# Practical 02: First iOS App in Swift

*In this lab, we will walk you through the basics of creating a new project and the main parts of the Xcode interface. This chapter will familiarize you with the main tools of iOS development by showing you tutorial on how to connect your controls created in your interface to your Swift classes written using Xcode.*

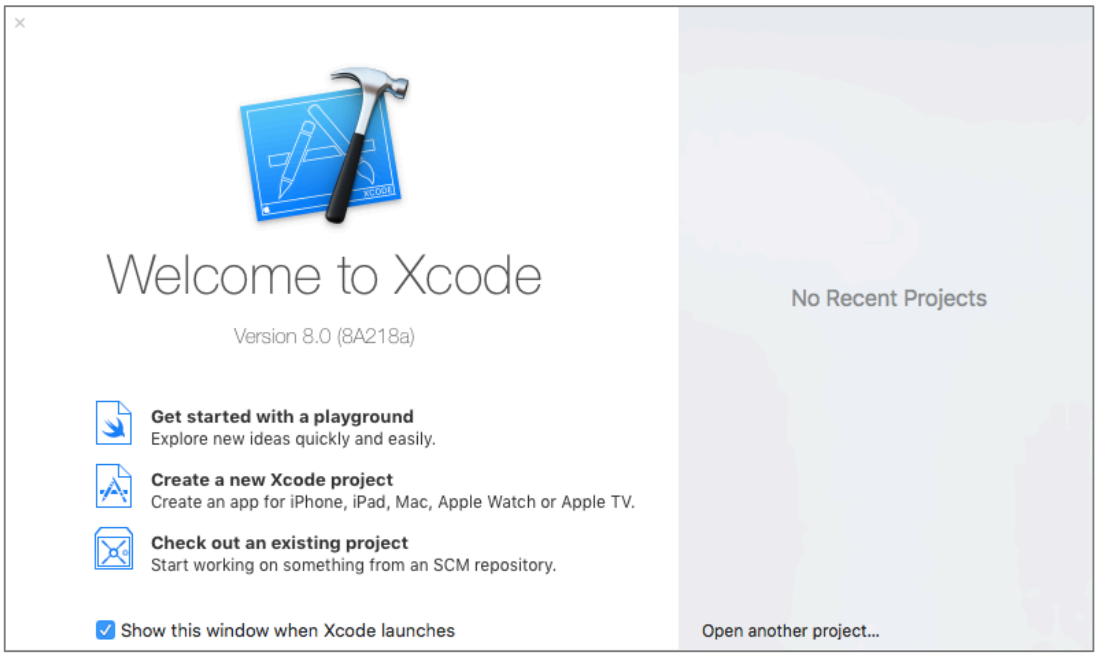
*This practical also makes use of various advanced topics we learnt in this lecture:*

*a. Optional chaining*
*b. Weak variables*
*c. Closures*

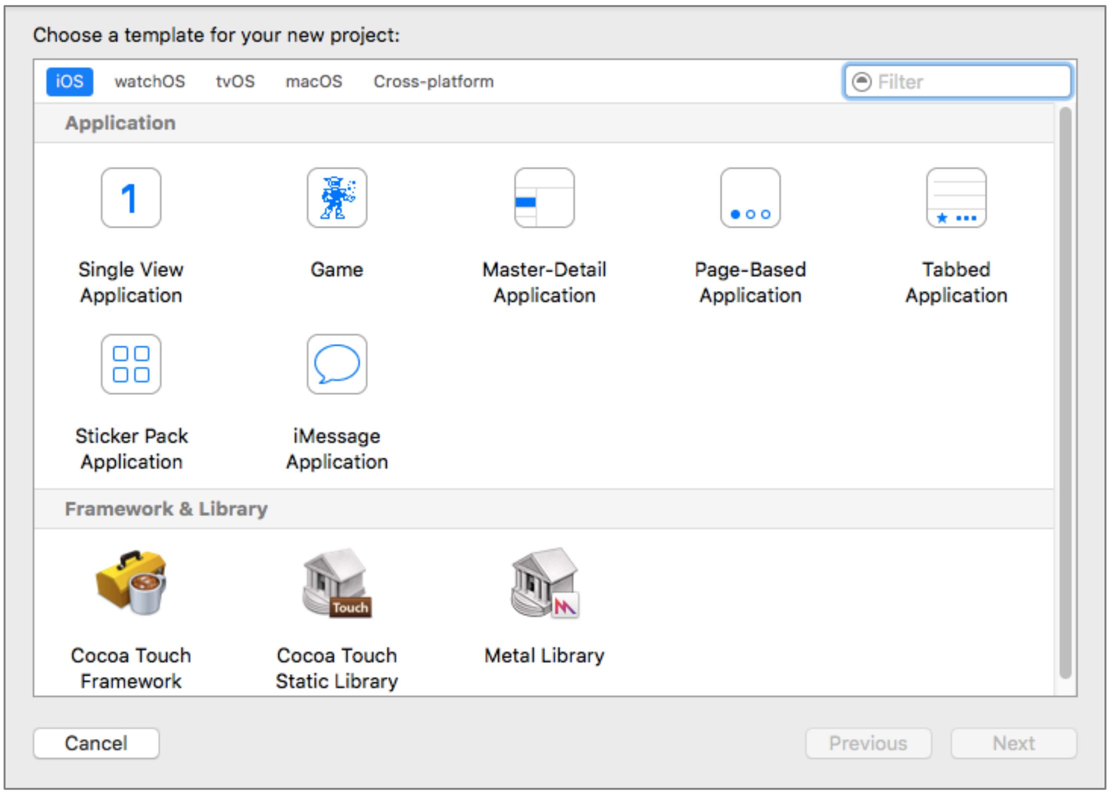
*We will have more hands-on on delegates in subsequent practicals.*

## Section 1: Working with Interface Builder

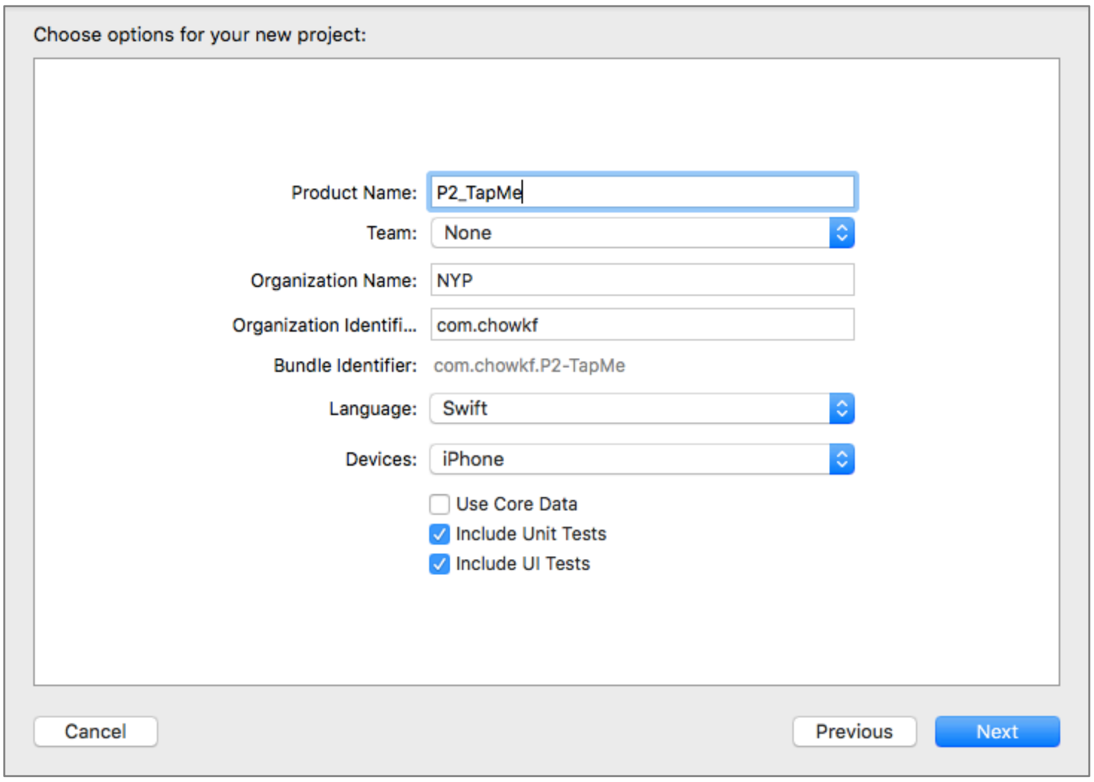
1. When you first launch Xcode, you are presented with a welcome screen as shown below.


Welcome to Xcode
Version 8.0 (8A218a)
Get started with a playground
Explore new ideas quickly and easily.
Create a new Xcode project
Create an app for iPhone, iPad, Mac, Apple Watch or Apple TV.
Check out an existing project
Start working on something from an SCM repository.
Show this window when Xcode launches
No Recent Projects
Open another project...

2. To create a new iPhone project, click **Create a new Xcode project**.
3. As an iOS developer, you are interested only in the **iOS** section of the List. The Xcode templates contain the files you need to quickly start on a new development effort.
   For this lab, select the **Single View Application** template and click **Next**.

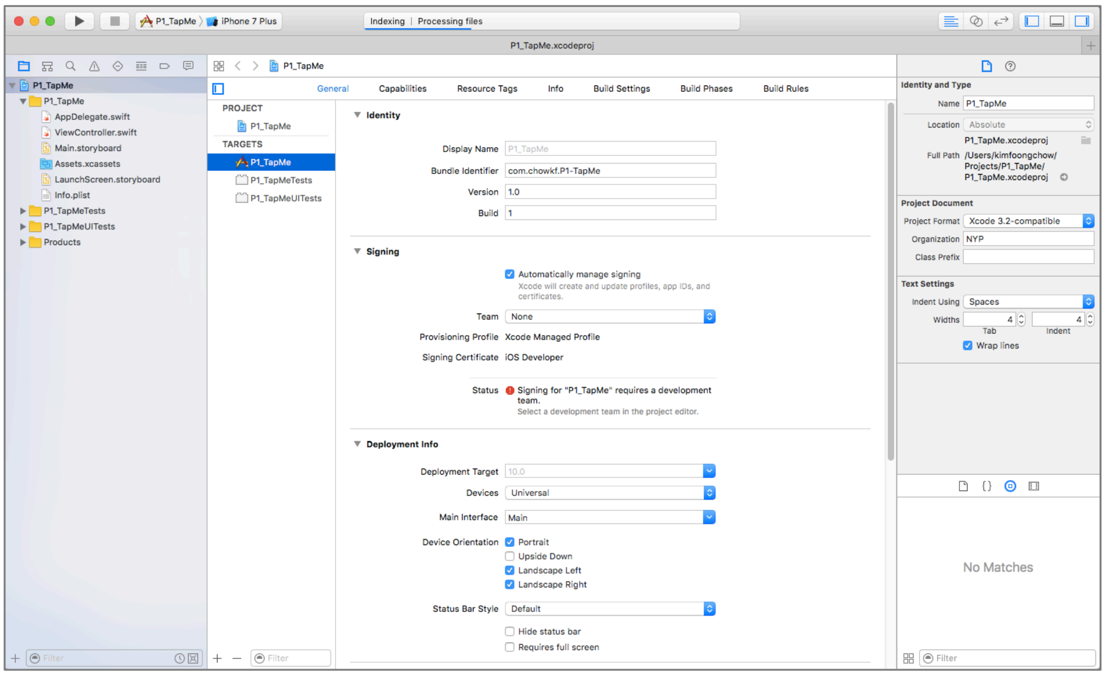


4. Name the project **TapMe** and under the company identifier type in **a suitable identifier**. Bundle identifier will be used to uniquely identify your app for distribution. Make sure the **iPhone** is selected under the **Devices** and **Swift** under the **Language**.



Product Name: P2_TapMe
Team: None
Organization Name: NYP
Organization Identifi... com.chowkf
Bundle Identifier: com.chowkf.P2-TapMe
Language: Swift
Devices: iPhone
☐ Use Core Data
☑ Include Unit Tests
☑ Include UI Tests

5. Xcode now displays your project in single window with a number of panes.



P1_TapMe.xcodeproj

Identity
Display Name: P1_TapMe
Bundle Identifier: com.chowkf.P1-TapMe
Version: 1.0
Build: 1

Signing
☑ Automatically manage signing
Xcode will create and update profiles, app IDs, and certificates.
Team: None
Provisioning Profile: Xcode Managed Profile
Signing Certificate: iOS Developer
Status: Signing for "P1_TapMe" requires a development team.
Select a development team in the project editor.

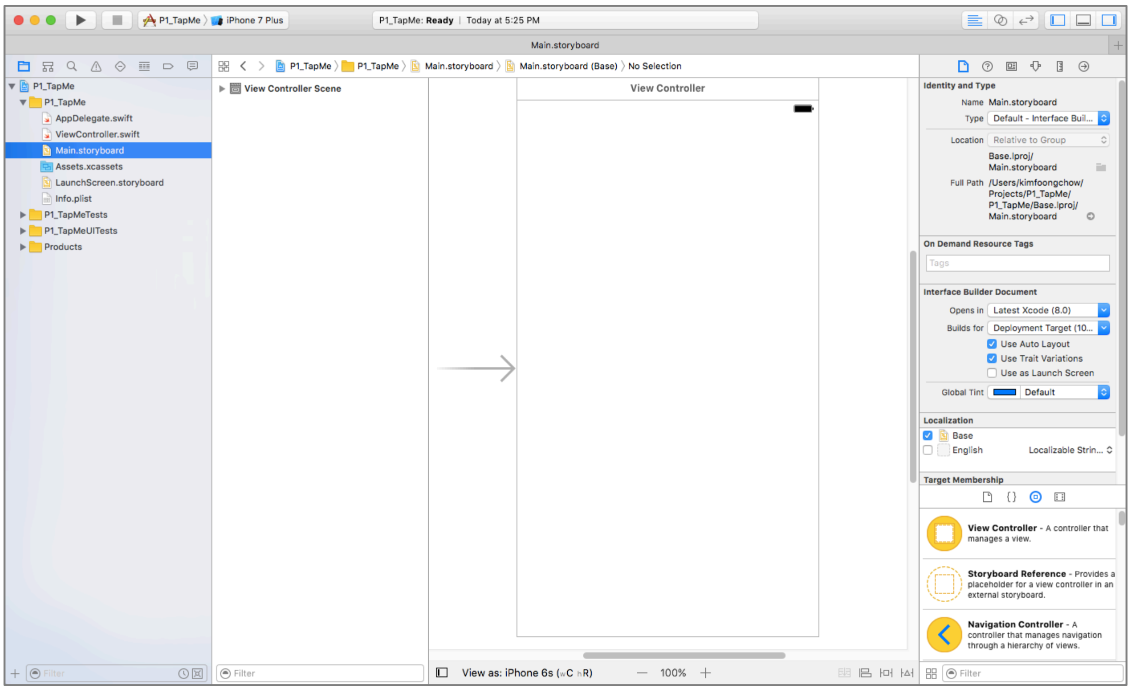
Deployment Info
Deployment Target: 10.0
Devices: Universal
Main Interface: Main
Device Orientation: ☑ Portrait ☐ Upside Down ☑ Landscape Left ☑ Landscape Right
Status Bar Style: Default
☐ Hide status bar
☐ Requires full screen

Identity and Type
Name: P1_TapMe
Location: Absolute
P1_TapMe.xcodeproj
Full Path: /Users/kimfoongchow/Projects/P1_TapMe/P1_TapMe.xcodeproj

Project Document
Project Format: Xcode 3.2-compatible
Organization: NYP
Class Prefix:

Text Settings
Indent Using: Spaces
Widths: 4 Tab, 4 Indent
☑ Wrap lines

6. In the project navigator, select the file **Main.storyboard**.



P1_TapMe: Ready | Today at 5:25 PM

Main.storyboard

Identity and Type
Name: Main.storyboard
Type: Default - Interface Buil...
Location: Relative to Group
Base.lproj/Main.storyboard
Full Path: /Users/kimfoongchow/Projects/P1_TapMe/P1_TapMe/Base.lproj/Main.storyboard

On Demand Resource Tags

Interface Builder Document
Opens in: Latest Xcode (8.0)
Builds for: Deployment Target (10...
☑ Use Auto Layout
☑ Use Trait Variations
☐ Use as Launch Screen
Global Tint: Default

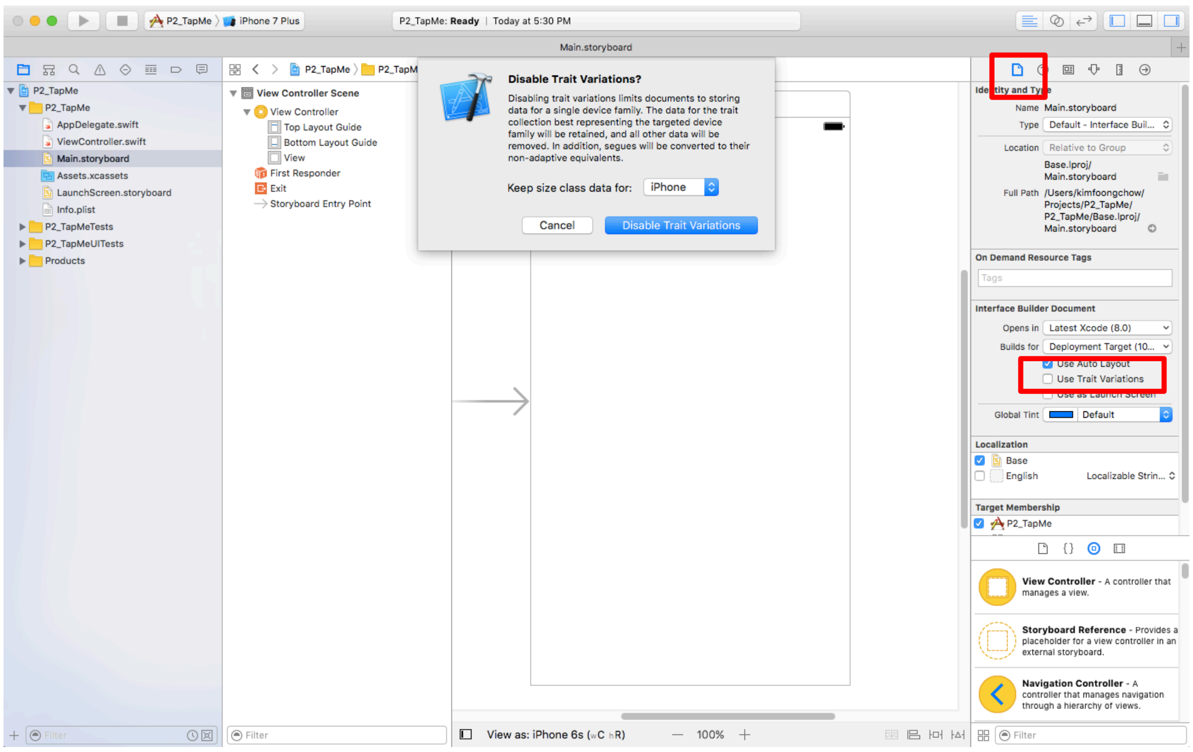
Localization
☑ Base
☐ English Localizable Strin...

Target Membership

**View Controller** - A controller that manages a view.
**Storyboard Reference** - Provides a placeholder for a view controller in an external storyboard.
**Navigation Controller** - A controller that manages navigation through a hierarchy of views.

View as: iPhone 6s (wC hR) 100%

7. Click the area outside of the View Controller and click on File inspector. Uncheck the **Use Trait Variations**, you will ask to keep size class data for: **iPhone**. Click the "Disable Trait Variations" button.



P2_TapMe: **Ready** | Today at 5:30 PM

Main.storyboard

**Disable Trait Variations?**

Disabling trait variations limits documents to storing data for a single device family. The data for the trait collection best representing the targeted device family will be retained, and all other data will be removed. In addition, segues will be converted to their non-adaptive equivalents.

Keep size class data for: iPhone

Cancel | Disable Trait Variations

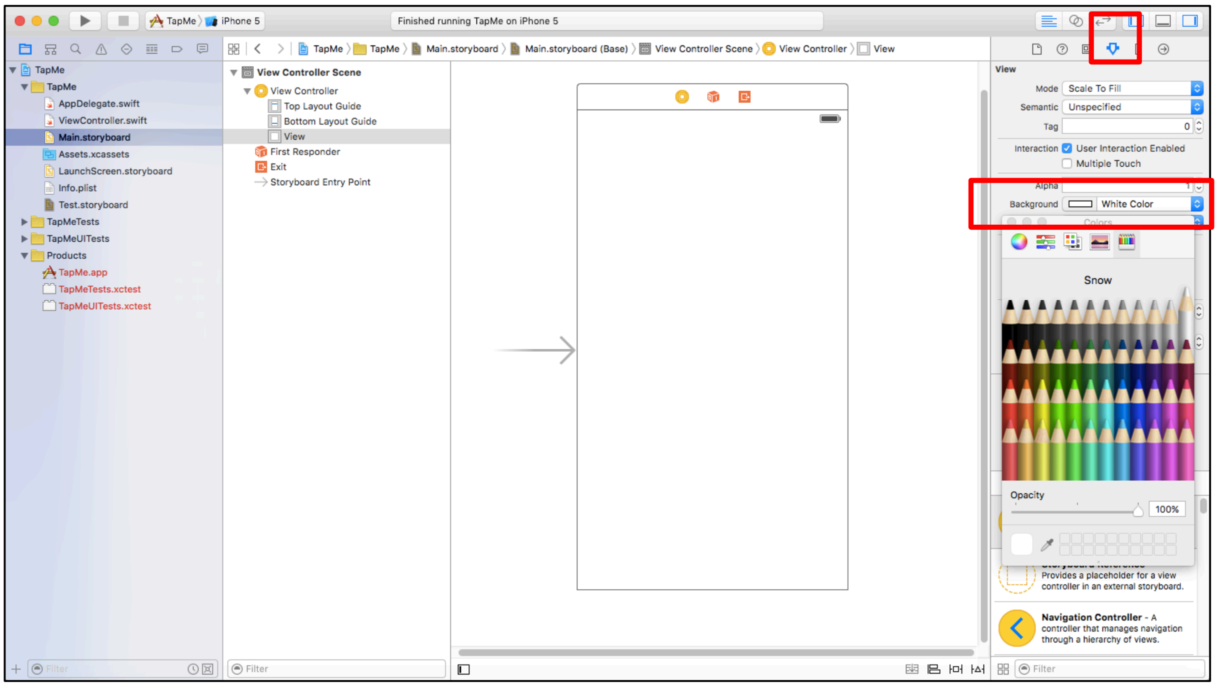
Identity and Type
Name: Main.storyboard
Type: Default - Interface Buil...
Location: Relative to Group
Base.lproj/Main.storyboard
Full Path: /Users/kimfoongchow/Projects/P2_TapMe/P2_TapMe/Base.lproj/Main.storyboard

Interface Builder Document
Opens in: Latest Xcode (8.0)
Builds for: Deployment Target (10...
☑ Use Auto Layout
☐ Use Trait Variations
Global Tint: Default

Localization
☑ Base
☐ English — Localizable Strin...

Target Membership
☑ P2_TapMe

8. Let's get started by changing the background colour of the view. Click on the **Attributes** inspector tab and change the background color.

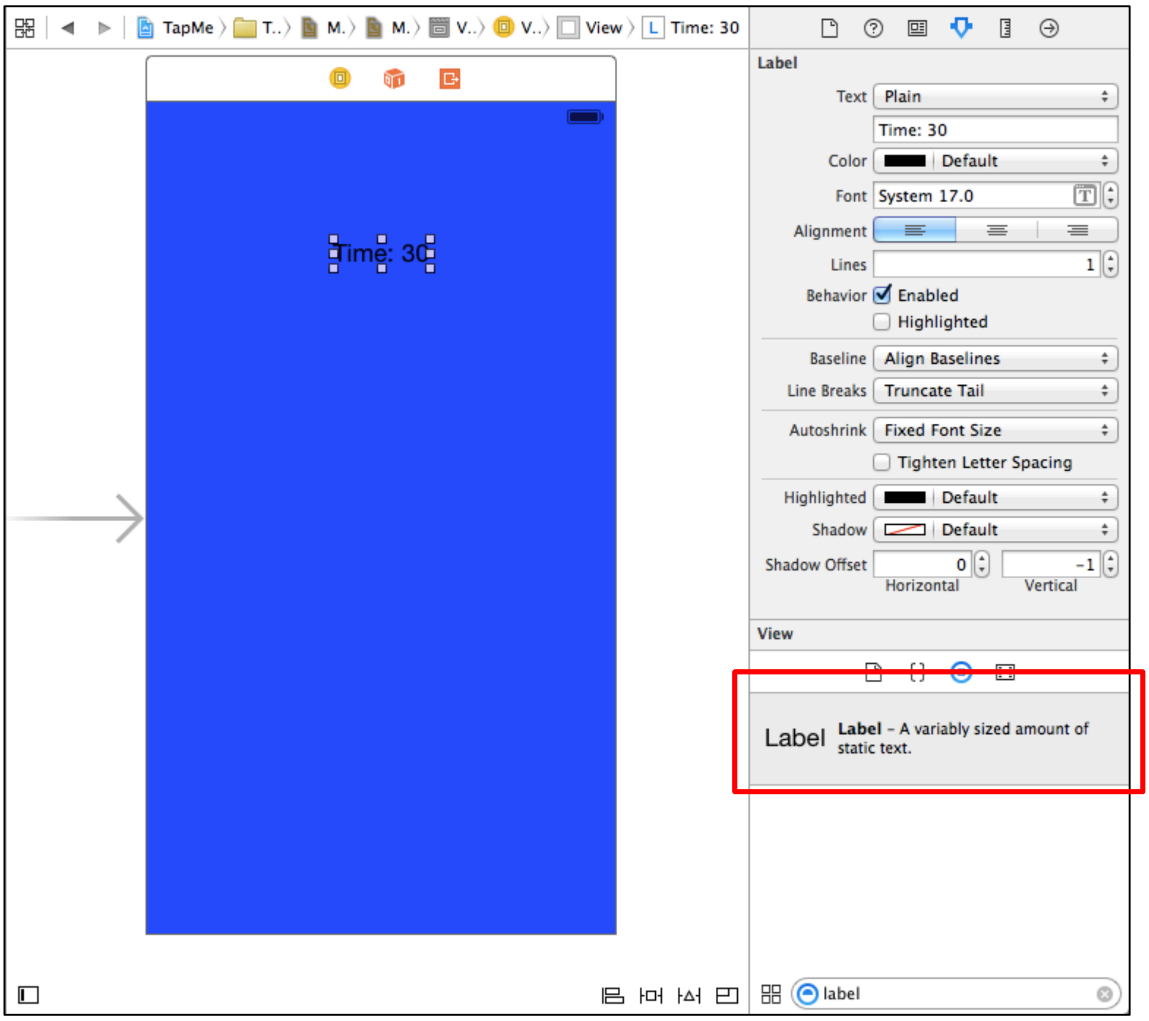


Finished running TapMe on iPhone 5

TapMe 〉 TapMe 〉 Main.storyboard 〉 Main.storyboard (Base) 〉 View Controller Scene 〉 View Controller 〉 View

View
Mode: Scale To Fill
Semantic: Unspecified
Tag: 0
Interaction: ☑ User Interaction Enabled
☐ Multiple Touch
Alpha: 1
Background: White Color

9. Next, let’s create a Timer Label. Drag and drop a **Label** view onto the UI and modify the attribute **Title** to “**Time: 30**”.

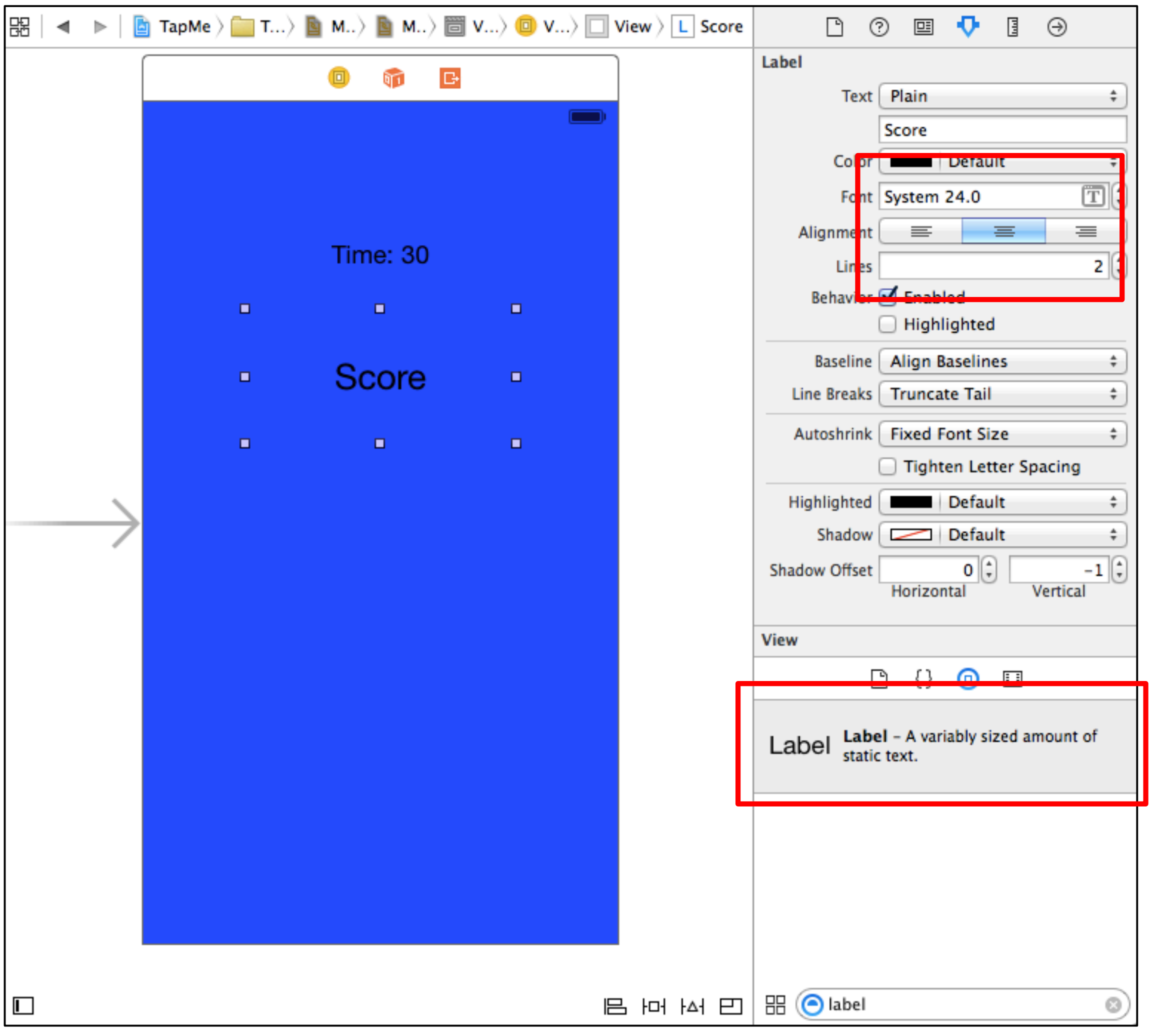


Label

Text: Plain
Time: 30
Color: Default
Font: System 17.0
Alignment
Lines: 1
Behavior: [✓ Enabled] Highlighted
Baseline: Align Baselines
Line Breaks: Truncate Tail
Autoshrink: Fixed Font Size
Tighten Letter Spacing
Highlighted: Default
Shadow: Default
Shadow Offset: 0 Horizontal, -1 Vertical

Label **Label** – A variably sized amount of static text.

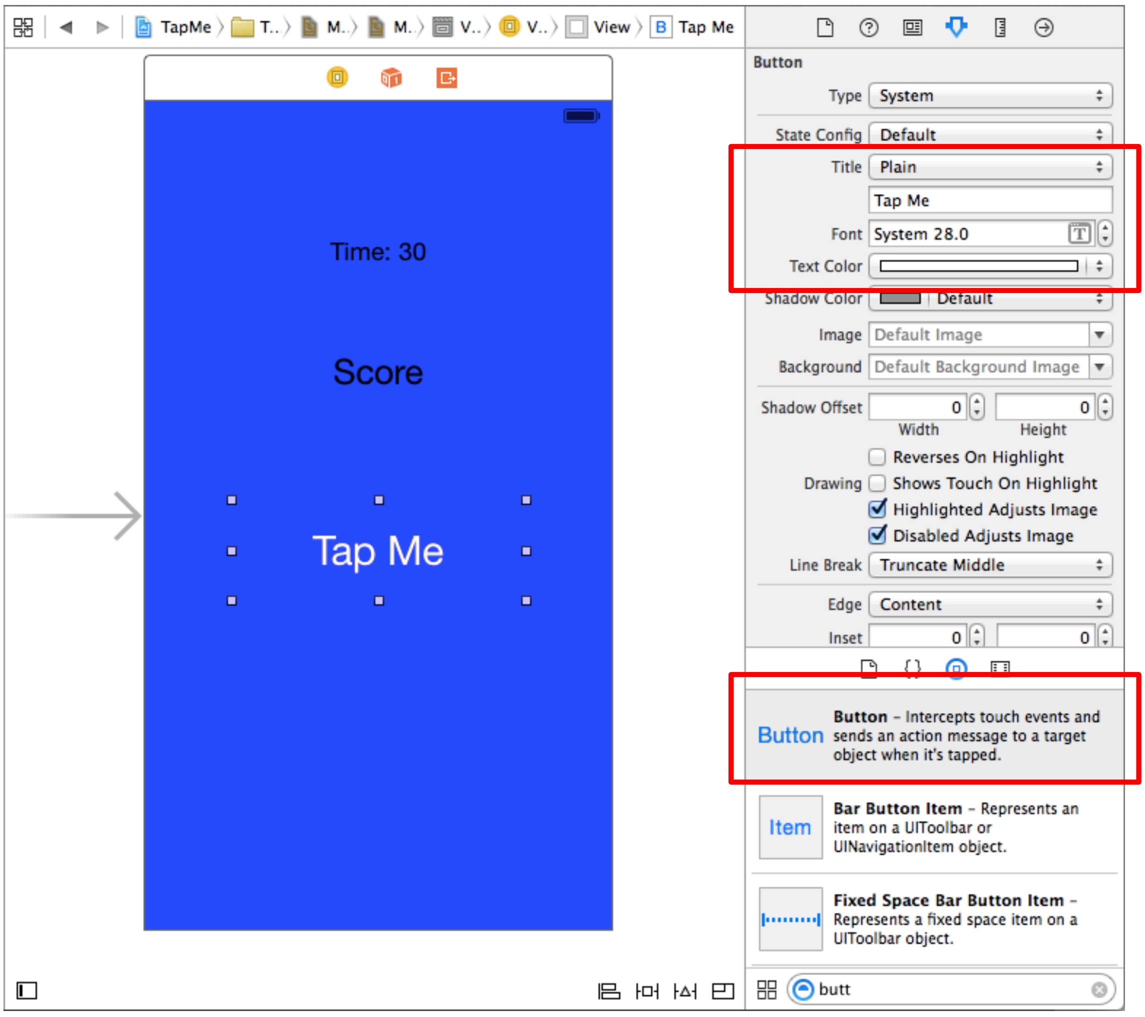
10. Repeat step 9 for Score Label. Dragging a **Label** view from the Library pane onto the UI. Click on the **Attributes** inspector tab and update the Label text to "**Score**" and make it centre-aligned. Make it the font size bigger to stand out in the screen and set the **Lines** to **2**.



Label
Text: Plain
Score
Color: Default
Font: System 24.0
Alignment
Lines: 2
Behavior: ☑ Enabled
☐ Highlighted
Baseline: Align Baselines
Line Breaks: Truncate Tail
Autoshrink: Fixed Font Size
☐ Tighten Letter Spacing
Highlighted: Default
Shadow: Default
Shadow Offset: 0 Horizontal, -1 Vertical
View

Label **Label** – A variably sized amount of static text.

11. Drag and drop a **Button** view onto the UI and modify the attribute **Title** to “**Tap Me**”. Make the font size bigger and the font color to **White**.



Button

Type: System

State Config: Default

Title: Plain

Tap Me

Font: System 28.0

Text Color

Shadow Color: Default

Image: Default Image

Background: Default Background Image

Shadow Offset: 0 Width, 0 Height

Drawing: Reverses On Highlight; Shows Touch On Highlight; ✓ Highlighted Adjusts Image; ✓ Disabled Adjusts Image

Line Break: Truncate Middle

Edge: Content

Inset: 0, 0

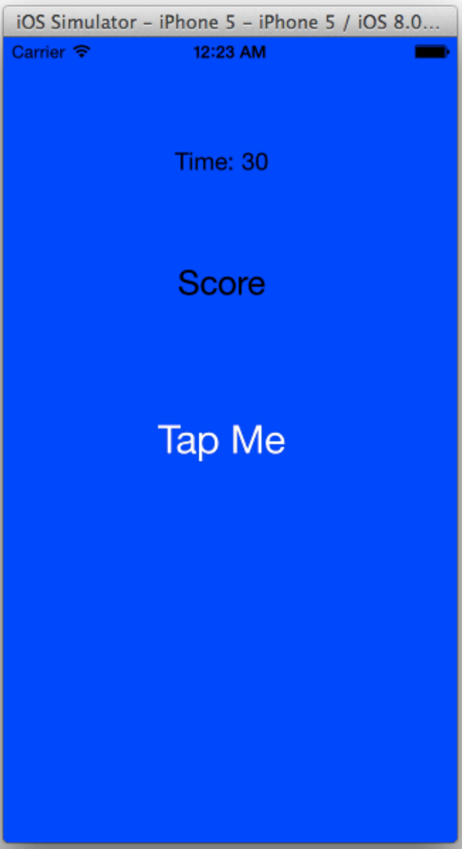
**Button** – Intercepts touch events and sends an action message to a target object when it's tapped.

**Bar Button Item** – Represents an item on a UIToolbar or UINavigationItem object.

**Fixed Space Bar Button Item** – Represents a fixed space item on a UIToolbar object.

butt

12. Save your project by pressing **Command-S**. To test your controls in the iPhone Simulator, press **Command-R** and your application will be deployed to the **iPhone Simulator**.



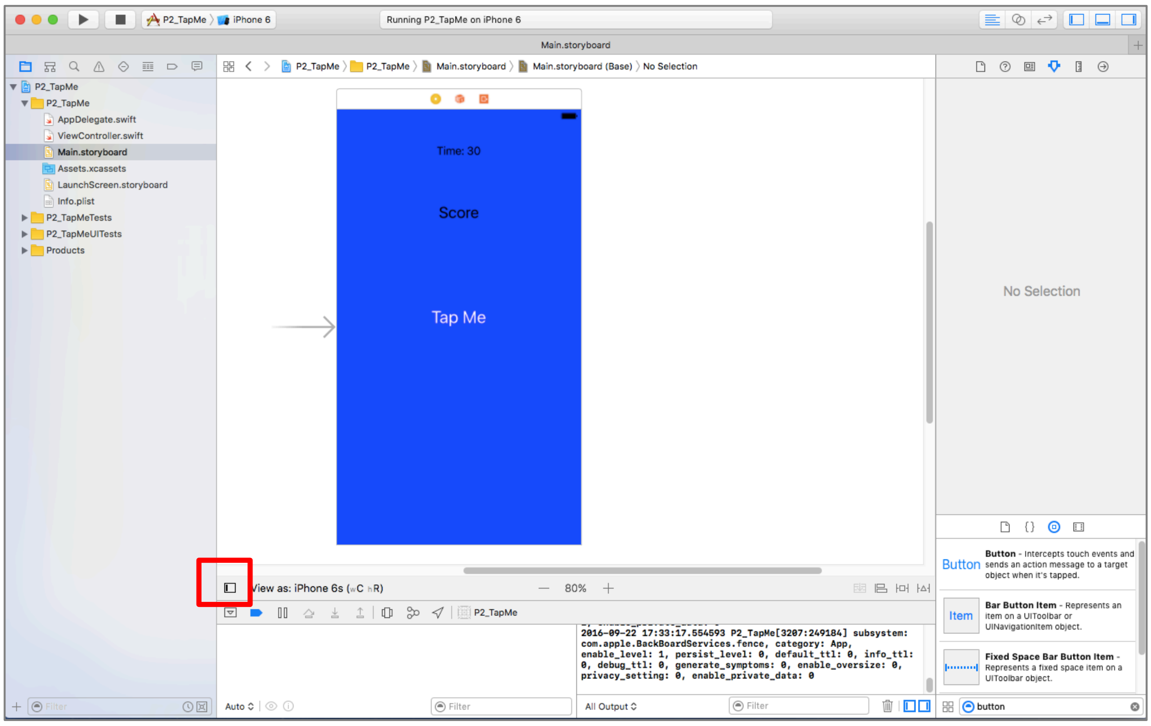
## Section 2: Connecting Outlets and Actions

*In section of this lab, we will connect the interface created previously to the code that makes it into a functional application.*

13. From the project navigator, click on the **ViewController.swift**. Enter the following highlighted statements.

```
import UIKit

class ViewController: UIViewController {

    var count = 0;

    // All IBOutlet UI elements are declared as weak variables
    // because we do not want to hold a reference to them.
    // They are also usually declared as implicitly unwrapped
    // optionals.
    //
    @IBOutlet weak var timerLabel: UILabel!
    @IBOutlet weak var scoreLabel: UILabel!

    @IBAction func buttonPressed(sender: AnyObject) {
        count = count + 1

        // Since this is implicitly unwrapped, we do not need
        // to write scoreLabel!.text to access the text property.
        scoreLabel.text = "Score\n\(count)";
        }
```

14. Back to the **Main.storyboard**, click on the icon below to show the document outline.

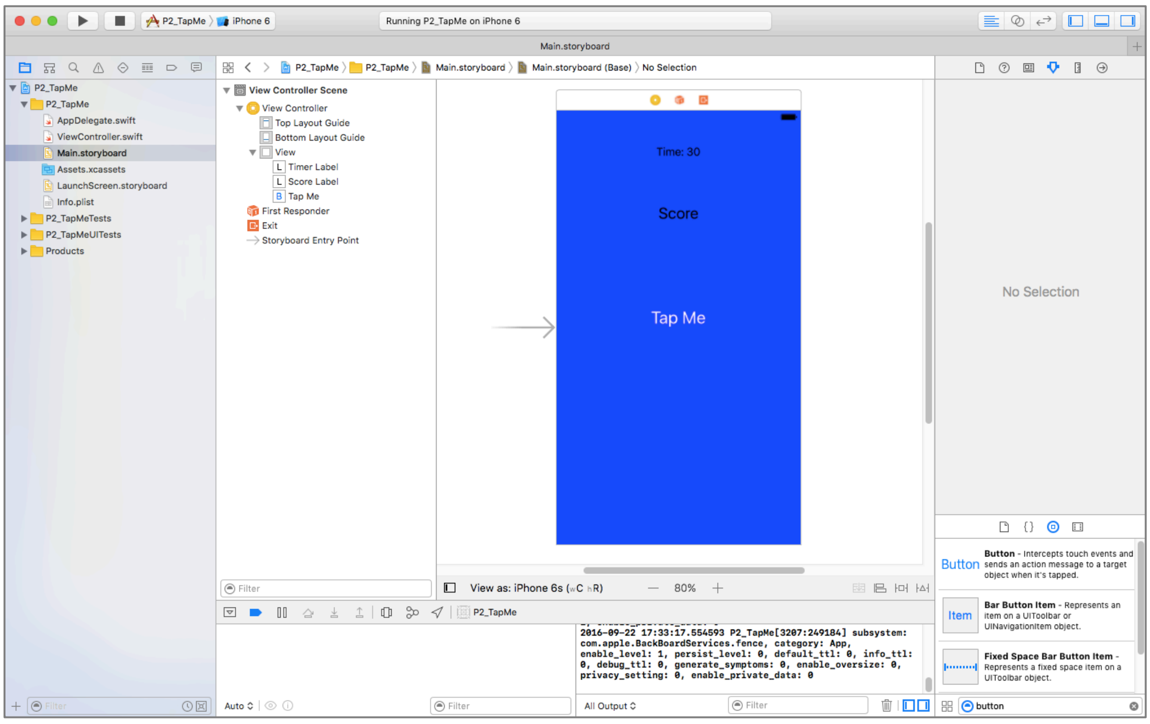


```
2016-09-22 17:33:17.554593 P2_TapMe[3207:249184] subsystem:
com.apple.BackBoardServices.fence, category: App,
enable_level: 1, persist_level: 0, default_ttl: 0, info_ttl:
0, debug_ttl: 0, generate_symptoms: 0, enable_oversize: 0,
privacy_setting: 0, enable_private_data: 0
```

Button - Intercepts touch events and sends an action message to a target object when it's tapped.

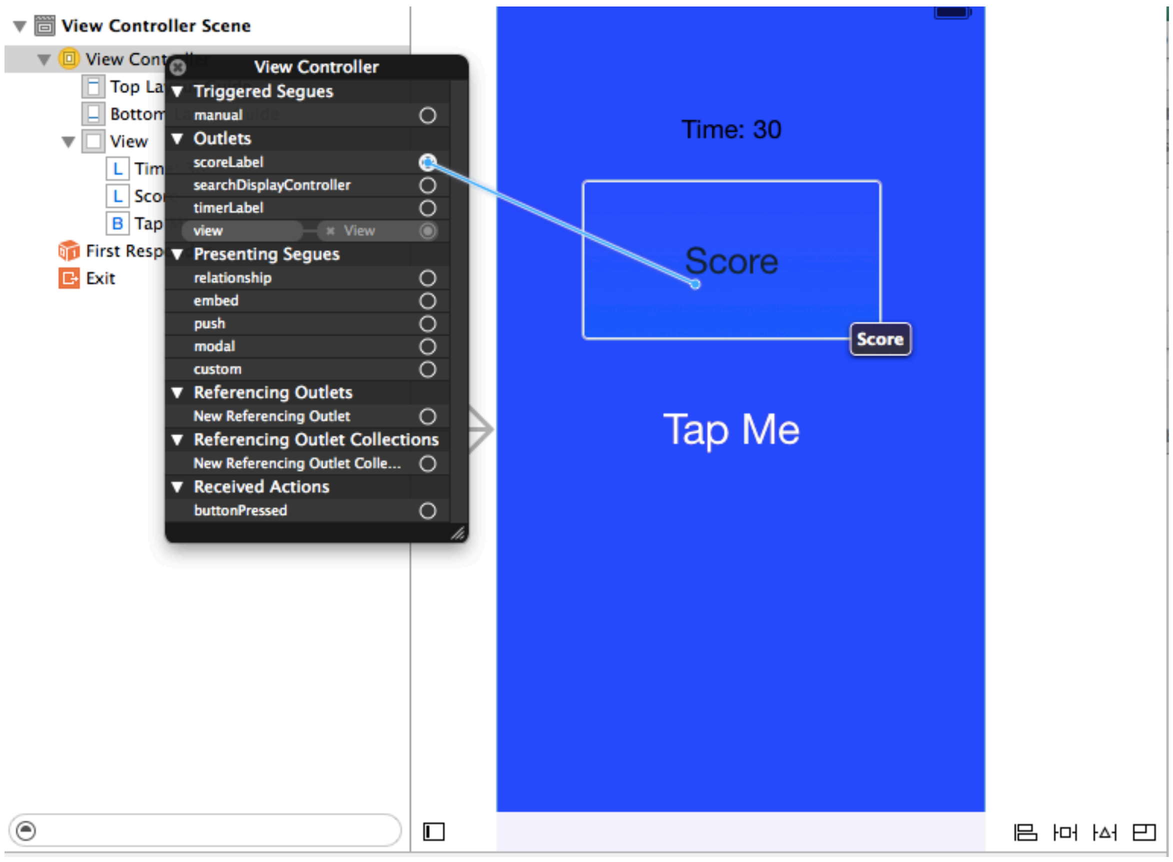
Bar Button Item - Represents an item on a UIToolbar or UINavigationItem object.

Fixed Space Bar Button Item - Represents a fixed space item on a UIToolbar object.



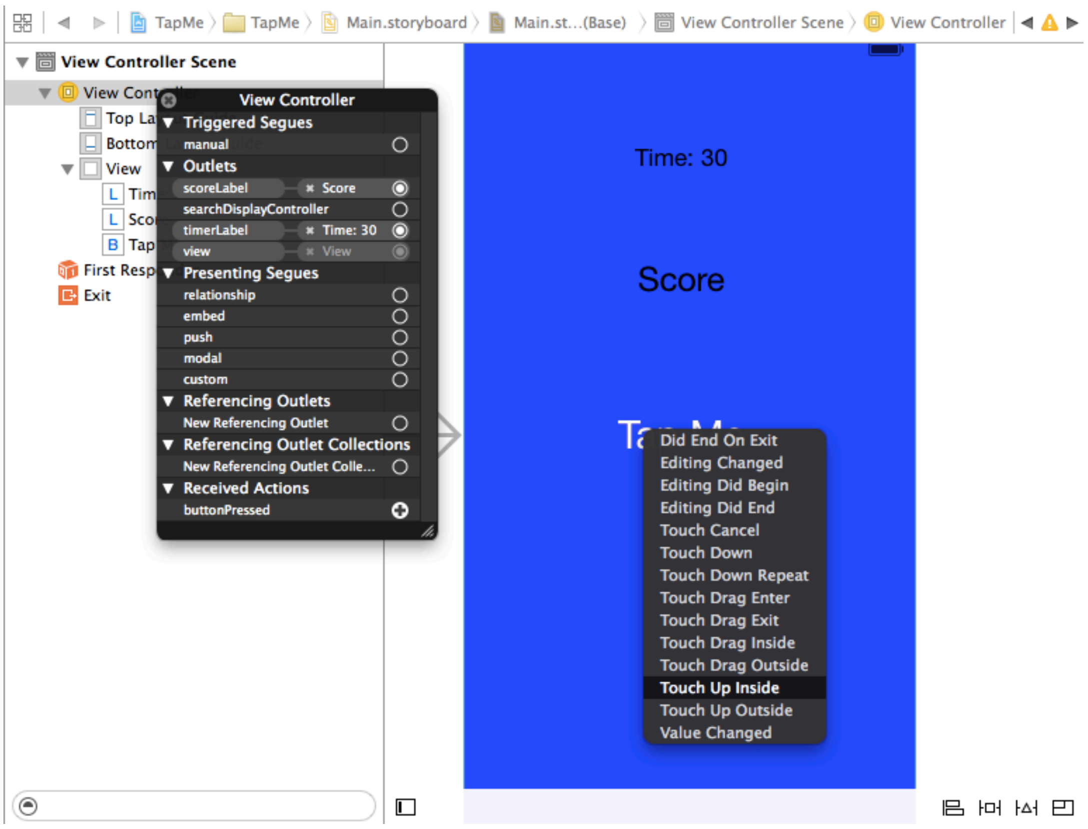
```
2016-09-22 17:33:17.554593 P2_TapMe[3207:249184] subsystem:
com.apple.BackBoardServices.fence, category: App,
enable_level: 1, persist_level: 0, default_ttl: 0, info_ttl:
0, debug_ttl: 0, generate_symptoms: 0, enable_oversize: 0,
privacy_setting: 0, enable_private_data: 0
```

15. Right click **View Controller** at the Document Outline. Connect the outlet by clicking the small circle on the right side of the label outlet. Keep your mouse button pressed and drag to the **scorelabel** on your **view** to make the connection.

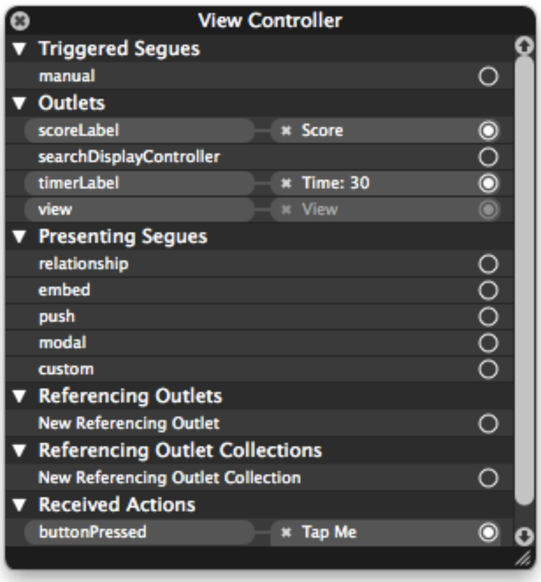


16. Repeat step 15 for **timeLabel**.

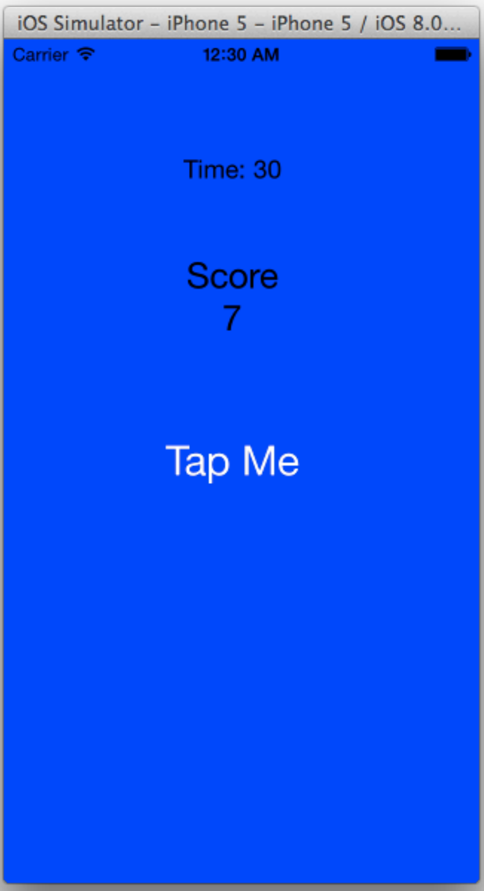
17. You can do the same to connect the action to the button on your view. Additional step is required to select the button event (buttonPressedWithSender) when it pops up to make the connection. For this lab, we select **Touch Up Inside**.



18. Ensure that the Connections Inspector window for the **File’s Owner** item in the **Main.storyboard** window looks as shown:



19. Build and run your application. When you click on the button, the label will update and show the score.



## Section 3: Create a Timer

*Next, we will need to set up a timer to count down from 30 seconds to zero.*

Now that game works and keeps score, you'll need to set up the timer to count down from 30 seconds to zero, when the player's turn will end.

20. In the **ViewController.swift**, edit the code:

```
class ViewController: UIViewController {

    var count = 0
    var seconds = 0
    var timer : Timer?


    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view,
        // typically from a nib.
        self.setupGame()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    func setupGame()
    {
        count = 0
        seconds = 30

        timerLabel.text = "Time: \(seconds)"
        scoreLabel.text = "Score\n\(count)"

        // Creates a timer that triggers every 1 second.
        // When it triggers, the subtractTime function will
        // be called.
        //
        timer = Timer.scheduledTimer(
            timeInterval: 1.0,
            target: self,
            selector: #selector(ViewController.subtractTime),
            userInfo: nil,
            repeats: true)
    }

    func subtractTime()
    {
        seconds = seconds - 1
        timerLabel.text = "Time: \(seconds)"

        if (seconds == 0)
        {
            // Optional chaining - don't call
            // invalidate if timer is null.
            //
            // The following optional chaining is more concise, and
            // it is equivalent to:
            // if timer != nil
            // {
            //     timer!.invalidate()
            // }
            //
            timer?.invalidate()
        }
    }
```

Understanding the codes for the timer:

```
Timer.scheduledTimer(
    timeInterval: 1.0,
    target: self,
    selector: #selector(ViewController.subtractTime),
    userInfo: nil,
    repeats: true)
```

- **Time Interval** (the first parameter) is to set the interval of the timer. For this case, we set it to be in 1 second difference.
- **Target** is which instance to send a message to every second. You're in the view controller now and you want the message to go to the view controller, so the target is **self**.

- **Selector** is what function you want to call.
- **User Info** is any extra info you want stored with the timer. You won't need anything for this timer so you say nil to represent nothing.
- **Repeats** says if the timer should repeat or just fire off once. If you want the timer to go off every second until you say stop, so this is set to **true**.

21. Debug the application on the iPhone Simulator.

## Section 4: Make use of UIAlertController

*Lastly, we are going to add an alert that will pop up the score and a button for the player to play the game again when the game is over. We will introduce to you closures, a design pattern. It is used to respond to an event from the host object that contains a pointer reference to the closure function. The object sends will trigger the closure function when it accomplishes a specific task.*

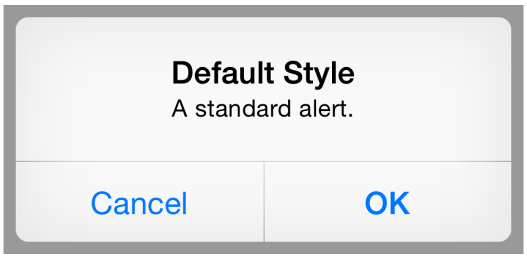
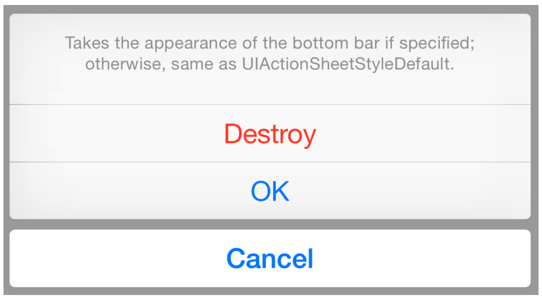
22. From the project navigator, click on the **ViewController.swift**. Edit the codes as shown:

```
import UIKit

class ViewController: UIViewController
{
    ...
    ...
    func subtractTime()
    {
        seconds--
        timerLabel.text = "Time: \(seconds)"

        if (seconds == 0)
        {
            // Optional chaining – don't call
            // invalidate if timer is null.
            timer?.invalidate()

            let uiAlert = UIAlertController(
                title: "Time is up!",
                message: "You scored \(count)",
                preferredStyle: UIAlertControllerStyle.alert)

            uiAlert.addAction(UIAlertAction(title: "Play Again",
                style: .default,
                handler: {
                    // handler is a closure that is called
                    // when the user clicks on the Play Again
                    // button.
                    //
                    // When called, set up the game to run again.
                    action in
                    self.setupGame()
                }))

            // Present the alert. Notice that the completion
            // parameter is also a closure that will be called
            // when the alert shows up.
            //
            self.present(uiAlert, animated: true, completion: nil)
         }
        }
}
```

Understanding the codes for the UIAlertController:

```
let uiAlert = UIAlertController(
    title: "Time is up!",
    message: "You scored \(count)",
    preferredStyle: UIAlertControllerStyle.alert)
```

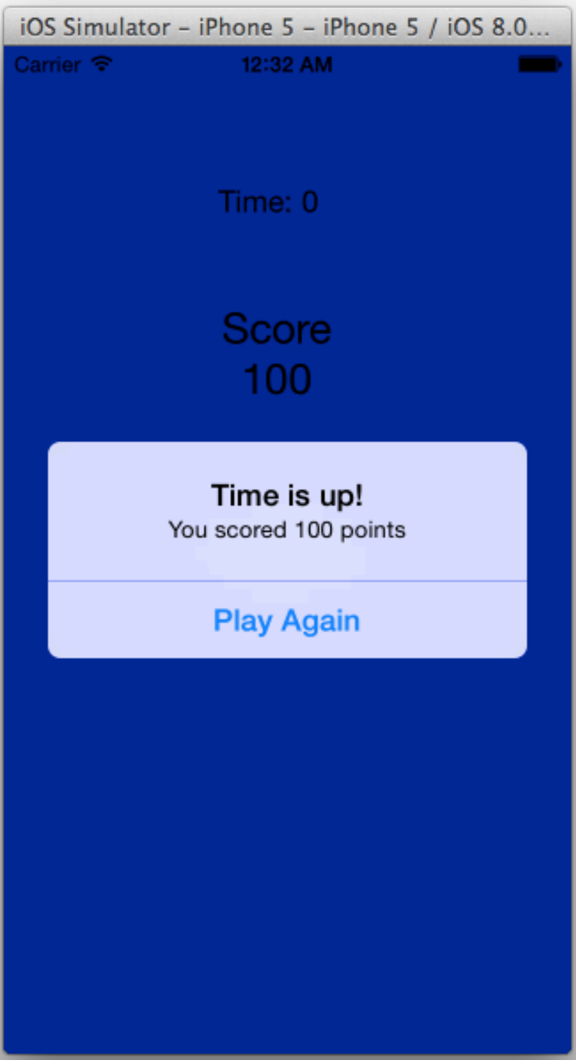
- **Title** – the title that will appear at the top of the alert.
- **Message** – the message. For this lab, we are displaying the score.
- **PreferredStyle** – This can be Alert or ActionSheet.

| Alert | ActionSheet (usually appears at the bottom of the screen) |
|---|---|
|  |   Takes the appearance of the bottom bar if specified; otherwise, same as UIActionSheetStyleDefault.<br>Destroy<br>OK<br>Cancel |

```
uiAlert.addAction(UIAlertAction(title: "Play Again",
    style: .default,
    handler: {
        // handler is a closure that is called
        // when the user clicks on the Play Again
        // button.
        //
        // When called, set up the game to run again.
        action in
        self.setupGame()
}))
```

- **Title** – the title that will appear on the button.
- **Style** – Indicates the style/color used to draw the button. This can be Default, Cancel, Destructive.
- **Handler** – A closure to be triggered when the button is clicked.

23. Build and run application again by pressing **Command-R**. You have successfully created your first iOS app.



Time: 0

Score
100

Time is up!
You scored 100 points

## Challenge

1. Consider creating four (4) buttons, and set the text of only one of them as "Tap Me" randomly, and set the remaining buttons' text to "Do Not Tap". You can also consider changing of the "Tap Me" button to white, and the colour of the "Do Not Tap" to red.
2. If the user clicks on a button with "Tap Me", then increment the score, otherwise, decrement the score.
3. Every time the user taps any button, reshuffle the "Tap Me" and "Do Not Tap" buttons.