THE UNIVERSITY OF CHICAGO

WATCH APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 5

UIKIT

WATCHKIT APPS

INSPIRATION

WATCHKIT APPS

WATCHKIT APPS

WATCHKIT APPS

WATCHKIT APPS

USER INTERACTION ELEMENTS

WATCHKIT APPS

SUBTITLE

- Very familiar collection of interface objects
- Objects are just a proxy for the views in your storyboard
  - MVC to the extreme

WKInterfaceController	UIViewController
WKUserNotificationInterfaceController	UIApplicationDelegate + UIAlertController
WKInterfaceDevice	UIDevice
WKInterfaceObject	UIView
WKInterfaceButton	UIButton
WKInterfaceDate	UILabel + NSDateFormatter
WKInterfaceGroup	UIScrollView
WKInterfaceImage	UIImageView
WKInterfaceLabel	UILabel
WKInterfaceMap	MKMapView
WKInterfaceSeparator	UITableView.separatorColor / .separatorStyle
WKInterfaceSlider	UIStepper + UISlider
WKInterfaceSwitch	UISwitch
WKInterfaceTable	UITableView
WKInterfaceTimer	UILabel + NSDateFormatter + NSTimer

WATCHKIT APPS

Labels

Labels use static text to convey short messages and are one of the most common elements in your app. They can span multiple lines and can be updated programmatically.

When you design a label, focus first on legibility. Use lighter colors for label text and use Dynamic Type to ensure that the text is sized appropriately. The built-in text styles offer the best legibility and automatically support Dynamic Type. If custom typefaces are necessary, avoid using ones that are overly stylized.

See Typography for more information about using text in an Apple Watch app.

WATCHKIT APPS

Images

An image element displays a single image or an animated sequence of images. Images may be in any format supported by iOS, but the preferred format is PNG. Animated sequences can be started and stopped programmatically.

Size images for 38mm and 42mm displays, respectively. You can use a single image resource for both display sizes as long as it maintains clarity.

Create images at 2x resolution. Apple Watch has a Retina display, so there is no need to create standard resolution images. Include "@2x" in image names.

WATCHKIT APPS

Groups

Groups can help you lay out other elements, such as images and labels. A group has attributes for specifying position, size, margins, and other layout-related properties. It also has a background image, color, and corner radius that can transform it into a visual element.

Use groups to arrange items horizontally or vertically. All items in a group are laid out in the same horizontal or vertical line. Use the group's inter-item spacing attribute to add space between items. Use the group's margins to add space between the group and its surrounding items.

Nest groups to mix horizontal and vertical content. You can use nesting to achieve specific layouts for your content. For example, you can place multiple vertical groups inside a single horizontal group.

WATCHKIT APPS

Tables

Tables present rows of content in a single column. A table row is dynamically configurable and its contents can be changed at any time. Tables are inherently scrollable, support various interactions, and can be assigned a background color or image.

Use row types consistently. Although a table can contain multiple row types, it should present a consistent overall appearance. Start with the row type you use for the table content, and add more as needed to support other types of content, such as headers or footers. Always use each row type for its intended purpose. For example, don't display content in a row type designed for headers or footers.

Limit the number of table rows to about 20. Display the most important rows at the top and let the wearer view more on demand. Fewer rows are easier to scan quickly.

Don't embed tables inside groups. Tables resize dynamically based on the number of rows they contain; they ignore height restrictions placed on them by groups.

WATCHKIT APPS

Buttons

A button performs an app-specific action. Buttons have customizable backgrounds and rounded corners. They can contain a label, an image, or a group object. The standard corner radius for a button is six points.

The background of the button is called the platter. You can assign a color or an image to the platter to change its appearance.

Create buttons that span the width of the screen. Full-width buttons look better and are easier to tap. If two buttons must share the same horizontal space, use the same height for both and use images or short text labels for each button's content.

Use the same height for vertical stacks of one- and two-line text buttons. As much as possible, use identical button heights for visual consistency. However, for buttons that contain dramatically different content, using different heights can improve the look of individual buttons.

Use the standard corner radius. It promotes a consistent visual style across apps and reinforces the intent of buttons.

WATCHKIT APPS

Switches

A switch lets people choose between two mutually exclusive choices or states, such as yes/no. A label must always be displayed next to a switch to specify which property the switch affects.

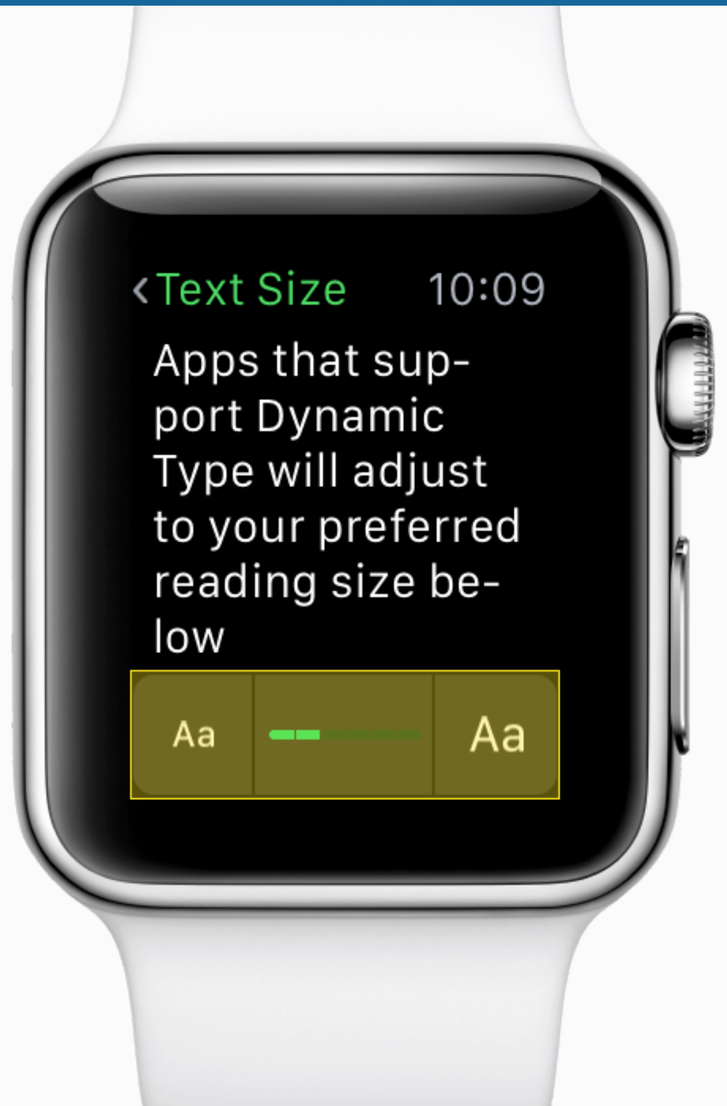
WATCHKIT APPS

Sliders

A slider lets people make adjustments to a value by tapping images at either end of the slider bar.

A slider displays its value as a set of steps or as a continuous bar. It always increases and decreases its value by a predetermined amount. Sliders are not meant to display numerical values.

Use custom images to communicate what the slider does. The system displays plus and minus signs by default.

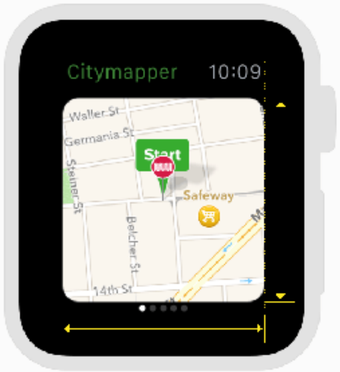


WATCHKIT APPS

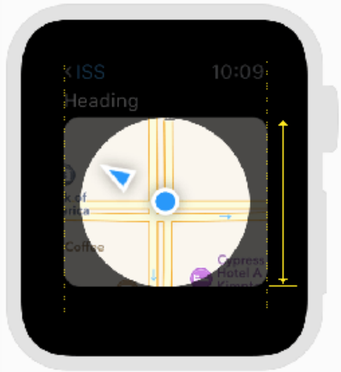
Maps

Maps are static snapshots of geographic locations. You place a map in your interface at design time, but you must configure the displayed region at runtime. The displayed region is not interactive, but tapping a map will open the Maps app on Apple Watch.

You can annotate a map to highlight points of interest or other relevant information. A map can display the standard green, red, and purple pins. It can also display custom images. The system lets you add up to five annotations to a map.



Size the map element to fit the screen. The wearer should be able to see the entire map element on the Apple Watch display without scrolling the screen.



Configure the displayed map region to be the smallest area that encompasses the points of interest. The contents of the map element itself do not scroll, so all intended content must be enclosed by the specified map region.

WATCHKIT APPS

Date & Timer Labels

Date and timer labels display real-time values on Apple Watch.

A date label displays the date, the time, or a combination of both. It can be configured to use a variety of formats, calendars, and time zones. After you configure it, a date label updates its value without further input from your app.

A timer label displays a precise countdown or count-up timer. It can be configured to display its count value in a variety of formats. After you configure it, a timer label counts down or up without further input from your app.

WATCHKIT APPS

Menus

Firm presses on the Apple Watch display cause the current screen's menu (if any) to appear. A menu can display up to four relevant actions for the current screen without taking away space from your interface.

Include a menu when the current screen has relevant actions. Menus are optional. If no menu is present, the system plays an animation when the wearer presses firmly on the display.

Use a label and an icon to convey the purpose of each menu action. Both the label and the icon are required. Labels are limited to two lines, so the text should be short.

For informati

Play

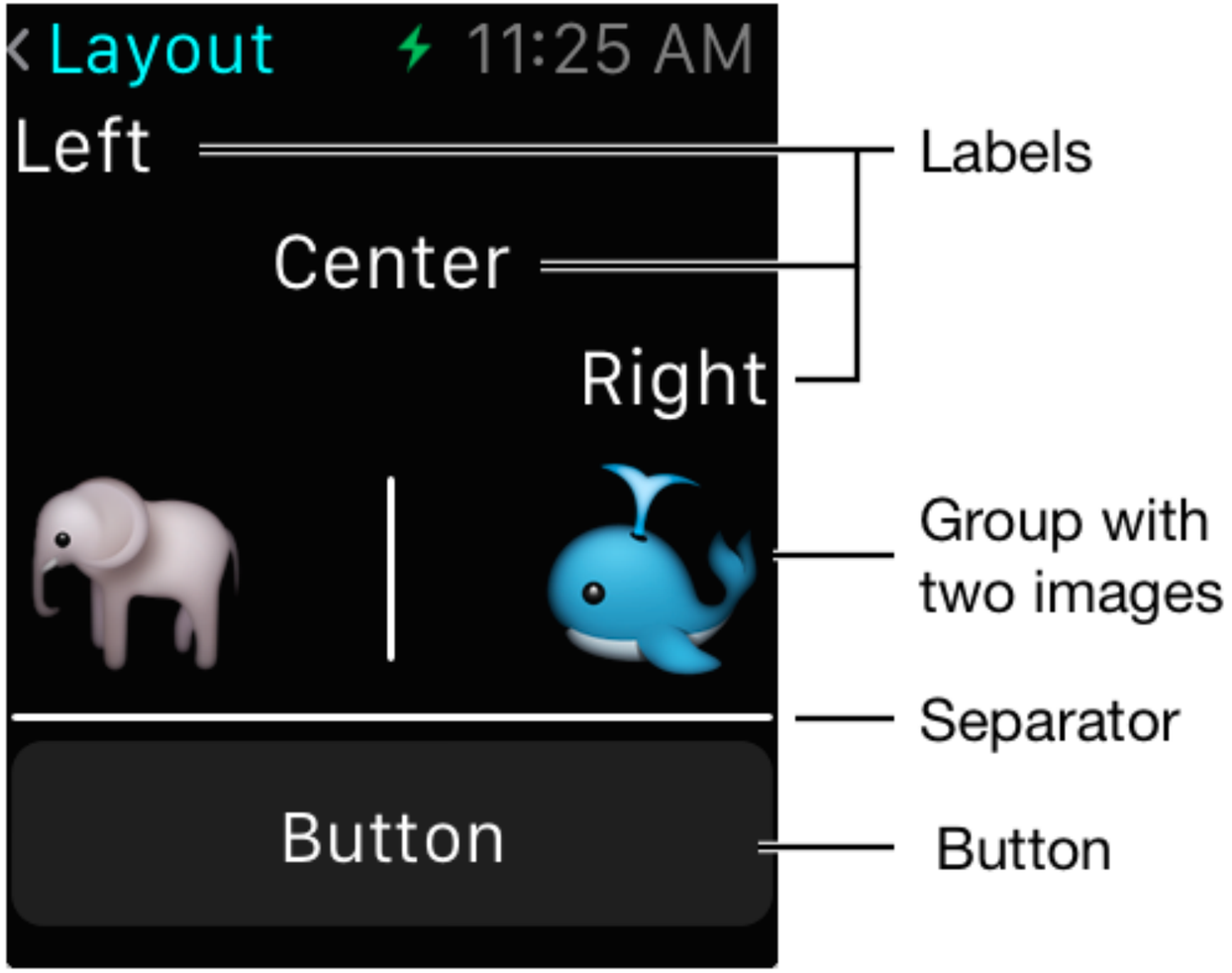
No visual design in IB; need to toggle presses in simulator

UI ESSENTIALS

WATCHKIT APPS

SUBTITLE

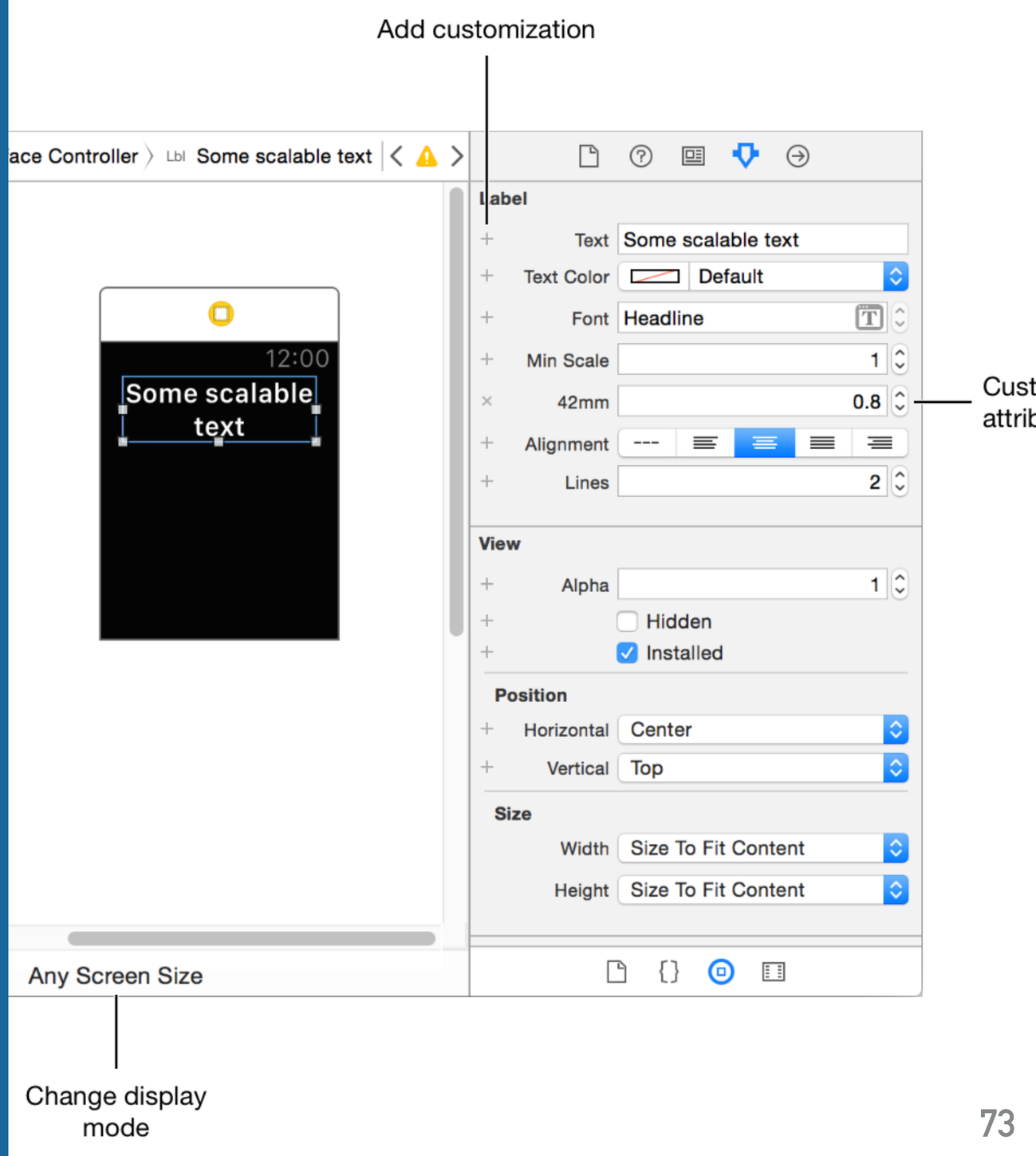
- WatchKit uses a different layout model
  - Xcode arranges items for you
  - Highly constrained, yet liberating
- Interface groups are used to define more complex layouts



WATCHKIT APPS

SUBTITLE

- What's 4mm?
  - Customize interface based on device (if you want)
  - Apple recommends same interface for all watches



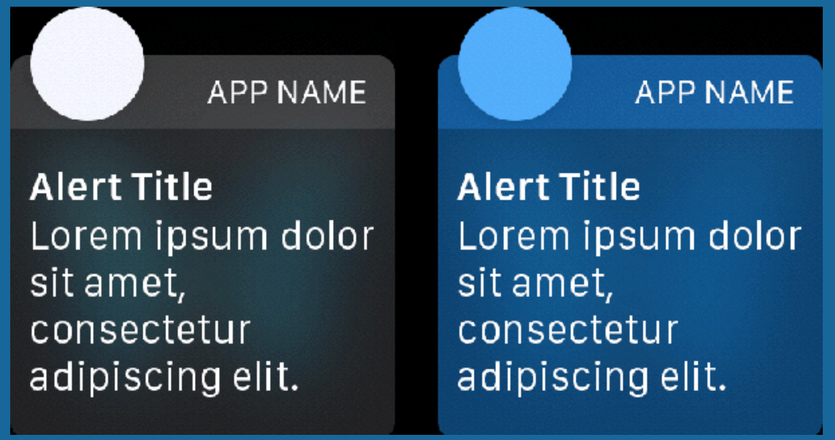
WATCHKIT APPS

- At runtime, an interface controller can make the following modifications to the objects in its corresponding storyboard scene:
  - Set or update data values
  - Change the visual appearance of objects that support such modifications
  - Change the size of an object
  - Change the transparency of an object
  - Show or hide an object

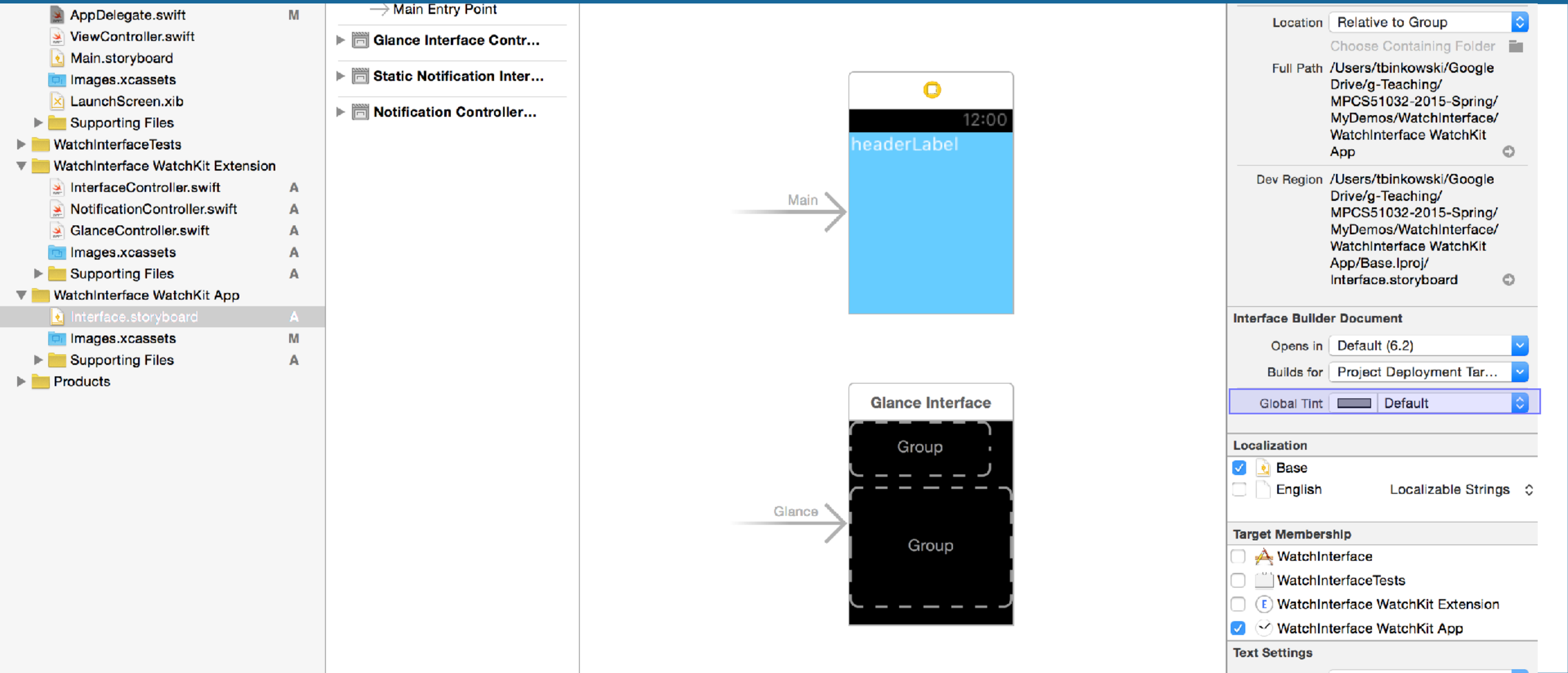
- Note: Objects have no getters, you need to keep track of state on your own

WATCHKIT APPS

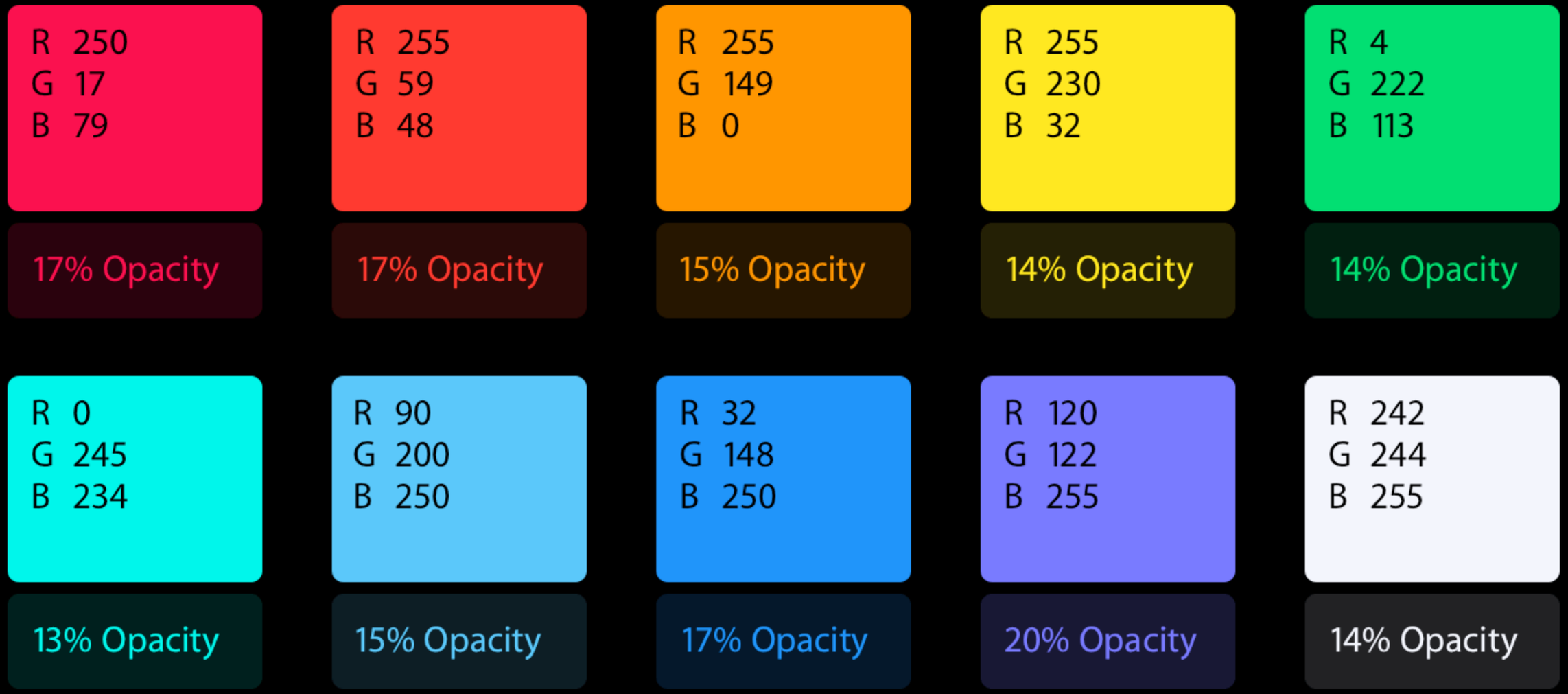
- Every WatchKit app has an associated key color, which is applied to the following UI elements
  - The title string in the status bar
  - The app name in short-look notifications
- An app's key color is stored in the Global Tint property of an app's storyboard



WATCHKIT APPS



WATCHKIT APPS

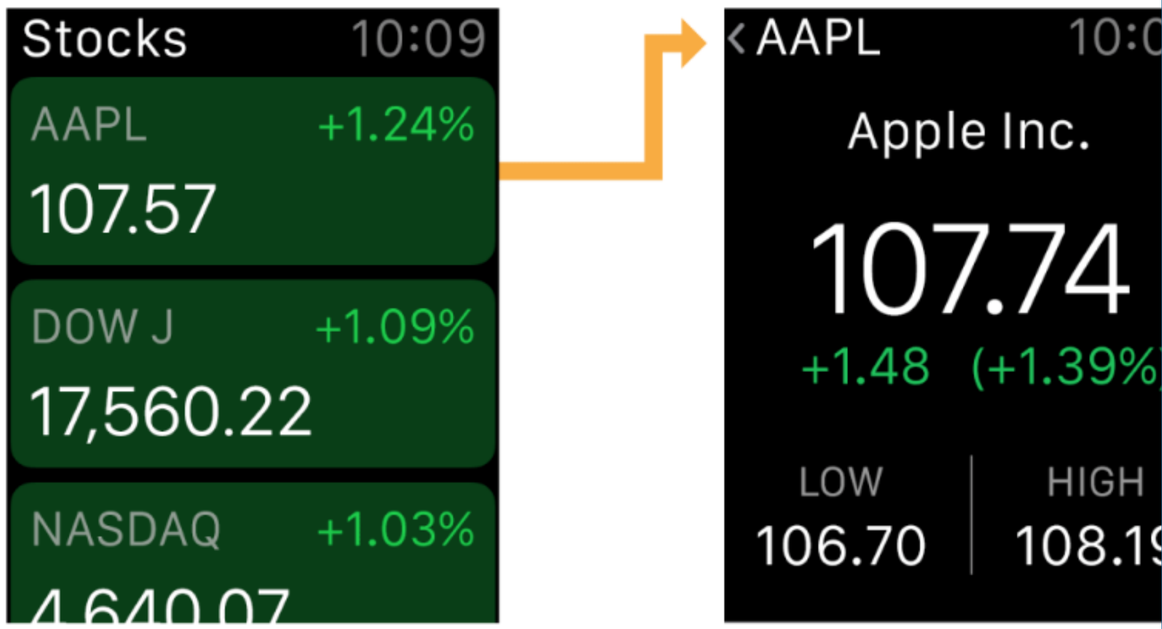


INTERFACE NAVIGATION

WATCHKIT APPS

Page-Based

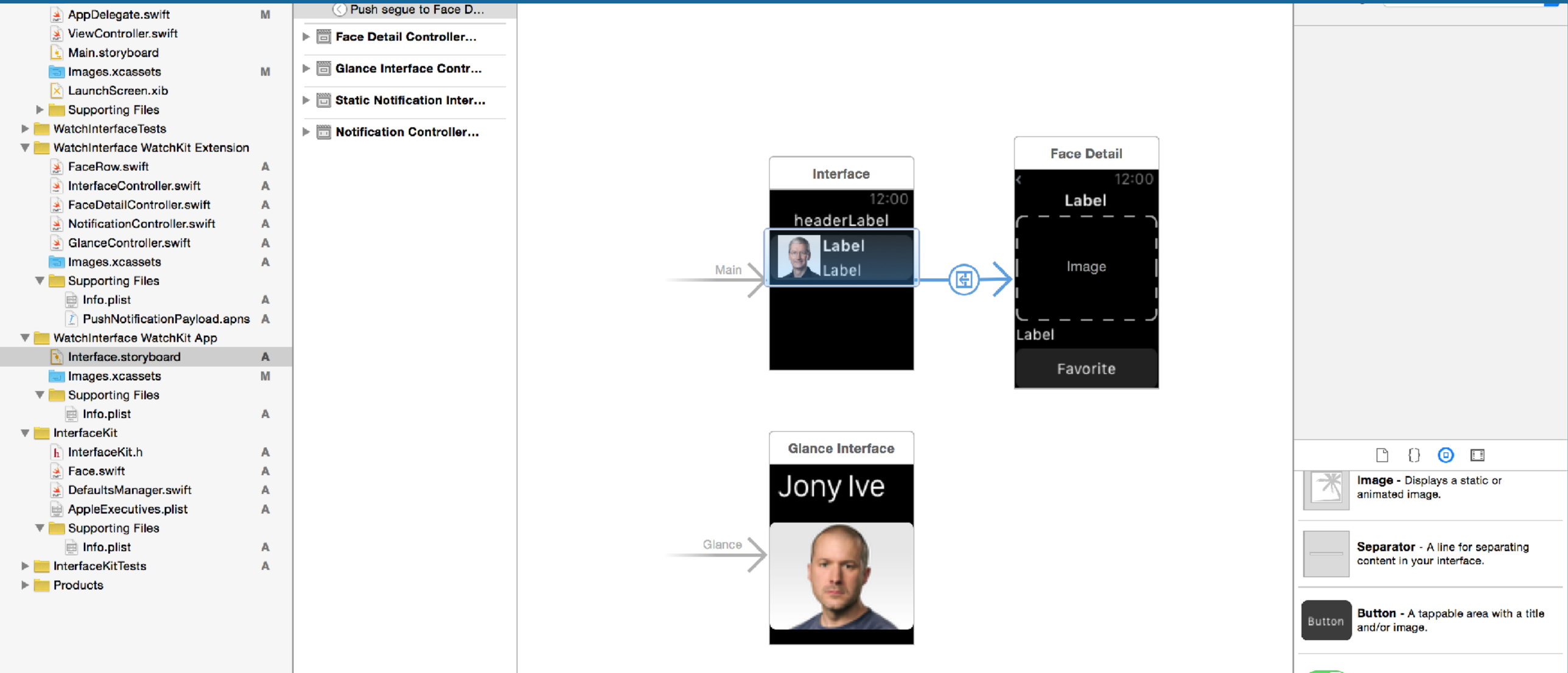
Hierarchical



WATCHKIT APPS

- Implementing a Page-Based Interface
  - reloadRootControllersWithNames:contexts:
  - becomeCurrentPage
- Implementing a Hierarchical Interface
  - pushControllerWithName:context:
  - popController
- Presenting Interface Controllers Modally
  - presentControllerWithName:context:
  - presentControllerWithNames:contexts:

WATCHKIT APPS



WATCHKIT APPS

```
49
50      // MARK: - Notifications
51      override func handleActionWithIdentifier(identifier: String?, forRemoteNotification
            remoteNotification: [NSObject : AnyObject]) {...}
58
59      // MARK: - Handoff Support
60      override func handleUserActivity(userInfo: [NSObject : AnyObject]?) {...}
63
64      // MARK: - Table Support
65      func reloadTable() {...}
81
82
83      // MARK: - Seuge
84      override func contextForSegueWithIdentifier(segueIdentifier: String, inTable table:
            WKInterfaceTable, rowIndex: Int) -> AnyObject? {
85
86          if segueIdentifier == "FaceDetailSegue" {
87              let name = faces[rowIndex].name
88              return name
89          }
90          return nil
91      }
```

WATCHKIT APPS

```
//

import Foundation
import WatchKit
import InterfaceKit

class FaceDetailController: WKInterfaceController {

    /** Store the name explicitly since we can't get it from the labels */
    var cachedName: String = ""

    // MARK: - Outlets and Actions
    @IBOutlet weak var name: WKInterfaceLabel!
    @IBOutlet weak var image: WKInterfaceImage!
    @IBOutlet weak var positionTitle: WKInterfaceLabel!

    /**
        Add current user as favorite
    */
    @IBAction func tapFavorite() {...}

    // MARK: - Lifecycle
    override func awakeWithContext(context: AnyObject?) {
        super.awakeWithContext(context)
        cachedName = context as! String
    }
```

You define the context

THE UNIVERSITY OF CHICAGO

WATCH APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 5