

Anyone Can Create an **App**

Beginning iPhone and iPad programming

Wendy L. Wise



manning





Anyone Can Create an App

by Wendy L. Wise

Chapter 5

Copyright 2017 Manning Publications

brief contents

PART 1 YOUR VERY FIRST APP.....1

- 1 ■ Getting started 3
- 2 ■ Building your first app 14
- 3 ■ Your first app, explained 23
- 4 ■ Learning more about your development tools: Xcode 30
- 5 ■ Capturing users' actions: adding buttons 39
- 6 ■ The button app, explained 53
- 7 ■ Capturing user input: adding text boxes 62
- 8 ■ Playing on the Playground 69

PART 2 THE KEYS TO THE CITY: UNDERSTANDING KEY DEVELOPMENT CONCEPTS.....79

- 9 ■ Go with the flow, man! Controlling the flow of your app 81
- 10 ■ While you're doing that... 91
- 11 ■ Collections 104
- 12 ■ Telling stories with storyboards 117
- 13 ■ ViewControllers in depth 125

- 14 ■ Put it on my tab: creating tab bars 136
- 15 ■ Table views: more than a coffee table picture book 144
- 16 ■ Patterns: learning to sew 159

PART 3 CREATING THE LIKE IT OR NOT APP167

- 17 ■ Putting it all together: the LioN app 169
- 18 ■ Adding data to your LioN app 180
- 19 ■ Displaying details of your LioN 190
- 20 ■ Creating the details of the detail view 201
- 21 ■ The AddEditView scene 212
- 22 ■ Delegates are everywhere 225
- 23 ■ Editing LioNs 236
- 24 ■ Saving LioNs 248
- 25 ■ Making your LioN prettier 260
- 26 ■ Working with Auto Layout 273
- 27 ■ Search your LioNs 285

5

Capturing users' actions: adding buttons

This chapter covers

- Creating an app with a button
- Buttons and how to use them
- Changing labels

You're going to create another app in this chapter. This one will have a button the user can tap. *Buttons* are used throughout iPhone and iPad apps to allow the user to do some kind of action, such as make a phone call. Each number on the phone number screen is a button, for example, and the call and hang-up buttons are buttons too.

In this chapter you'll add a label and a button to the app and write some code. The code you write will make the button change what the label displays. You're also going to change how the label looks by implementing cosmetic changes.

Looking at the code

You can download all the projects from this book at www.manning.com/books/anyone-can-create-an-app or <https://github.com/wlwise/AnyoneCanCreateAnApp>, and you can refer to them anytime.

5.1 *Adding a label and a button*

As usual, we're going to start with pseudocode steps so that we follow a logical road map for completing this project. Here are the steps:

- 1 Start a new project using the Single View Application template.
- 2 Add a button and label to the storyboard, and run the app to test it.
- 3 Connect the button and the label to the code (wire them up), and run the app to test it.
- 4 Add code to change the text on the label when the button is clicked, and run the app to test it.
- 5 Change how the label looks, and run the app to test it.

You'll notice that steps 2, 3, 4, and 5 all end with running the app and testing it. I always find it helpful to run the app often to make sure it works the way I expect it to. As you begin adding more and more code to your apps, it's easier to find problems with the code if you test it more frequently. Let's get started.

5.1.1 *Step 1: Start a new project using the Single View Application template*

I want you to get used to starting new projects, so go ahead and create a new one:

- 1 Click File > New > Project.
- 2 Remember to select Single View Application, and name it ButtonApp.
- 3 Make sure the language selection is still Swift. Click OK.
- 4 Save the project to the dev folder on your computer that you created in chapter 2.

5.1.2 *Step 2: Add a button and label to the storyboard, and run the app to test it*

As before, the project is loaded, and the files are listed down the left side of Xcode. Click Main.storyboard so the storyboard shows up in your Standard Editor panel.

STEP 2A: ADDING THE BUTTON

In the bottom-right palette on the Utilities panel, make sure you have the Object Library showing by clicking the circle with a square inside of it (as you did in chapter 2). Then, at the bottom of the panel, search for *button* instead of *label*. You'll notice that you have three options to choose from for the buttons. In this case, you want to add the top one—the one that says Button. Grab it, and drop it on your storyboard (see figure 5.1). Double-click the new button that you dropped on your storyboard, and change the text to *My Button*.

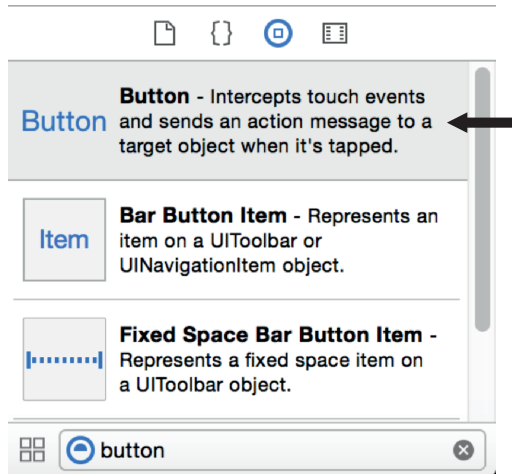


Figure 5.1 Drag a button from the Object Library onto your storyboard.

STEP 2B: ADDING THE LABEL TO THE STORYBOARD

Go back to the Object Library, and search for *label* to add a label to your storyboard. Double-click that label, and change the text to *Button Demo*. (You also did this in chapter 2.)

Now run the app to make sure it looks okay (click the right arrow at top left in the Xcode window). Your Simulator should look similar to figure 5.2.

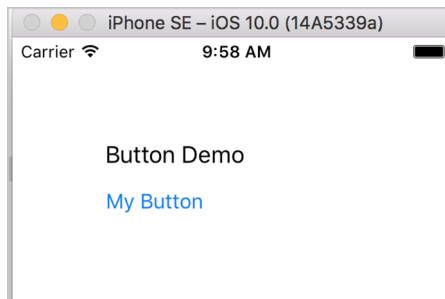


Figure 5.2 The Simulator running with the button and the label

Click the My Button button: it dims when you click it, to show that you're interacting with it. Great! It works, although it doesn't do anything yet. Get ready. You're going to add some code now that will change the title of the button when it is clicked. I'll walk you through it. Stop your Simulator by clicking the square in Xcode (this looks like a stop button on most electronics).

5.1.3 Step 3: Connect the button and the label to the code (wire them up), and run the app to test it

In the next chapter, I'll explain *why* you need to do this step, but for now you need to follow along and make the connections. Okay, let's change the real estate so you can see both the storyboard (with your label and button) and the code.

STEP 3A: CONNECTING THE BUTTON TO THE CODE

You're going to use the Assistant Editor for this, so click the Assistant Editor button (the one that looks like two interlocking circles at top right). You should see the storyboard on the left side and ViewController.swift on the right side, as shown in figure 5.3.

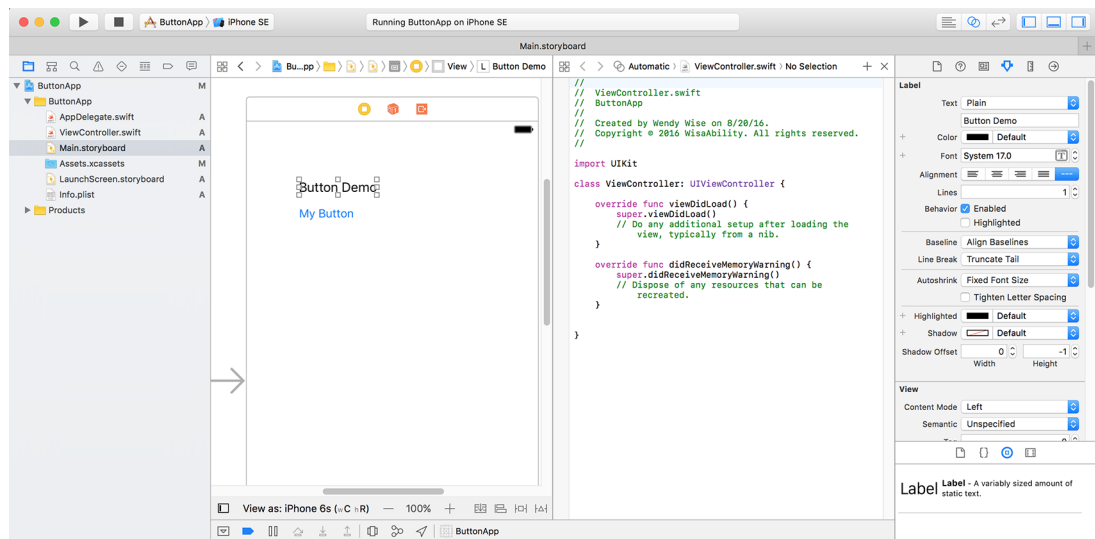


Figure 5.3 The Assistant Editor shows the storyboard on the left and ViewController.swift on the right.

I know the ViewController may look daunting, but we'll walk through it in chapter 13, I promise. For now, think of it as Vanna White from *Wheel of Fortune*. The premise of the show is that three contestants guess letters to a puzzle, and eventually enough letters are guessed so that someone can guess what the puzzle spells. A contestant on the show yells out a letter to the puzzle, and if the letter is in the puzzle, Vanna walks over and turns the letter around so everyone can see it. If the letter is not in the puzzle, she doesn't move. The ViewController is like Vanna: the user presses a button (yells out a letter), and if the button is wired up to your code (if the letter is in the puzzle), the ViewController (Vanna) performs that code (turns the letter around). If the button isn't wired up (the letter is not in the puzzle), nothing happens (Vanna stays still). You can see episodes of the game show here: www.youtube.com/watch?v=rZmWwPN3H2Y.

Again, I'm going to show you how to "wire up the button" first and then I'll explain what you did afterward. I talk more about wiring up buttons in Chapter 6, but for now

think about it as if you're connecting Vanna to the puzzle board. Vanna needs to know which puzzle to use:

- 1 Hold down the Control button on the keyboard, and click My Button on your storyboard.
- 2 With the mouse button and Control still held down, drag the mouse pointer onto the right side of the screen in the ViewController.
- 3 Hover the mouse under the words `class ViewController: UIViewController {`. You should see a blue link that links over to the ViewController and a box that says "Insert Outlet, Action, or Outlet Collection."
- 4 When you see that, let go of both the mouse and the Control button (with your mouse pointer right under the `class ViewController: UIViewController {` line. You now see a pop-up that looks like figure 5.4.

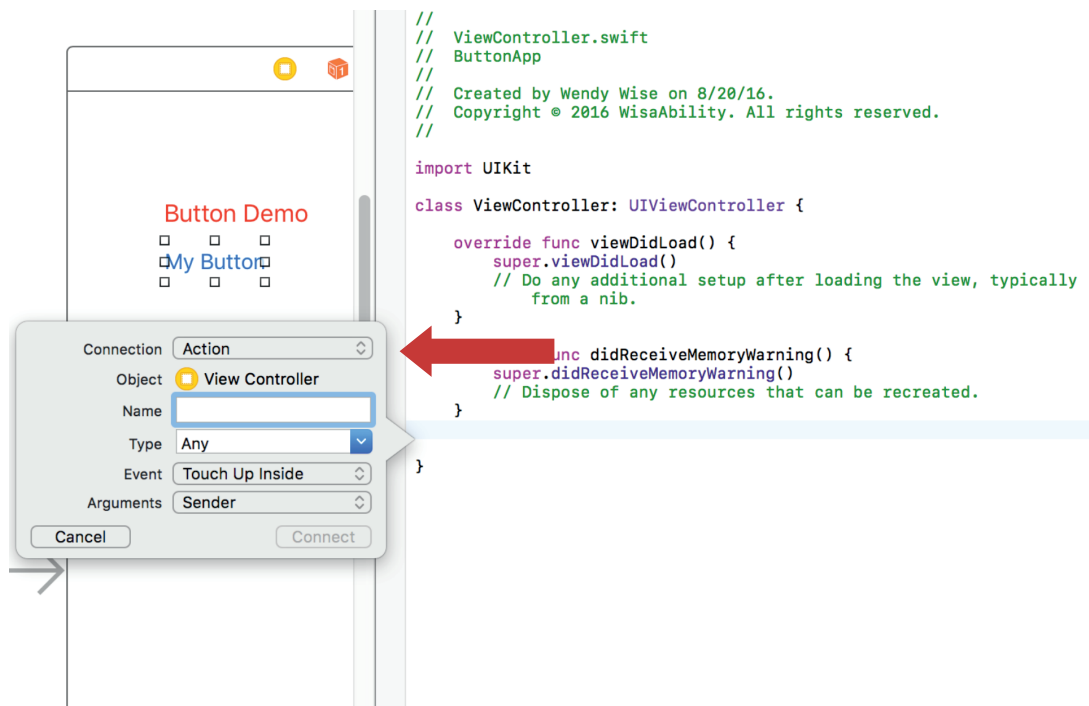


Figure 5.4 The pop-up box should appear when you release the mouse button and Control button. Change the Connection option to Action.

- 5 Change the Connection type (at the top of the pop-up) from Outlet to Action, type `buttonClick` in the Name field, change Type from Any to UIButton, and click Connect (see figure 5.5). You should see on the right panel (the ViewController.swift side) that Xcode added a new line for you. It should state the following:

```
@IBAction func buttonClick(sender: AnyObject) {
```

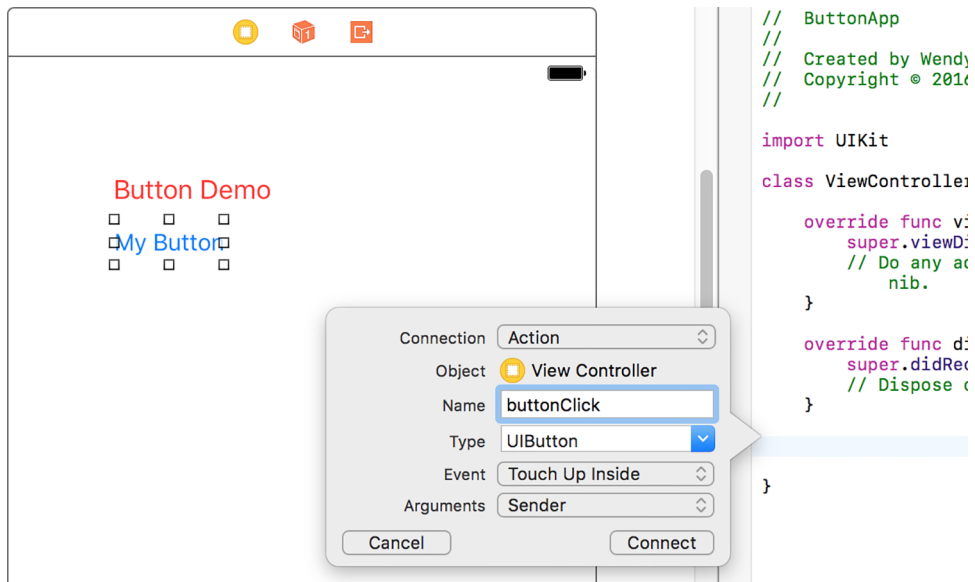


Figure 5.5 Change the Connection, Name, and Type values.

- 6 If the line says `@IBOutlet` instead, you need to delete the button and repeat the steps again, but select Action instead of Outlet in the dialog window that pops up. Your code should now look like figure 5.6.

```
//
// ViewController.swift
// ButtonApp
//
// Created by Wendy Wise on 8/20/16.
// Copyright © 2016 WisaAbility. All rights reserved.
//

import UIKit

class ViewController: UIViewController {

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

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

    @IBAction func buttonClick(_ sender: UIButton) {
    }

}
```

Figure 5.6 Button demo code after you click Connect. The `@IBAction` line is new. Make sure it looks like this.

STEP 3B: CONNECTING THE LABEL TO THE CODE

Now you're going to connect the label in almost the same way that you connected the button:

- 1 Control-click your label on the storyboard. With the Control key still held down, drag your mouse pointer over to the ViewController.
- 2 Let go of the Control key when your mouse is positioned under the class `ViewController: UIViewController {` line and above the `@IBAction` line that you added.
- 3 This time, when the dialog pops up, leave the connection as `Outlet` (at the top of the box, as shown in figure 5.7).

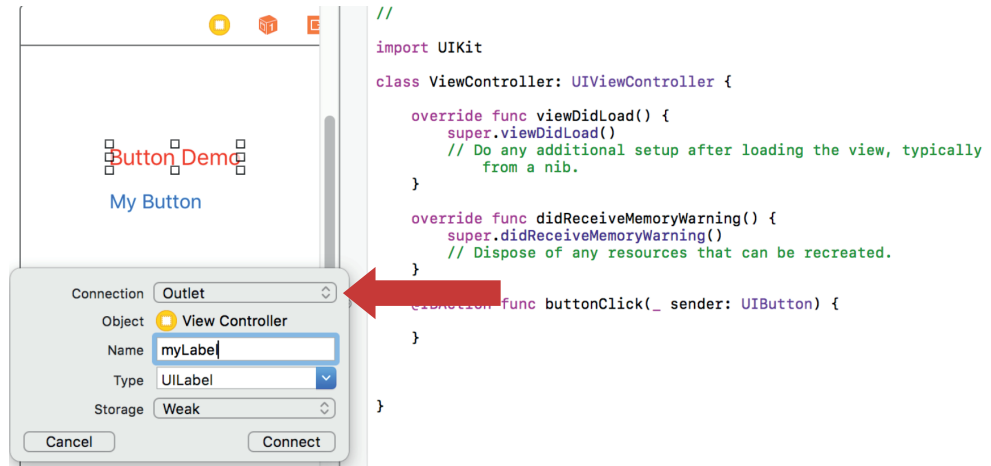
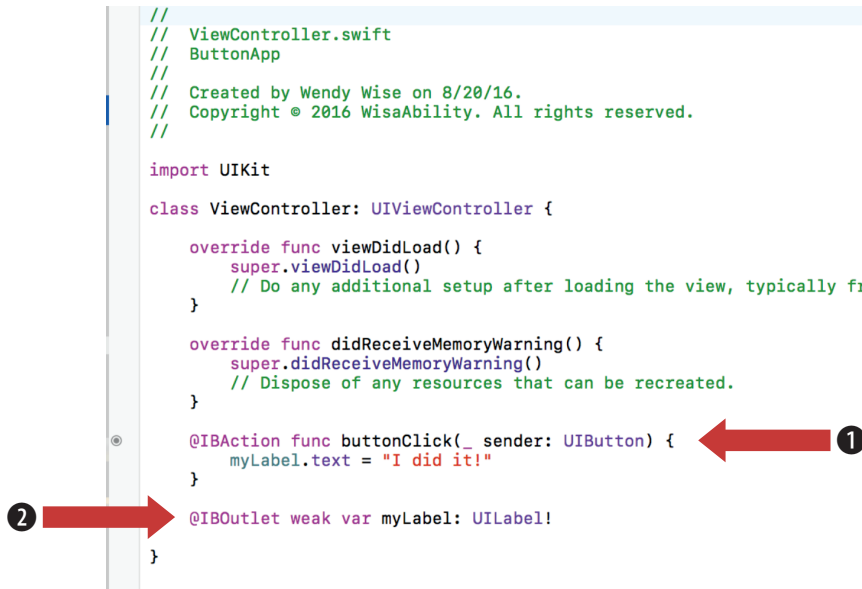


Figure 5.7 Connect the label to the code with the `Outlet` connection, and change the name to `myLabel`.

- 4 Change the name to `myLabel` and leave all the other options set as they are.
- 5 Click the `Connect` button.

Your code should look similar to figure 5.8.

Now run the app again to make sure it still works. The app should do exactly what it did before you wired up the label, because you haven't added any code yet. You run it here to make sure that it still works.



```
//
// ViewController.swift
// ButtonApp
//
// Created by Wendy Wise on 8/20/16.
// Copyright © 2016 WisaAbility. All rights reserved.
//

import UIKit

class ViewController: UIViewController {

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

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

    @IBAction func buttonClick(_ sender: UIButton) {
        myLabel.text = "I did it!"
    }

    @IBOutlet weak var myLabel: UILabel!

}
```

Figure 5.8 There should be two additions: ① @IBAction and ② @IBOutlet.

5.1.4 Step 4: Add code to change the text on the Label when the button is clicked, and run the app to test it

Next you'll go back to the code and add the following line right under the @IBAction func buttonClick(sender: UIButton) { line:

```
myLabel.text = "I did it!"
```

Make sure you add this line after the button-click line, but before the next } line. Your code should look like figure 5.9.

That code may look daunting, but I'll explain more about what you're doing and why in chapter 6. Now run the code again, and click the button. If everything worked, your button should now change the label to "I did it!" when you click it, as shown in figure 5.10. Great job!

If it didn't work, look back at your ViewController file and see if there are any red exclamation points with a big red highlight. If so, you have an error in your code. Check to make sure your code looks like my code in figure 5.9, and try again. If you accidentally created the button as an outlet or the label as an action, it won't work. Unfortunately, you can't delete the code in the ViewController because the storyboard still thinks there's a link to the code. To fix this (only if you accidentally created them backwards), you'll need to delete the linkage (the wires) between the

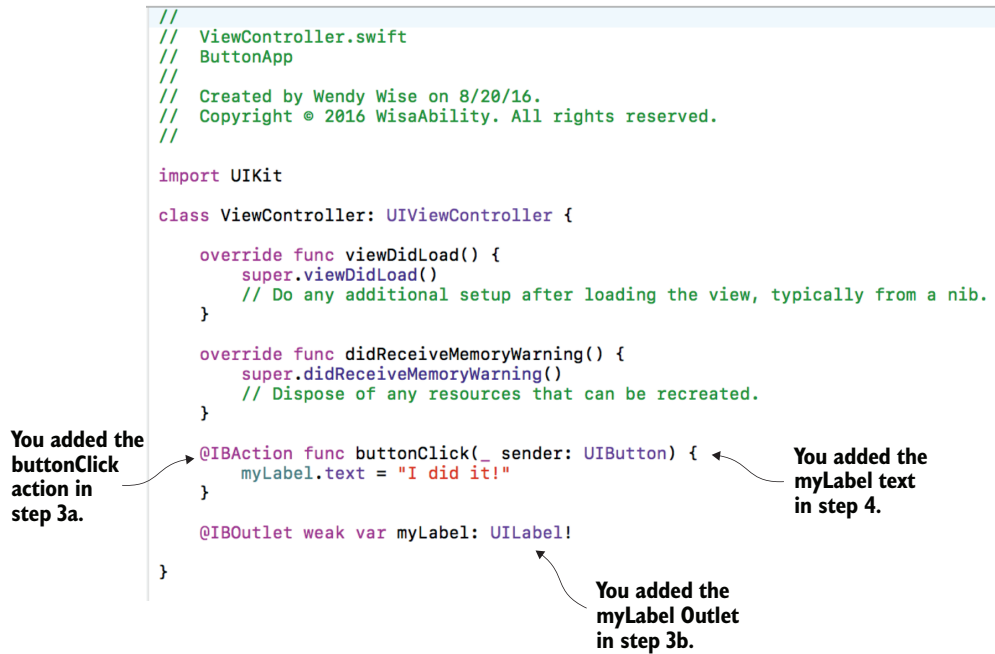


Figure 5.9 The final code for the My Button app, with @IBOutlet, @IBAction, and the line of code to change the text of the label

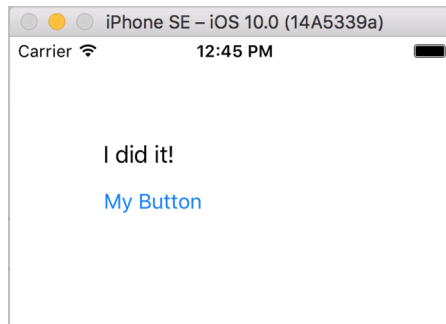


Figure 5.10 The working buttonDemo app after you clicked My Button should change the label text to “I did it!”

storyboard and the ViewController. Figure 5.11 assumes you accidentally created your button as an outlet instead of an action, but you can fix any incorrect “wiring” this way.

This is exciting—you made an app with a button that changed the text of a label! Now you’ll change how the label looks, and then I’ll explain what you did in chapter 6.



Figure 5.11 If you accidentally connected the objects to the code incorrectly, you need to delete the connection (wire) and the code following these steps.

5.2 Changing how the label appears

In this section, you're going to change some attributes of the label. We use the term *attribute* to describe information about something (such as a label). A label has *properties*, or descriptors of that label. Think back to the pen we talked about in chapter 3. Remember, the underlying definition of a pen was a *class*, and the pen in your hand was the *object*. Thinking about that class (the pen definition), what things can be changed on a pen without changing the fact that it's a pen?

You could change the type of pen to ballpoint, felt tip, or fountain. You could change the color of the ink to red, blue, or purple, so the ink color can change and yet it is still a pen. We could say that the color of the ink is a *property* of the pen. In other words, all pens *must* conform to the property of having an ink color. What if you tried to make the ink into lead? Well, that breaks the definition of a pen and makes it a pencil. *Lead* does not conform to the property of the pen. The color of the ink is a pen property, and the available colors that you can make the ink are *attributes* of that individual pen.

Think about the label you added to the My Button app. What was the font? What was the color? How big was the label? Those are all *properties* of the label. One property of the label is color. Color is a property of label, and *red* may be attributed to that property. You can do the same thing with the font. The font of the label is a *property*, and Comic Sans may be the attribute. Again, you can do the same thing with the type

of pen. The type is a property of pen, and an attribute could be *ballpoint*. These are key terms in the coding world.

5.2.1 Step 5: Change how the label looks, and run the app to test it

Now you'll change some properties of the label you added in the last section:

- 1 Click once on the label in your storyboard. With the label still highlighted, look at the right side of the Xcode screen.
- 2 If it isn't selected already, click the icon that looks like a fat arrowhead pointing down—this is your Attributes Inspector panel. It should look like figure 5.12.

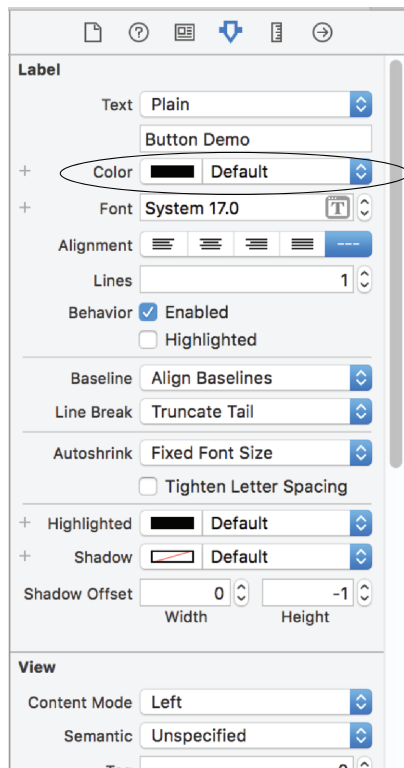


Figure 5.12 Select the label on the storyboard, and the Attributes Inspector will show the options available to change how the label looks. Click the Color attribute to change the color of the label.

These are the options for changing the properties of the label. You can change these settings now. Click the little up arrows next to Font System 17.0. Notice that the size of the font on your label increases as you click the up and down buttons. As you do this, you're changing the *attribute* for the font property:

- If you make the font too large, some of the letters on the label disappear and are replaced with ellipses (. . .). You can fix this easily by grabbing the corner of the label and dragging it out until your label is completely visible again. Grab

the squares on the right side of the label and drag to the right. Voilà! You can see the entire label again.

- Set the Font to System Font 20. Make sure your label is fully visible.

Now you'll change the color property of the label too, because you can:

- 1 Click the Colors selector right above the Font selector. A color wheel window will open, like the one in figure 5.13.

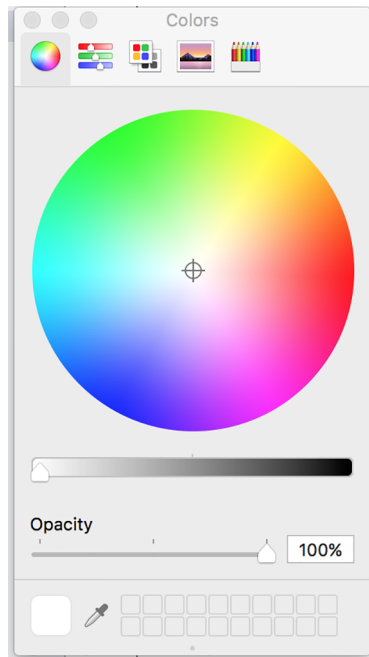


Figure 5.13 The color wheel opens after you click on the text label so you can change the color. Click any color in the wheel.

- 2 Pick a color you like by clicking inside the circle of colors. When you click the circle, the small box in the lower-left corner changes color to show you what you selected. You can continue to click different colors until you find the one you like.
- 3 Once you have it, close the panel. The label on your storyboard should now be the color you selected, and the Utilities panel should also show the color. I selected red for the attribute in figure 5.14.

Run the app again, and check that your label looks good in the Simulator. Take a little time and click through the label attributes on the Utilities panel to see what happens to the label. There are a lot of things you can do to make the label look different. If something changes and you can't figure out how to change it back, delete the label by selecting it on your storyboard and pressing the Delete key. Then add another label by following step 2 again.

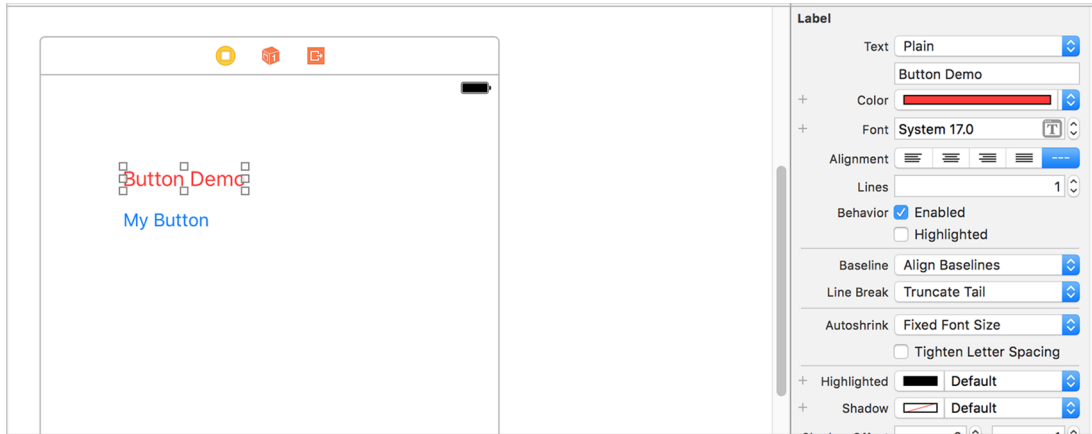


Figure 5.14 Change the label Color property to red by choosing red on the color wheel.

Concepts to remember

- Outlets let you access something on the user interface (the UI) by providing a reference to it (like Vanna being able to touch the letters on the game board and turn them around).
- Actions tell the code what should happen when a user interacts with a control object (like your button) on the UI.
- The most important thing to take away from this chapter is a high-level understanding of outlets, actions, and attributes. Chapter 6 talks more about outlets and actions, so stay tuned. You'll use these concepts for almost every app you build from here on.

5.3 Summary

This chapter covered a lot. You created an app with a label and a button, and you made the words on the button change when you clicked it.

You learned about outlets, but I didn't explain them in depth. Chapter 6 covers them, but for now, think of them as ways for the code to "talk to" the UI elements. The example we used was how our game-show host Vanna knew which puzzle to turn the letters around for.

You learned about actions, which are the way the user interface directs the flow of your code or tells the code to do something differently. In the code example, the "action" was for the user to tap the button and the code to then do something. In our real-world example, it was when the game show contestant yelled out a letter for Vanna. Chapter 6 covers this concept as well. You also learned about changing the attributes of a label, such as the font and the color.

The next chapter further explains the concepts from this chapter. This chapter taught you the basics of wiring up your user controls (buttons and labels) to your code. You'll use these actions throughout the rest of this book, so make sure you understand how to do it.

Anyone Can Create an App

Beginning iPhone and iPad programming

Wendy L. Wise



Do you have a fantastic idea for an iPhone app but no idea how to bring it to life? Great news! With the right tools and a little practice, anyone can create an app. This book will get you started, even if you've never written a line of computer code.

Anyone Can Create an App begins with the basics by introducing programming concepts, the Swift language, and the tools you'll need to write iOS apps. As you explore the interesting examples, illuminating illustrations, and crystal-clear step-by-step instructions, you'll learn to

- Get started programming, no experience necessary!
- Add controls like text boxes and buttons
- Keep track of your favorite things by creating the Like It or Not (LioN) app

By the end, you'll be able to create and run your own apps, and you'll have the confidence to learn more on your own.

This book is written especially for nonprogrammers—no experience needed!

Wendy Wise is a web and mobile developer with a talent for teaching app development to new programmers.

Updated for Swift 3

"An excellent way to get started on iOS development. A beginner-friendly approach that lets you learn at your own pace."

—Stephen Byrne, Dell

"A clear, concise guide for the uninitiated. Quick results guaranteed in simple steps!"

—Mark Cooper, Cooper and Lansbury Associates

"Highly recommended for anyone wanting to get their first app into the App Store."

—Jason Pike
Atlas RFID Solutions

"Approachable, easy to read, and easy to follow."

—Jocelyn Jeriah
MBO Partners

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit manning.com/books/anyone-can-create-an-app



manning

US \$29.99 / Can \$39.99
[including eBook]



9 781617 292651