

My Life Quick Start

Welcome to the My Life App!

This app is a starting point for building your first iPhone or iPad app. You'll create an app dedicated to the important people in your life while learning the basics of iOS development. In the app, you can describe the things that make your friends and family memorable and special to you.

This guide assumes that you're familiar with basic programming concepts but have never used Xcode to make an iPhone or iPad app.

This is version 1.0 of the MyLife sample app and guide. These are meant to be used with Xcode 8 and the iOS 10 SDK or later.

How to Use This Guide

This quick start will help you understand how the My Life app works and will teach you several ways to customize it. Feel free to use or ignore any of the suggestions in this guide, and modify your app as you see fit—there's no right or wrong design!

This guide gives you ideas for how to modify the app to further describe your friends and family. You can even modify the app to be about your favorite movie characters, or the places in the world that you most want to visit (or have visited). Don't feel limited by the ideas in this guide; if you want to add something, search online to find out how to do it.

This guide might look long, but that's because it helps you with various app-development tasks that are broken into steps with accompanying screenshots. If you get stuck or need help following any of the instructions in this guide, check out the [Apple Developer Forum](#) that's dedicated to learning to code.

Tip: If you are reading this in Xcode, you might find it easier to read this document by double-clicking the file to open it in a new window. Or, if you want Xcode to display it side-by-side with your code, you can hold the option key (⌥) and click the file to open it in a second window called the Assistant Editor.

Getting to Know the App

To get a sense of what the app does, you can start by running it in a simulator. At the top of the Xcode window, click the device you want to simulate (for example, the iPhone 6s), and then click the Run button or press ⌘R




When the app runs, you'll see a table with three rows. In each row is a person's name and image. You can tap any row to go to a second page (in coding, this is known as performing a *segue* to a new *view controller*).

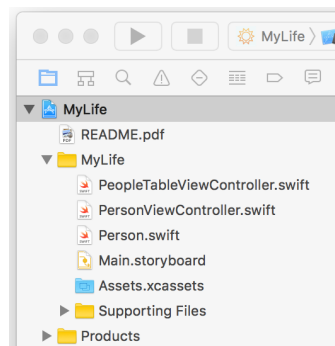
On the second page, you can edit the text that represents the person's name, or drag the slider to adjust how much they like dogs. Your changes are stored only if you tap the Done button; if you tap Cancel, the changes won't appear on the main page.

You can also add a friend by tapping the Add button (+) on the first page, but there's no way to add an image for your friend yet. This is a feature you'll learn how to add later!

Note: You'll notice that if you run the app a second time, your changes are gone. Right now, the app doesn't save your data between app launches. This is another feature you can add later.

Understanding the App Project Files

The app contains a number of files that it needs to make everything work. On the left side of the window is the navigation area, which lists all the files in the app. If this window isn't present at any point, you can press  to make it visible.



App files

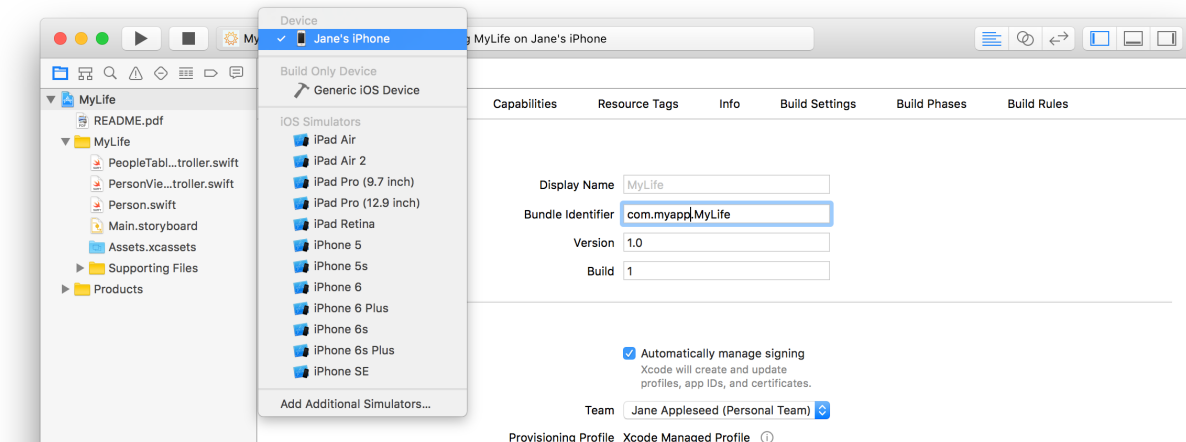
- `Person.swift`—A class that defines the data stored for each person displayed in the app. To change the information you store about someone, you modify this file.
- `PeopleTableViewController.swift`—The initial view controller displayed when you launch the app. This controller handles populating the table view using data from an array of people (one person per row). It also handles transitions that occur when you tap a table cell or the Add button.
- `PersonViewController.swift`—The view controller for viewing and editing the information for a specific person. This controller configures the display based on a `Person` class when it loads, and it saves your changes when you tap Done.
- `Main.storyboard`—A file where you can lay out view controllers and the views they display such as text, images, and other interface elements. You can also set up transitions (*segues*) between view controllers.
- `Assets.xcassets`—The asset catalog, which contains images that will be used in your project, including the app icon and the images you see when the app loads.

There's also a folder called `Supporting Files` that contains additional files the app requires. You don't need to worry about these files for now, because you won't be modifying them yet.

Note: You'll notice in this guide that `Person` is sometimes capitalized. This is a reference to the `Person` class.

Running the App on Your Device

If you have an iPhone or iPad, you can run the app directly on your device. Any app you install on your device needs to be code-signed, but if you have an Apple ID, Xcode can take care of most of those steps for you.

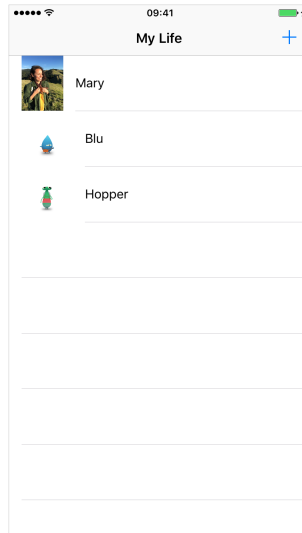


1. Connect your device to your Mac.
2. In the toolbar at the top, click the iOS Device to the right of MyLife, and then choose your device from the pop-up menu.
3. In the navigation area on the left, click MyLife.
4. Change the Bundle Identifier to remove “example.apple-samplecode” from the name and put in something different here; for example, com.myapp.MyLife.
5. Click Add Account, and enter your Apple ID and password.
6. In the provisioning selection, choose the Team selector and select your name.
Xcode connects to Apple and generates a code-signing certificate for you.
7. Click the Run button, and when prompted to enable Developer mode, click Enable.
8. When prompted, enter your system password.
9. On your device, choose Settings > General > Profile & Device Management, and tap the row that contains your Apple ID.
10. Tap Trust, and tap again to confirm.

Now you're all set to use and run the app on your device!

Customizing Your App

When you're building an app, you're in charge. There's a limitless variety of ways you can customize your app to do whatever you want. This section describes modifications you can make, although you shouldn't feel you have to follow this list exactly (or at all):

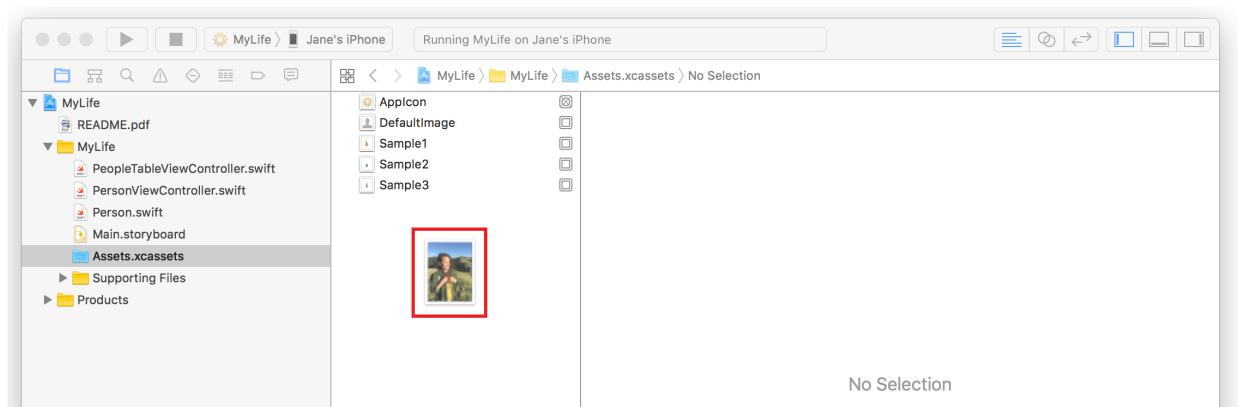


Customization 1: Displaying Your Friends

The app contains sample data, but you can make it more personal by including the names and images of your friends (and how much they like dogs!). This customization will show you how to do that. When you're finished, your app will look something like this:

First, you'll need to add your friends' images to Xcode:

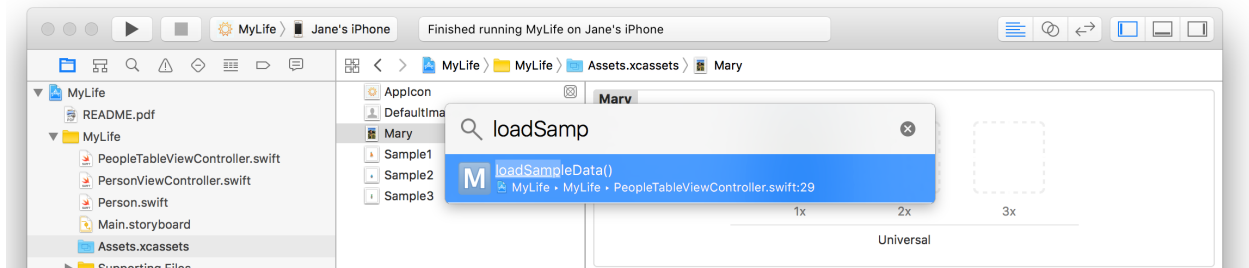
1. In the project navigator, select `Assets.xcassets`.



2. Find an image you want to use and drag it to the left window of the asset catalog.
3. Select the image you dragged over, and press Return to rename it.

Next, you'll need to modify the code to use this image. The code that loads the sample data is the `loadSampleData()` function in `PeopleTableViewController.swift`.

4. You can quickly jump to the `loadSampleData()` function with Xcode's Open Quickly command. Press Command-Shift-O to bring up the Open Quickly menu, start typing `loadSampleData`, and press enter when `loadSampleData()` appears as the first result.



5. Now that the `loadSampleData()` function is onscreen, you can load your new image by changing "Sample1" to the name of the image you added.
6. To change the name, change "Byte" to the name of your friend or family member.

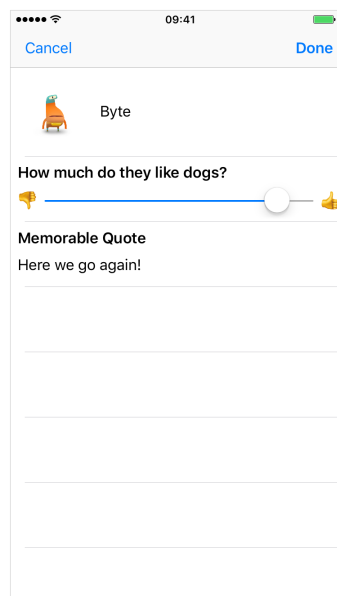
Build and run the app to show the updated name and the image you added!

Customization 2: Adding Information about a Person

Right now, each `Person` contains an image, a name, and a number describing how much they like dogs, but what if you wanted to add new descriptions?

There are a lot of things you can add to describe your friends; for example:

- You can add a memorable quote for each person. In coding terms, you can store the quote as text (`String`) and allow editing of the quote in the app by including an editable text field (`UITextField`). You should also describe the text field with a label that can't be edited (`UILabel`), which can say "Memorable Quote." An example of this is below.



- You can also add a yes or no question, such as “Do they like cats?”. You place the question in a label (`UILabel`) and represent the possible responses with a switch that you interact with in the app to change from yes to no (`UISwitch`).

In each case, you need to think about what type of information you want to store, and the corresponding interface elements that will be used to show the data in the app. This guide walks you through the first example below, but there are many other interface options.

The following are some of the interface elements that you can use:

- `UITextField`—Displays a single line of editable text.
- `UISwitch`—Provides a switch with on and off positions for Boolean data.
- `UISlider`—Provides a slider for selecting a value in a number range.
- `UIImageView`—Displays an image.
- `UILabel`—Displays uneditable text.

But you're not limited to only these options. More are included in the user interface framework for iOS (called `UIKit`). You can also create your own.

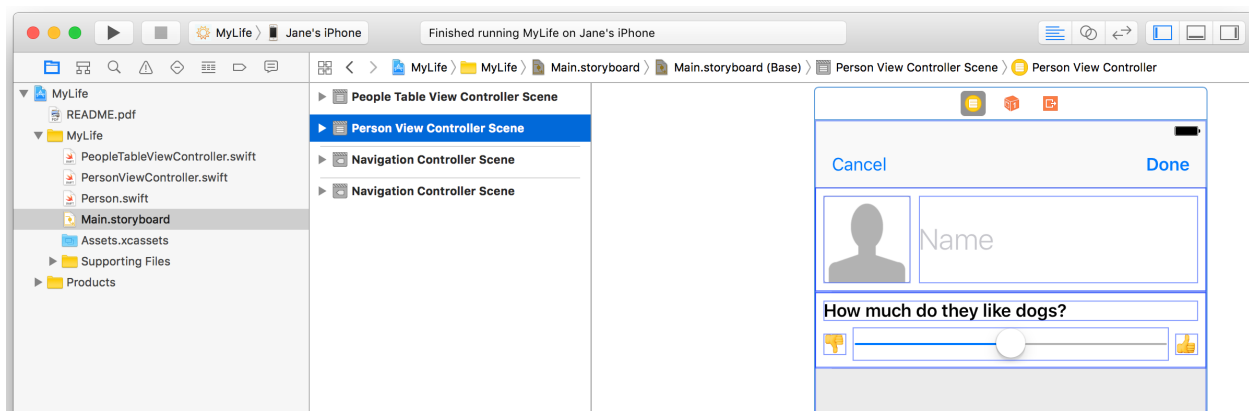
A few general steps are needed to add new ways to describe your friends and family:

1. Adding the interface element to your view controller in the storyboard.
2. Making your `PersonViewController` aware of the view in your storyboard so it can modify interface elements and respond to input.
3. Updating the `Person` class with a new variable for the data that will be stored.
4. Modifying your `PersonViewController` to load and save this new data when the page loads and when you tap Done.

Now you'll go through the steps to add a memorable quotation to each `Person` and allow quotations to be edited within the app. Throughout the example, you'll also get tips for doing something similar for other types of data.

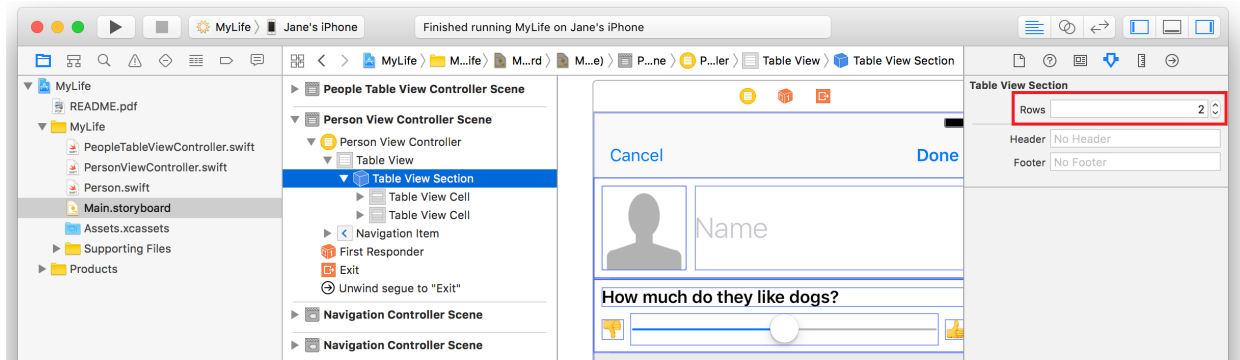
Adding Interface Elements to the Storyboard

1. Open the `Main.storyboard` file, and click the `Person View Controller Scene` in the menu on the left side.



In the view controller, you'll see a table view that's set to always have two rows; this is known as a *static* table view.

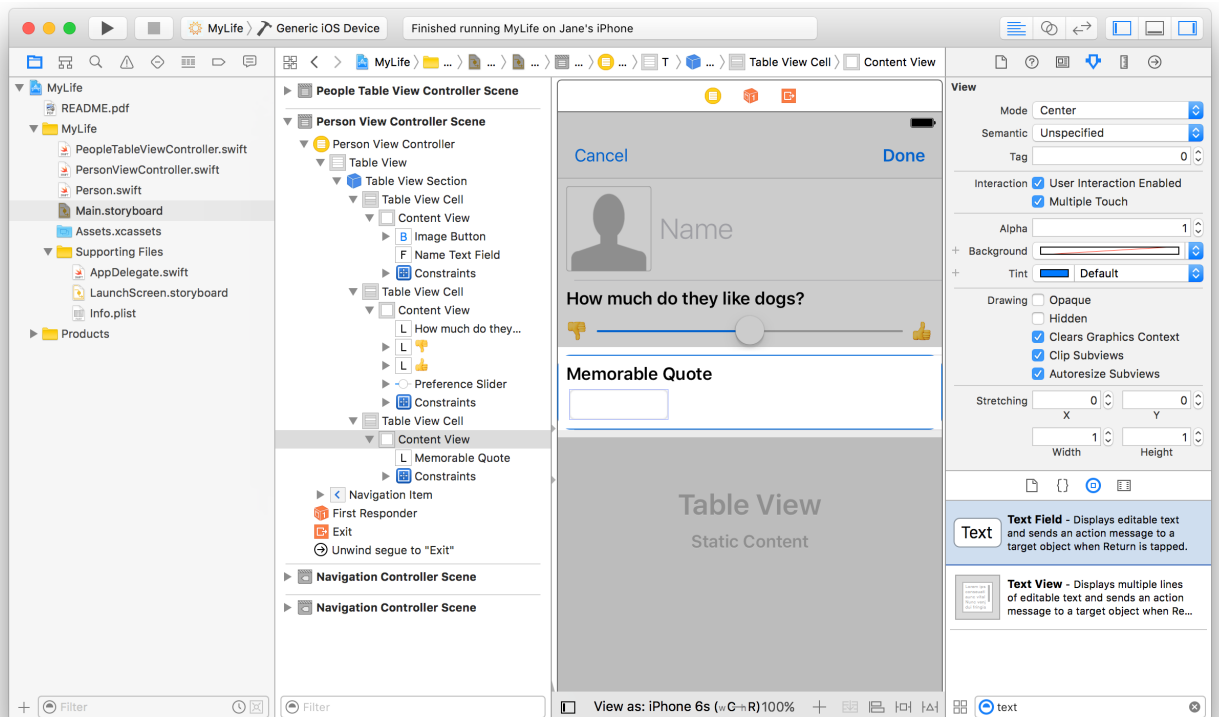
- Expand the Person View Controller menu on the left, choose Table View Section, and then choose View > Utilities > Show Attributes Inspector, or press option-command-4 ($\text{⌘} \text{⌘} 4$).



- Change the Rows value to 3.

You'll see a new table row appear, with the same text and slider as in the second row.

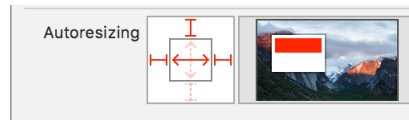
- To remove the new slider and views displaying the emojis in the new row, select them and press Delete.
- Click the label, and in the attributes inspector, change the Text attribute to something that will help you understand what this row represents when the app is run; for example, Memorable Quotation.
- Next, add a text field that the user can type in. To do this, you need to open the Objects Library in View > Utilities > Show Object Library. The library contains various objects that you can drag onto your storyboard and use. You'll need a "Text Field"; so type "text" in the search bar to filter the list, or scroll through the list to find it. Drag this text field onto your third row below the label.



7. Drag the text field around to change its position, and modify its width however you like by dragging either end of it.
8. To ensure that the text field will be resized to fit across any size device, you can fix the text field to always be this same distance from the edges of the table cell. With the text field selected, open the Size Inspector by pressing option-command-5 (⌥⌘5) or by going to View > Utilities > Show Size Inspector.
9. Next to Autoresizing, click to highlight all of the horizontal lines as in the image below. The text field will now always be the same distance from the left and right sides of the table cell.

To further customize, you can also use the Attributes Inspector to modify how the text field and label are displayed. Try experimenting with changing the border style, modifying the font, or even providing placeholder text that appears when the text field is empty.

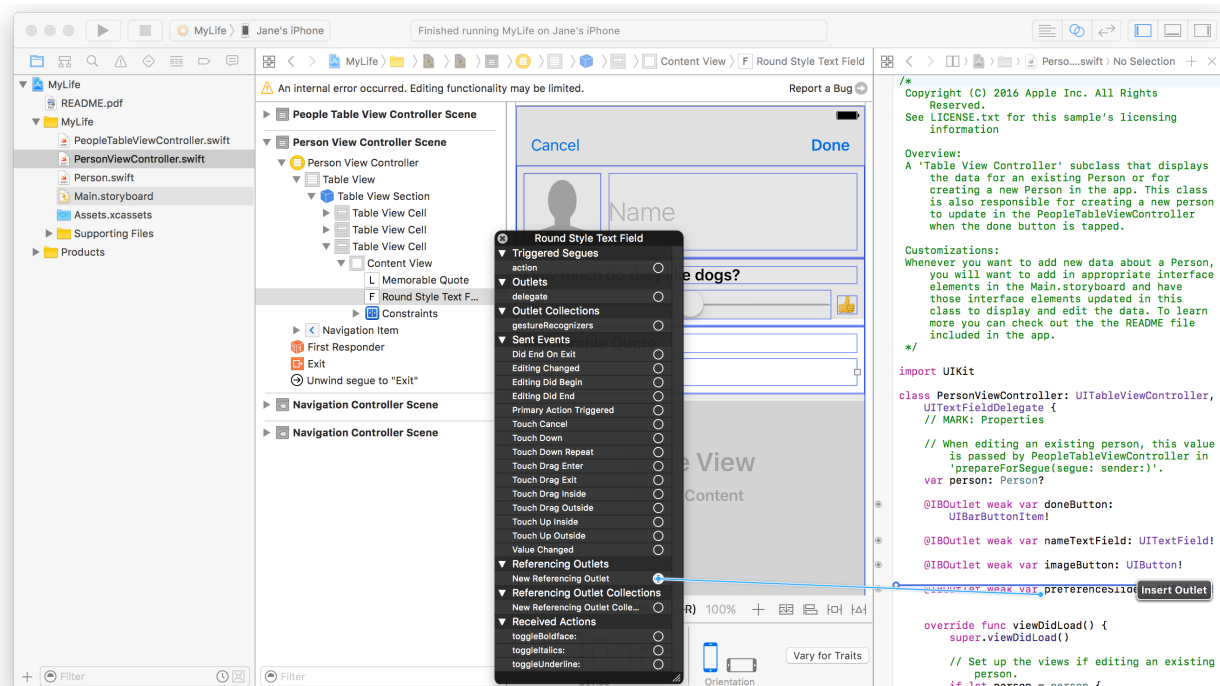
Run the app, and you'll see it updated with your new views! You can type in the text field, but you'll notice that when you modify the text and leave the screen, none of the changes are saved. You'll fix that in the next steps.



Connecting Interface Elements to Code

Now you'll need to make the `PersonViewController` aware of the text field so that in code, the view controller can read the text that a user types, and set the initial text.

1. Open the storyboard and the `PersonViewController` side by side. You can double-click a file to open it in a new window and place it next to another file, or use the assistant editor by option-clicking the file.
2. In the storyboard, control-click the text field and open the Connections Inspector by pressing option-command-6 (⌥⌘6), or by choosing View > Utilities > Show Connections Inspector.
3. Drag the circle next to New Referencing Outlet into your `PersonViewController` file, releasing when the cursor is under `preferencesSlider`.



4. You need to give the text field a name that you will use to refer to it in code. It's helpful to have a descriptive name that tells what the field is used for and what type of view it is. Type `memorableQuoteTextField`, and then click Connect.

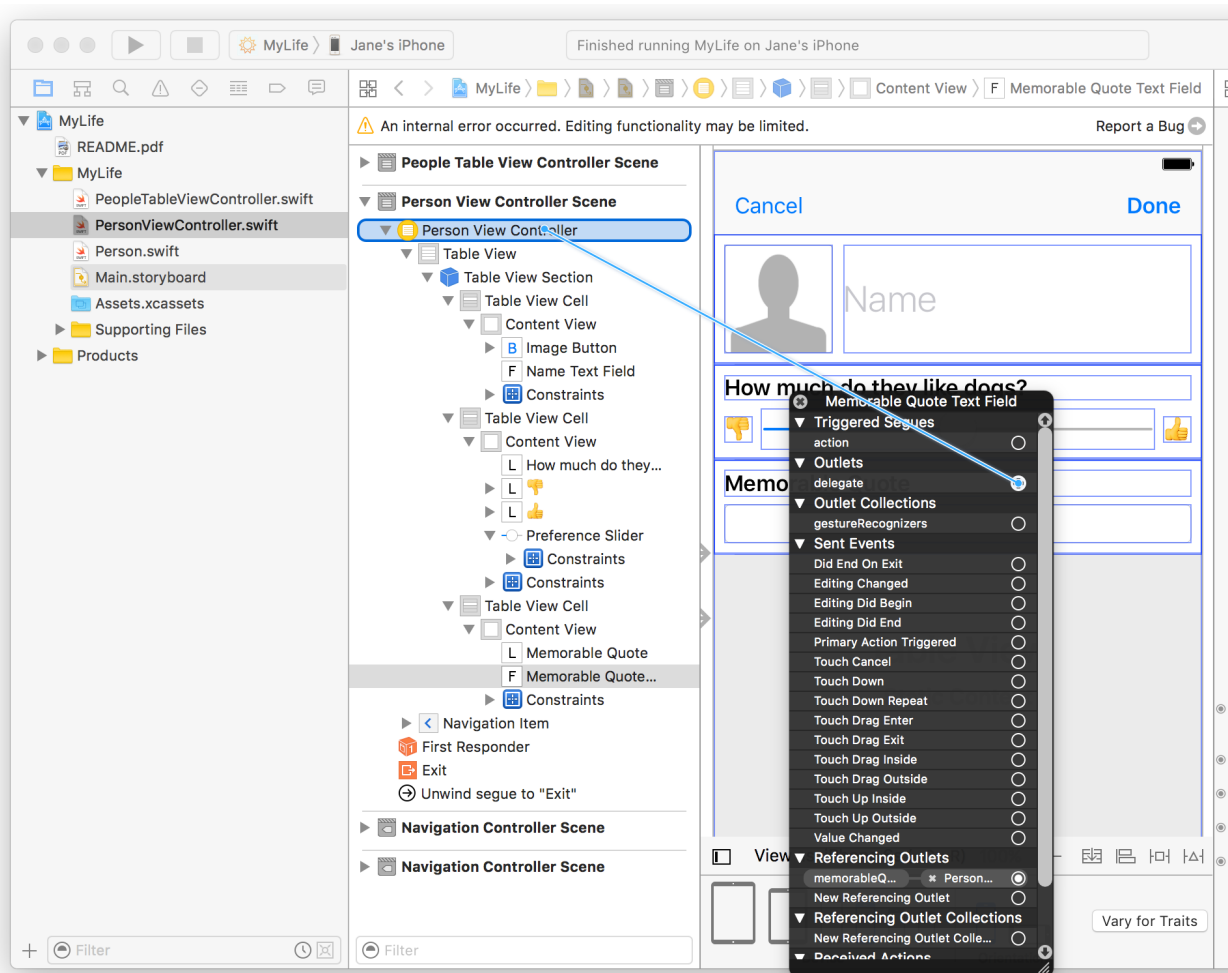
The following line is added:

```
@IBOutlet weak var memorableQuoteTextField: UITextField!
```

Note: You don't need to add an outlet for the label, because the text you're displaying is always the same, and there's no need for the view controller to read or modify the label. However, any interface element that you want to modify in code will need to be connected similarly to the text field.

When you type in the text field, you might want the keyboard to be dismissed when Return is tapped, similar to how it works in the name text field. Dismissing the keyboard requires setting the delegate for the text field (known as `UITextFieldDelegate`) to the `PersonViewController`. The delegate is the object that is told about changes to the text field, so when you do this, the view controller is informed when the Return key is tapped.

In the Connections Inspector, click and drag the circle next to the delegate to the yellow page icon in your storyboard that represents your `PersonViewController` class.



Now when the text field's return button is tapped, the `textFieldShouldReturn(textfield:)` function will be called in the `PersonViewController`. This function was already implemented for the `nameTextField`, and it calls `resignFirstResponder()` on the text field, dismissing the keyboard, so you don't have to do anything else.

Try running your app again to see if the text field is dismissed when you tap Return.

Updating the Person Class

To store a memorable quotation for a person, you need to add a new variable to the `Person` class to store the quotation. Text information is stored as a `String`, the same way you're storing the person's name in the `name` variable.

1. In `Person.swift`, you can follow the same format and add a new variable below the line where the `dogPreference` variable is declared, which you can name "`memorableQuote`". The entire line should look like this:

```
var memorableQuote: String?
```

The `?` after `String` is to show that the string is an `Optional`, which means it might have a text value, or it might have no value (represented as `nil`). This app allows a `Person` to be created without requiring them to have all of the descriptions filled out, which is why the variables are optional.

2. You also need to modify the initializer to be able to create a new `Person` and set the `memorableQuote` variable at the same time. You'll follow the same format for the other variables. By having the `"= nil"`, you're not required to pass in a `memorableQuote` parameter during initialization, and it defaults to `nil` in that case (representing no value). The new initializer should look like this:

```
// Person.swift
init(name: String? = nil, image: UIImage? = nil, dogPreference: Float? = nil,
     memorableQuote: String? = nil) {
    self.name = name
    self.image = image
    self.dogPreference = dogPreference
    self.memorableQuote = memorableQuote
}
```

Note: If you were to pass in other types of data, it would follow a similar format but store some variable other than a `String`. For example, an image would be `UIImage`, decimal numbers for a slider are `Float`, and yes or no input on a switch are `Bool`.

Updating in the View Controller

The last step is connecting the values stored in the `Person` class to the text field and enabling your edits in the app to update the corresponding person. This is all done in our `PersonViewController` class, because it's the job of the view controller to handle communication between the `UITextField` and the `Person` class where the data is stored.

Displaying Memorable Quotations when the View Loads

1. To update the text field with the stored `memorableQuote` when the `PersonViewController` is loaded, you need to open the `PersonViewController` file and go to the `viewDidLoad` function.

2. Notice in the `viewDidLoad` function that it is setting the `text` property on the `nameTextField` with the name of the `Person`, which is the same general format you need to follow for your new text field.
3. Right below where it sets the `nameTextField`'s text, you can add a line to do the same thing for `memorableQuoteTextField`, making sure you match the name of your text field to the name of the variable in your `Person` class:

```
memorableQuoteTextField.text = person.memorableQuote
```

Note: If you run the code now, nothing different will happen, because there is no initial value for `memorableQuote` on any `Person`. If you want to set the text in the `memorableQuote` for a `Person` when you first run your app, you can do this in `PeopleTableViewController.swift` in the `loadSampleData()` function. You need to follow the same format that you used in the `init` function above, putting the name of the variable (`memorableQuote`) and the text that it should be set to at the end of the `Person` declaration; for example:

```
// PeopleTableViewController.swift
let person1 = Person(name: "Byte",
                    image: UIImage(named: "Sample1"),
                    dogPreference: 9.0,
                    memorableQuote: "Here we go again!")
```

Updating the Person Class After Editing the Text

Now you need to handle updating any edits the user makes when they tap the Done button. The app currently is updating any edits to the name and dog preference in the `prepareForSegue` function of the `PersonViewController`. This segue is called if you tap on Cancel or Done, though the function checks to only save the data if "Done" was tapped.

1. Inside the if statement, you'll add code to read the text from the `memorableQuoteTextField`. You'll use the same text properties you used earlier, and store the information in a variable.

```
let memorableQuote = memorableQuoteTextField.text
```

2. This variable can then be passed into your `Person()` constructor, which calls the `init` function you implemented earlier.

```
person = Person(name: name, image: image, dogPreference: dogPreference,
memorableQuote: memorableQuote)
```

Your new `prepareForSegue` function should look like this:

```
// PersonViewController.swift
override func prepare(for segue: UIStoryboardSegue, sender: AnyObject?) {
    if sender == doneButton {
        let name = nameTextField.text
        let image = imageButton.image(for: [])
        let dogPreference = preferenceSlider.value
        let memorableQuote = memorableQuoteTextField.text

        person = Person(name: name, image: image, dogPreference:
dogPreference, memorableQuote: memorableQuote)
    }
}
```

Customization 3: Taking a Photo with the Camera

You can update the image for a person within the app by taking a photo with the camera or using your existing photo library.

Note: If you are running the app in the iPhone Simulator, you will not be able to use the camera, but you can still add photos with a simulated photo library.

Taking photos with the image picker

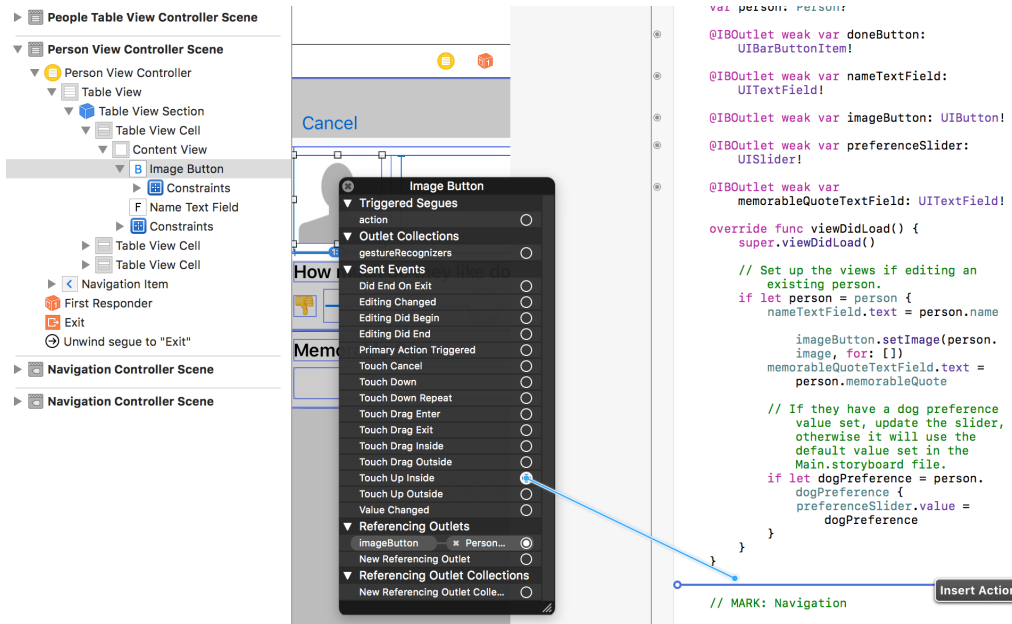
First, you need a way to tell the app that you want to take a photo, such as tapping on the image. The image you see on the `PersonViewController` page is actually a button that displays an image. You can modify this button so that when you tap it, it takes a new photo.

1. Open the storyboard (`Main.storyboard`) and go to the `PersonViewController`.

In this view, you'll see a silhouette of a person, which is the placeholder image for any new person you create. This image is displayed in a button (`UIButton`) that responds to different types of taps. You'll need to connect this button to the `PersonViewController` so the view controller can execute code when it's tapped.

2. You'll need to have the storyboard and `PersonViewController` open next to each other. You can double-click a file to open it in a new window and arrange the windows next to each other, or use the assistant editor by option-clicking a file.
3. With both files open, control-click the person image to bring up a list of touches this button can respond to.

- Click and hold on the circle next to “Touch Up Inside,” and drag it below the `viewDidLoad()` function in the `PersonViewController`. Release it at a point where Xcode says “Insert Action.”



“Touch up inside” is an event that occurs when you tap the button and then release your finger while still on the button (as opposed to dragging your finger elsewhere on the screen).

- Now you need to name the function that's called whenever the user taps the image. You can set the name to `imageTapped`.

If you run the app now, nothing will seem different, although if you tap the image, it will call the `imageTapped` function. To test it, you can add a print statement inside `imageTapped` to put a message on the console:

```
// PersonViewController.swift
@IBAction func imageTapped(_ sender: AnyObject) {
    print("Image Tapped")
}
```

If you run the app again and open the console (choose View > Debug Areas > Activate Console from the top menu, or press command-shift-C), tapping the image should make “Image Tapped” appear in the console at the bottom.

Now you need to add code so that when you tap the image, it takes a photo. Apple provides a class called `UIImagePickerController` to take photos with the camera or using the device’s image library.

You need to create and display an image picker controller and configure how it will work before presenting it to the user. The following code specifies using the camera and disables editing of the photos, but you can experiment with the other options.

```
// PersonViewController.swift
@IBAction func imageTapped(sender: AnyObject) {
    let imagePicker = UIImagePickerController()
    imagePicker.sourceType = .camera
    imagePicker.allowsEditing = false
    imagePicker.delegate = self
    present(imagePicker, animated: true, completion: nil)
}
```

Right now if you try to compile the app, there will be an error. Notice that the code also sets the delegate to “self.” This is done so that the `PersonViewController` class can receive delegate messages when a photo is taken or when the user taps Cancel, which is needed to use the selected photo. Once you write the code to use the delegate in the next steps, the error will go away.

Using the selected photo

At the top of the file, you need to modify the `PersonViewController` class definition to show that it conforms to the `UIImagePickerControllerDelegate` and the `UINavigationControllerDelegate`, both of which are required for getting the delegate messages. The class definition should look like the following:

```
// PersonViewController.swift
class PersonViewController: UITableViewController, UITextFieldDelegate,
UIImagePickerControllerDelegate, UINavigationControllerDelegate {
```

After a new photo is taken, the following delegate function is called:

```
func imagePickerController(imagePicker: UIImagePickerController,
didFinishPickingMediaWithInfo info: [String : AnyObject])
```

The `info` variable is a dictionary that contains the image the user picked. You need to get the image out of the `info` dictionary and update the image button. The button is displaying the previous image in the normal state, so you'll set a new image to the same state. Finally, you'll dismiss the image picker controller with an animated transition.

```
// PersonViewController.swift
func imagePickerController(_ imagePicker: UIImagePickerController,
didFinishPickingMediaWithInfo info: [String : AnyObject]) {
    let image = info[UIImagePickerControllerOriginalImage] as? UIImage
    imageButton.setImage(image, for: [])
    dismiss(animated: true, completion: nil)
}
```

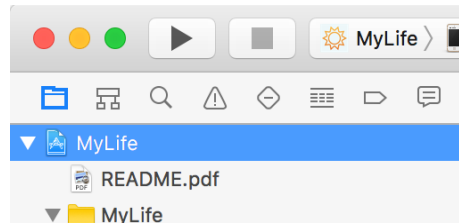
Now you need to handle cases where the user cancels taking a photo. In this situation, you need to dismiss the image picker and return to the previous page.

```
// PersonViewController.swift
func imagePickerControllerDidCancel(_ picker: UIImagePickerController) {
    dismiss(animated: true, completion: nil)
}
```

Customization 4: Changing the Name of your App

The app is called MyLife, but you can call it whatever you want. You can even name it after yourself so it appears, for example, as “Jane’s Life.”

1. Select the `MyLife` at the top of your project navigator window in Xcode.



2. Press Return to edit and type your new project name.
3. Now you should clean the project to ensure that any references to the old name are removed. On the top toolbar, click Product > Clean, or use the shortcut command-shift-K (⌘⇧K)

If you run the app and go to your home screen, you should see the new name!

Where to Go from Here

After you've modified and customized the My Life app, you'll know some of the basics of app development. But don't stop there! Search online to learn how to add to the app or start building a new one.

Here are a few places you might want to check out, depending on what you want to do next:

- **Build another app from scratch.** [Start Developing iOS Apps](#) walks you through building a FoodTracker app.
- **Learn to design beautiful app interfaces.** [iOS Human Interface Guidelines](#) teaches you how to make your app consistent with the user interface conventions for iOS.
- **Learn all about Swift programming.** [The Swift Programming Language](#) describes everything you need to know about Swift.
- **Learn about the technologies available to you.** [iOS Technology Overview](#) describes the frameworks and other technologies that are available to your app in iOS.
- **Debug and test your app.** [Debugging with Xcode](#) teaches you how to thoroughly debug and test your app in Xcode.
- **Ship your app.** [App Distribution Guide](#) walks you through the process of provisioning devices for testing, and submitting apps to the App Store.