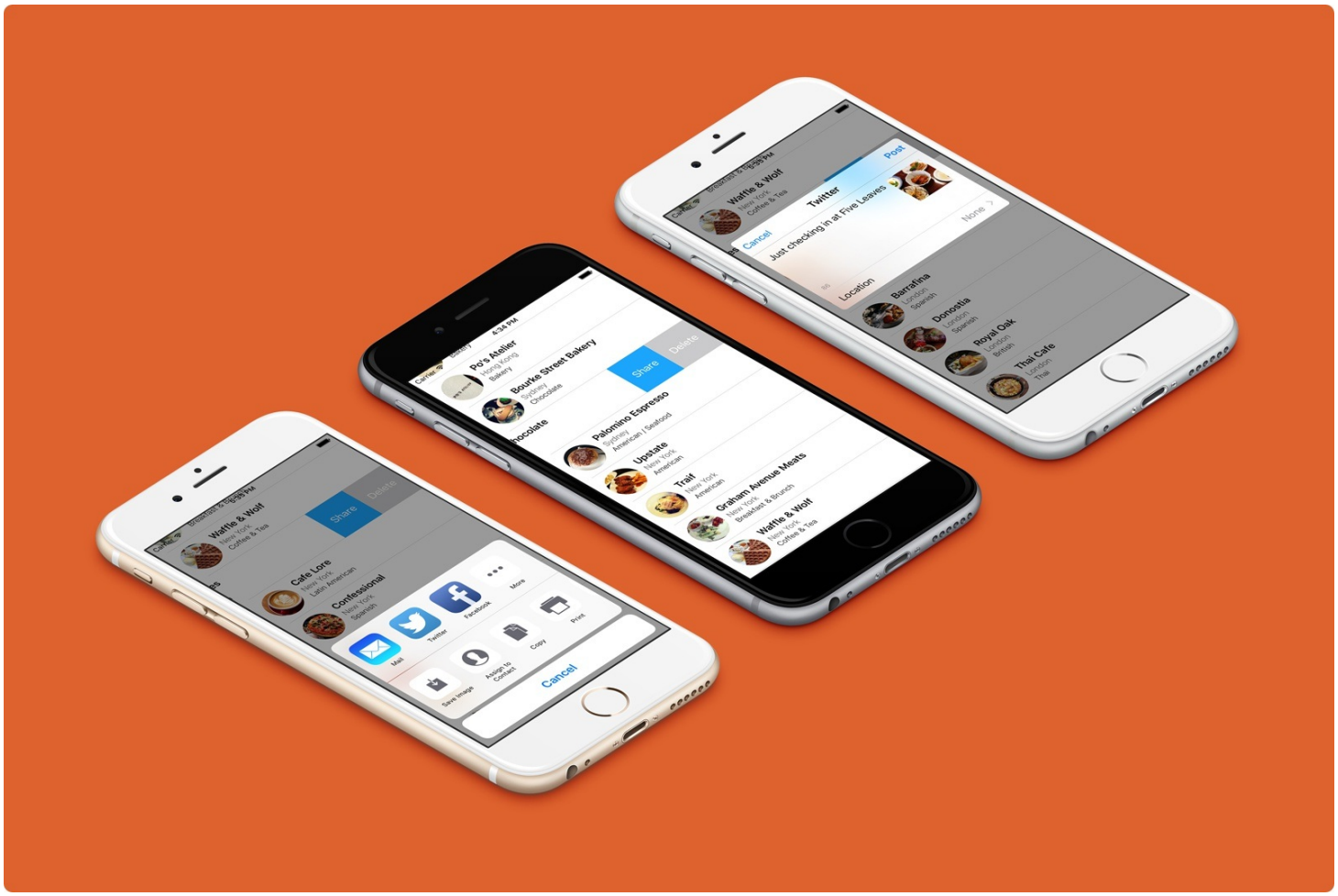
Chapter 11
Table Row Deletion, Custom Action Buttons, Social Sharing and MVC



> If you spend too much time thinking about a thing, you'll never get it done. Make at least one definite move daily toward your goal.
>
> – Bruce Lee

Now you know how to handle table row selection. But how about deletion? How can we delete a row from UITableView?

It's a common question when building a table-based app. Select, delete, insert and update are

the basic operations when dealing with data. We've discussed about selection. Let's talk about deletion in this chapter. In addition, we'll go through a couple of new features to the FoodPin app:

1. Adding a custom action button when a user swipes horizontally in a table row. This is usually known as *Swipe for More* action.
2. Adding a social sharing feature to the app, that enables users to share the restaurants on Twitter or Facebook.

There are a lot to learn in this chapter, but it's going to be fun and rewarding. Let's get started.

A Brief Introduction to Model View Controller

Before jumping into the coding part, I would like to give you an introduction of Model-View-Controller (MVC) model, which is one of the most quoted design patterns for user interface programming.

I try to keep this book as practical as possible and seldom talk about the programming theories. That said, you can't avoid from learning Model-View-Controller, especially your goal is to build great apps or become a competent programmer. MVC is not a concept that applies to iOS programming only. You may have heard of it if you've studied other programming languages, such as Java or Ruby. It is a powerful design pattern used in designing a software applications, whether it is a mobile app and a web app.

Understanding Model-View-Controller

> At the heart of MVC, and the idea that was the most influential to later frameworks, is what I call Separated Presentation. The idea behind Separated Presentation is to make a clear division between domain objects that model our perception of the real world, and presentation objects that are the GUI elements we see on the screen. Domain objects should be completely self contained and work without reference to the presentation, they should also be able to support multiple presentations, possibly simultaneously. This approach was also an important part of the Unix culture, and continues today allowing many applications to be manipulated through both a graphical and command-line interface.

> By Martin Fowler

No matter which programming language you learn, one important concept that you need to know is *Separation of Concerns (SoC)*. The concept is pretty simple. Here, the *Concerns* are different aspects of software functionality. This concept encourages developers to break a complicated feature or program into several areas of concern so that each area has its own responsibility. The delegate pattern, that we explained in the earlier chapters, is one of the examples of SoC.

The *model-view-controller (MVC)* concept is another example of SoC. The core idea behind MVC is to separate an user interface into three areas (or groups of objects) that each area is responsible for a particular functionality. As the name suggests, MVC breaks an user interface into three parts:

- **Model** – model is responsible for holding the data or any operations on the data. The model can be as simple as an array object that stores the table data. Add, update and delete are examples of the operations. In business world, these operations are usually known as business rules.
- **View** – view manages the visual display of information. For example, UITableView displays data in a list format.
- **Controller** – controller is the bridge between the model and the view. It translates the user interaction from the view (e.g. tap) into the appropriate action to be performed in the model. For example, a user taps a delete button in the view. Consequently, the controller triggers a delete operation in the model. Once finished, the model requests the view to refresh itself so as to reflect the update of the data model.

To help you better understand MVC, let's use the SimpleTable app (the one that we have built in chapter 8) as an example. The app displays a list of restaurants in the table view. If you turn the implementation into a visual illustration, here is how the table data is displayed:

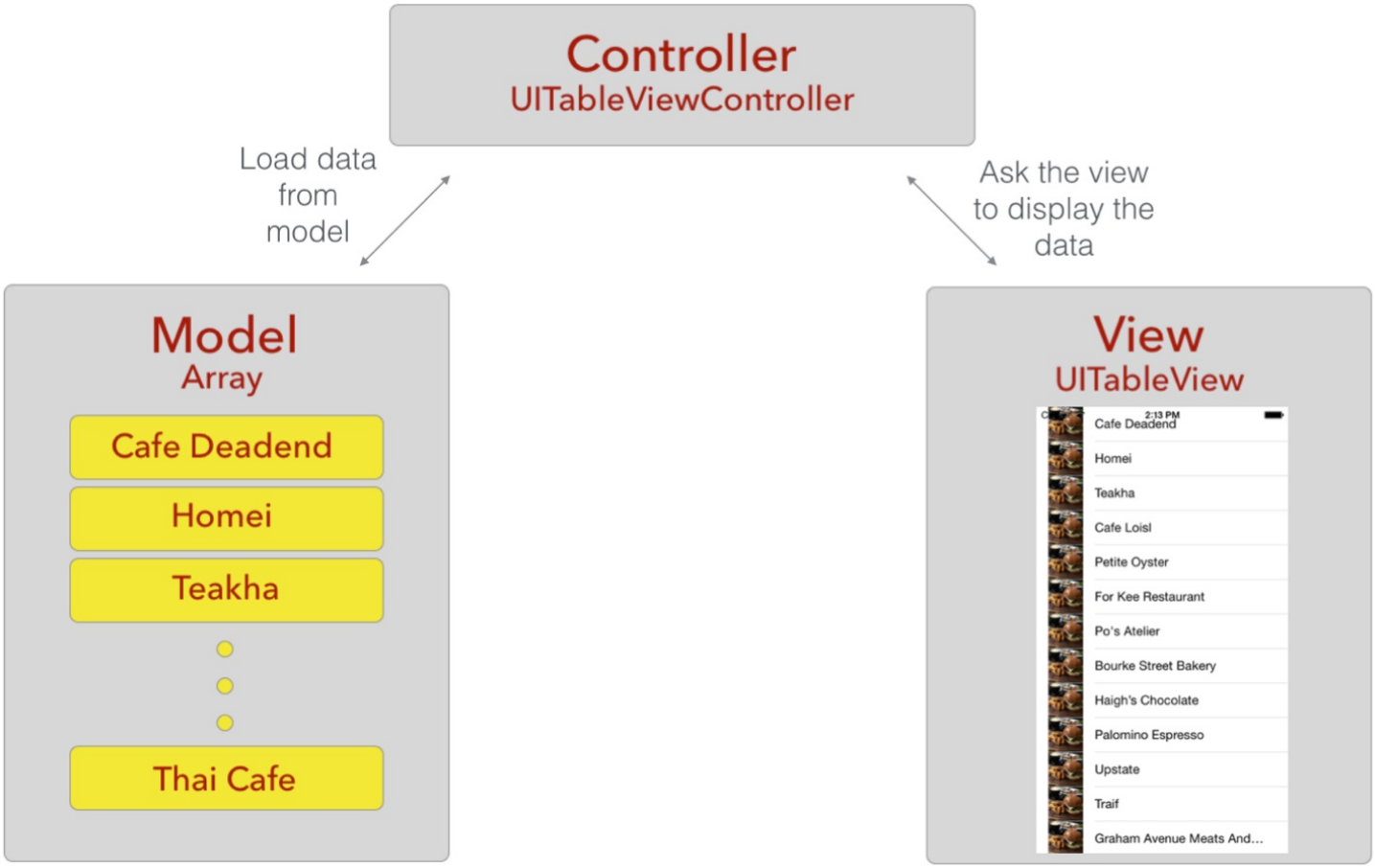


Figure 11-1. MVC Illustrated

The `restaurantNames` object, which is an array, is the *Model*. Each table row maps to an element of the `restaurantNames` array. The `UITableView` object is the actual *View* to be seen by a user. It's responsible for all the visuals (e.g. color of the table rows, style of the tableview, separator style, etc). The `UITableViewController` object, which is the *Controller*, acts as the bridge between the table view and the data model. It manages the table view and is responsible to load the data from the model.

Deleting a Row from UITableView

I hope you now have a better understanding of Model-View-Controller. Let's move onto the coding part and see how we can delete a row from a table view. We'll continue to develop the FoodPin app (if you haven't completed the previous exercise, you can start with by downloading the project from

https://www.dropbox.com/s/uf84e3bv3z9uatm/FoodPinTableSelection.zip?dl=0) and add the "delete" feature.

If you understand the MVC model, you probably have some ideas on the implementation of row deletion. There are three main tasks we have to do:

1. Enable the swipe-to-delete feature of the table view so that the user can select the Delete option
2. Delete the corresponding table data from the data model
3. Reload the table view to reflect the change of table data

Enable the Swipe-to-delete feature

In iOS app, users normally swipe horizontally across a table row to reveal the Delete button. Recalled that we have adopted the `UITableViewDataSource` protocol, there is a method called `tableView(_:commitEditingStyle:forRowAtIndexPath:)`. To enable the swipe-to-delete feature of a table view, all you need to do is implement the method. If the method exists, the table view will automatically display a "Delete" button when a user swipes across a row.

Simply add the following code to the `RestaurantTableViewController.swift` file:

```
override func tableView(tableView: UITableView, commitEditingStyle
editingStyle: UITableViewCellEditingStyle, forRowAtIndexPath indexPath:
NSIndexPath) {

}
```

Now let's have a quick test. Run the app on an iPhone simulator. Though the method is currently without any implementation, you should see the "Delete" button when swiping across a row.

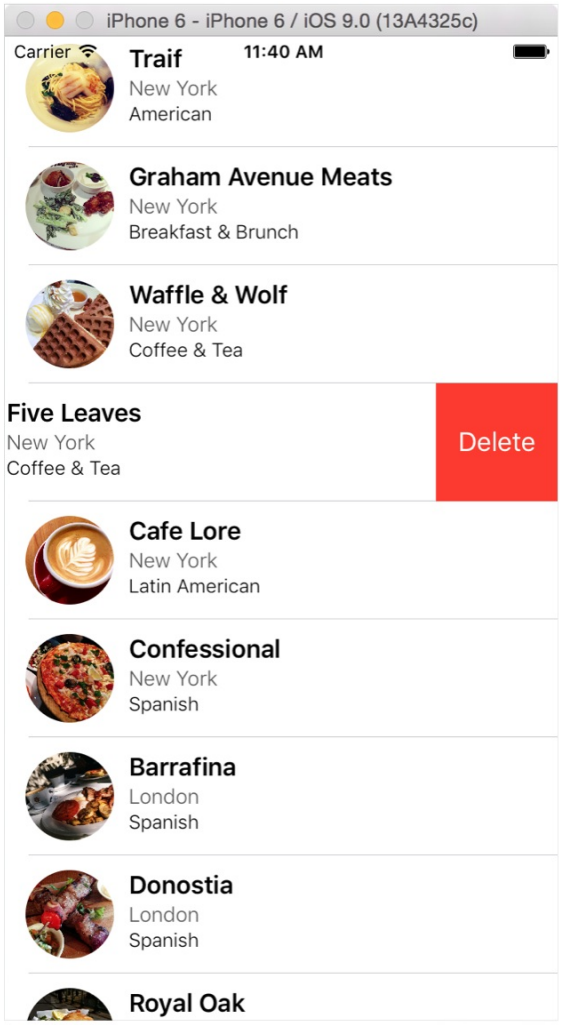


Figure 11-2. Swipe-to-delete feature

Delete Row Data from the Model

The next thing is to implement the method and write the code for removing the actual table data. From the method declaration, the `indexPath` parameter contains the row number of the cell, which is about to delete. You can make use of this information to remove the corresponding element from the data arrays.

In the FoodPin app, the `restaurantNames`, `restaurantLocations`, `restaurantTypes`, and `restaurantIsVisited` are the data model. Obviously, we have to remove the data of the selected restaurant from all of the arrays. To remove an item from an array, you can simply call the `removeAtIndex` method of the array object. So update the method with the code below:

```
override func tableView(tableView: UITableView, commitEditingStyle
editingStyle: UITableViewCellEditingStyle, forRowAtIndexPath indexPath:
NSIndexPath) {

    if editingStyle == .Delete {
        // Delete the row from the data source
        restaurantNames.removeAtIndex(indexPath.row)
        restaurantLocations.removeAtIndex(indexPath.row)
        restaurantTypes.removeAtIndex(indexPath.row)
        restaurantIsVisited.removeAtIndex(indexPath.row)
        restaurantImages.removeAtIndex(indexPath.row)
    }
}
```

The method supports two types of editing styles: *Insert* and *Delete*. Because we only remove the data when the user select the Delete button, we first check `editingStyle` before executing the code block.

Now run and test your app again. Oops! The app doesn't work as expected. When you tap the Delete button, the cell is not removed. You may think that the data was not removed properly. Let's do some debugging here. Insert the following lines of code before the end of the method to print out the content of the array:

```
print("Total item: \(restaurantNames.count)")
for name in restaurantNames {
    print(name)
}
```

In Swift, you use the `print` method to output a message to the console. Printing the content of a variable is a very basic way of debugging. The above code prints the total number of item in the `restaurantNames` array, and its content after the cell deletion. By default, the console is hidden in Xcode. Go up to the Xcode menu, select View > Debug Area > Activate Console to bring up the console.

Now compile and run the app again. Delete the first row (i.e. Cafe Deadend) from the table view. You'll find the output in the console under the debug area (see figure 11-3). At first, we have 21 restaurant items in the array. After deleting a row, the number reduces to 20. And from the output, "Cafe Deadend" was deleted completely.

As you can see, the app actually deletes the item from the arrays. It looks like the view doesn't

show the update. Yes, that's true. We only remove the data from the model but haven't notified the table view to update its content.

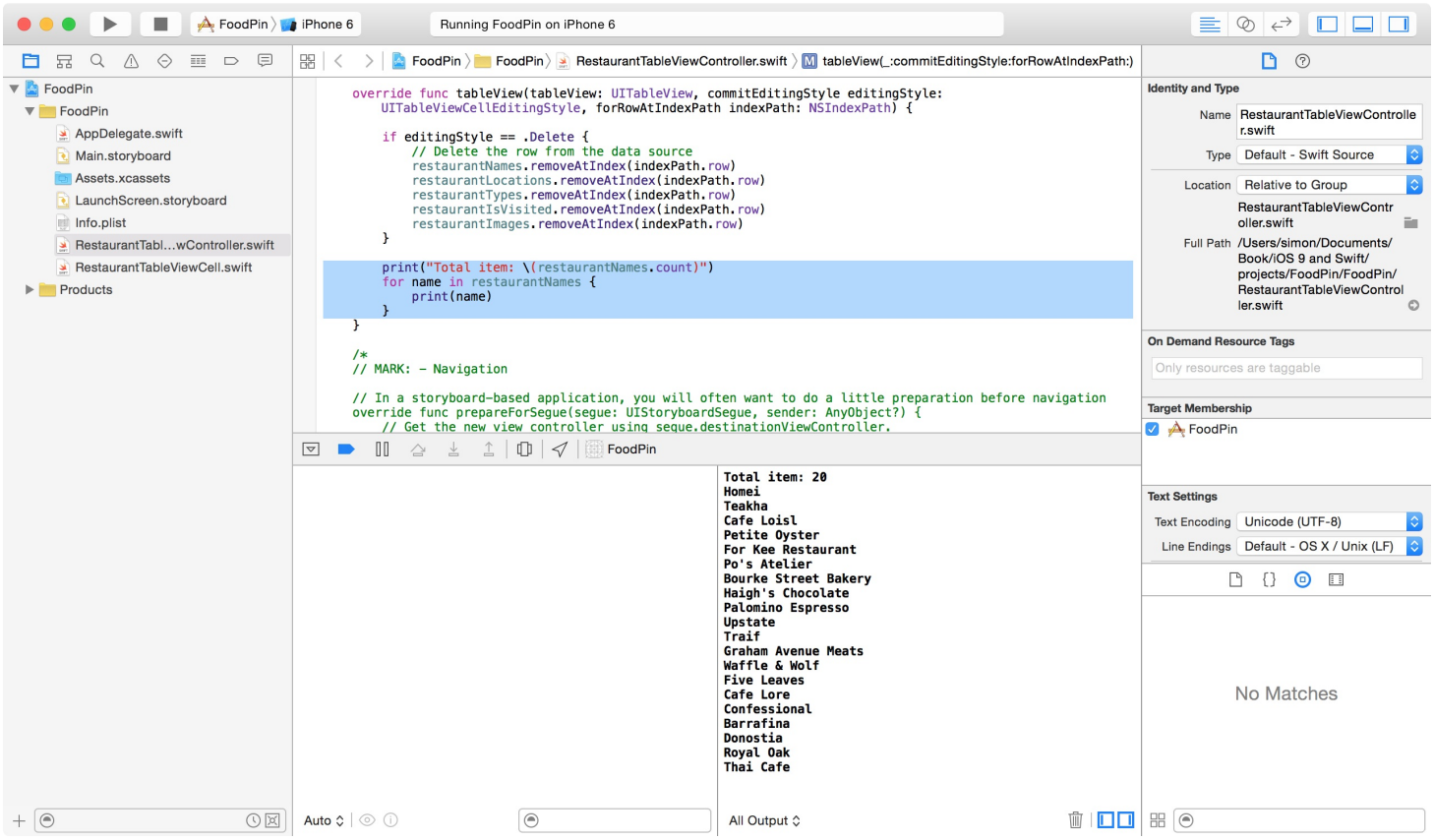


Figure 11-3. Debug area and console

Reloading UITableView

One way to ask the view to reload its content is by calling the `reloadData` method. So insert a line of code in the method to reload the data in the table view:

```
override func tableView(tableView: UITableView, commitEditingStyle
editingStyle: UITableViewCellEditingStyle, forRowAtIndexPath indexPath:
NSIndexPath) {

    if editingStyle == .Delete {
        // Delete the row from the data source
        restaurantNames.removeAtIndex(indexPath.row)
        restaurantLocations.removeAtIndex(indexPath.row)
        restaurantTypes.removeAtIndex(indexPath.row)
        restaurantIsVisited.removeAtIndex(indexPath.row)
```

```
        restaurantImages.removeAtIndex(indexPath.row)
    }

    tableView.reloadData()

    print("Total item: \(restaurantNames.count)")
    for name in restaurantNames {
        print(name)
    }
}
```

When the `reloadData` method is called, the table view clears its content and reload the current data from the restaurant arrays to display. Now compile and test the app again. When you delete a restaurant, the table row should be removed.

Delete a Row from UITableView

The app works, but there is a better way to refresh the table view. Consider that we only need to delete a single row, why don't we just remove that particular row from the table view? You're allowed to use a method called `deleteRowsAtIndexPaths` to delete a specific row (or multiple rows) from the table view. Replace the `reloadData` method with the following line of code:

```
tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation: .Fade)
```

The `deleteRowsAtIndexPaths` method takes in two parameters: *an array of index path* and *the row animation*. Here we just pass the method with the current index path and specify to use the fade animation. The row animation indicates how the deletion is to be animated. `.Fade` animation is commonly used. Optionally, you can change it to other animations such as `.Right`, `.Left` and `.Top`. Compile and run the app again. When you confirm to delete a record, the row fades out of the table view.

Swipe for More Actions Using UITableViewRowAction

When you swipe across a table cell in the stock Mail app, you'll see a Trash button, and a More button. The More button will bring up an action sheet providing a list of options such as Reply, Flag, etc.

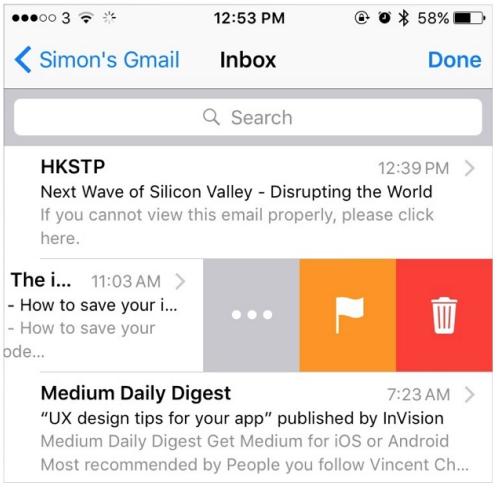


Figure 11-4. Swipe for more in the Mail app

This *swipe-for-more* feature was first introduced in the stock iPhone apps in iOS 7. At that time, Apple didn't make this feature available to developers. Starting from iOS 8, the iOS SDK comes with a new class named `UITableViewRowAction`. You use this class to create custom actions for table rows of any table view. To add custom actions to the table view's row, all you need to do is implement the `tableView(_:editActionsForRowAtIndexPath:)` method, and set the custom actions as return objects.

Let's see how it works. Insert the following method in the `RestaurantTableViewController.swift` file:

```
override func tableView(tableView: UITableView, editActionsForRowAtIndexPath
indexPath: NSIndexPath) -> [UITableViewRowAction]? {

    // Social Sharing Button
    let shareAction = UITableViewRowAction(style:
UITableViewRowActionStyle.Default, title: "Share", handler: { (action,
indexPath) -> Void in

        let defaultText = "Just checking in at " +
self.restaurantNames[indexPath.row]
        let activityController = UIActivityViewController(activityItems:
[defaultText], applicationActivities: nil)
        self.presentViewController(activityController, animated: true,
completion: nil)
    })

    // Delete button
```

```
    let deleteAction = UITableViewRowAction(style:
UITableViewRowActionStyle.Default, title: "Delete",handler: { (action,
indexPath) -> Void in

        // Delete the row from the data source
        self.restaurantNames.removeAtIndex(indexPath.row)
        self.restaurantLocations.removeAtIndex(indexPath.row)
        self.restaurantTypes.removeAtIndex(indexPath.row)
        self.restaurantIsVisited.removeAtIndex(indexPath.row)
        self.restaurantImages.removeAtIndex(indexPath.row)

        self.tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation:
.Fade)
    })

    return [deleteAction, shareAction]
}
```

The usage of `UITableViewRowAction` is very similar to that of `UIAlertAction`. You specify the title, the style and the block of code to execute when a user taps the button. In this example, we name the custom action as "Share". When a user taps the button, it brings up an activity controller for social sharing.

The `UIActivityViewController` class is a standard view controller that provides several standard services, such as copying items to the clipboard, sharing content to social media sites, sending items via Messages, etc. The class is very simple to use. Let's say you have a message for sharing. All you need to do is create an instance of `UIActivityViewController` with the message object, and then present the controller on screen. That is what we have done in the above code snippet.

You may notice that we added a delete action button. When you implement the `tableView(_:editActionsForRowAtIndexPath:)` method, the table view will no longer generate the Delete button for you. This is why we need to create our own Delete button.

The last line of the code is probably the most important part. It returns an array of `UITableViewRowAction` objects (i.e. `deleteAction` and `shareAction`) telling the table view to create the buttons when someone swipes across the cell.

Compile and run the app. Swipe across a table row and it'll show you both Share and Delete buttons. Tapping the Share button will bring up a share menu like the one shown in figure 11-5.

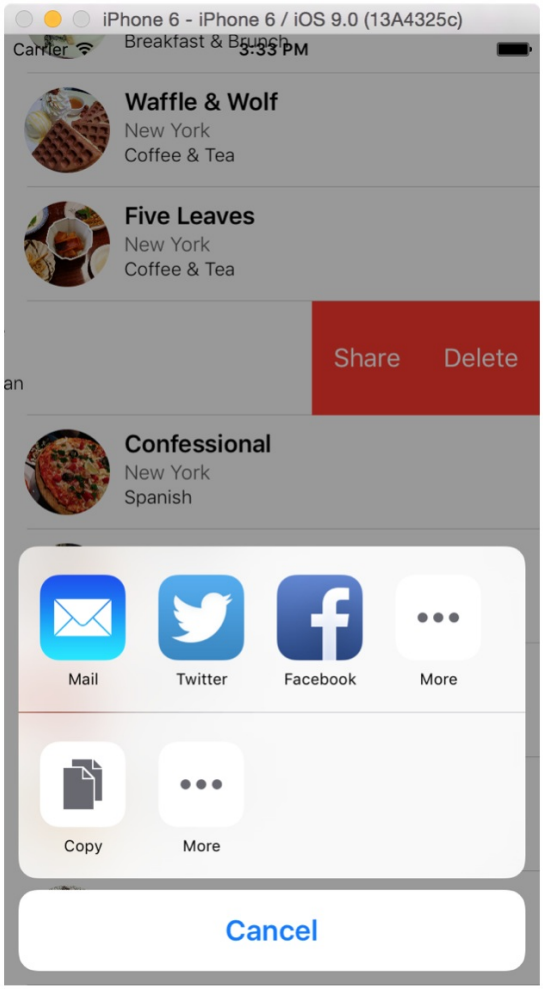


Figure 11-5. Swipe to Share button

The Twitter and Facebook buttons may not appear in your simulator. In this case, press shift-command-H to go back to the Home screen. Select Settings > Twitter / Facebook, and sign in with your account. Then re-launch the app. You should be able to share the content on social media sites.

The `UIActivityViewController` class does not limit you from sharing content in text format. If you pass it a `UIImage` object during initialization, your app will allow users to post images to Twitter or Facebook. Modify the `tableView(_:editActionsForRowAtIndexPath:)` method to the following:

```
override func tableView(tableView: UITableView, editActionsForRowAtIndexPath
indexPath: NSIndexPath) -> [UITableViewRowAction]? {

    // Social Sharing Button
    let shareAction = UITableViewRowAction(style:
UITableViewRowActionStyle.Default, title: "Share", handler: { (action,
indexPath) -> Void in

        let defaultText = "Just checking in at " +
self.restaurantNames[indexPath.row]
        if let imageToShare = UIImage(named:
self.restaurantImages[indexPath.row]) {
            let activityController = UIActivityViewController(activityItems:
[defaultText, imageToShare], applicationActivities: nil)
            self.presentViewController(activityController, animated: true,
completion: nil)
        }
    })

    // Delete button
    let deleteAction = UITableViewRowAction(style:
UITableViewRowActionStyle.Default, title: "Delete",handler: { (action,
indexPath) -> Void in

        // Delete the row from the data source
        self.restaurantNames.removeAtIndex(indexPath.row)
        self.restaurantLocations.removeAtIndex(indexPath.row)
        self.restaurantTypes.removeAtIndex(indexPath.row)
        self.restaurantIsVisited.removeAtIndex(indexPath.row)
        self.restaurantImages.removeAtIndex(indexPath.row)

        self.tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation:
.Fade)
    })

    return [deleteAction, shareAction]
}
```

We just add a couple of lines in the above code to create an `imageToShare` object for sharing. We first load the image by using the `UIImage` class, and then pass it to `UIActivityViewController` during initialization. `UIActivityViewController` will automatically embed the image when the user shares the restaurant to social media sites.

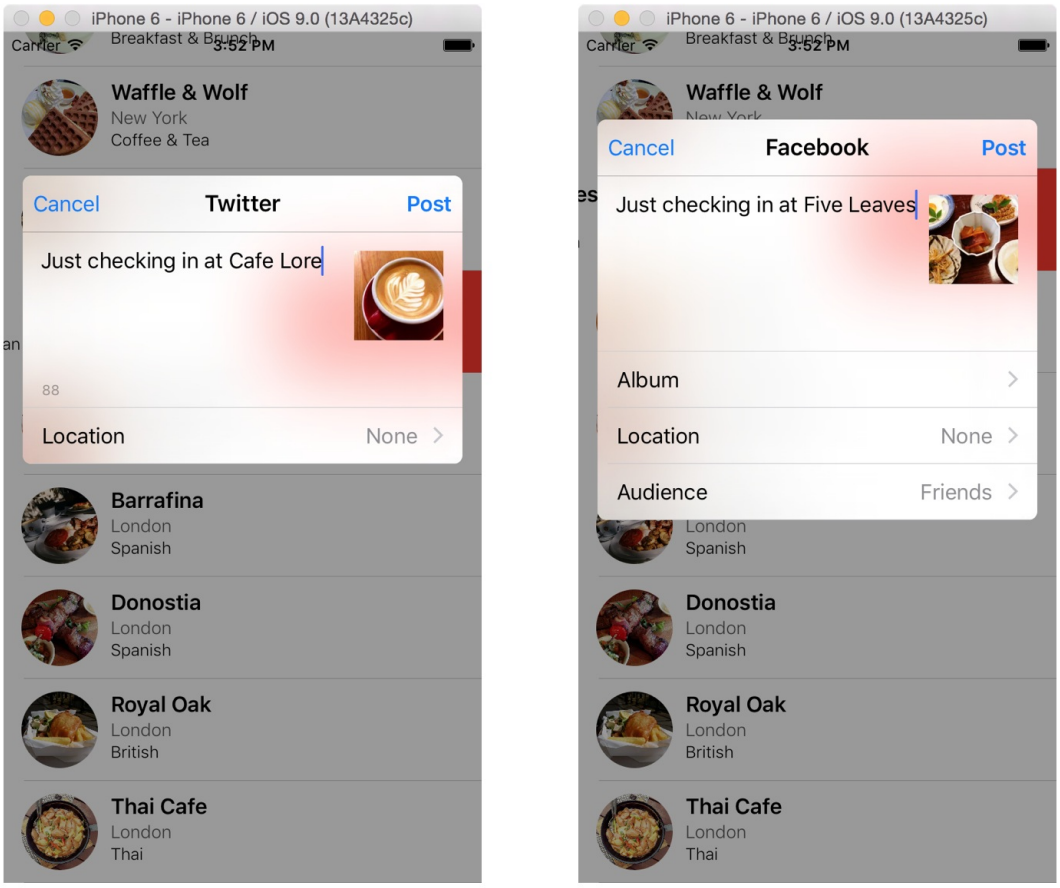


Figure 11-6. Posting content to Twitter and Facebook

if let for Optionals

When loading an image, it is possible that the image is failed to load. This is why the UIImage class returns an optional during initialization. In Swift, we use "if let" to verify if an optional contains a value or not. For details, you can refer to the Optional section in the appendix.

Customize UITableViewRowAction

By default, the action buttons are in red. The `UITableViewRowAction` class provides an option for developers to customize its background color through the `backgroundColor` property:

```
shareAction.backgroundColor = UIColor(red: 28.0/255.0, green: 165.0/255.0,
blue: 253.0/255.0, alpha: 1.0)
deleteAction.backgroundColor = UIColor(red: 202.0/255.0, green: 202.0/255.0,
```

```
blue: 203.0/255.0, alpha: 1.0)
```

The UIKit framework provides a `UIColor` class to represent color. Many methods in UIKit require you to provide color using a `UIColor` object. The class comes with a number of standard colors such as `UIColor.blueColor()` and `UIColor.redColor()`. If you want to use your own color, you can create your own `UIColor` object using RGB component values. The component values should be within 0 and 1. If you're a web designer or have some experience with graphics design, you know that the RGB values are usually on a scale of 0 to 255. To conform with the requirement of `UIColor`, you have to divide each of the component values by 255 when creating the `UIColor` object. You can paste the code above before the following line of code in the `tableView(_:editActionsForRowAtIndexPath:)` method:

```
return [deleteAction, shareAction]
```

Here we go. Test the app again and see if you like the new color. Otherwise, modify the color code and change to your preferred color.

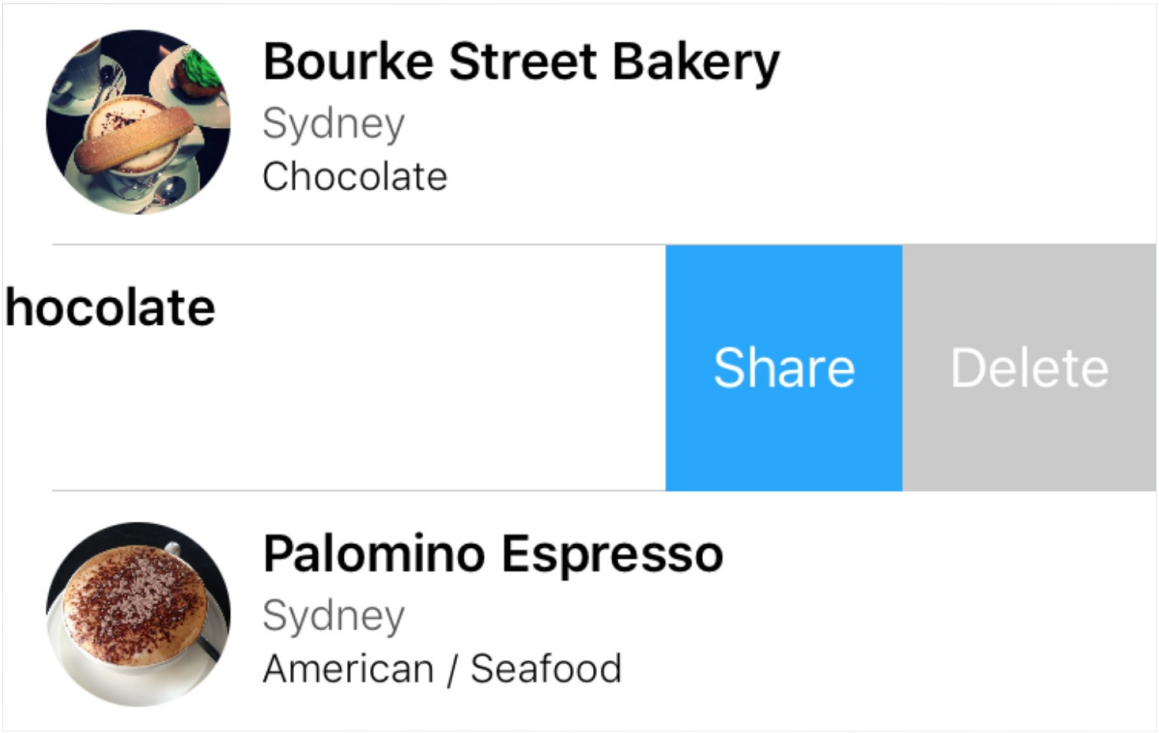


Figure 11-7. Posting content to Twitter and Facebook

Quick note: Like me, if you are not a designer, you may need some color inspiration. You can check out Adobe Color CC (color.adobe.com) and Flat UI Color Picker (flatuicolorpicker.com). You'll find a lot of color combinations for designing your app.

Summary

In this chapter, I gave you a brief overview of MVC, showed you how to manage deletion in table view, and taught you how to create a swipe-for-share button. You also learned how to use `UIActivityViewController` for implementing social sharing feature. The FoodPin app is getting better and better. You should be proud of your achievement so far.

The MVC concept is important. If you're a programming newbie, it may take you some time to fully understand the material. Feeling confused? Just take a break and grab yourself a coffee. After your break, go over the chapter again. Probably you'll find it easier to digest.

For reference, you can download the complete Xcode project from https://www.dropbox.com/s/xdy8uu40any3hza/FoodPinDeleteRow.zip?dl=0.

In the next chapter, we'll take a look at something new and create a navigation controller.