ShowMe App

You create a working multi-view application; then you learn how to pass information from one view to another and display that information in the ShowMe application. Passing information between views is essential for many applications. Here, you simply pass text between views.

Once you've started the process of creating this app, you need to specify which application template you want to use. As with any other application created using Xcode, you need to start by creating a new project. Let's begin:

1. Create a new Xcode project by going to File ➤ New ➤ Project (+Shift+N) or, alternatively, clicking Create A New Xcode Project from the Welcome screen (+Shift+1).

2. Select Single View Application from the Project Templates dialog and click Next.

3. You're required to provide Xcode with those all-important little details such as Product Name, Organization Name, and so on. Figure illustrates the values to put in (enter your own first and last names in the relevant fields). For Product Name, use ShowMe.

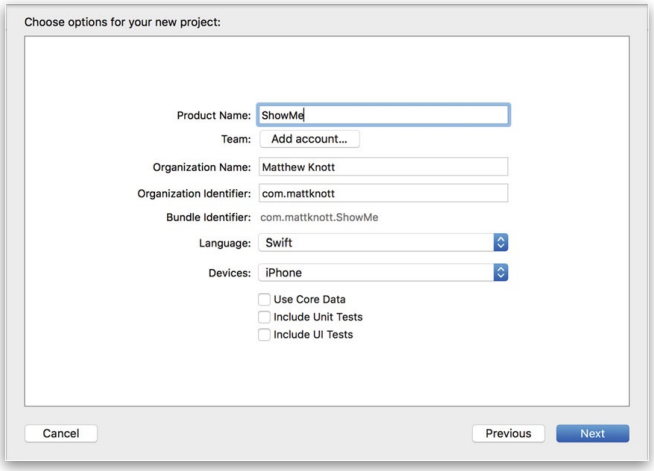

Figure. *Inputting the options to create the application*

4. For this project, you can specify your own name (that is, your first and last name)
   as the Organization Name.

5. For Organization Identifier, use reverse domain notation and enter com.YOURSURNAME.

6. Ensure that Device is set to iPhone and Language is set to Swift, and that you've
   unchecked Use Core Data, Include Unit Test, and Include UI Tests.

7. To finish, click Next. You're prompted to choose a location for your project. Save it somewhere that's easy for you to find, and ensure that Source Control is unchecked. Click Create. Now you're now ready to explore the many areas of Xcode.

Working with Interface Builder.

1. Open Main.storyboard from the Project Navigator. You have only a single view in the storyboard, but this will be a multi-view application, so let's add a second view. You can drag view controllers in from the Object Library. Drag in a view controller from the Object Library and position it to the right of the current view controller, as shown in Figure.

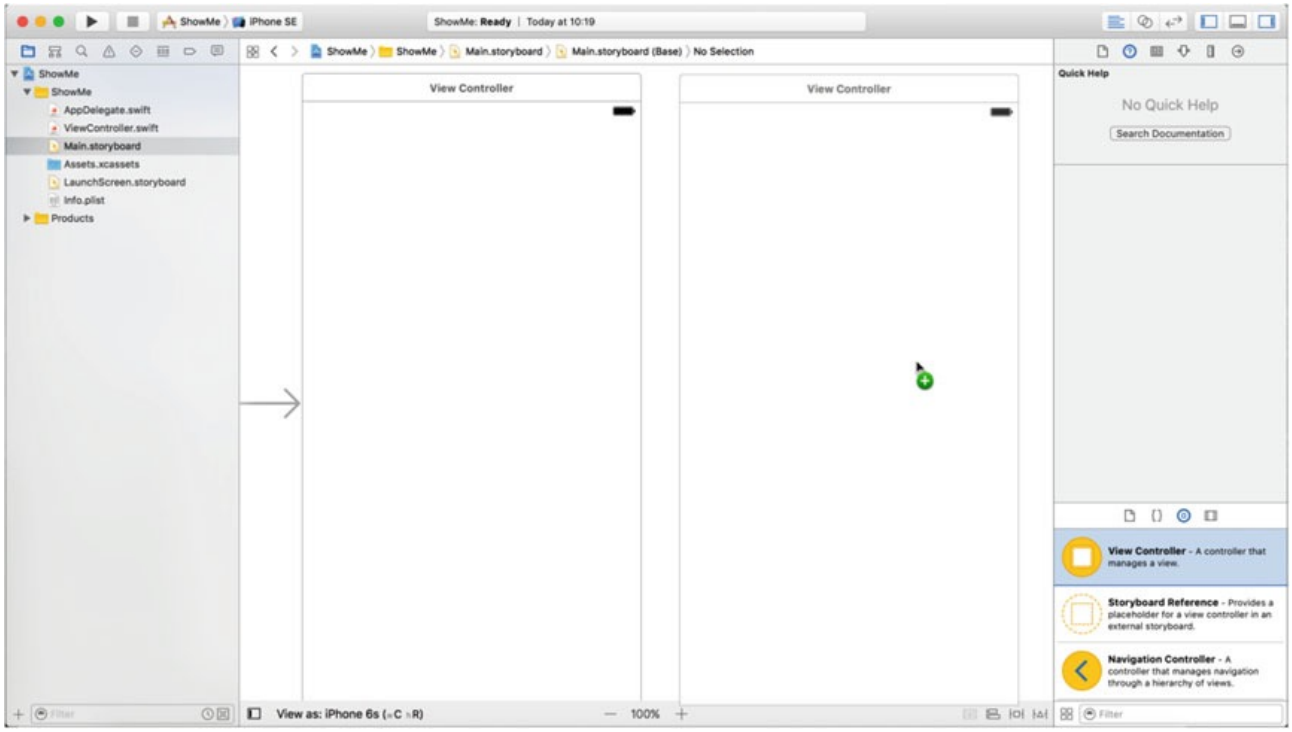

Figure. *Dragging a view controller from the Object Library to the design area*

2. Once you've released the view controller, use the small bar at the top of the view controller to maneuver it neatly beside the existing view controller, as shown in Figure.

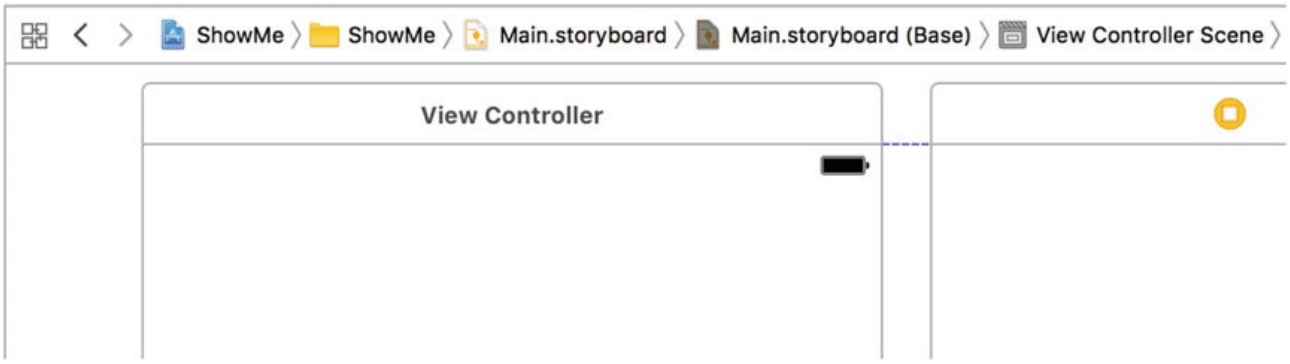

Figure. *Moving the view controller so that it snaps neatly in to place beside the existing view controller*

3. You now have two view controllers on the storyboard. A view controller in a storyboard needs a corresponding view controller code file in order to interact with the visual portion. You need to create a new view controller file called MessageViewController that subclasses UIViewController. Create the file by going to File ➤ New ➤ File... (+N) and selecting Cocoa Touch Class, as shown in Figure. Click Next.

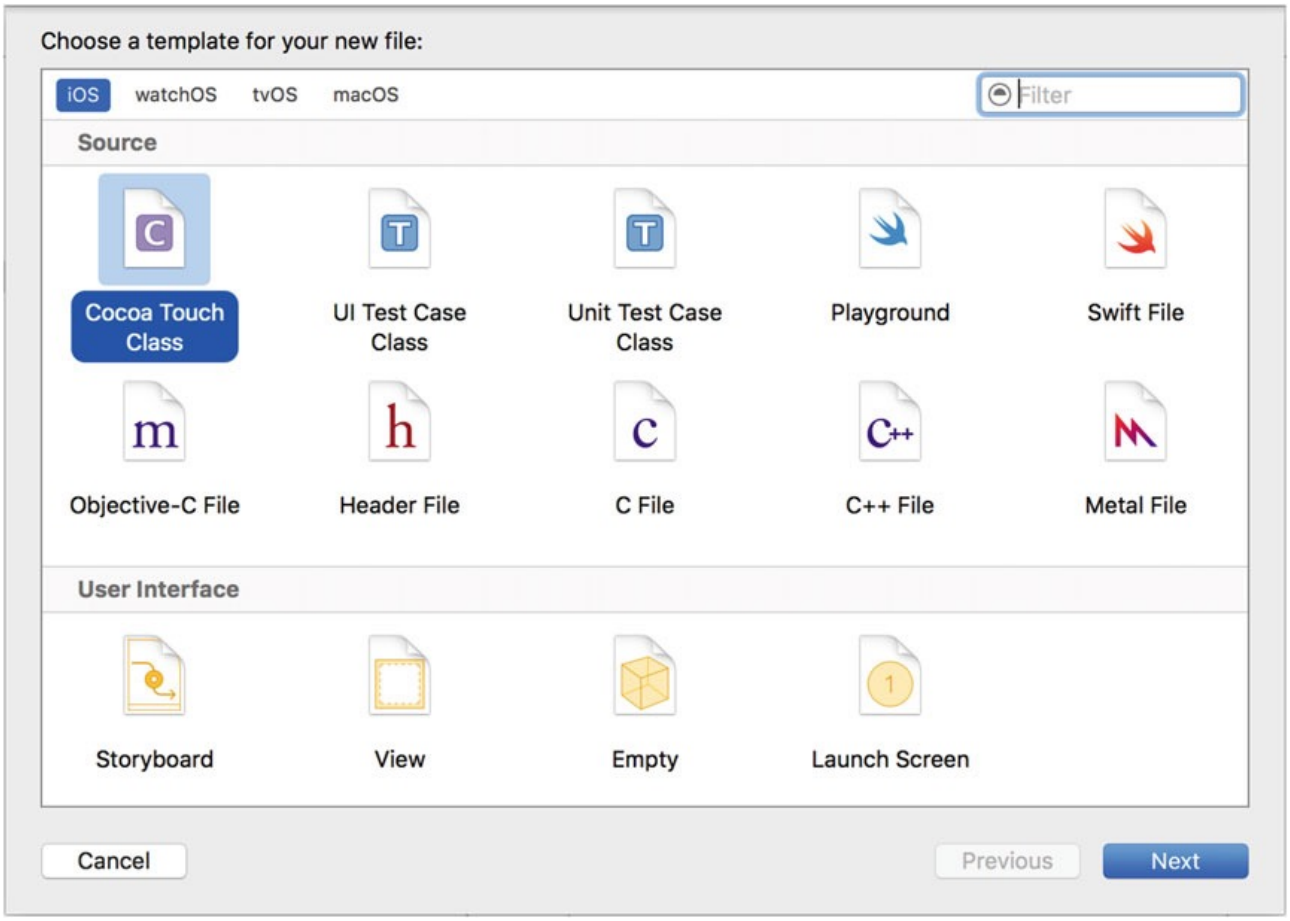

Figure. *The new file template selection*

4. On the next screen, you're asked for two values: Class and Subclass Of. For the Subclass Of value, you need to tell Xcode which class your new class is based on. If you were creating a class to hold custom properties, such as a Car class or an Animal class, you would use a generic NSObject; but in this instance you need a view controller, so set the value to UIViewController.

5. The Class value is largely up to you; this is the name you use to instantiate this view controller. When naming classes, always try to make the names semantically accurate—that is, they should describe the function of the class. This view controller displays the message it's sent, so let's name it MessageViewController.

6. Ensure that Language is Swift. Check that your values match Figure and click Next.

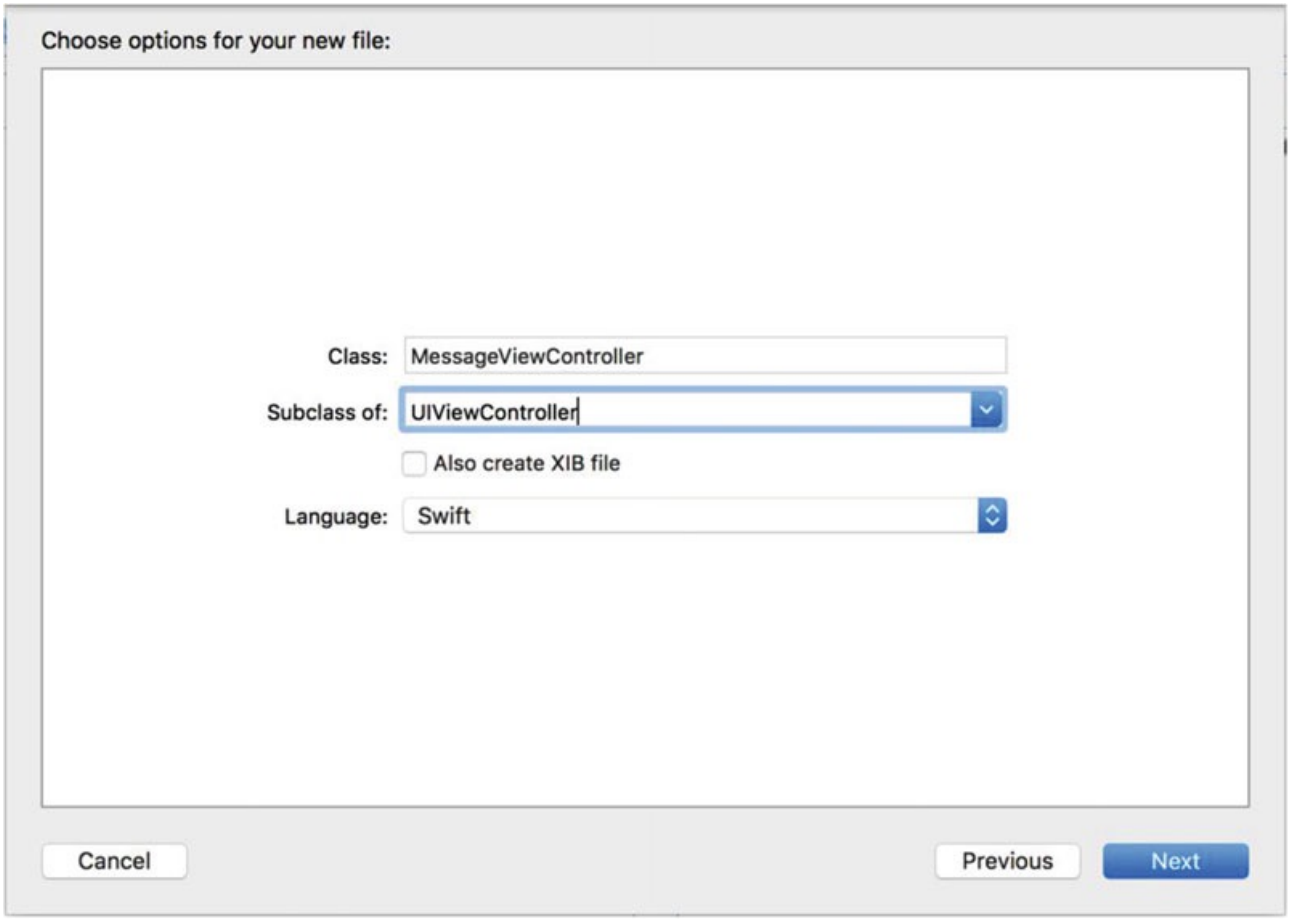

Figure. *Subclassing UIViewController*

7. On the next screen, you're prompted for a save location for your new file. Stick with the default settings. Ensure that in the Targets box, ShowMe is selected; then click Create.

You should now have a new file in your Project Navigator called MessageViewController.swift, which means you have all the files needed for the project and you're ready to put together the interface for the first view controller.

Reopen Main.storyboard from the Project Navigator. Before you add anything to the view, click the left view in the storyboard, and then look back to the utilities area on the right. You should see that the two tabs have become six tabs, as shown in Figure.

Figure. *The utilities tabs in Interface Builder*

. Before adding any objects to your view, you need to add an element called a ***navigation controller*** to manage the navigation back and forth through the different views. Xcode makes this very easy to do:

1. If you haven't already, select the left-most view in the storyboard by clicking the large white area in the design area. This should be the view with a large arrow pointing to its left side.
2. Go to Editor ➤ Embed In ➤ Navigation Controller, as shown in Figure. You should see a navigation controller appear to the left of the view. Once it's there, you can pretty much ignore it for the rest of the project, because the focus will be on the two views. Without it, navigating views would be extremely troublesome.

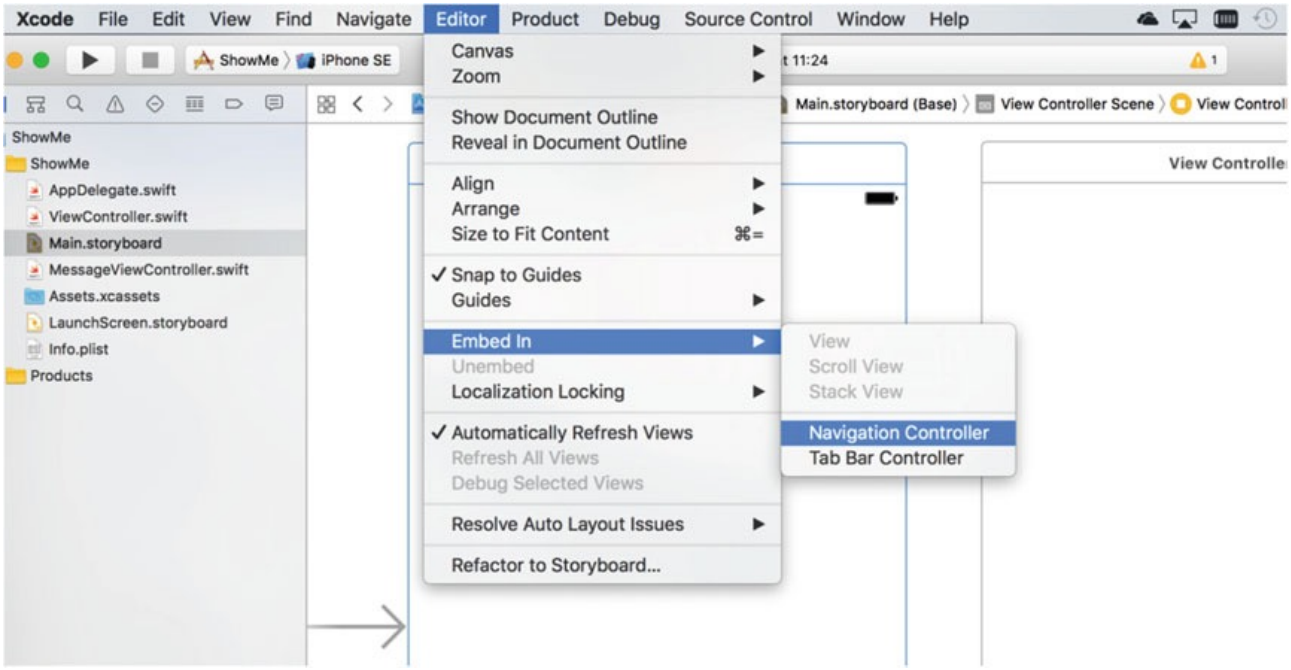

Figure. *Adding a navigation controller to the view controller*

3. Position the design area so you can see the first view, which is now attached to the navigation controller.

4. From the Object library, you need to add three items to the view. First, drag a label onto the view and position it toward the top of the view in the center.

5. Click the label once to select it, and then select the Attributes Inspector. Here you can really appreciate the range of minute customizations available to you.

The second property in the Attributes Inspector for this label is a text field that says Label. Change this to say “Text to Send”. You can also edit labels by double- clicking them on the view, but this method helps you see some of the minute adjustments you can make in the Attributes Inspector.

6. Click the T in the Font box and select Headline from the font list, as shown in Figure.

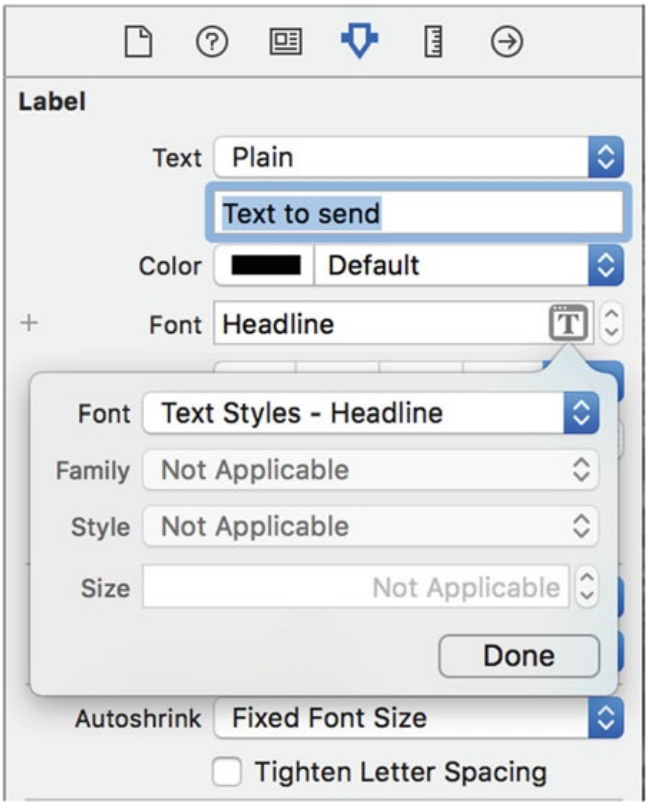

Figure. *Selecting the Headline text style for your label*

7. The label in the view is now too small to display the text you entered; rearrange the label by dragging one of the small handles in the corners until all the text is visible. You may need to re-center it after this.

8. Now that the label is in place, drag a text field from the Object Library onto the view and position it below the label. As you move the text field, you should get a feel for the vertical positioning as the object snaps into place below the label. Drag out the sides of the text field until a blue line appears on the side of the view; this indicates the view’s margin.

9. Drag a button onto the view from the Object Library. Position it a little below the text field. Then double-click the button and change its text to “Show Me”. That’s it! You’ve built your interface, and you should have something resembling Figure.

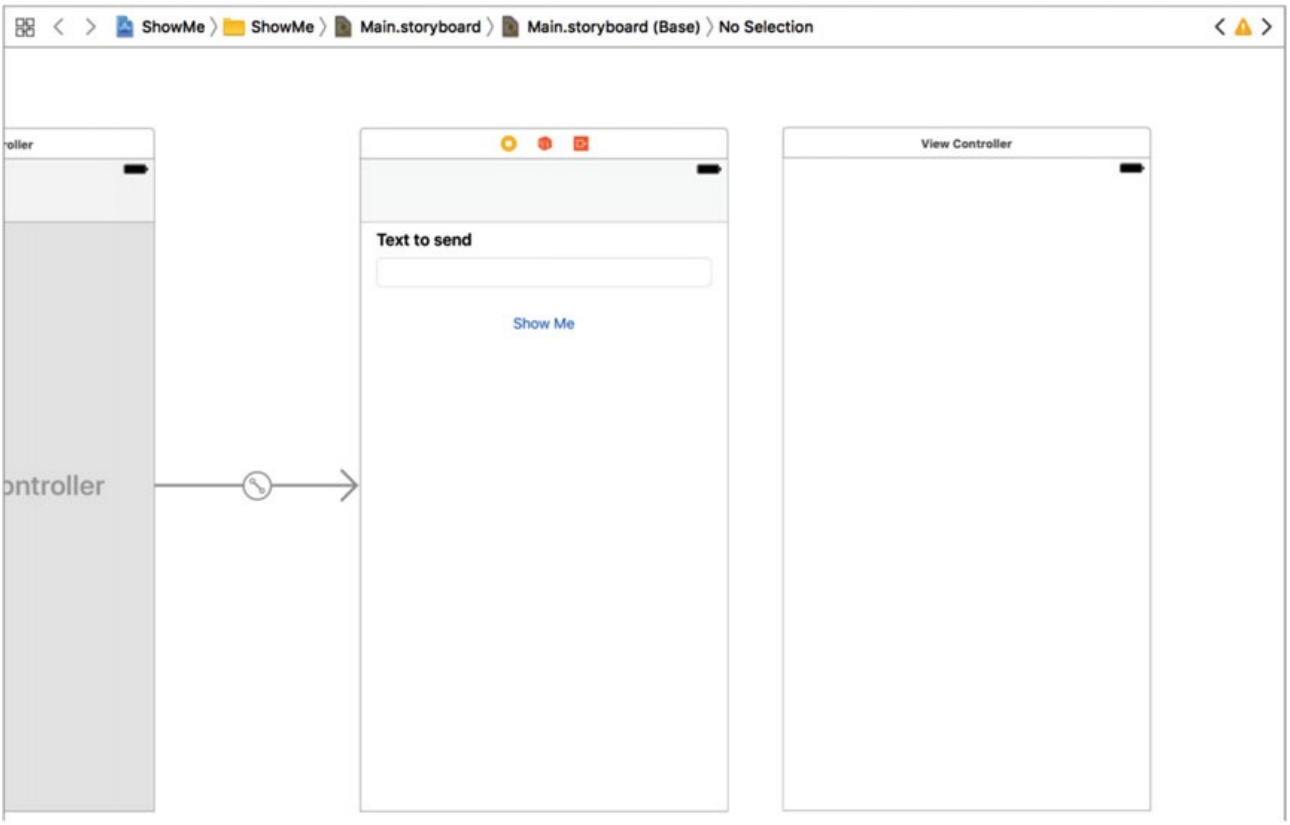

Figure. *The Main.storyboard with all of its elements*

10. You need to add some constraints in order to make sure the elements line up correctly when the application runs. Click a white area of the view, and then click the Resolve Auto Layout Issues button in the bottom-right corner of the design area. Select Add Missing Constraints, and Xcode will do the hard work for you, as shown in Figure.

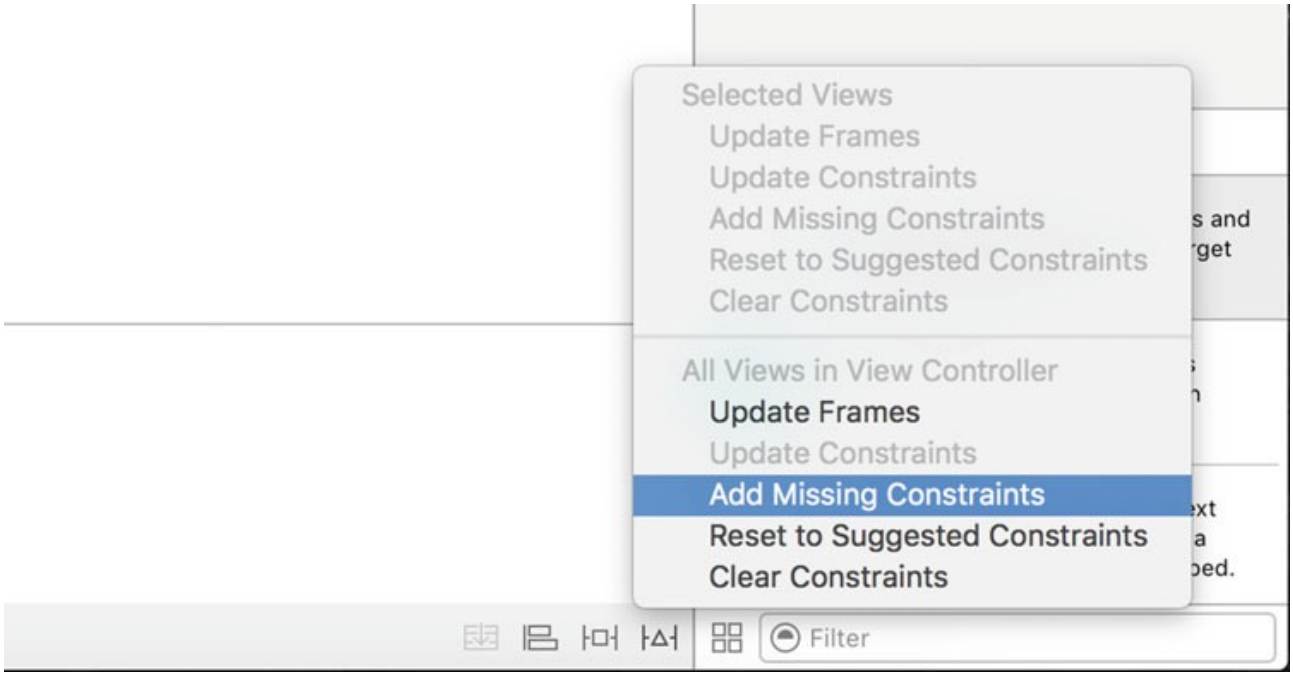

Figure. *Adding missing constraints to the view to get all elements lined up nicely*

The final thing you need to do is link the objects from Interface Builder into the view controller code as outlets and actions. As a challenge, do the steps without any visual aids. You can see if everything matches up at the end.

1. Open the Assistant Editor. You should see ViewController.swift displayed in the code editor portion. If not, ensure you have selected the correct view in the design area.

2. Select the text field. Holding down the Ctrl key, click and drag a connection to the classfile,positioningitjustbelowclass ViewController: UIViewController. Create an outlet named textToSendField.

3. Perform a similar action on the Show Me button. Control-drag another connection to the class file, positioning it below the outlet you just created; but this time when you release the mouse button, specify that you're creating an action, not an outlet, and name it showMe.

That's it for the first view for now. But before you move on, check that the code in your ViewController. swift file is the same as the following:

```
import UIKit
class ViewController: UIViewController {
    @IBOutlet weak var textToSendField: UITextField!
    @IBAction func showMe(sender: AnyObject) {
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view.
    }
    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

}
```

You've covered a lot of ground so far in this practice, so let's take a look at the application in action. Run the application in the Simulator. The output should resemble Figure.

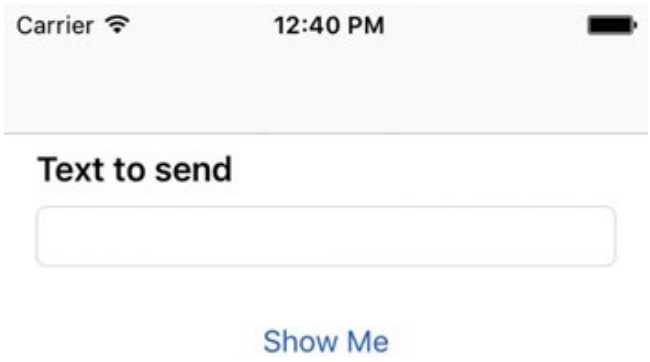

Figure. *The application running in the Simulator*

At this point, if your app runs successfully and you're staring at the result of your hard work, give yourself a pat on the back. You've built the first part of the application; now it's time to configure the second view controller and bring the two together using a linkage called a ***segue***.

Configuring the Second View Controller

Hopefully you're feeling pleased with what you've done so far. You should be—but the application isn't finished. The idea behind this application is to type some text in

the text field and have it display on another view controller when you click the Show Me button. Using storyboards to build an application comes into its own when you're working with multiple views. First you need to create a linkage called a ***segue*** between the Show Me button and what will be the Message view controller; by creating this linkage, you can move between view controllers without writing a single line of code. After you've created the segue, you need to create the second view's interface and, finally, the underlying code to tie it all together.

Now that you know what you're aiming for, let's get started. Again, you'll be tested on how much you remember from creating the previous view controller:

1. Switch back to the Standard Editor by clicking the button to the left of the Assistant Editor on the toolbar.

2. Position the storyboard so that you can see both view controllers.

3. Click the Show Me Button, and then Control-drag a connection from it to the view controller on the right, as shown in Figure.

Figure. *Making a connection from the Show Me button to the second view controller*

4. When you release the mouse button, a dialog appears with a number of options in it; these are the different types of segues available. Select Show, as shown in Figure.

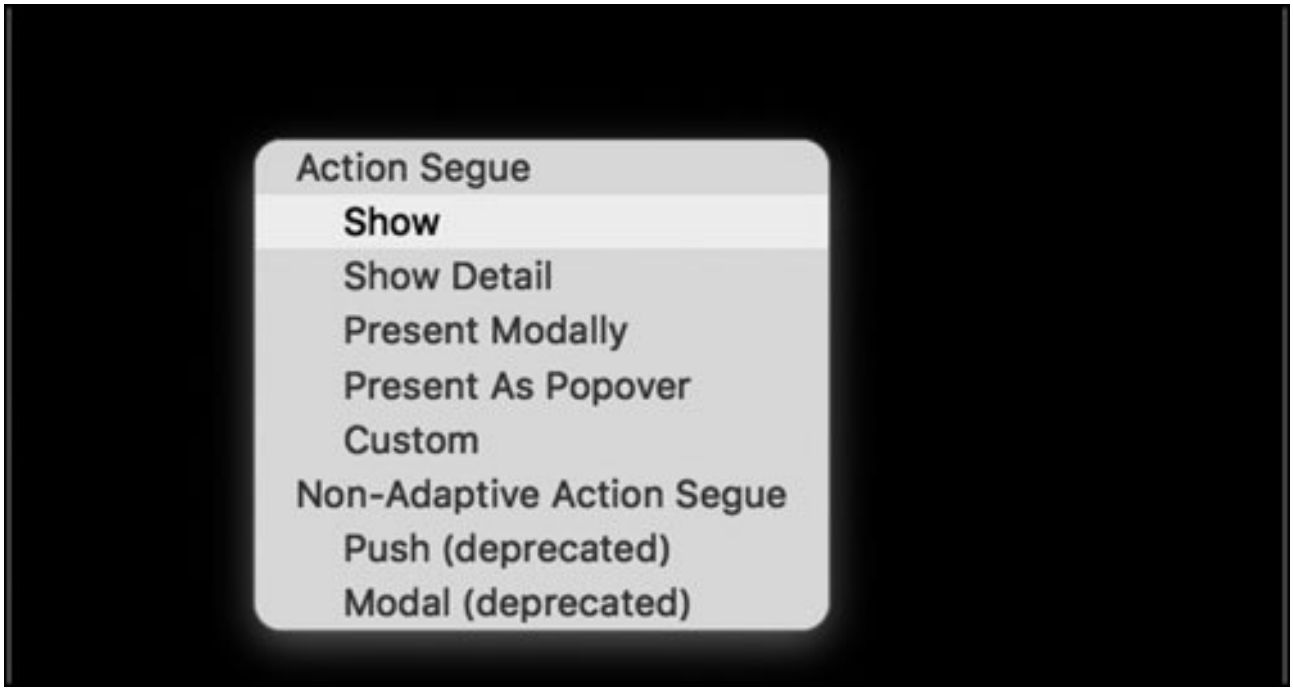

Figure. *Selecting the Show segue type*

You've created a relationship between the button and the second view controller. To see what this means, run the application again. Note that when you click the button, the second view slides in nicely and there is a Back button to take you back to the initial view, but it's all a bit simplistic at the moment. Let's finish configuring the second view controller.

5. Stop the Simulator and then align the storyboard so that you can see the second view controller.

6. Drag two labels onto the view, one below the other, and position them near the top of the view under the navigation bar placeholder. Double-click the first label and set its text to "You Said...". Then resize the second label so that it's the width of the view.

When you created MessageViewController.swift, it was so that you could interact with the visual output of the second view. The next step is to link this view controller to the custom MessageViewController class, using the Identity Inspector.

7. Click the bar at the very top of the right view controller and then open the Identity Inspector, just to the left of the Attribute Inspector. Change Class to MessageViewController, as shown in Figure.

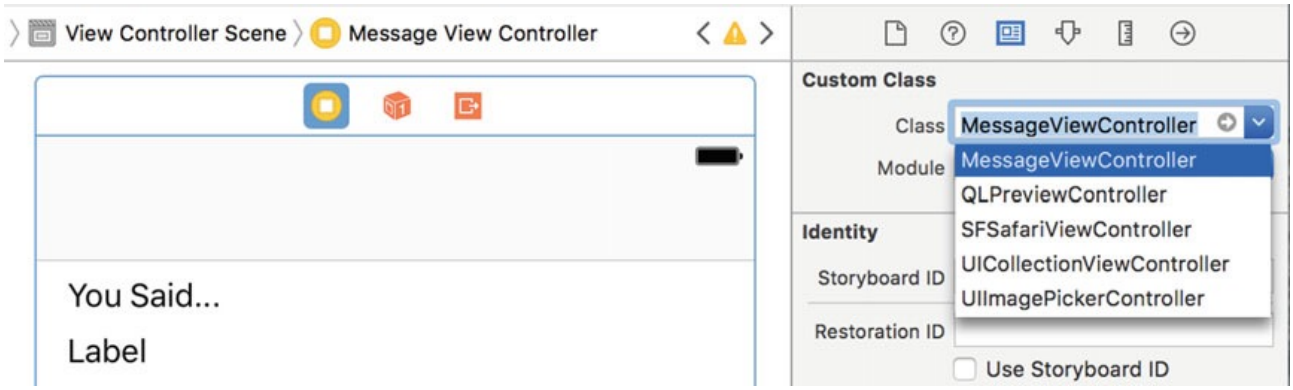

Figure. *Selecting the MessageViewController class for the second view controller*

8. Now that you've created this relationship, turn on the Assistant Editor and, with MessageViewController.swift showing in the code portion, Control- drag a connection from the bottom label to below the line starting class MessageViewController and create an outlet named messageLabel.

9. Click the view and then add the constraints required for a flexible layout by clicking the Resolve Auto Layout Issues button and clicking Add Missing Constraints.

Hopefully, your view should now resemble Figure.

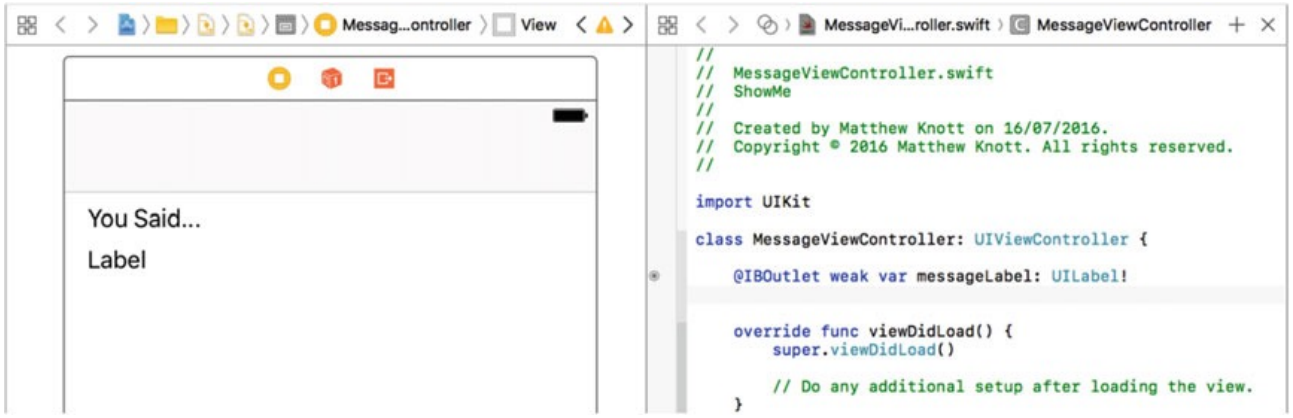

Figure. *The complete,very simple layout for the second view controller, called MessageViewController*

Before you go any further, let's recap what the objective is for this project. The users should be able to push the Show Me button on the ViewController and have whatever

they have written in the text field appear in the MessageViewController. To make this happen, you need to add some code to the MessageViewController class file so it can receive the message from the initial ViewController.

▪ ***Note*** *When a new view is loaded on to the screen by a navigation controller replacing another one in an iOS application, this is referred to as* ***pushing*** *a view.*

The ViewController will interface with the MessageViewController using a custom initializer that will accept the text passed from the ViewController.

1. Switch back to the Standard Editor and open MessageViewController.swift from the Project Navigator.

2. Create a variable to hold the message data to be displayed. The data coming from the first view controller is text, so create a String variable by adding the following highlighted code just below your outlet:

class MessageViewController: UIViewController { @IBOutlet weak var messageLabel: UILabel! var messageData: String?

3. Next, you need to take the supplied text and display it on the label. You do this by setting the text property of the messageLabel outlet. Add the highlighted code below to the viewDidLoad method:

```
override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view.
    messageLabel.text = messageData
}

// MARK: - Navigation
// In a storyboard-based application, you will often want to do a little preparation
// before navigation
override func prepareForSegue(segue: UIStoryboardSegue, sender: Any?) {
    // Get the new view controller using segue.destinationViewController.
    // Pass the selected object to the new view controller.
}
```

Now all that remains is to complete the code to send the data to the MessageViewController.

1. Open ViewController.swift from the Project Navigator.

2. Underneath the didReceiveMemoryWarning function, paste in the code you copied from the other view controller using Edit ➤ Paste (+V) so that the bottom of your code file looks exactly like this:

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

3. The prepare(for:sender:) method is called just before a segue linking view is executed. Normally, you would name your segue and control the actions based on the segue that has been triggered, but in this instance, you only have one segue, so when this method is called, you know that the destination of the segue is the MessageViewController. You're going to create an instance of MessageViewController based on the segue's destination controller, and then set the messageData string with the text of your text field. Add the following highlighted code into the prepare(for:sender:) method, removing the two comments:

```
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    let messageController = segue.destination as! MessageViewController
    messageController.messageData = textToSendField.text
}
```

Xcode automatically understands that you have a MessageViewController class without having to make any specific references or include statements in your code file, and when you type messageController. on the second line of the method, it knows you have a variable called messageData ready and waiting for you to pass it the contents of the text field.

That's it; you should now be able to run the application and find that you can click in the text field, type a message, and click Show Me. This will take you to the MessageViewController and display whatever you typed, as shown in Figure.

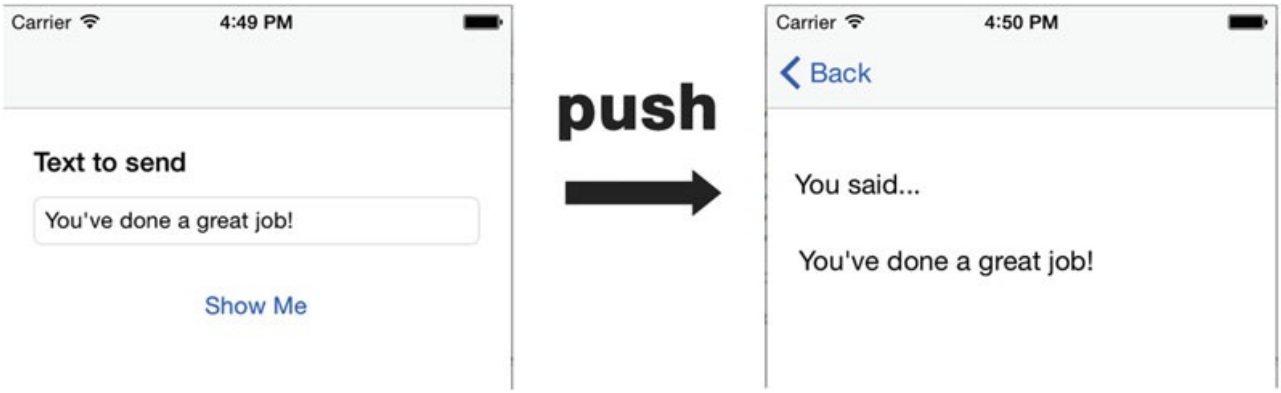

Figure. *The finished application*

Debugging Area

The final focus is the debugging area. In order to see this in action, you need to add some code to the project that captures the text that was entered when the button is clicked, and you use NSLog to add a message to the console. Go to ViewController.swift and go to the showMe action that you've left empty so far. Add the following highlighted code:

```
@IBAction func showMe(sender: AnyObject) {
   NSLog("User Wrote: %@", textToSendField.text!)
}
```

The debug area allows you to pin down any issues with your program. Because the debug area can quickly become very complex and can be used for a variety of different things. Now run your application and try to click the Show Me button without adding any text to the text field. You should see that when you click the button, a message is added to the output console, as shown in Figure.

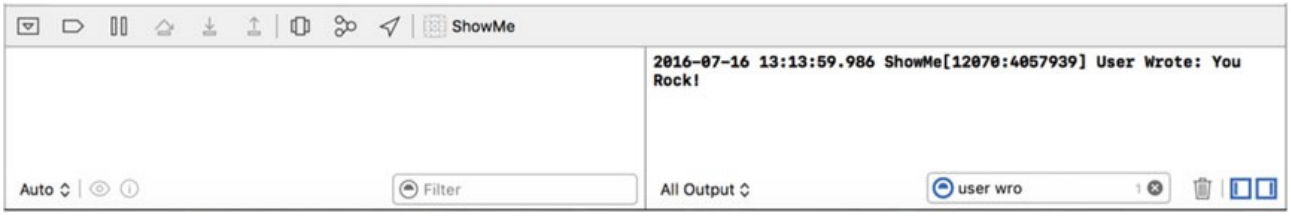

Figure. *The result of the NSLog method call is displayed in the output console*

For now all you need to know is that the debugger included with Xcode is the LLVM-GCC debugger, this means you can debug a variety of code in a variety of languages. This is especially useful as Swift can use frameworks and libraries written in Objective-C.

Summary

This practice covered two rather contrasting topics: project templates and the basics of Xcode's interface. The purpose of doing so was, first, to give you the confidence to start an Xcode project and choose correctly a project template suitable for your projects and, second, to drop you in at the deep end to understand how to start with nothing and build a working application, while providing a basic tour of the key parts of Xcode's interface.

More specifically, you:

- Created an application with multiple views
- Passed an object from one view controller to another
- Learned about segues and the hugely important prepareForSegue method.

***