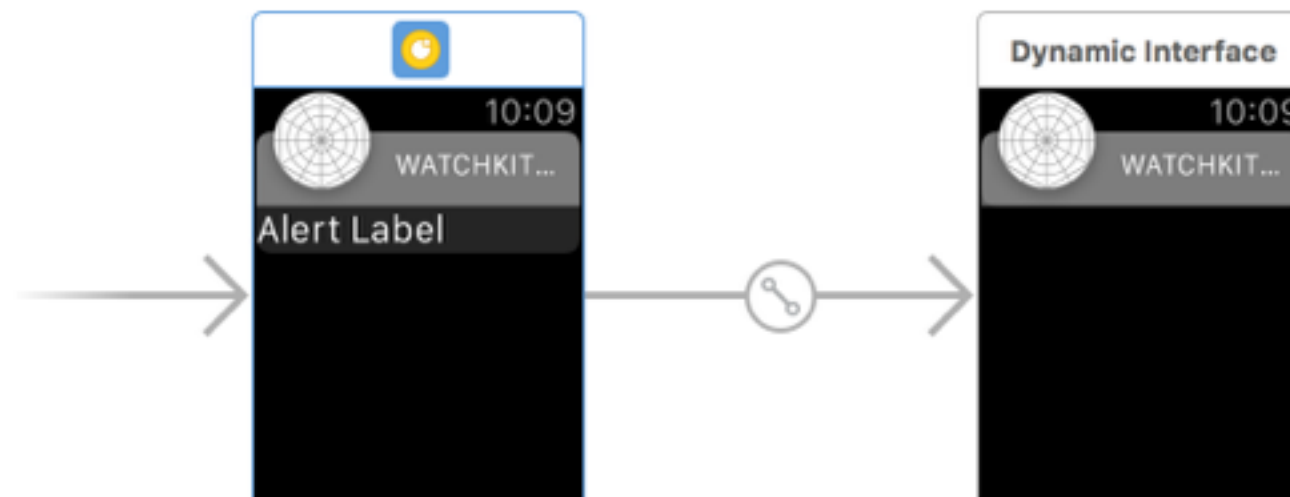
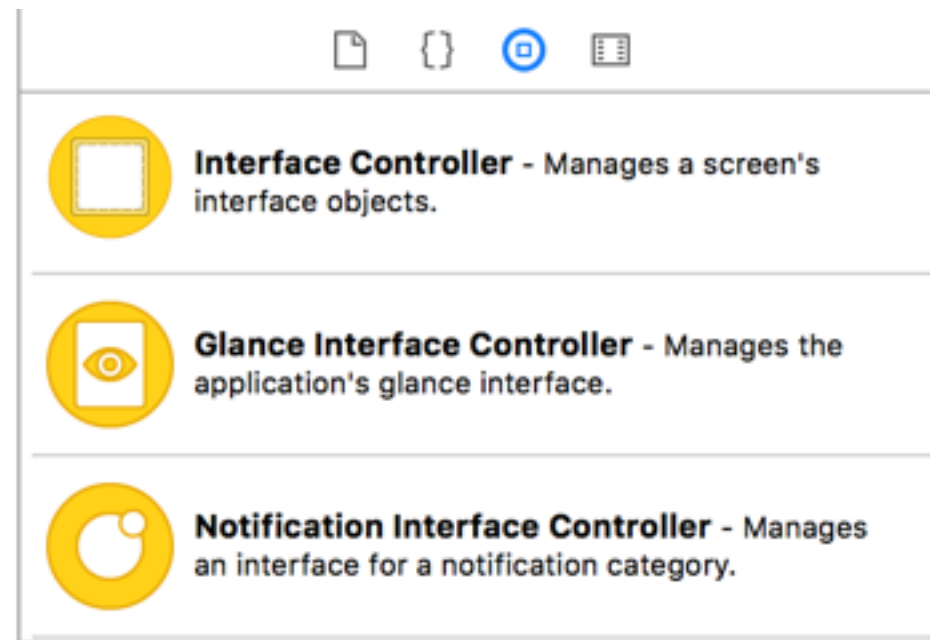
Static Notification Interfaces

- All images must reside in the WatchKit app bundle.
- The interface must not include controls, tables, maps, or other interactive elements.
- The interface's `notificationAlertLabel` outlet must be connected to a label. The label's contents are set to the notification's alert message. The text for all other labels does not change.

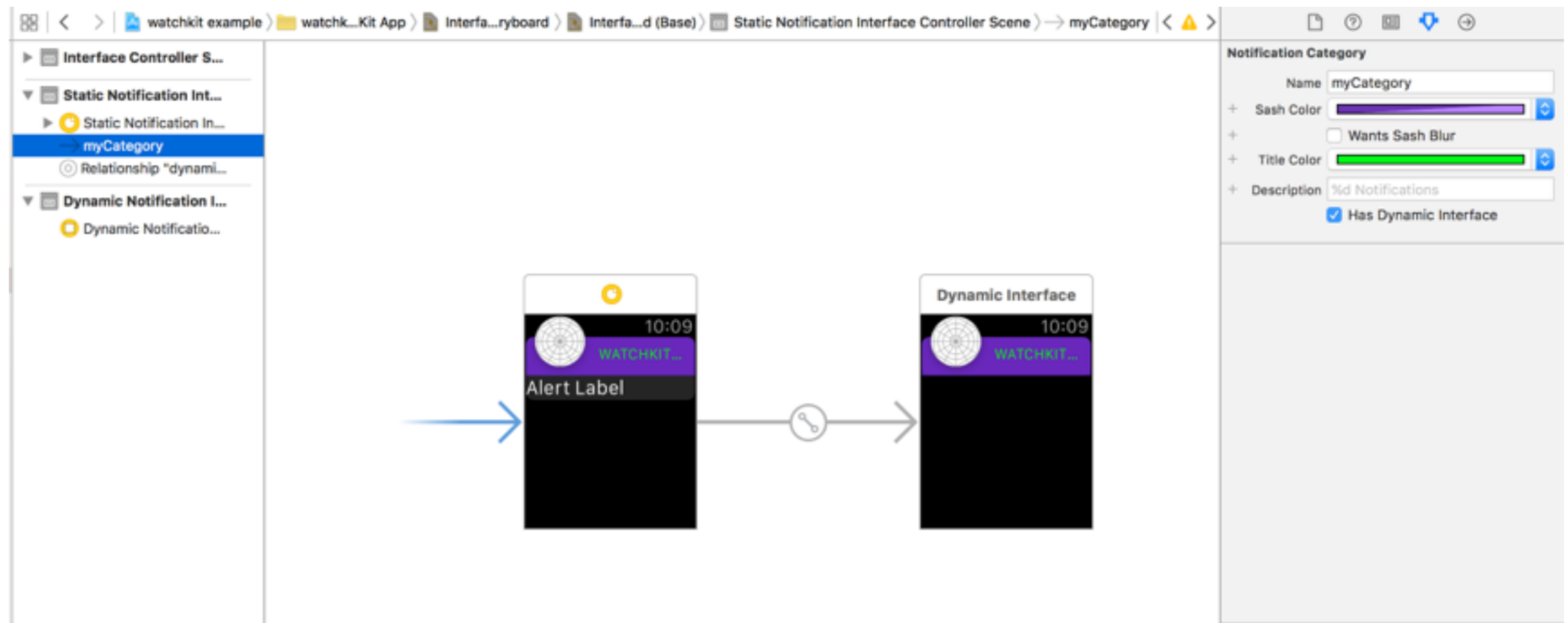
Dynamic Notification Interfaces

- Use labels, images, groups, and separators for most of your interface.
- Include tables and maps only as needed in your interface.
- Do not include buttons, switches, or other interactive controls.

Adding a Notification Interface Controller in the Storyboard

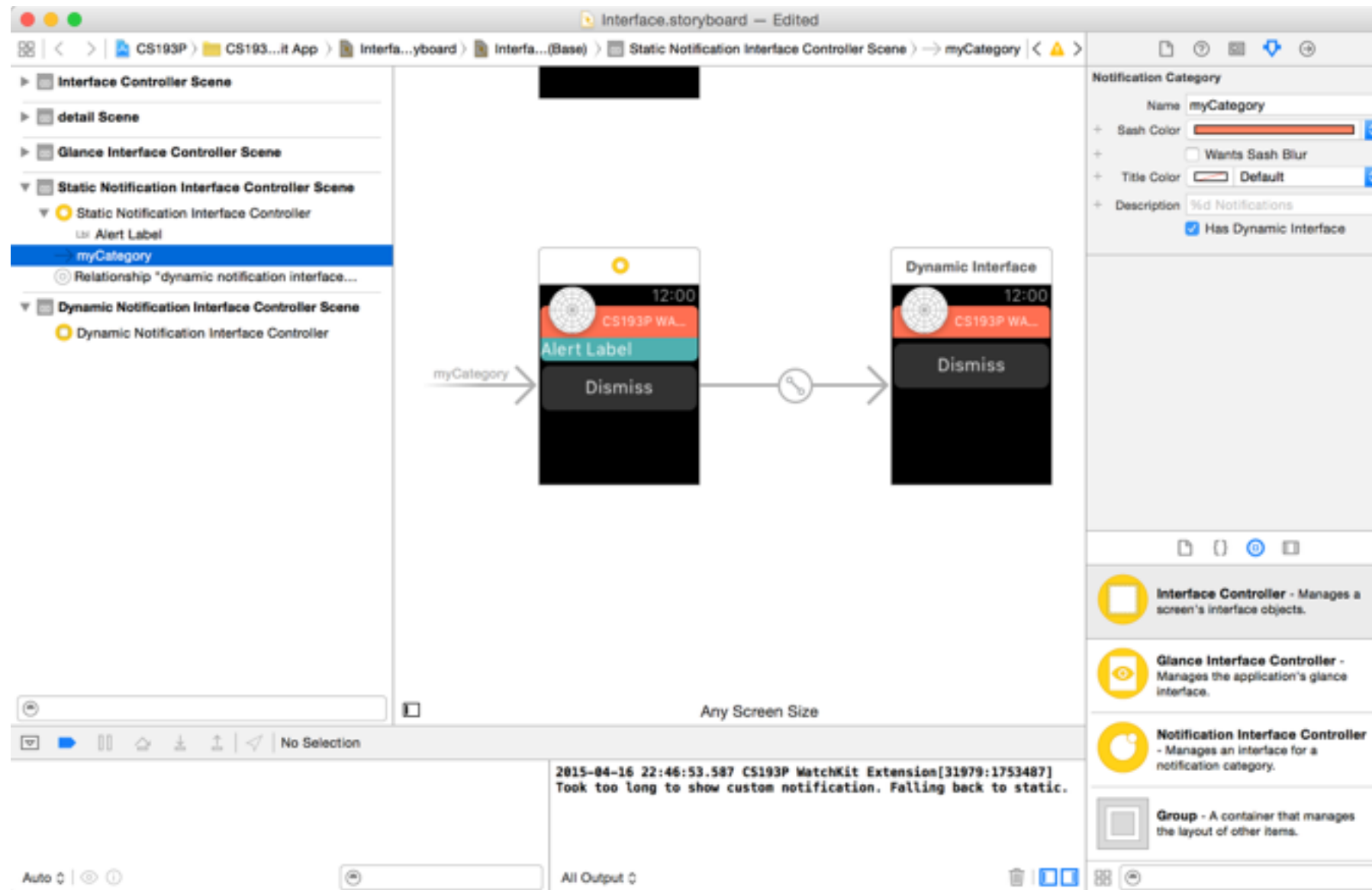


Sash and Title Color

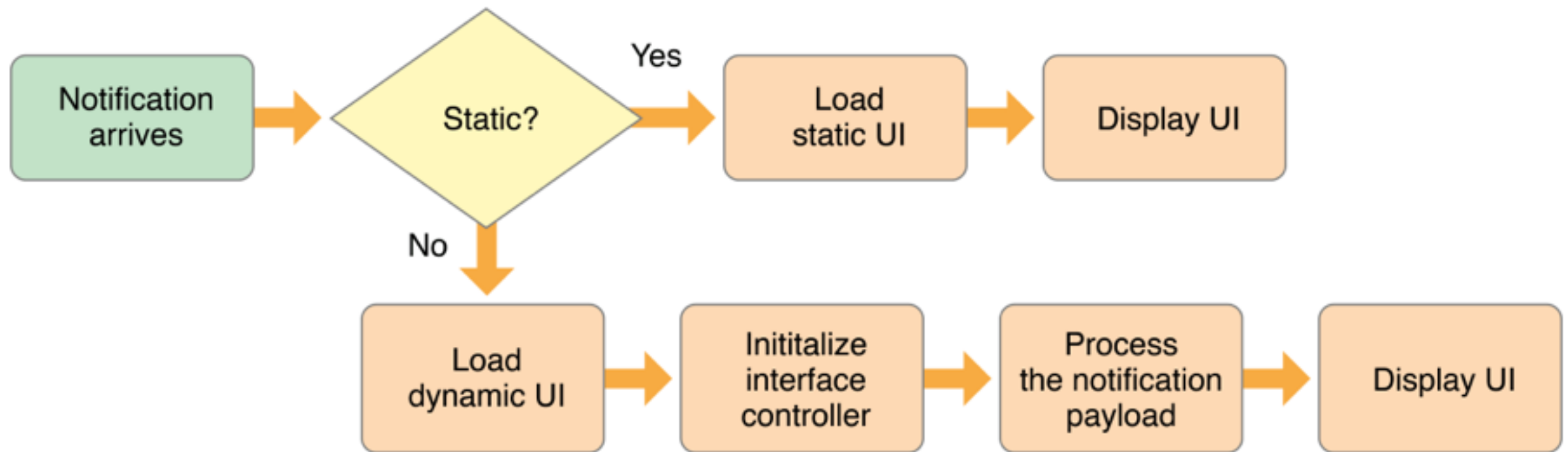


Note also the **Description**, which is a format string displayed when multiple notifications are received at once.

Assigning a Category to a Notification Interface



Preparing the notification interface



WKUserNotificationInterfaceController

- A subclass of `WKInterfaceController`
- Contains two additional methods:

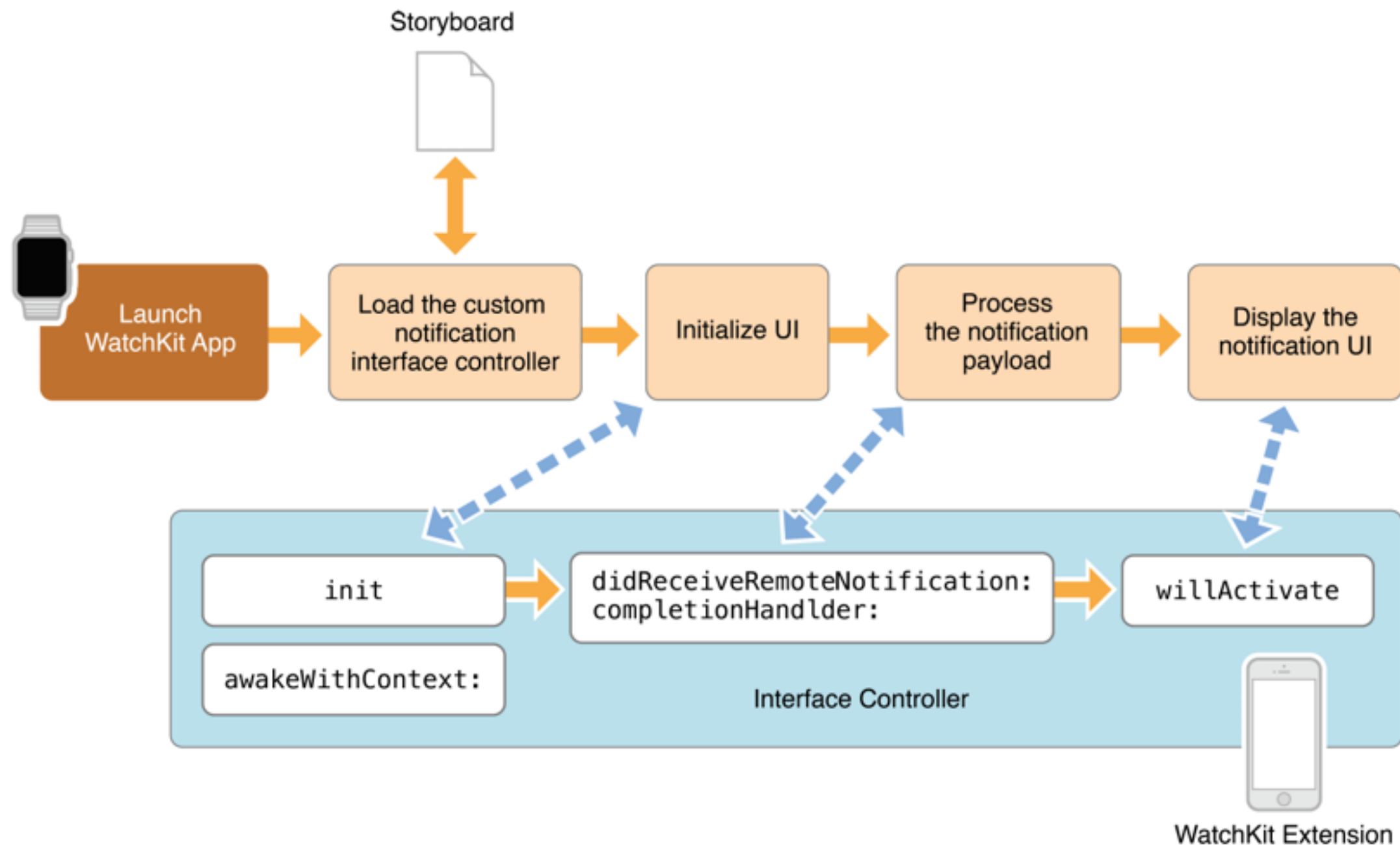
```
func didReceiveLocalNotification(_ localNotification: UILocalNotification,  
                                withCompletion completionHandler: (WKUserNotificationInterfaceType) -> Void)
```

```
func didReceiveRemoteNotification(_ localNotification: UILocalNotification,  
                                  withCompletion completionHandler: (WKUserNotificationInterfaceType) -> Void)
```

`WKUserNotificationInterfaceType` is either:

- .Default - Show the static interface
- .Custom - Show the dynamic interface

Configuring a Dynamic Notification Interface



Receiving Notifications While the Watch App is Running

In `WKExtensionDelegate`:

```
optional func didReceiveLocalNotification(_ userInfo: [NSObject : AnyObject])
```

```
optional func didReceiveRemoteNotification(_ userInfo: [NSObject : AnyObject])
```

Suggestions for Text Replies

Like a Text Input Controller, you can provide suggestions for text replies on Apple Watch:

```
func suggestionsForResponseToActionWithIdentifier(_ identifier: String,  
                                                forLocalNotification localNotification: UILocalNotification,  
                                                inputLanguage inputLanguage: String) -> [String]
```

```
func suggestionsForResponseToActionWithIdentifier(_ identifier: String,  
                                                forRemoteNotification localNotification: UILocalNotification,  
                                                inputLanguage inputLanguage: String) -> [String]
```

Handling Notification Actions in WKExtensionDelegate

```
optional func handleActionWithIdentifier(_ identifier: String?,  
                                       forLocalNotification localNotification: UILocalNotification)
```

```
optional func handleActionWithIdentifier(_ identifier: String?,  
                                       forLocalNotification localNotification: UILocalNotification,  
                                       withResponseInfo responseInfo: [NSObject : AnyObject])
```

```
optional func handleActionWithIdentifier(_ identifier: String?,  
                                       forRemoteNotification remoteNotification: [NSObject : AnyObject])
```

```
optional func handleActionWithIdentifier(_ identifier: String?,  
                                       forRemoteNotification remoteNotification: [NSObject : AnyObject],  
                                       withResponseInfo responseInfo: [NSObject : AnyObject])
```

responseInfo contains a `UIUserNotificationActionResponseTypedTextKey` which contains the user's quick reply response.

Debugging Notification Handlers

Testing Remote Notifications

- Make a `PushNotificationPayload.apns` file:

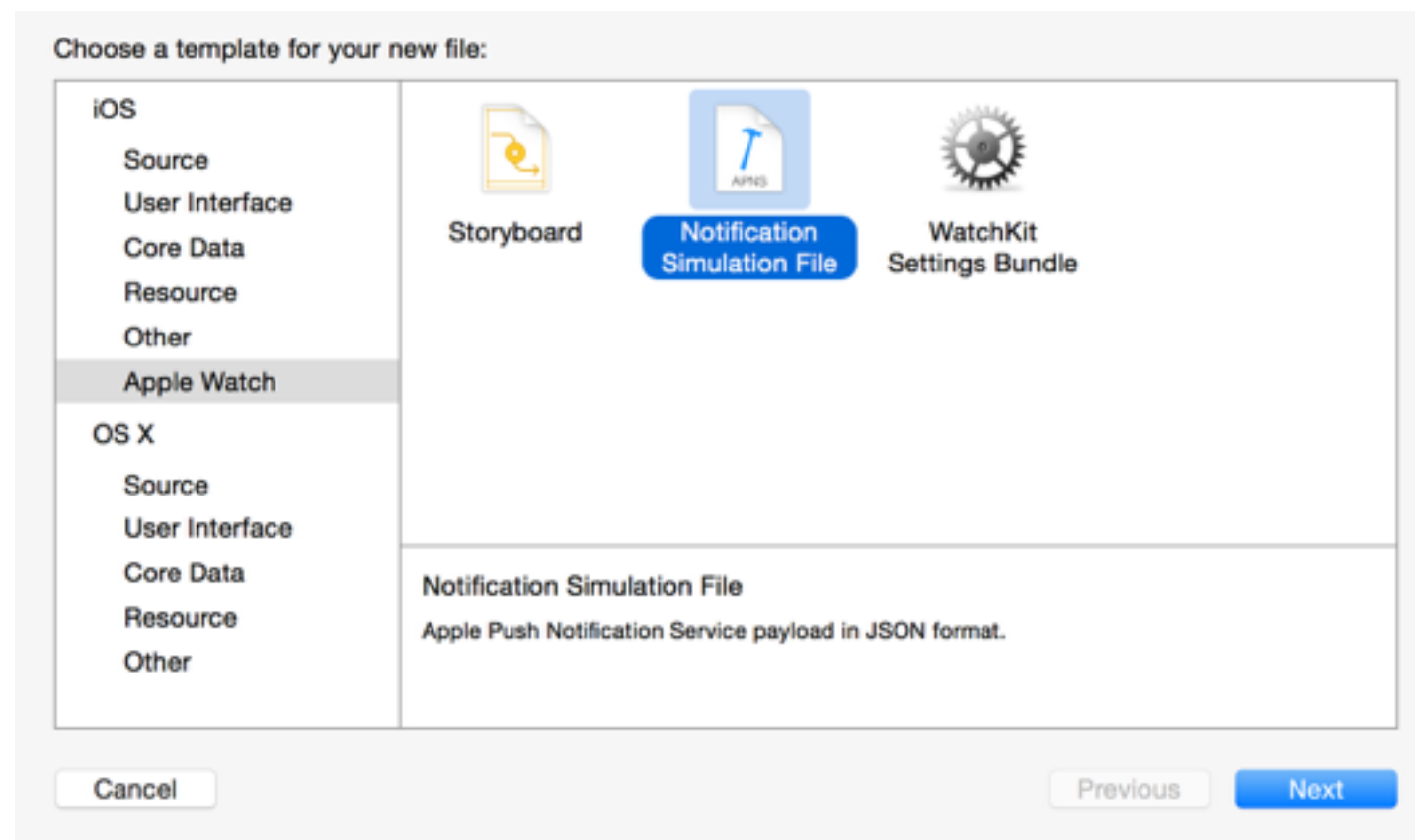
```
{
  "aps": {
    "alert": {
      "body": "Test message",
      "title": "Optional title"
    },
    "category": "myCategory"
  },
  "WatchKit Simulator Actions": [
    {
      "title": "First Button",
      "identifier": "firstButtonAction"
    }
  ],
```

`"customKey": "Use this file to define a testing payload for your notifications. The aps dictionary specifies the category, alert text and title. The WatchKit Simulator Actions array can provide info for one or more action buttons in addition to the standard Dismiss button. Any other top level keys are custom payload. If you have multiple such JSON files in your project, you'll be able to select them when choosing to debug the notification interface of your Watch App."`

```
}
```

How to make a .apns File

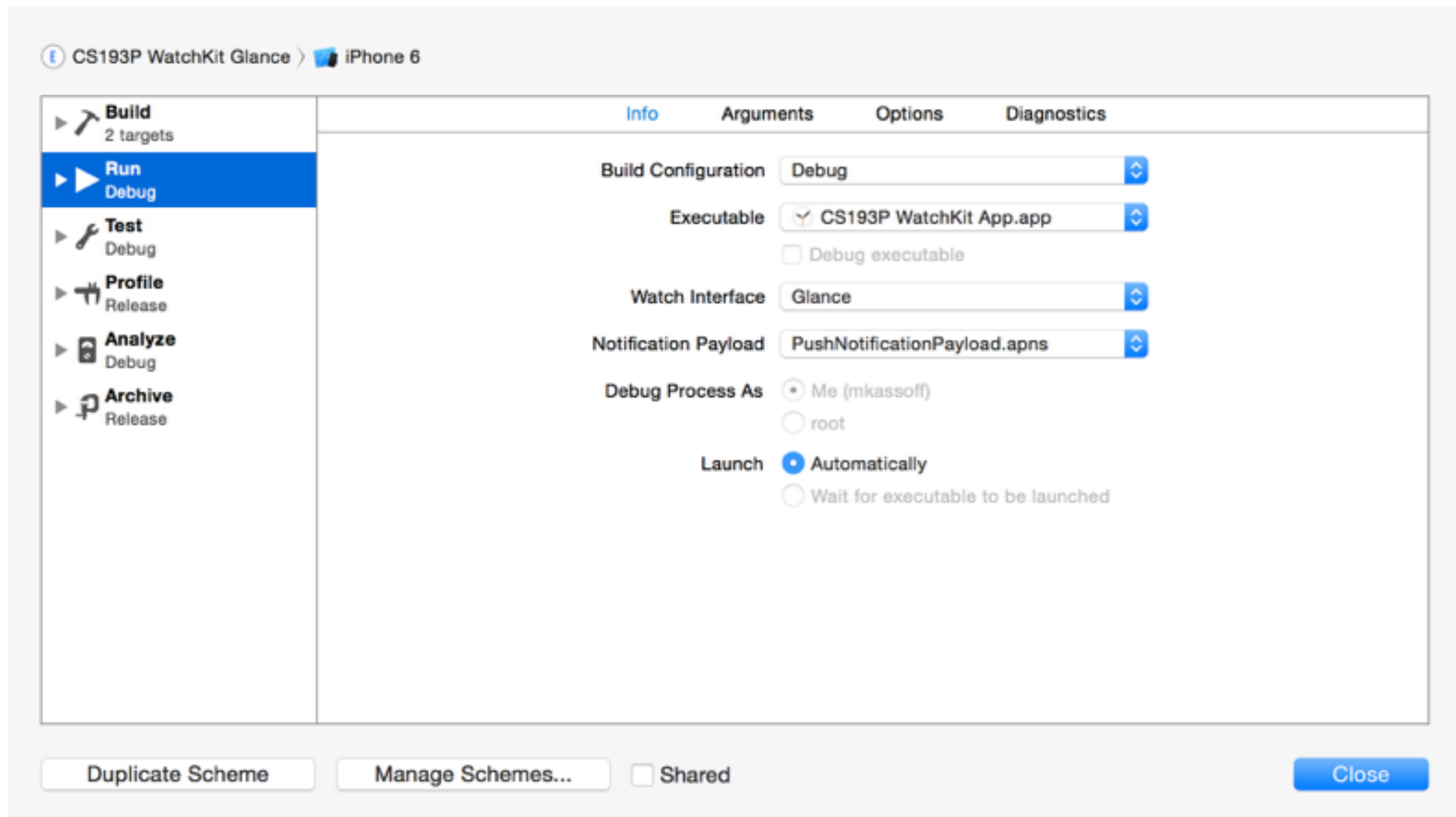
- File -> New -> File...



Does not need to be in a target.

Choosing a .apns file

Go to Product -> Scheme -> Edit Scheme...



Debugging Local Notifications

- One strategy for debugging local notifications is to fire off a local notification on a timer once the app has gone to the background
- e.g. use `dispatch_after` in `applicationWillResignActive:`