



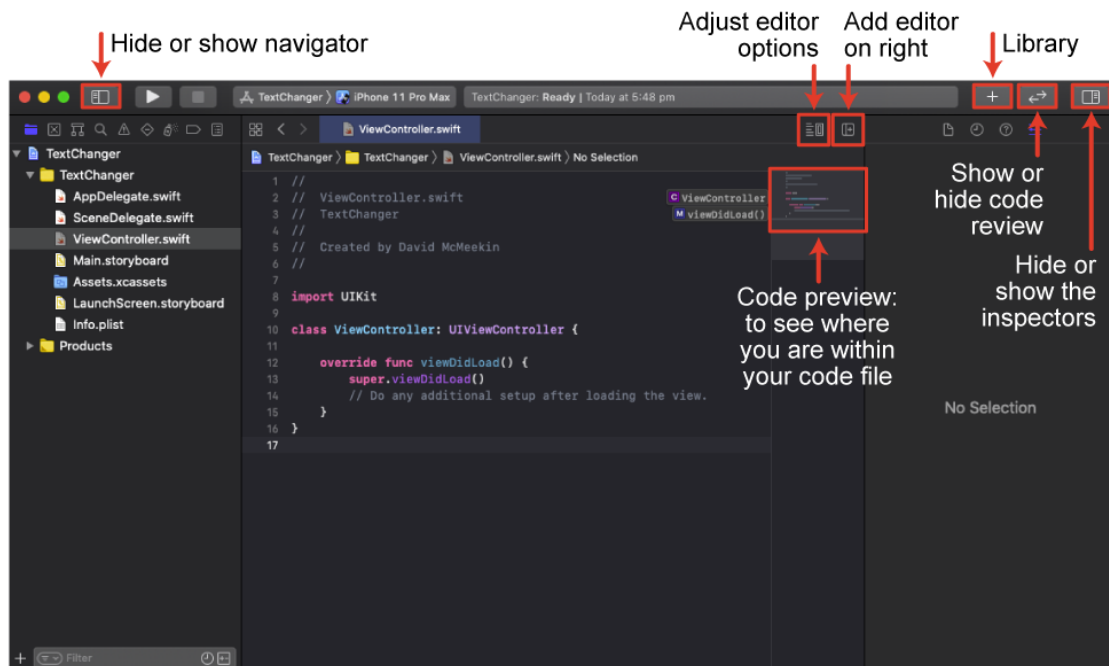
Lesson 3

PDF Slides Lesson 3

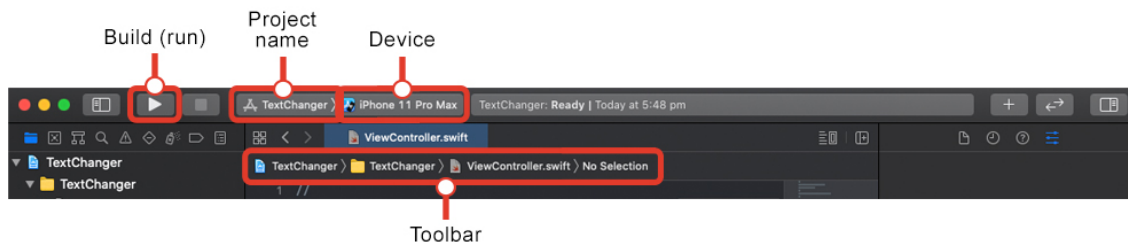
 Lesson3_slides.pdf <https://drive.google.com/file/d/1getygUcxyQ04U6P37FQyzFTlcYXe6MYs/view?usp=drivesdk>

Xcode layout

1. Launch Xcode by going to **Applications > Xcode**.
2. Go to **File > New > Project > iOS > App > Next**.
3. Save your new project with the Product Name of **'TextChanger'** and save it to your computer.
4. The main Xcode window consists of three main sections:
 - Navigator Area (left);
 - Editor Area (centre); and the
 - Utility Area (right).
5. Take note of the following highlighted areas.

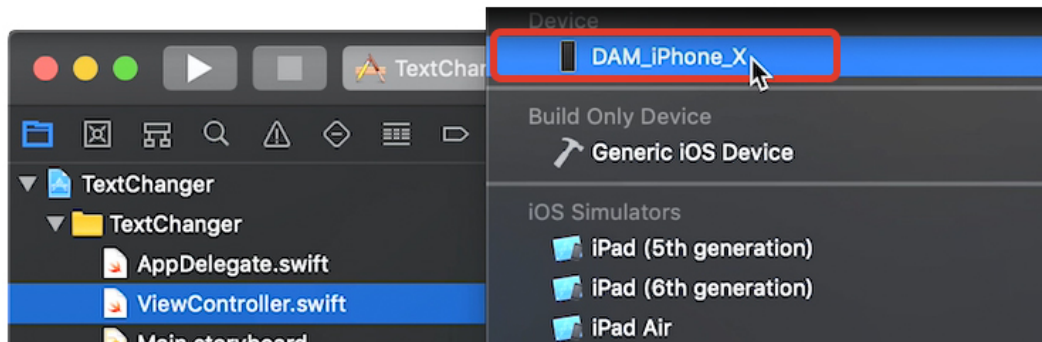


6. Across the top of the window you will see the following: the Toolbar area, the Build (Run) button, the name of our app (project) and the kind of device we are building for.



7. Click on the **Device** type and look at the huge number of different iOS devices that you can build for. Select **iPhone XS** from the list.
8. Click the **Build** (Run) button to run your app in the Simulator. You can drag up from the bottom of the simulator to close your app and go to your home screen, just as you would with an iPhone XS.
9. Go back to Xcode and click on the **Stop** button to stop your program from running on the Simulator.
10. If you have an iPhone or other iOS device, you can connect it to your Mac via cable. Follow the prompts in Xcode if you want to do this. Once successfully connected, if you click on **Device** your connected device should appear at the top of the list. This lets you build, connect and run the

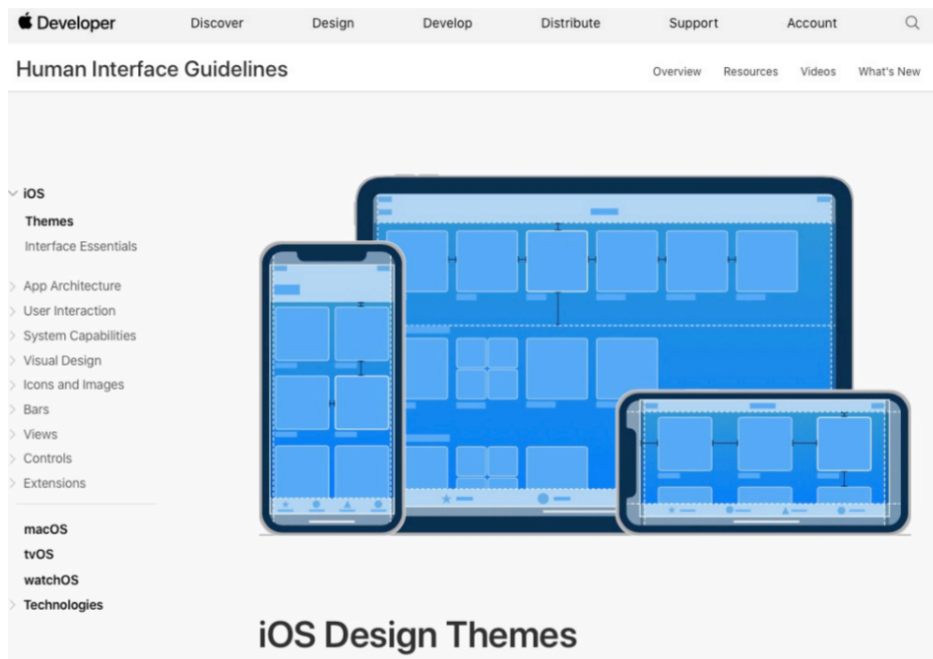
app on your device. In the example below you can see David's iPhoneX (DAM_iPhone_X) has been connected.



1. You can **also** connect your device wirelessly to Xcode. Check out the Xcode Help documentation on how to [Pair a wireless device with Xcode \(iOS, tvOS\)](#).
2. Click on the **Assets.xcassets** folder in the Navigator area. This folder is used to store assets (such as icons, graphics or resources) into our app.

Interface Builder

- Interface builder is a tool within xcode that helps you to build interfaces for your app
- Apple have a set of guidelines for people to follow



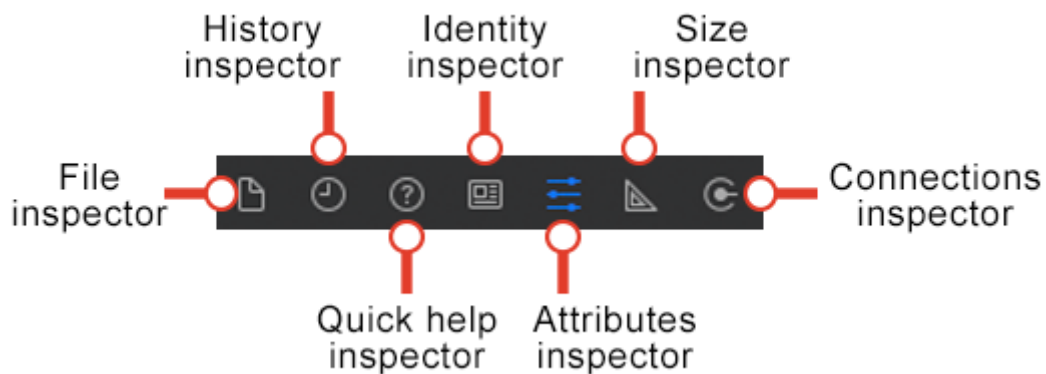
<https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/>

1. If you haven't done so already, open up your '**TextChanger**' project in Xcode.
2. Click on **Main.storyboard** to open up your storyboard canvas. The grey arrow points to the initial view of your app. This is the view that you will see when your app first launches. You may need to navigate around in the window a little to see the storyboard fully. This will depend on the size that it opens within the Xcode window. What we're going to do next is add a button and a label to the canvas and then look at how we can constrain these objects within our app so that when we rotate our device, the interface responds in the way we want.
3. Click on the **Library** icon (📖) to open up your object library.



4. Scroll down to find the **Button** object. Drag it onto the canvas of your storyboard and use the guides to place it in the centre of the canvas.
5. In your object Library, scroll down to find the **Label** object. Drag it onto the canvas of your storyboard above the button and line it up with the left hand side of the button.
6. Over in the **Utilities** area (on the right hand side), hover your mouse over the different Inspectors to see the options available. File inspector Quick

Help inspector Identity inspector Attributes inspector Size inspector
Connections inspector



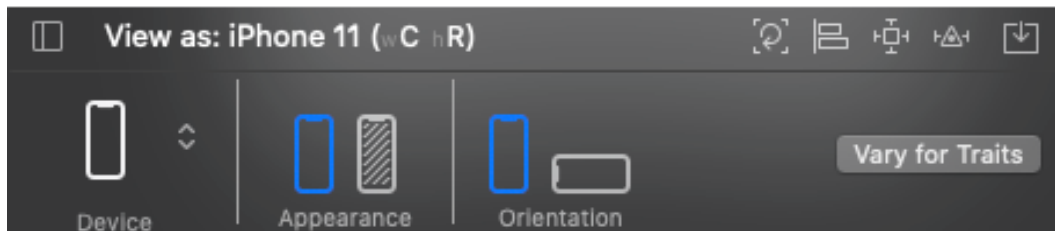
7. Select the **Attributes** inspector and then click on your **Label**. You will see a whole range of options will appear in the Attributes inspector – which relate to the Label.
8. Click on the **Button** and look at the Attributes inspector. Click on your canvas and then at the Attributes inspector. Notice how the Attributes inspector changes depending on what is selected.
9. Select your **Button**. In the Attributes inspector, change the **Title** of your button from Button to **textChange**. The button on your canvas will update to reflect your new title. Please note: You can **also** double-click on your Button to edit the title of it as well.
10. With the **textChange** button still selected, click on the **Size attributes inspector** (). Look at the various settings which relate to the size and location of this button.



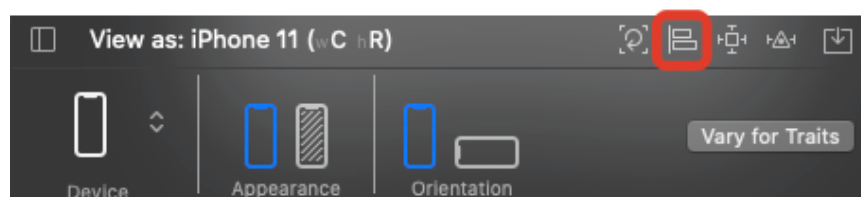
11. Click on the **Label** and look at the various size attribute settings available. Within the Interface Builder you can use these settings to start constructing and adjusting the look of your app.
12. Click on the **textChange** button and go back to the **Attributes inspector** ().



13. Make sure your '**View as**' option is set for **iPhone 11**. Now we want to change our **View**. Below your canvas you will see there is a **View As:** button. You may need to click on it to expand or contract it depending on the size of your window. This button controls the **view of your app on a certain device** and the **orientation of that device**.



1. Currently our view is set to iPhone 8. Click on the **View As:** and select **iPad Pro 12.9**. You'll see that your canvas has resized and your label and button have moved and are no longer centred.
2. Switch your view back to **iPhone 8** to return to your original view.
3. Change your Orientation from Portrait to **Landscape**. Again you will see that as your canvas has rotated, your button and label are no longer in the centre. How can we fix this problem? We want to ensure that no matter what device (or orientation of that device!) that we build our app for – our interface displays the way we want it to. To do this – we use **Auto Layout**.
4. Select your textChange button and click on the **Align** icon.



1. In the **Add New Alignment Constraints** box that pops up, select the check boxes next to: and click on **Add 2 Constraints**. This will set the constraints for our button.
 - Horizontally in Container
 - Vertically in Container
2. Change your view to **iPhone Xs Max**. You'll see that your canvas size has changed BUT your button remained where it was – due to the constraints you added. However you will see that your Label has moved.

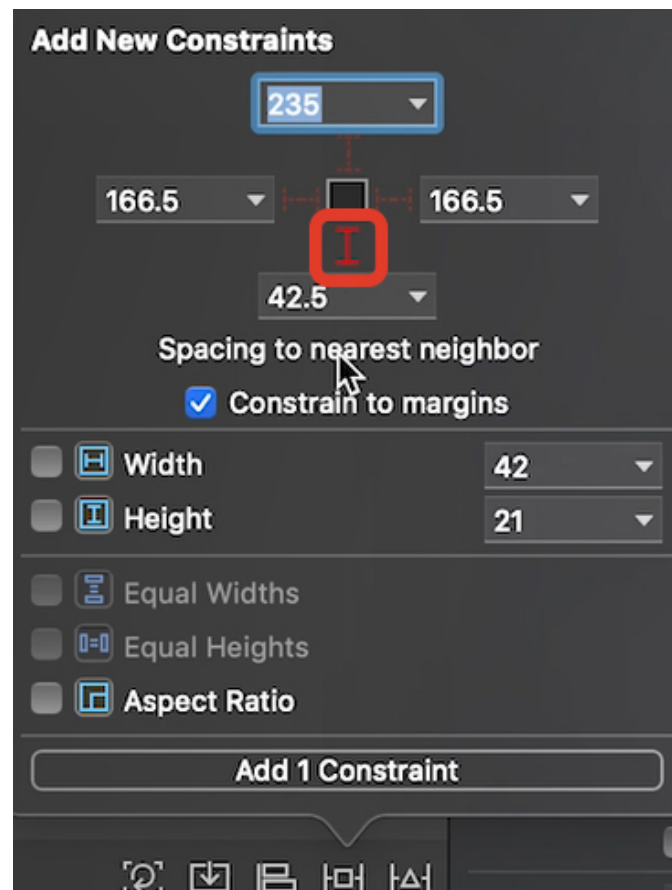
3. Change your Orientation to **Landscape** and you'll see the same thing – your button remains where you set it. But your Label has moved again.
-



4. **Hide the Navigator** and adjust your windows so that your canvas fits nicely into the middle of your screen. We are now going to set the Label to be constrained along with the button.
5. Set your View as: **iPhone XS** and set the Orientation to **Portrait**.
6. Select the **Label** on your canvas and then select **Align () > Horizontally in Container > Add 1 Constraint**.



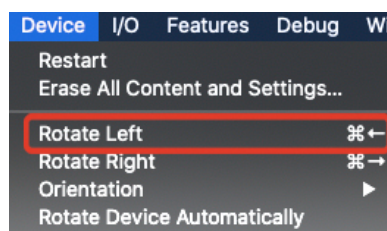
7. Let's change device. Change your View to an **iPad Pro 9.7**. Your **Label** should remain in the middle.
8. Change your View back to an **iPhone 8** and change the Orientation back to **Landscape**.
9. We're now going to add another constraint to link the Label to the textChange button. Select the **Label** and then click on **Add New Constraints**.
10. Click on the red line **below** the black box in the middle to select the Spacing to nearest neighbor. You will see a constraint line appear on your canvas between your two objects.



1. Change your orientation to **Landscape**. Now change your View to a different device. You should find that your label and button remain together in the centre of the screen (just where you want them!).
2. Click on the **Run** () icon to build your app for the iPhone XS and see how it looks in the Simulator.



3. Once it loads, rotate your device in the Simulator by going to **Device > Rotate Left**.





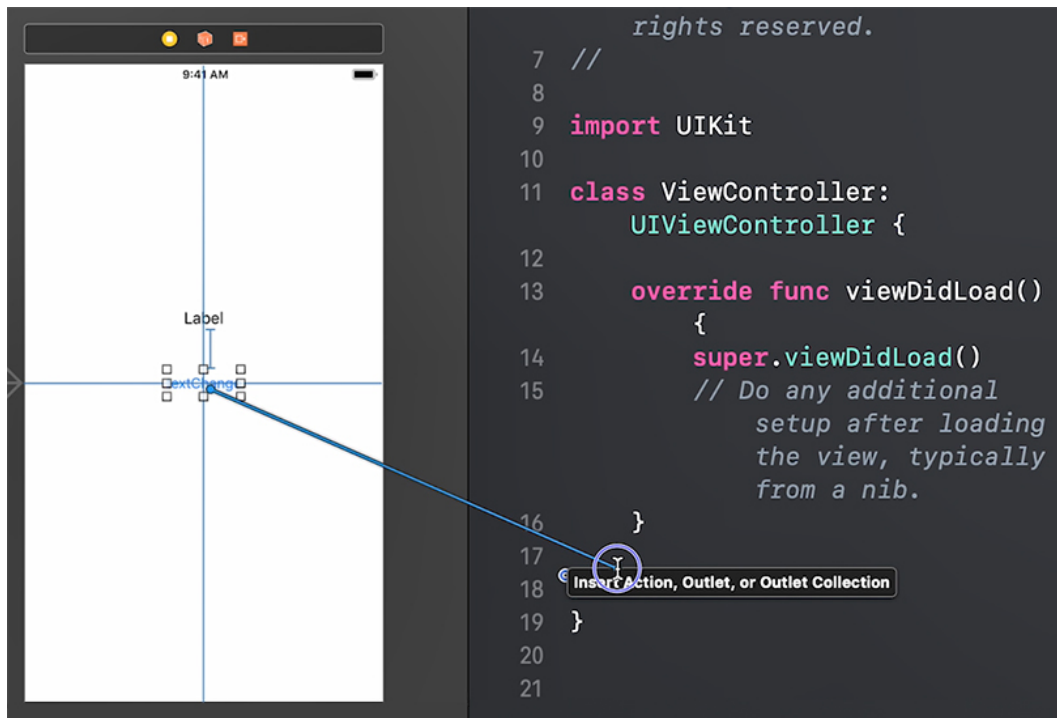
4. Click on **Hide or show the Navigator** icon to bring back your Navigation panel.
5. Click on **View Controller Scene** and then select the **Identity Inspector** (). You'll see in the inspector that there is a Class listed. In this scene, our Class is set to **ViewController**. This means that for this scene, our code is contained in the **ViewController.swift** file over in the navigation panel.



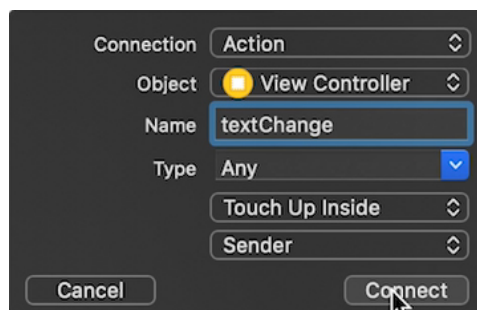
6. Click on the **ViewController.swift** file to bring the code into the Editor window. It will be blank apart from the default code contained within it.
 - Please note that every scene in a storyboard references a ViewController class.
 - Each ViewController class contains the code required to manipulate the scene.
 - If there were multiple scenes in our application, we would need **multiple** ViewController.swift files and we'd have to ensure that each one is correctly connected.
7. We now want to start connecting code to our objects so they actually do something. Click on **Show the Assistant editor** () which will display the code window.



8. Click on **Hide or show the Navigator** to close the navigator window to give yourself more room if needed.
9. Select the **textChanger** button on your canvas. Hold the **Ctrl** key down and click and drag the button into the code window between the last two brackets. You should see a message appear which says: **'Insert Action, Outlet, or Outlet Collection'**. Let go and a pop up window will appear.



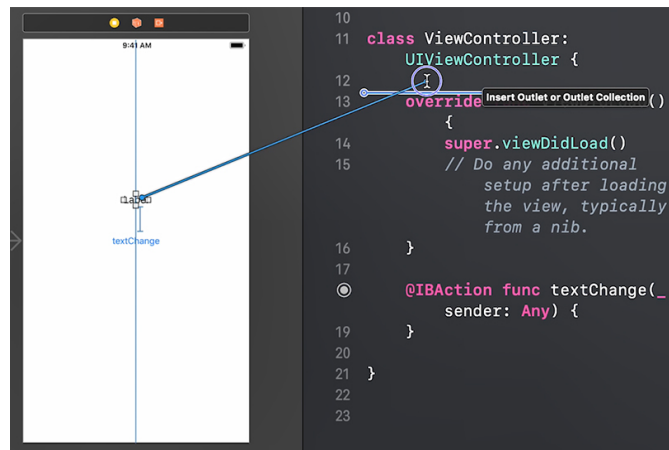
1. You will see the **Connection** is an **Action** – and this is what we want. An action performs something when tapped. In the Name field enter **'textChange'** (which is the name of your button) and click **Connect**.



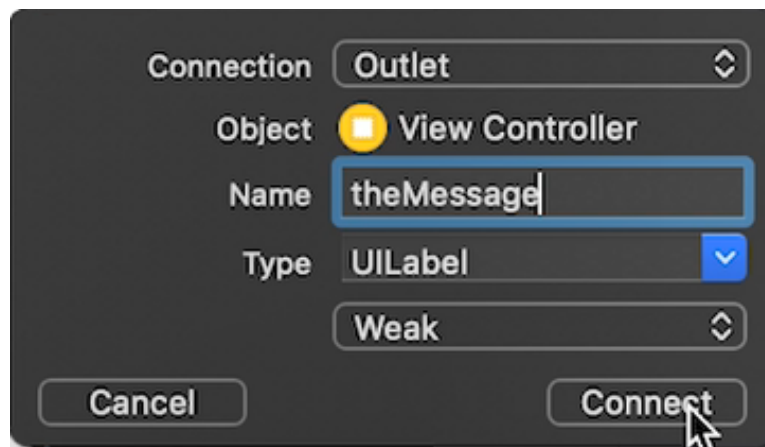
1. This will add the following line of code into the code window.

```
@IBAction func textChange(_ sender: Any) {
}
```

2. Select the **Label** on your canvas and hold the **Control** key down. Then drag the label across into your code window *under* the `UIViewController` line of code. You should see a message appear which says: **'Insert Outlet or Outlet Collection'**. Let go and a pop up window will appear.



1. You will note that this time the **Connection** is an **Outlet** (whereas before the textChange Button was connected to an Action). In the Name field enter '**theMessage**' and click **Connect**.



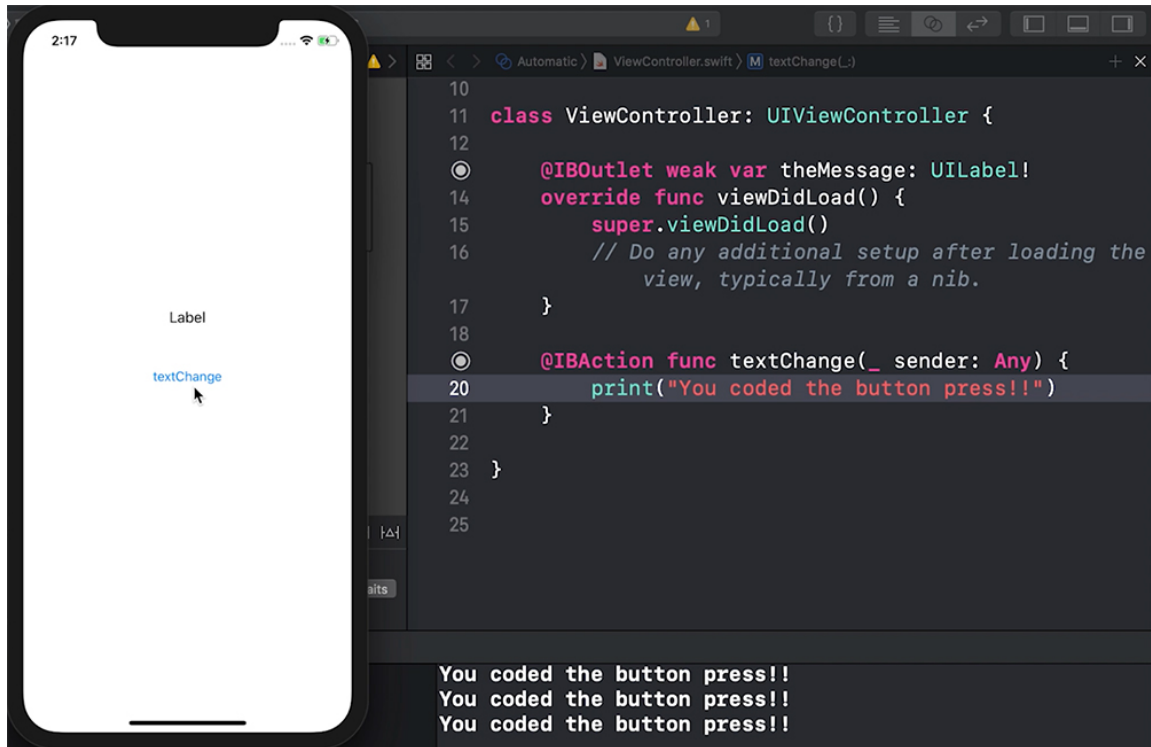
1. This will insert the following line of code into the code window.

```
@IBOutlet weak var
    theMessage: UILabel!
```

2. Now we want to add the following line of code to your @IBAction func so it looks like this.

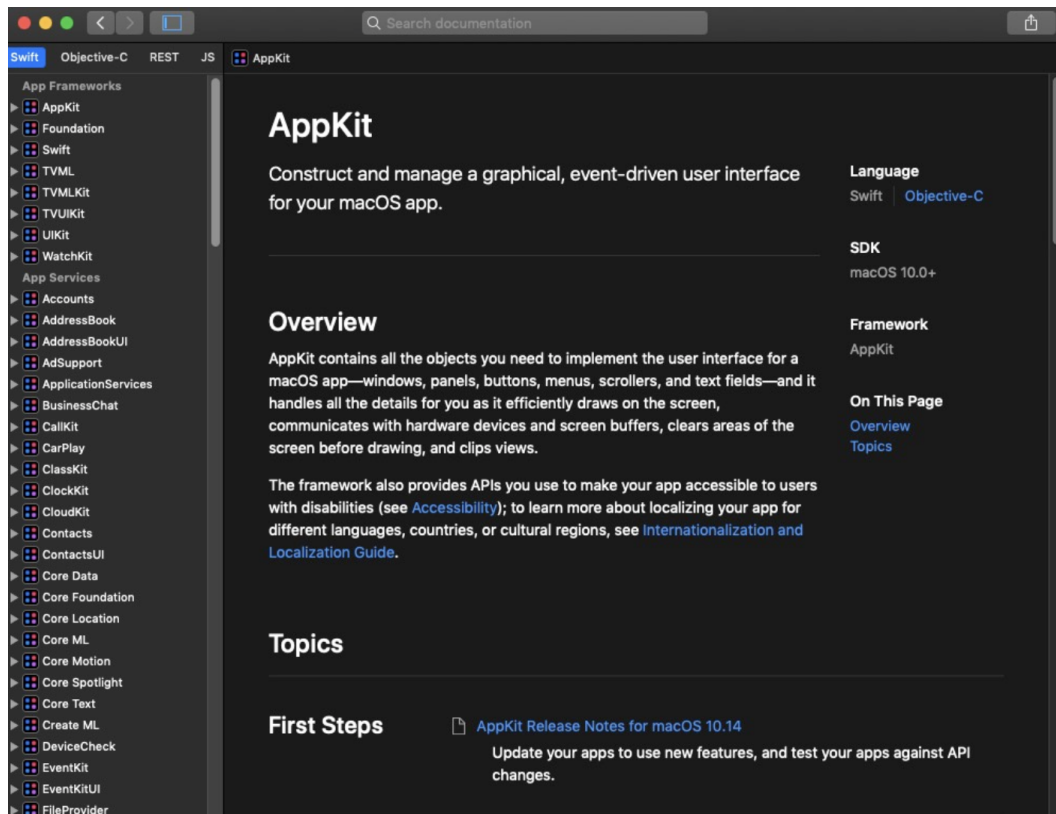
```
@IBAction func textChange(_ sender: Any) {    print("You coded the button pres
s!!")
}
```

3. Build (Run) your app in the Simulator to see it in action. Click on your `textChange` button in the Simulator and you should see the text "You coded the button press!!" appear at the bottom of your screen, in the debug area.

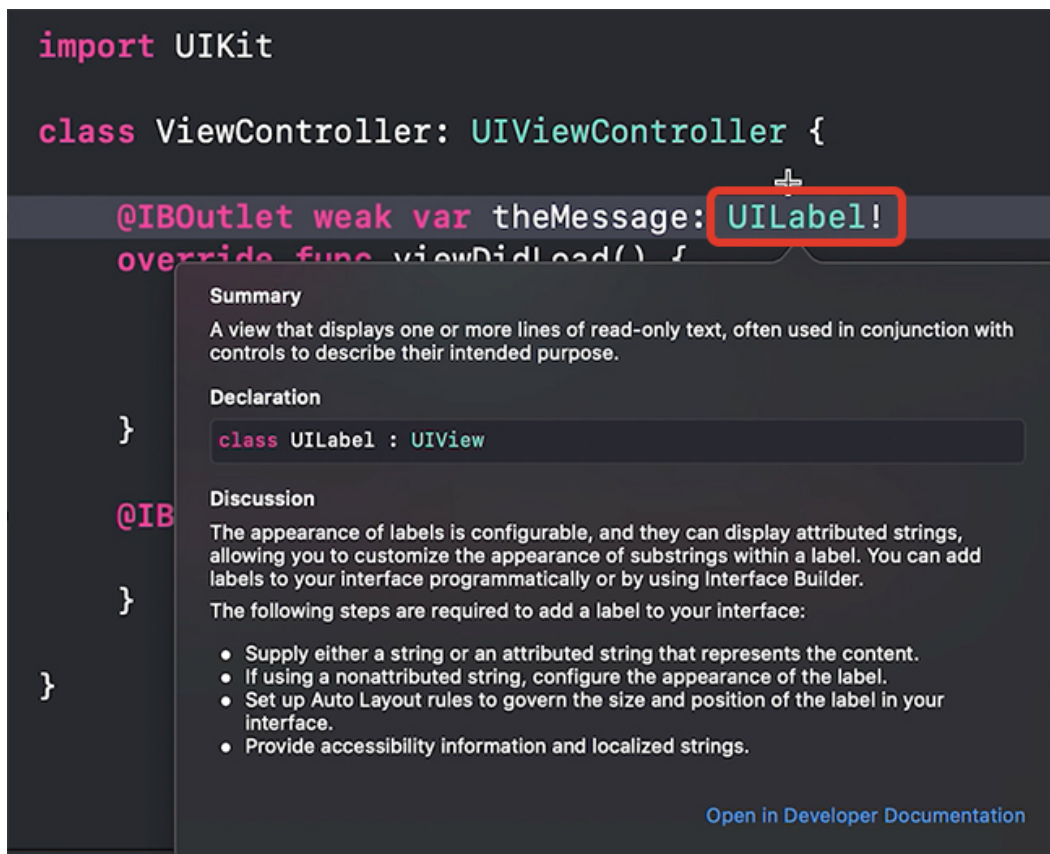


Documentation

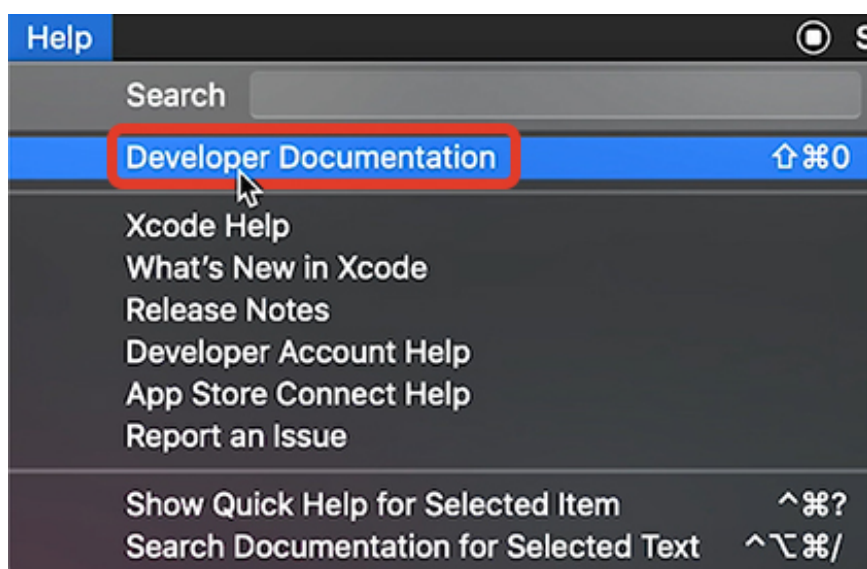
- Everything that is required for you to know is in the documentation
- important to be able to read it



1. Open up your '**TextChanger**' project in Xcode and click on the **Main.storyboard** to open up your storyboard canvas.
2. In the code window, Option + Click on **UILabel!** and a summary of the developer documentation will pop up.



1. Click on **Open in Developer Documentation** to open the developer documentation relating to this item.
2. Alternatively, you can also go to **Help > Developer Documentation** and you can then search the documentation. We're now going to add some more code to our app.



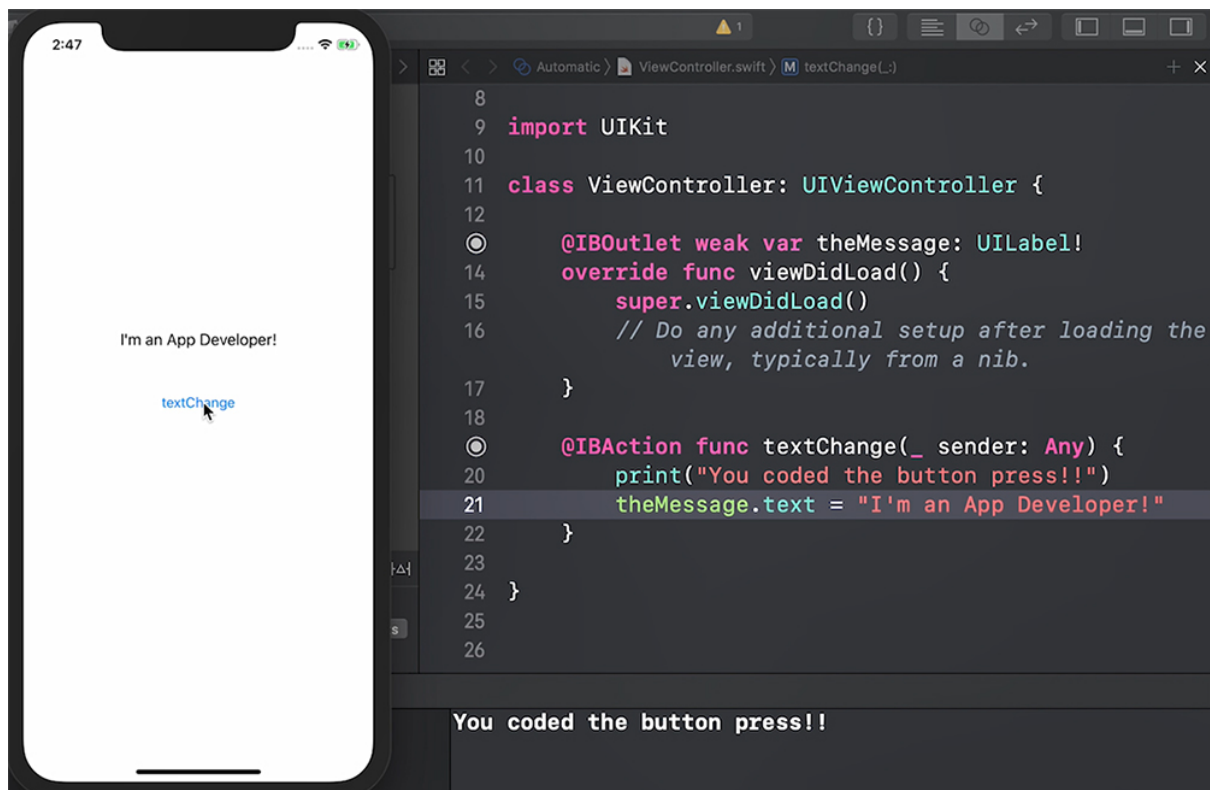
1. Enter the following line of code into your code window underneath the print statement. Your code should now look like this.

```
@IBAction func textChange(_ sender: Any) {  
    print("You coded the button press!!")  
    theMessage.text = "I'm an App Developer!"  
}
```

2. Build (Run) the app and see what happens.



When you click on textChange your label should change to “I’m an App Developer!”.



So from reading the developer documentation, we’ve found out about a label and found out that it has a property called Text. Then, using the name of the variable, .text, we’ve been able to assign that a new value.

Debugging

Warning

- Will not prevent your compiling or running;
- These occur often when you:
 - Create a variable and its value doesn't change;
 - Create a piece of code that never executes;
 - Use code that is from an older version of Swift
- Usually quite simple to resolve & remove.

Errors

- When the run button is clicked, Xcode takes the code you wrote, combines it with the code in things such as UILabel & creates an executable file;
- This is called compiling your code;
- The executable file is what runs on the device;
- When the code can't compile, a compiler error occurs.
- These usually occur when the written code has an error in it.

Run Time Errors

- These are what are really referred to as Bugs;
 - The code compiles without errors and starts executing;
 - While running the code stops or crashes;
 - This is where the code did something it wasn't meant to and can't continue on;
 - Most of us have experienced these when using software, and they are frustrating for both users and developers.
1. Let's now try a debugging exercise of our own. If you haven't done so already, you will firstly need to [download the exemplar code](#).



1. Open **FirstTimeDebugging** in XCode.

2. Click on **ViewController.swift** on the left side menu. You should now be able to see some code, and some **errors** and **warnings** that have been flagged.



1. Before we start debugging the code, let's **run** it. Click the run button on the top left of the screen. You will get a message saying: **'Build failed'**.
2. We need to work through the different types of errors. Note that at the top of the screen that there are **4 warnings** (yellow triangle symbol) and **2 errors** (exclamation mark in a red octagon symbol).



1. In the Editor area, note the highlighted error that says **Expected '{' in body of function declaration**. Click on the red error symbol for more information. An additional line of error text appears below: **Expected parameter name followed by ':'**



1. To resolve the errors that are occurring, have a look at **line 5** of the code, which is highlighted alongside to the error messages. We're looking at a function call **viewDidLoad**. Note that directly after, there is an '(' open bracket but no ')' close bracket. Remember that with a function call there's always an open and close bracket.
2. Close the bracket, so it now reads: **viewDidLoad()**. The error message should disappear.
3. Let's now deal with the second error. Further down the page, highlighted in red is the error: **Use of unresolved identifier 'someMetho'; did you mean**

'someMethod'? This means a function called 'someMetho' has been called, but Xcode cannot find it.

4. Click on the red error symbol for more information. The expanded message says: **Replace 'someMetho' with 'someMethod'**. So Xcode is taking an educated guess at what you might have meant to type, and is making a suggestion. Click **'Fix'** in the error message box. The error disappears. You have now fixed the 2 errors.



1. **Run** the code again. This time you will get a message that says **'Build succeeded'**.
2. We now need to address the **warnings**. Click on the first yellow warning nearest the top of the page of code: **Variable 'sample' was never mutated; consider changing to 'let' constant**. This means Xcode is trying to set this variable, but it hasn't been initiated or mutated. So it's suggesting we change it from **var** to **let**. If you click on the warning (yellow symbol) for more information, the message says: **Replace 'var' with 'let'**.

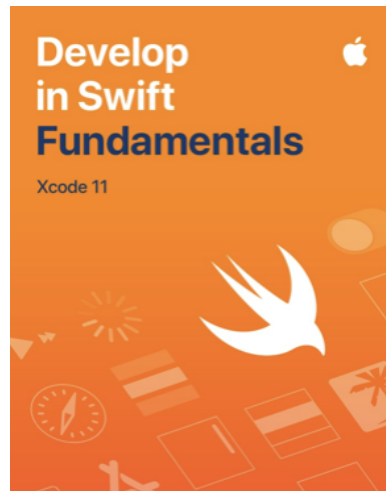


1. Click **'Fix'** in the warning message to replace **var** with **let** and resolve the warning.
2. Go down the page to the next warning: **Will never be executed**. If you click on the warning symbol, it reads: **1. Condition always evaluates to false**. So the code will never be executed, because the condition always evaluates to false. So in this example, you can see that the warning has appeared because there is some code present that is not required.



Extra Resources


- In Apple Books:



Source Code:

Ace5584/IOS-Dev-Notes

Contribute to Ace5584/IOS-Dev-Notes development by creating an account on GitHub.

 <https://github.com/Ace5584/IOS-Dev-Notes/tree/main>

Ace5584/**IOS-Dev-Notes**



 1
Contributors

 0
Issues

 0
Stars

 0
Forks

