SocialApp #5

Although this project won't have all the bells and whistles you might expect from a full Twitter client, you can still choose from multiple accounts, see a full Twitter feed, and compose and post tweets. This application will look like, see Figure 1.

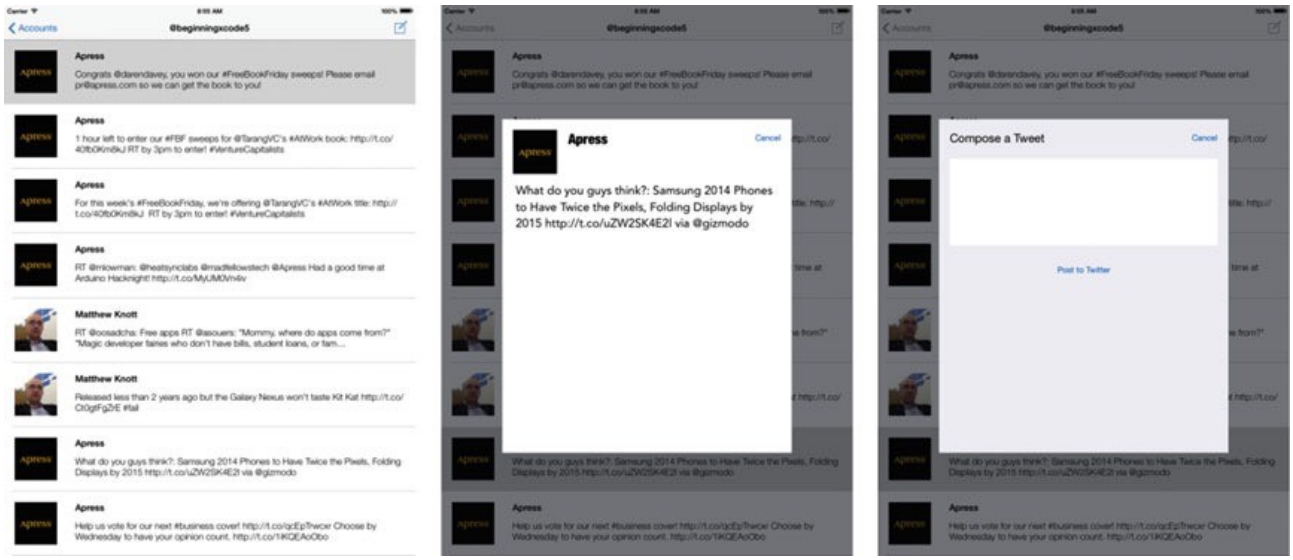

Figure 1. *Some of the key screens in SocialApp, your functional Twitter client*

Figure 2 shows an overview of the storyboard you will develop for the SocialApp Twitter application.

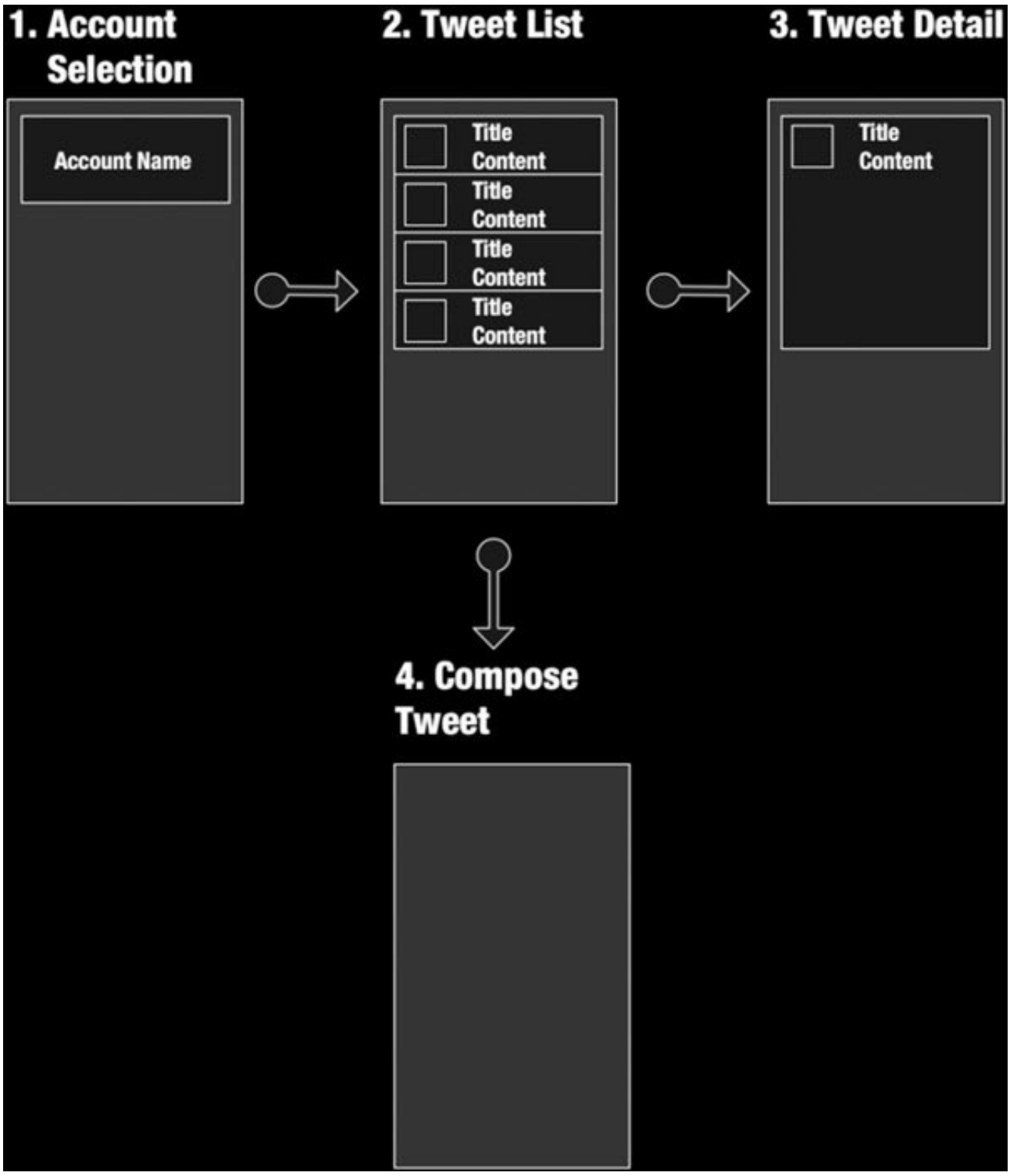

Figure 2. *The basic composition of Social App, the example Twitter client*

When dealing with storyboards in Xcode, Apple refers to the views as *scenes*. As you can see in Figure 2, this app has four scenes, and one scene leads to another. Before you begin developing this app, let's look at each scene in more detail:

1. *Account Selection:* Today people often manage several Twitter accounts, so the first scene is a *grouped style* table view controller that lists each account available on the device.

2. *Tweet List:* Once the user has selected a preferred account, you want to show the 20 most recent tweets on that user's timeline. These are displayed in a *plain style* table view with a custom table cell.

3. *Tweet Detail:* The user can see more details about the tweet and its author in the Tweet Detail scene. This is based on a *standard* view controller and lists the user's name, avatar, and the full tweet content in a text view.

4. *Compose Tweet:* Accessed from the compose icon in the Tweet List, this *standard* view controller uses a text view to compose a tweet and then posts it to Twitter on behalf of the user.

If you want to reference the scenes in this storyboard to the actual application screenshots shown in Figure 1, the first screenshot is scene 2, Tweet List; the second is 3, Tweet Detail; and the third is 4, Compose Tweet.

Now that you know a little more about storyboards, their origin, and how they're used by developers every day, it's time to begin putting this application together. Now we'll focuse on laying out the scenes in the storyboard and putting the connecting segues in place, as well as embedding navigation controllers and creating the custom classes behind the view controllers, so let's get started.

You need to create the project and then lay out the views for this application:

1. Open Xcode and create a new project by clicking Create a New Xcode Project from the Welcome screen or going to File ➤ New ➤ Project (+Shift+N). Select the Single View Application template and click Next.

2. Name your project SocialApp and ensure that the targeted device is set to iPad, not iPhone or Universal. Configure the other settings as you've done in previous applications and click Next.

3. You don't need to create a Git repository this time, so leave that option unchecked and make sure your project will be saved in the right place. Then click Create.

4. As should be familiar by now, you're ready to begin your application in earnest. Open Main.Storyboard by selecting it from the Project Navigator.

5. If the view is an iPhone view, change it to iPad by clicking the View as: button at the bottom of the design area and select "iPad Pro 9.7".

At this point in the process, you may need to refer back to the initial layout storyboard created for Figure 2. The first scene in the application is a Table view controller that lists the available

accounts. You could add a table view to the default view controller that was added to the storyboard, but it's easier to add a completely separate Table view controller:

1. Drag a Table view controller from the Object Library onto the design area and drop it next to the existing view controller, as shown in Figure 4. You may need to move it around a little to get a neat and tidy design area.

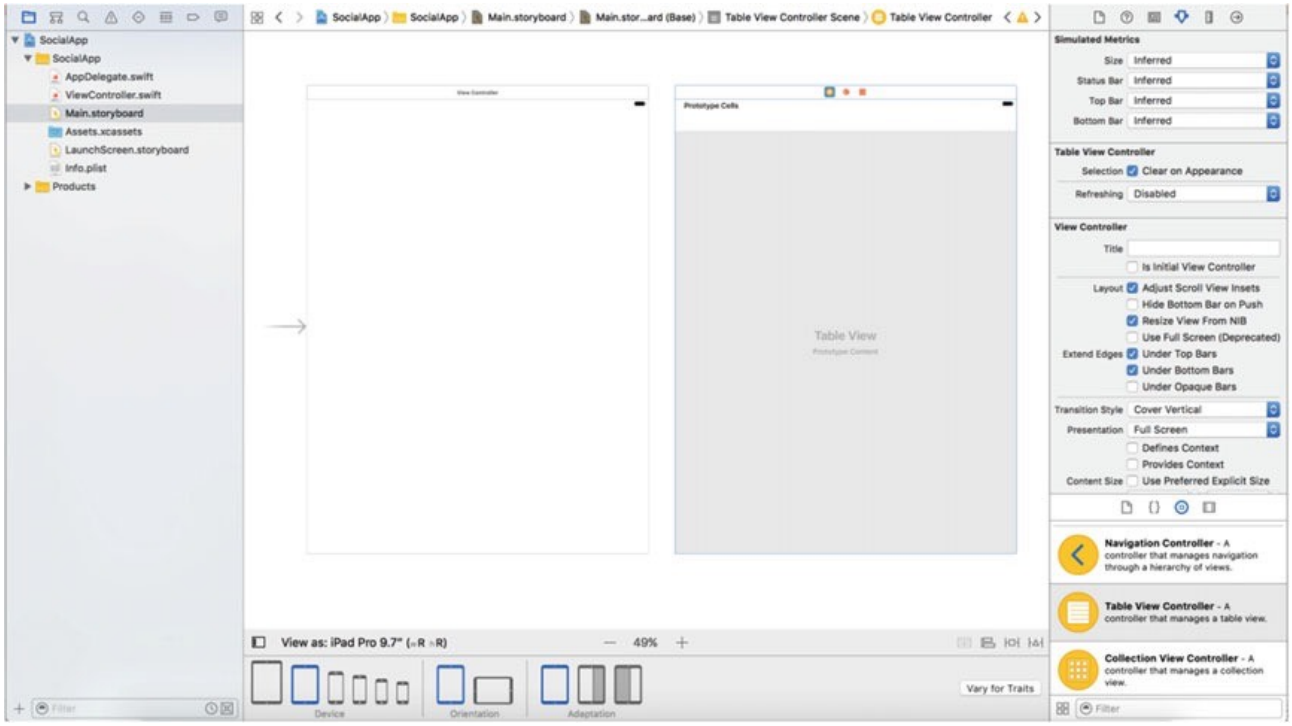

Figure 4. *The Table view controller scene, next to the initial scene you started with*

2. Before you add any more scenes, let's run the application in the simulator. Click the Run button on the Toolbar, or press +R. Notice that the default view controller is loaded instead of the Table view controller, and there is no obvious way of accessing the Table view controller within the application.

3. Quit the simulator and return to Xcode.

4. The initial view controller is the starting point; you can tell this because there is an arrow pointing to the left side of it, known as the storyboard entry point. Drag and drop the starting arrow onto the Table view controller. When the arrow is over the Table view controller, the scene becomes highlighted in blue, as shown in Figure 5. The starting arrow now points to the Table view controller, just as it once pointed to the default view controller.

Figure 5. *The starting arrow being hovered over the Table view controller*

If you run the application again, you should be greeted with an empty table view. It was that easy to change the starting point for this application! Before storyboards, you had to modify the application delegate to tell it which view controller to start with; now you can just drag and drop a visual aid.

An alternative way to set the initial scene, when you don't want to drag and drop the starting arrow, is to:

1. Select the Table view controller in the design area, either by clicking the scene while zoomed out or by selecting Table View Controller from the Document Outline.

2. Open the Attributes Inspector and look down to the View Controller section. Notice that Is Initial View Controller is selected. Unselect it, and the starting arrow disappears! You'd better bring it back; otherwise the application will run with a black screen.

3. It would be a good idea to set a title while you're here. To do so, click in the Title box and set the title to Accounts.

4. With the initial view controller set, you must now delete the default view controller. Select the blank view by clicking its scene while zoomed out or by selecting View Controller from the Document Outline, as shown in Figure 6.

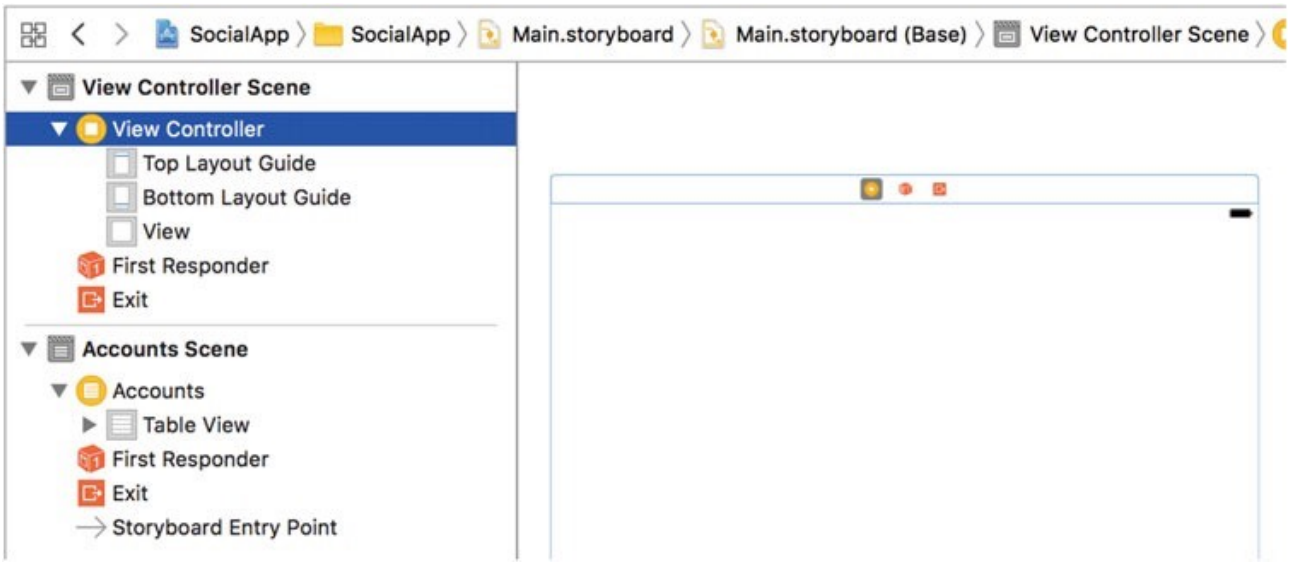

Figure 6. *Selecting the initial view controller from the Document Outline*

5. Delete the view controller by pressing the Backspace key or by choosing Edit ➤ Delete.

6. You also need to remove the file that Xcode added for this view controller. Using the Project Navigator, select ViewController.swift and, again, press the Backspace key or choose Edit ➤ Delete.

7. You're presented with the dialog shown in Figure 7, giving you options for file deletion. The Remove References button removes the files from the project but leaves them in place in the project folder on your Mac. In this case you want to delete the file altogether, so select the Move to Trash option.

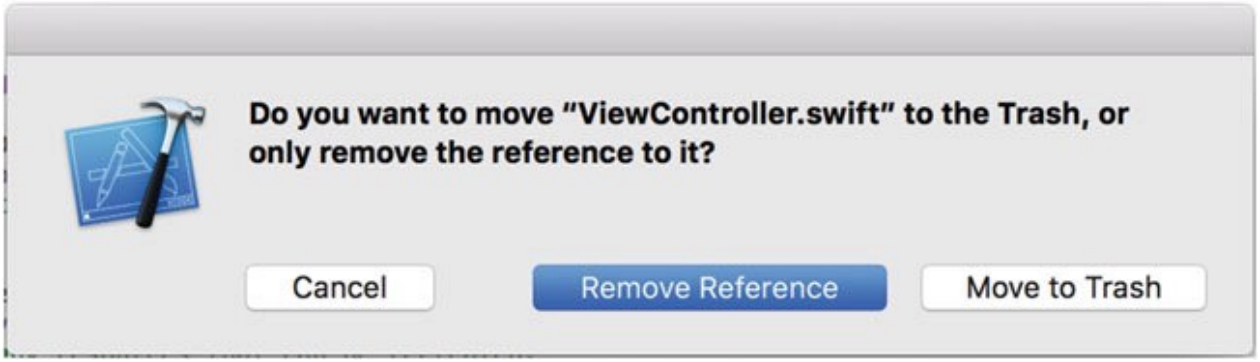

Figure 7. *The dialog presented by Xcode when removing files via the Project Navigator*

You've removed all the unnecessary files and views from the project. You're now going to step away from the storyboard for a moment to create four custom view controllers for the views by subclassing either UITableViewController or UIViewController.

Creating View Controllers

You can add as many view controllers to the storyboard as you like. But if you don't tie them to a view controller class file, the application will be extremely limited, because you'll have no way of interacting with the view controllers or controlling them using code.

SocialApp currently has a view, and you know you're going to add several more. Before you do, let's create all the view controllers so that when you add the views, you can tie them directly to a controller. All your view controllers subclass either UITableViewController or UIViewController to create an individual class file for each view. This time you need to create four view controllers of different types.

Subclassing UIViewController

UIViewController is the class name given to the standard view controller that has been used so far in all our applications. When you declare a class that subclasses UIViewController, you type, for example,

class MyCustomViewController: UIViewController

This says that the view controller called MyCustomViewController *subclasses* UIViewController. *Subclassing* means taking on all the attributes of another class, but with the ability to add your own methods and properties and override others. Two of your views subclass UIViewController. The two view controllers will be called TweetViewController and ComposeViewController. First, let's create TweetViewController, and then see if you can repeat the process for ComposeViewController:

1. Right-click the SocialApp group in the Project Navigator and select New File, as shown in Figure 8; or select the SocialApp group and press +N.

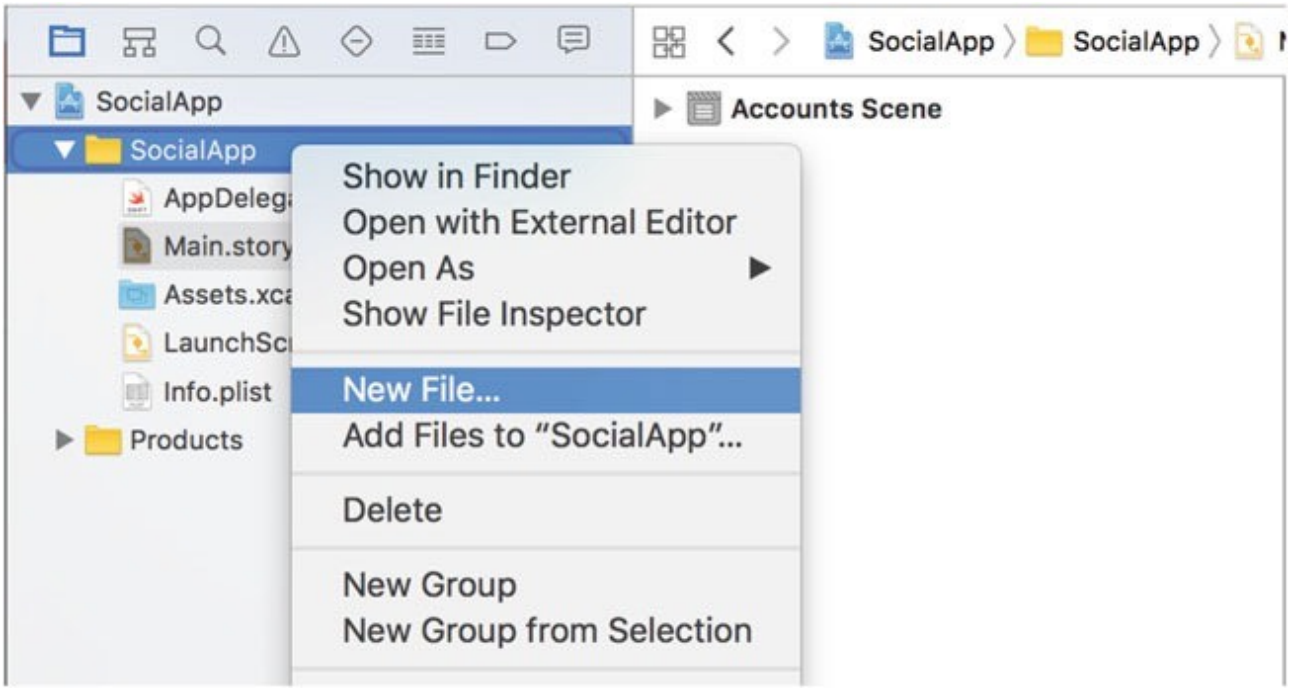

Figure 8. *Adding a file to the SocialApp group in the Project Navigator*

2. You're presented with the file template selection screen. Ensure that iOS is selected from the options at the top of the list, then under the Source heading, select Cocoa Touch Class, as shown in Figure 9, and click Next.

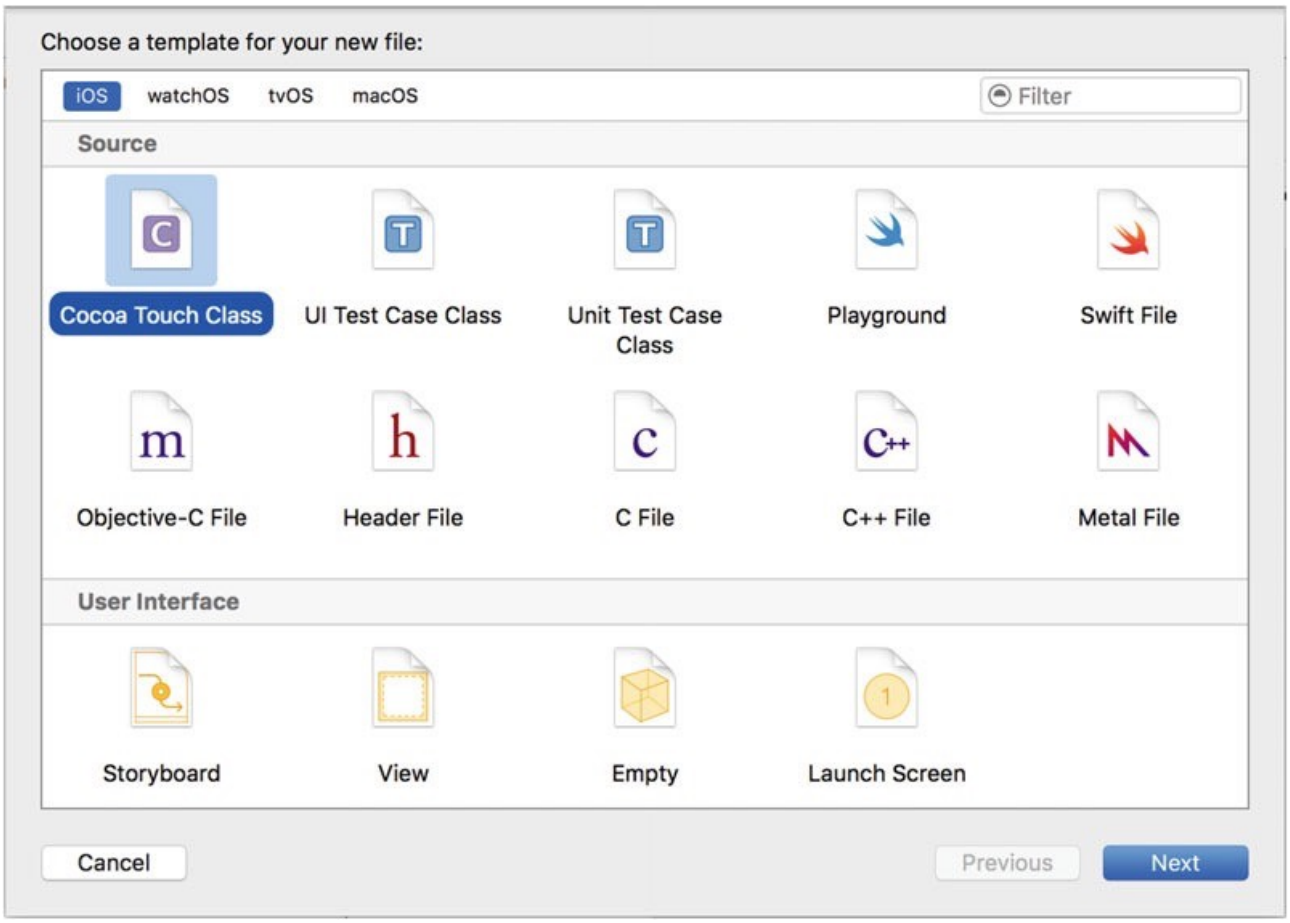

Figure 9. *Selecting the Cocoa Touch Class file template*

3. Set the Subclass Of field to UIViewController. This defaults the Class field to ViewController, making it easy for you to change it to TweetViewController. Be sure the Also Create XIB File check box is unchecked, as shown in Figure 10, and click Next.

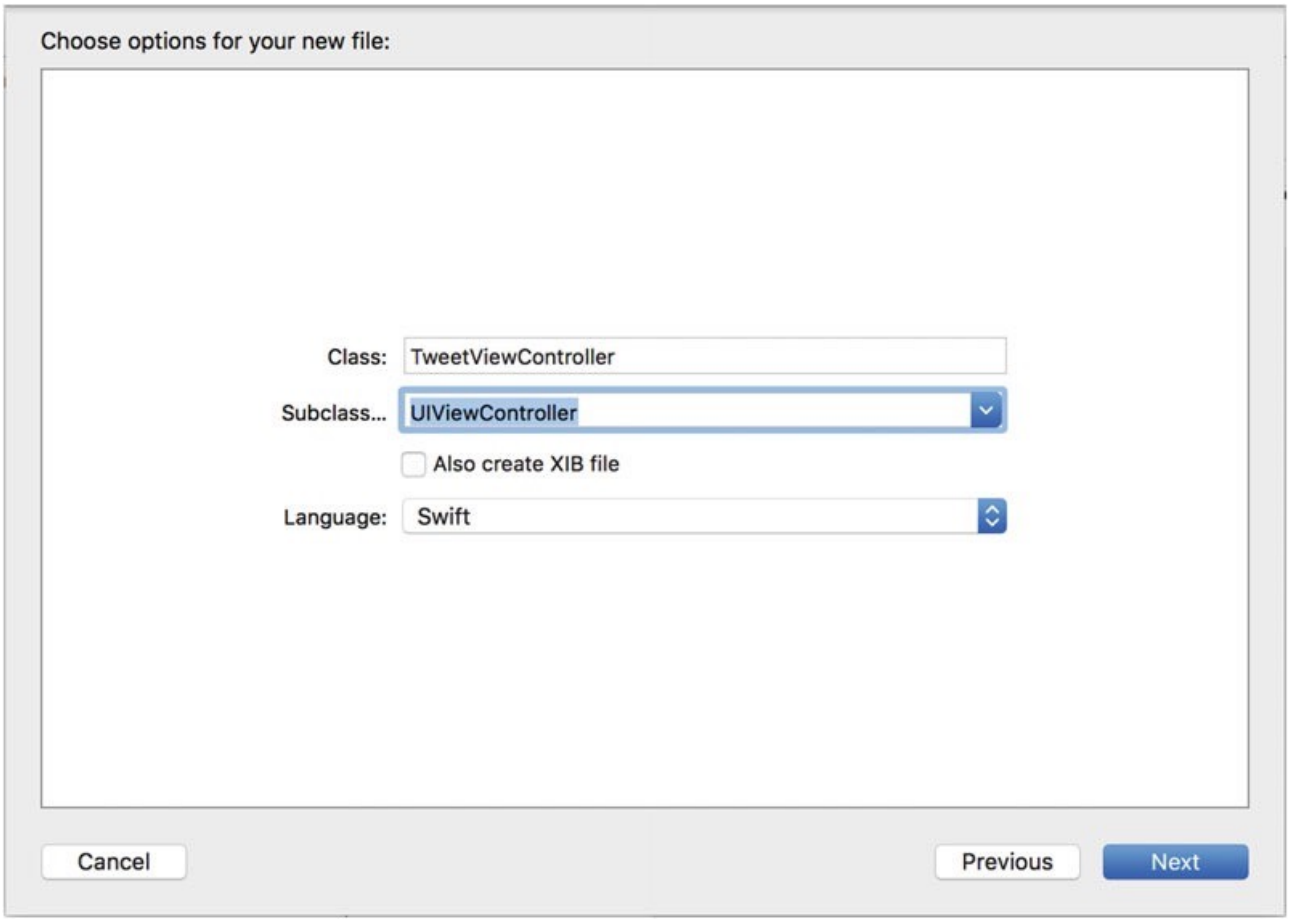

Figure 10. *Setting the options for the TweetViewController file*

4. You need to choose a location to save the file. Xcode automatically suggests the project folder, which is what you want. Be sure the Group option is set to SocialApp and that the SocialApp target is selected, as shown in Figure 11. Click Create.

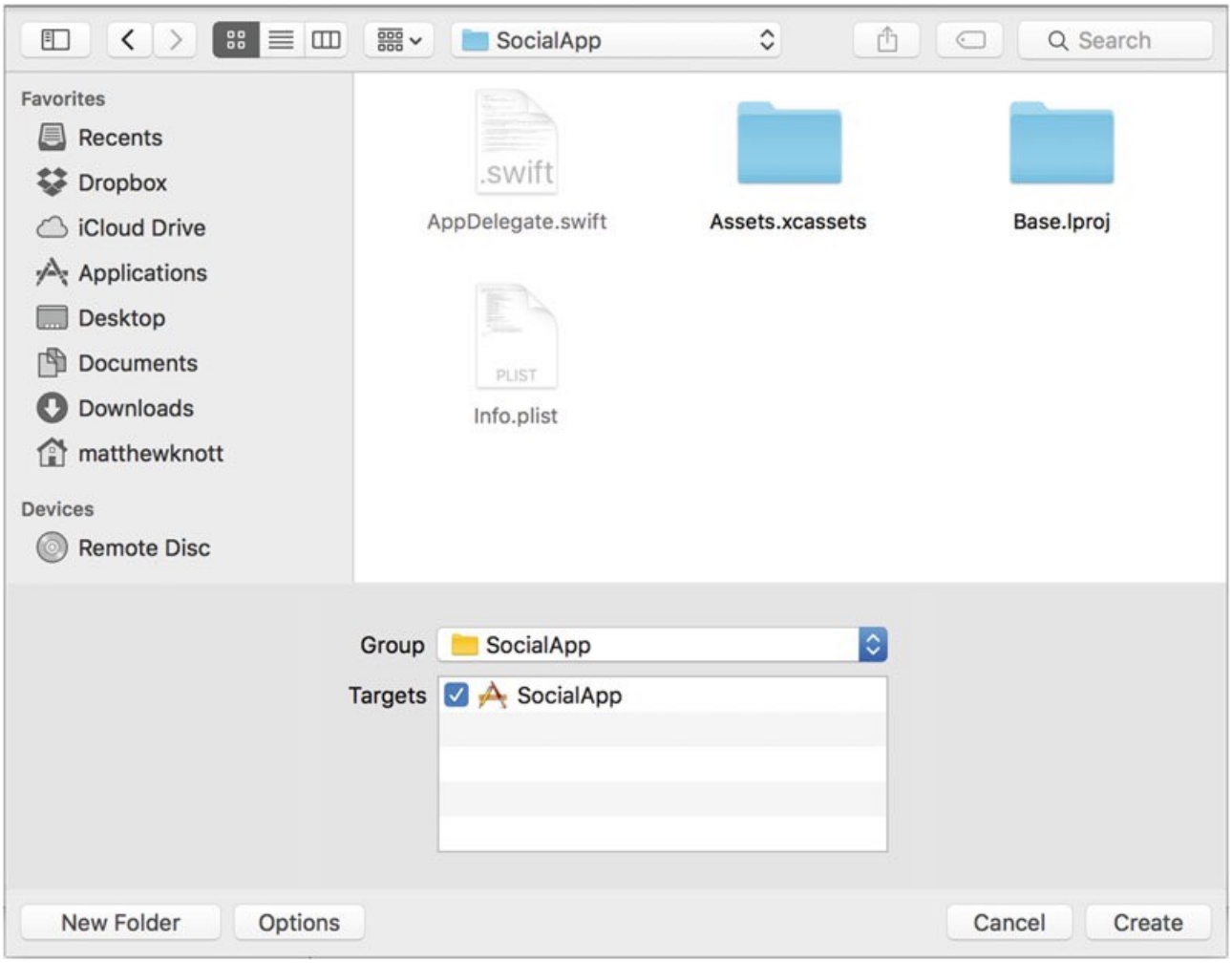

Figure 11. *Choosing a save location and specifying the group and target*

You're returned to Xcode, where you can see that TweetViewController.swift has been added to the project. That's one view controller; now repeat this process for ComposeViewController. When you're done, your Project Navigator should resemble that shown in Figure 12.

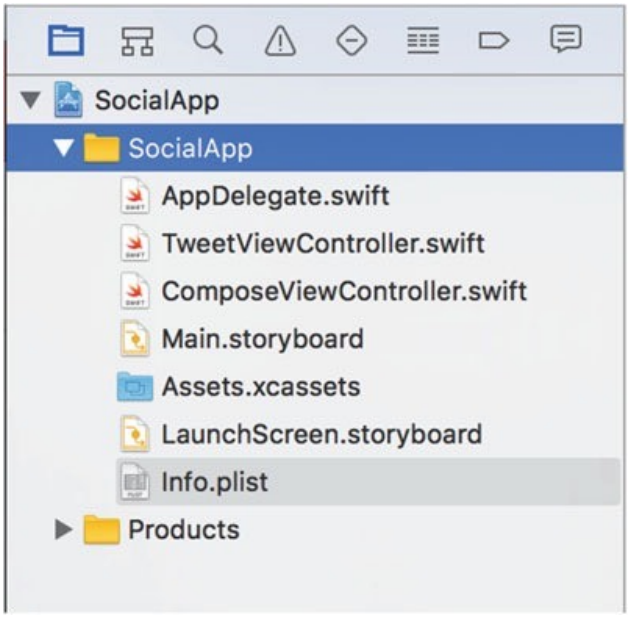

Figure 12. *The growing project in the Project Navigator*

One thing you can foresee, looking at Figure 12, is that the file list is growing and you still have three more view controllers to add. You need to tidy up the structure by grouping the view controllers together:

1. At the bottom of the Project Navigator, type View in the Show Files With Matching Name filter to make sure you only see your view controllers.

2. Click the first view controller file (in my case, it's TweetViewController.swift). Holding the Shift key, select the last file (in my case, ComposeViewController. swift). Both view controllers should be selected, as shown on the left in Figure 13.

3. Right-click the selected files, and choose New Group from Selection, as shown on the right in Figure 13.

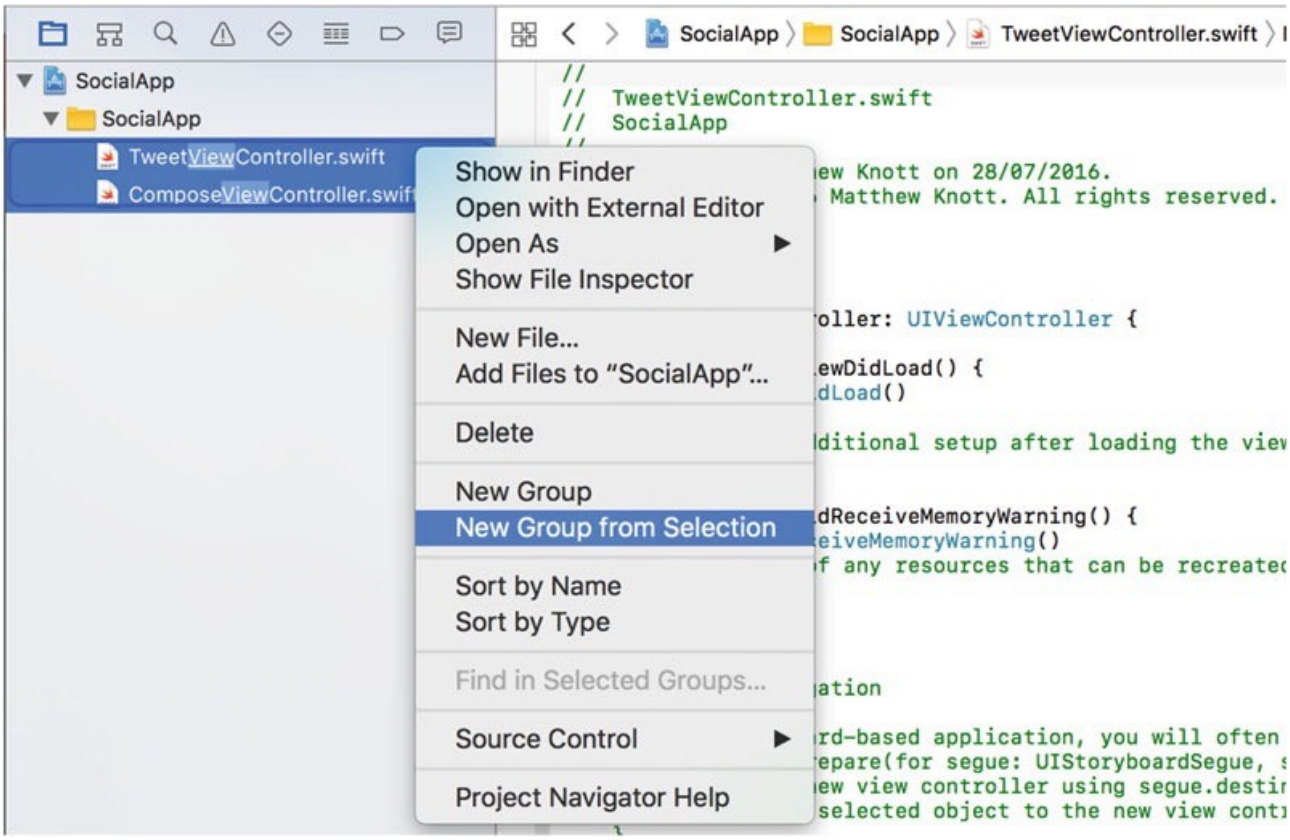

Figure 13. *Selecting the view controllers and creating a new group to contain them*

4. When prompted, name your new group View Controllers. Clear the filter by clicking the X at the end of it. You should be left with a neat project, as shown in Figure 14.

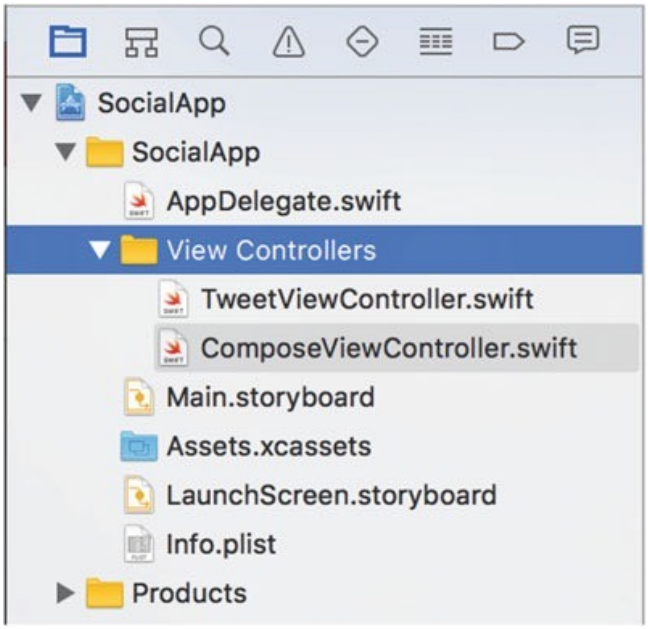

Figure 14. *The view controllers grouped neatly together*

Organizing your project neatly and logically ensures the admiration of your development team. Organizing files in big projects means if someone else has to work on your project, they don't have to hunt for the custom classes or the view controllers, because you've applied logic and good housekeeping to your project structure.

Subclassing UITableViewController

You've created two of the four view controllers, and it's time to create the remainder. The process is more or less the same, but there are some subtle changes. UIViewControllers are fairly straightforward: the views themselves are blank canvases, ready for you to add controls; and their methods are also very minimal, giving you just a viewDidLoad function and a handler for low memory. UITableViewController, however, is a more complex system, designed for displaying large amounts of data through a structured interface. It has a number of intrinsic attributes that result in the code files containing methods that are used to control the number of rows, sections, and more. Let's just create them so you can get back to the storyboard.

The process is largely the same as before, but this time you start by selecting the View Controllers group instead of SocialApp in the Project Navigator. You need to create the two instances of UITableViewController that this application uses, called AccountsViewController and FeedViewController:

1. Right-click the View Controllers group and select New File, as shown in Figure 15, or press +N. You're presented with the file template selection screen.

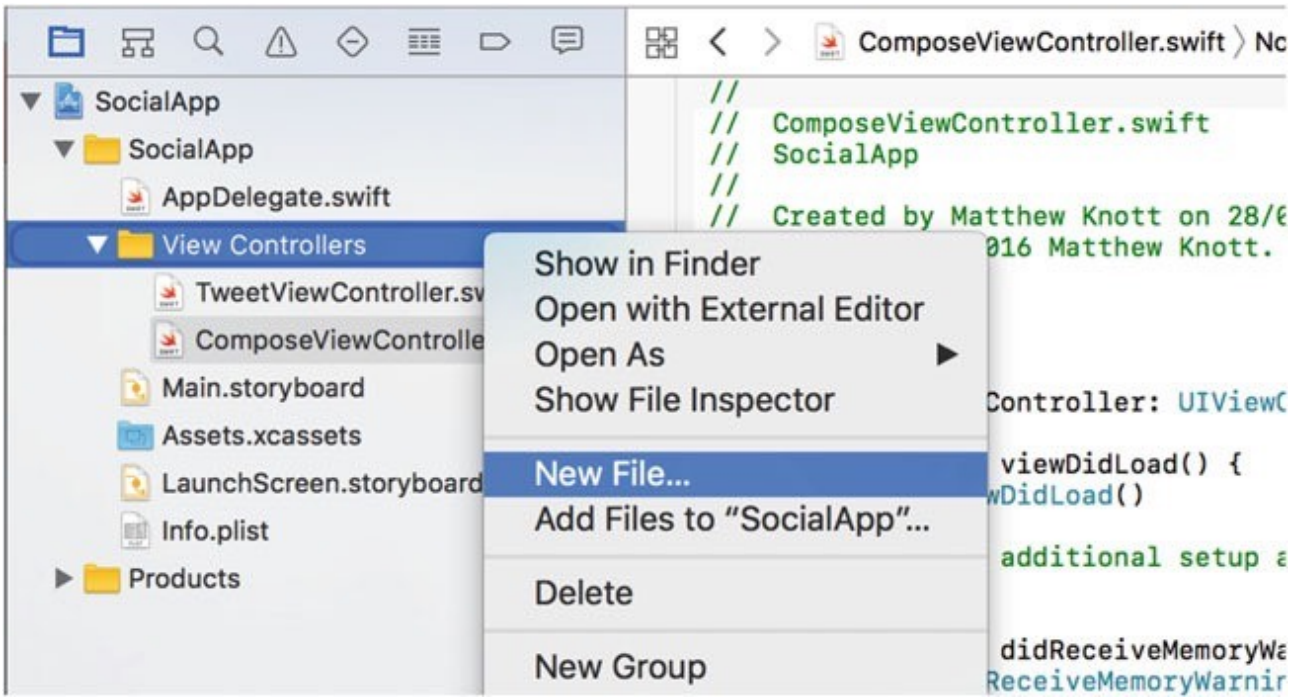

Figure 15. *Adding a new file to the View Controllers group*

2. As with the UIViewControllers, select the Cocoa Touch Class file template and click Next. Set the Class field to AccountsViewController and the Subclass Of field to UITableViewController, as shown in Figure 16, and click Next.

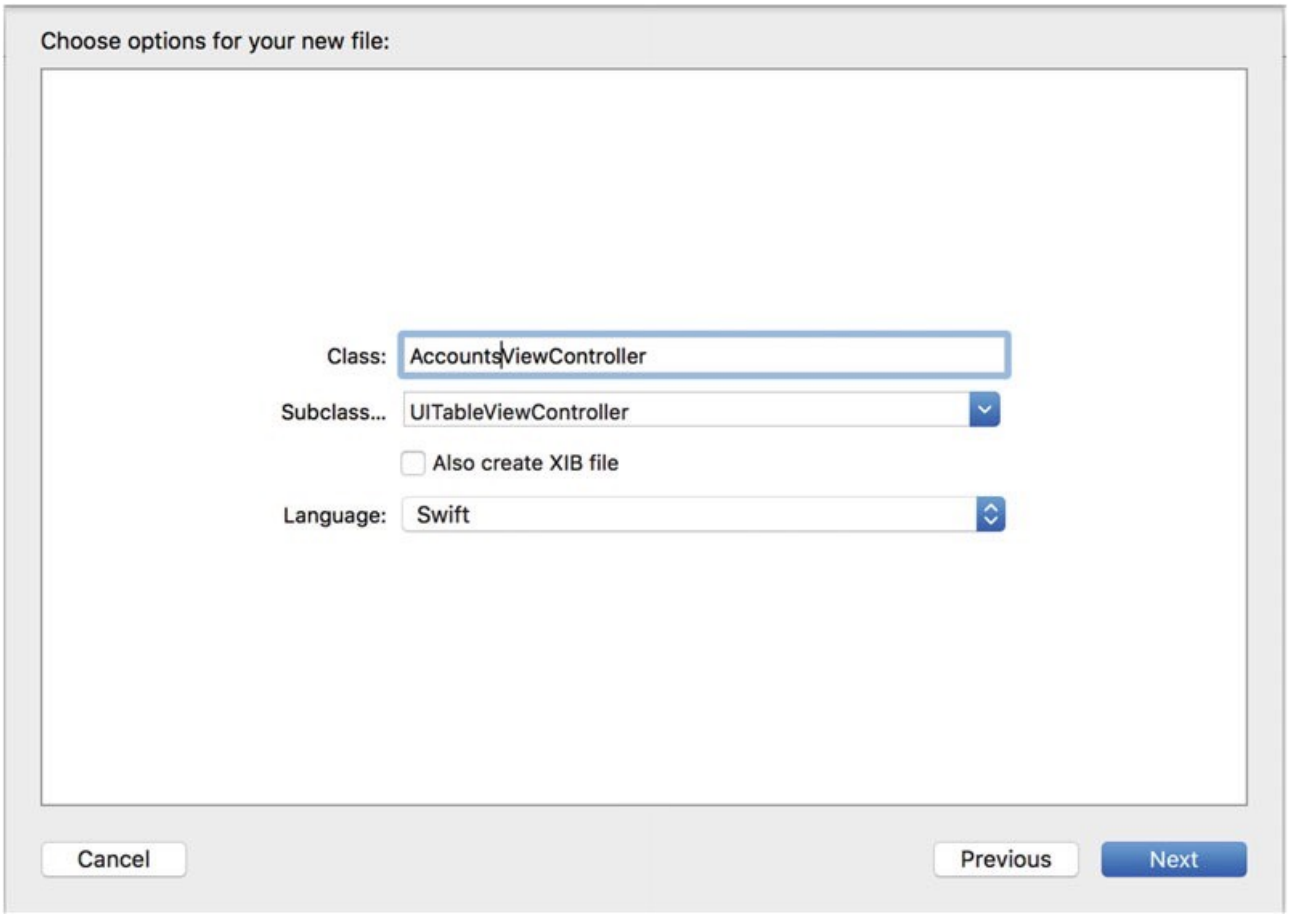

Figure 16. *Specifying the options for the UITableViewController*

3. Accept the suggested file location and click Create.

4. You should be a master at creating new view controllers, so repeat these steps and create another UITableViewController called FeedViewController.

You've now created four view controllers. Let's finish adding the views to the storyboard and tie them to their respective controllers. Your Project Navigator should look something like the screen shown in Figure 17.

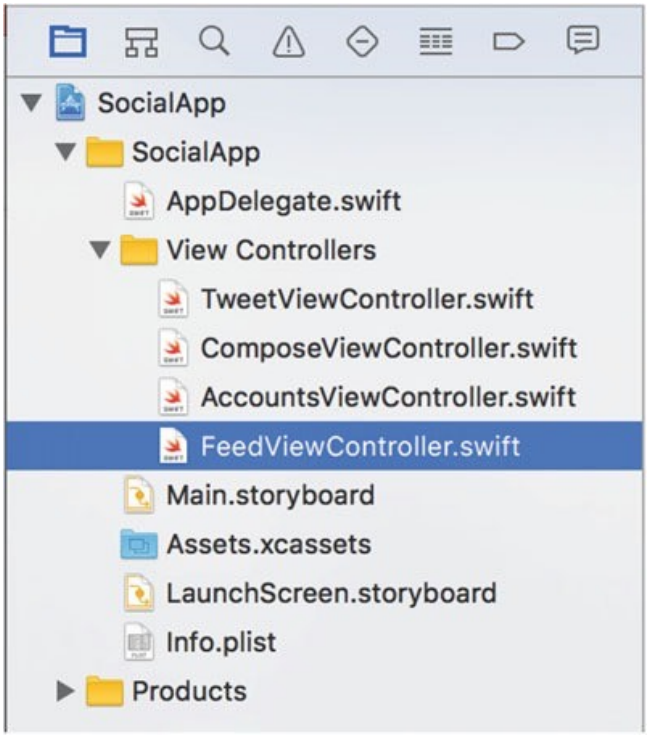

Figure 17. *The Project Navigator with all the view controllers nicely organized*

Pairing the View to the Controller

You've created the controllers, and next you need to add the views to the storyboard and tie them to their specific view controller by using the Identity Inspector from the Utilities bar:

1. Open Main.storyboard and select Accounts from below Accounts Scene in the Document Outline. When setting the class of a view, you need to select the correct view controller before you apply the class; otherwise things get messy. Use the Document Outline to make sure of your selection.

2. Open the Identity Inspector from the Utilities bar, or press + +1. Xcode should resemble the screen shown in Figure 18.

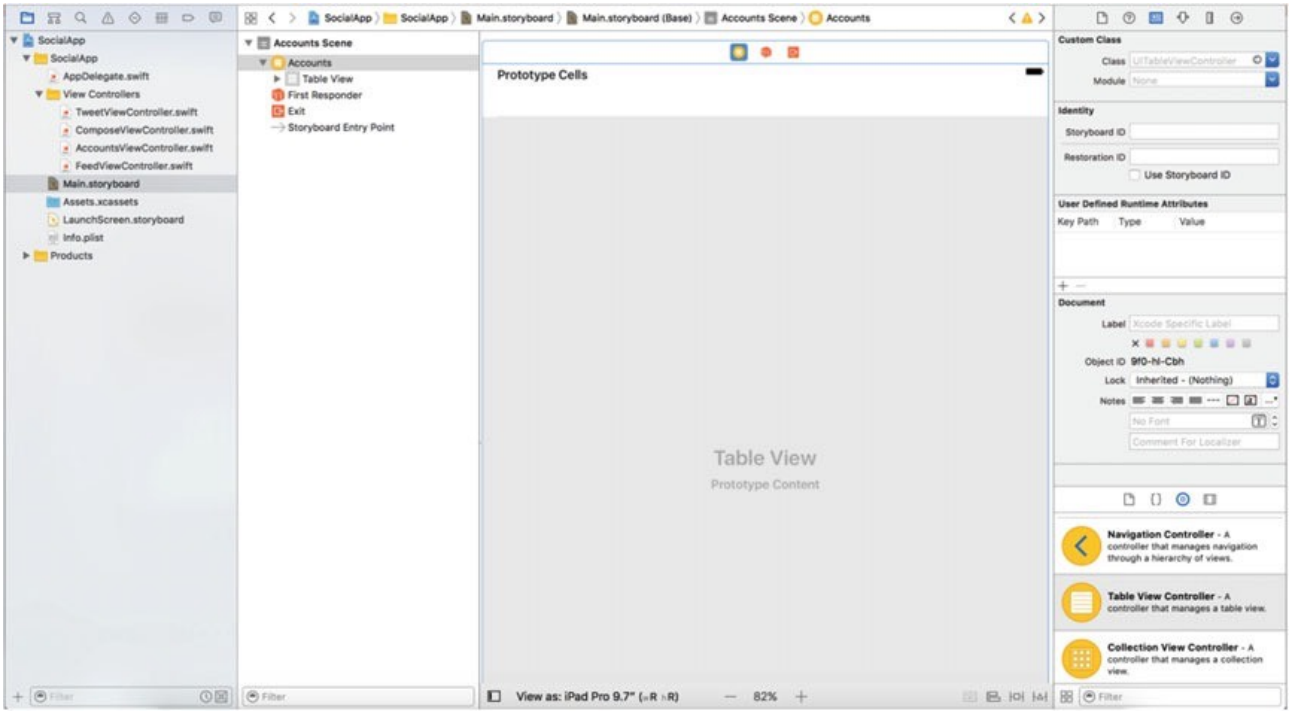

Figure 18. *Xcode with the view controller selected and the Identity Inspector open*

3. In the Identity Inspector, look for the Custom Class section. This is where you bind your view controller's visual element to the actual view controller. In the Class field, it currently says that this view controller's class is UITableViewController. It's grayed out because although it knows what its base type is, you haven't yet tied it to a custom view controller.

4. Click the down arrow at the end of the field. You should see three selections: the base class and the two custom view controllers, as shown in Figure 19. Select AccountsViewController.

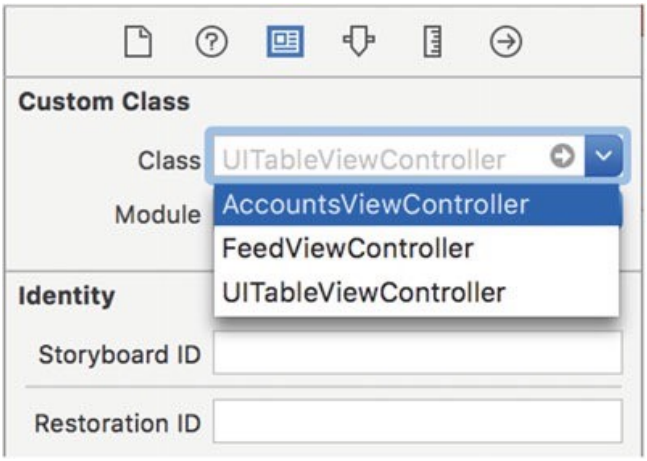

Figure 19. *The list of available view controllers*

Note If at any time you don't see one of your view controllers listed as an option in the Custom Class list and you have the view controller selected in Interface Builder, either quit Xcode and relaunch it or write the class name in yourself—but remember, it's case sensitive.

You've told the view controller on your storyboard that it no longer complies with its base class UITableViewController; now it's controlled by the AccountsViewController class. Xcode immediately reacts to the change in custom class. Try opening the Assistant Editor: it displays the implementation file for the Accounts view controller, confirming that the class is valid and the link to the storyboard is working.

Building Up the Storyboard

With Main.storyboard open, let's get back to the focus of this practice: storyboards. You've created the first scene for account selection; it's time to create the second scene, which is the list of tweets, more commonly known as the Twitter feed, which is controlled by FeedViewController:

1. This is another UITableViewController class, so you need to drag a Table view controller from the Object Library and drop it next to the first scene, as shown in Figure 20.

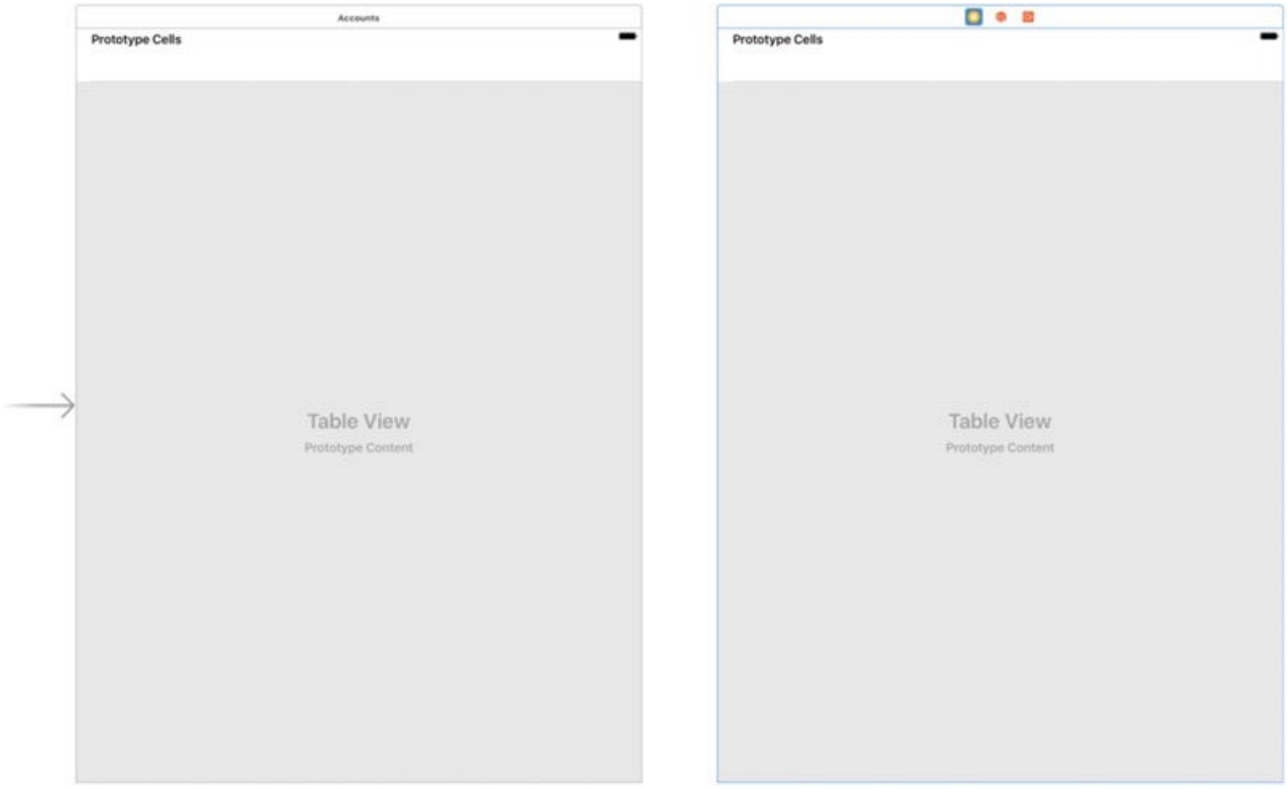

Figure 20. *The storyboard with two Table view controllers side by side*

2. Select the new Table view controller and open the Attributes Inspector. Set the Title attribute to Feed; this will help you keep track of your scenes in what will be a full storyboard.

3. Select the Identity Inspector, click the Class drop-down list, and choose FeedViewController.

4. You have two scenes on your storyboard. Let's add the last two so you can move on to a careful examination of one of the key features of any storyboard: segues. The third scene is a regular view controller that is used to show the details of the tweet. Drag a view controller from the Object Library and drop it above and to the right of the Feed view controller.

5. Select the new view controller and open the Attributes Inspector. Set the Title attribute to Tweet.

6. To set the class, open the Identity Inspector and set the Class value to TweetViewController. Your growing storyboard should resemble that shown in Figure 21.

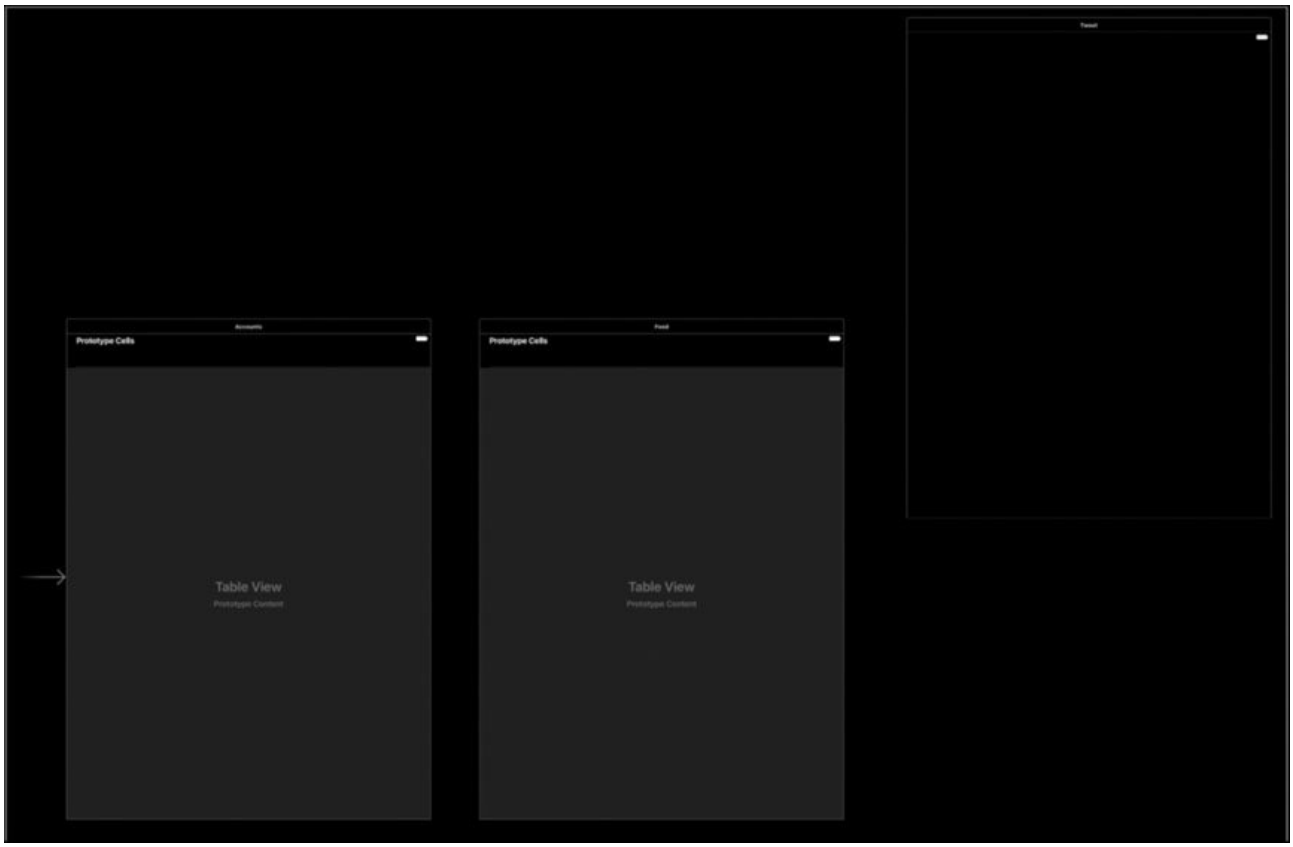

Figure 21. *The three scenes, positioned nicely and ready for the fourth*

7. By now you can start to appreciate how it can sometimes be challenging to work on large, multi-view applications using storyboards. This is why it's best to meticulously name every scene as it's created, so you can quickly identify a scene via the Document Outline. The next scene is ComposeViewController, which you will use to write tweets and post them to Twitter. Drag in another view controller and position it directly below the Tweet view controller.

8. Click the new view controller and open the Attributes Inspector. Set the Title attribute to Compose.

9. As you've done previously, open the Identity Inspector and set the Class value to ComposeViewController.

All the scenes of the storyboard are laid out neatly. Each view controller on the storyboard is tied to its respective view controller class. Although you have the basic structure in terms of the scenes, the scenes themselves are largely empty. Let's focus on the individual scenes and begin adding the elements that will make up the interface. Before you move on, check that your interface resembles the one shown in Figure 22.

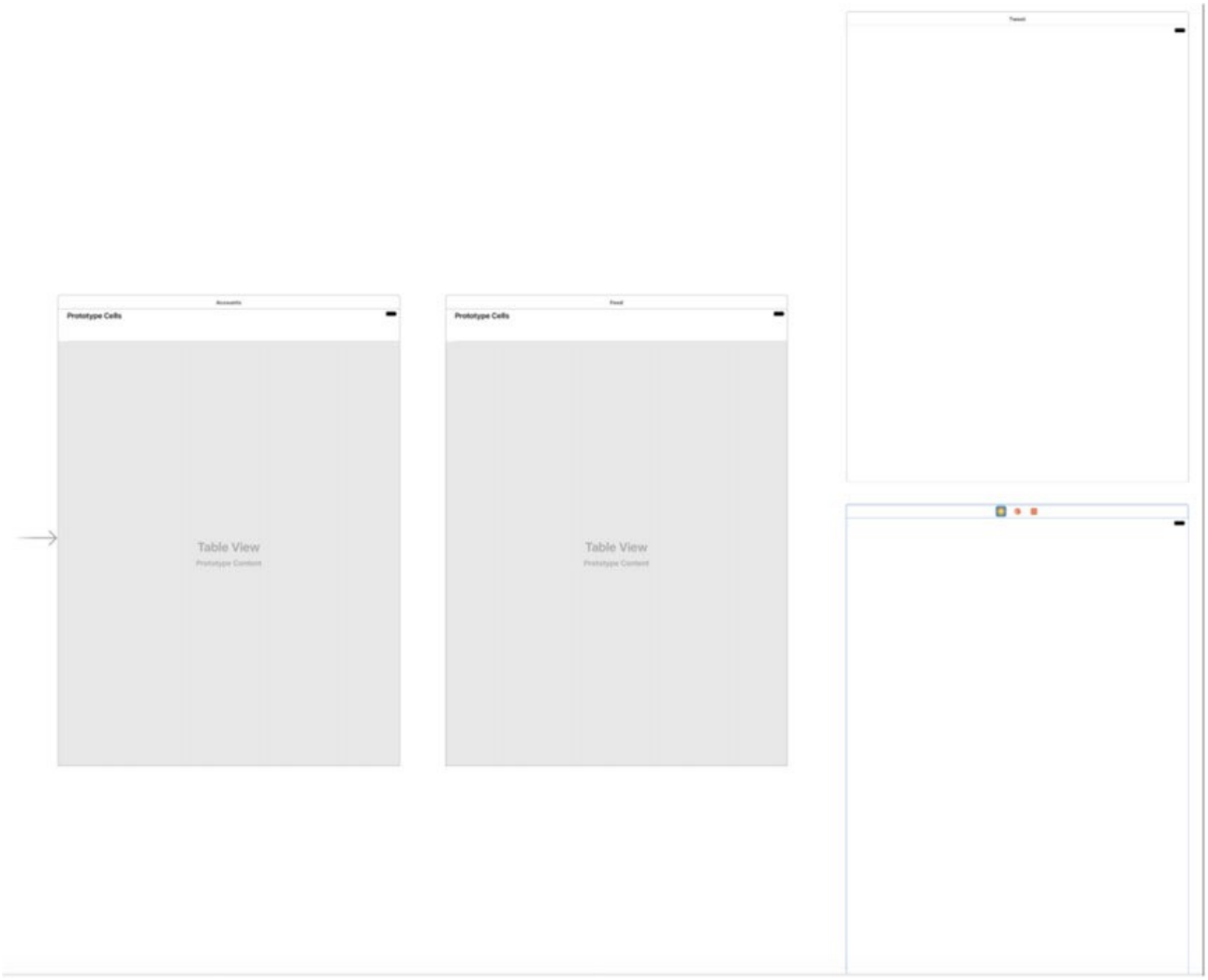

Figure 22. *The storyboard as it stands, with all the scenes in place, but without their interface elements*

Linking Scenes and Building Interfaces

One message that should be coming across in this practice is that building an application with storyboards is part of a structured process. First you plan your application and its scenes, and then you create the view controllers before tying them into their respective visual counterparts on the storyboard, giving you separate view controllers that have their classes set but are ultimately strangers to one another. To address this issue, you need to progress through each scene from 1 through 4, building and connecting the interface with segues.

Let's link the Accounts view controller to the Feed view controller. It's important to note that the Accounts view controller doesn't need any specific interface work at this point. Here are the steps:

1. Double-click the design area to zoom back to 100%. Move the storyboard around so you can get as much of the Accounts view controller and the Feed view controller in view as possible, as shown in Figure 23.

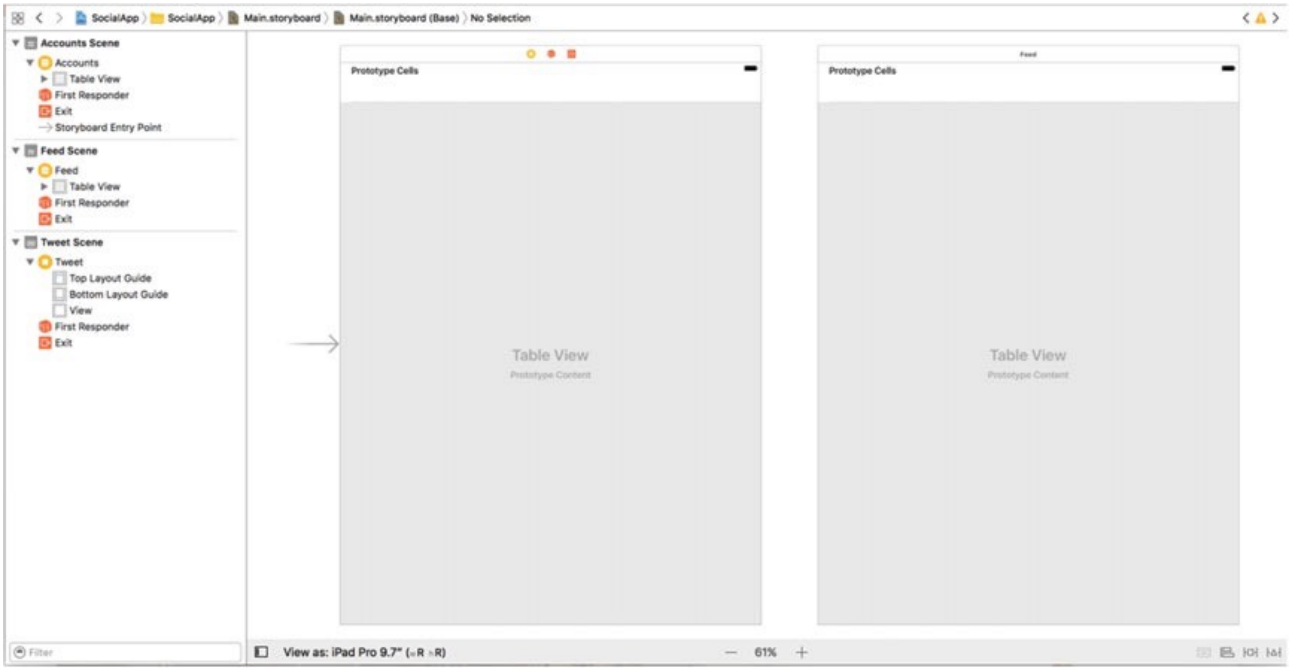

*Figure **23.** Scenes 1 and 2 side by side: the Accounts view controller and the Feed view controller*

2. The Accounts view controller lists all the Twitter accounts that are set up on the device in a table view. Selecting one of the rows takes you to the Feed view controller. To make this happen, you need to create a Show segue from the table cell to the Feed view controller. Highlight the table cell in the Accounts view controller by clicking it, as shown in Figure 24.

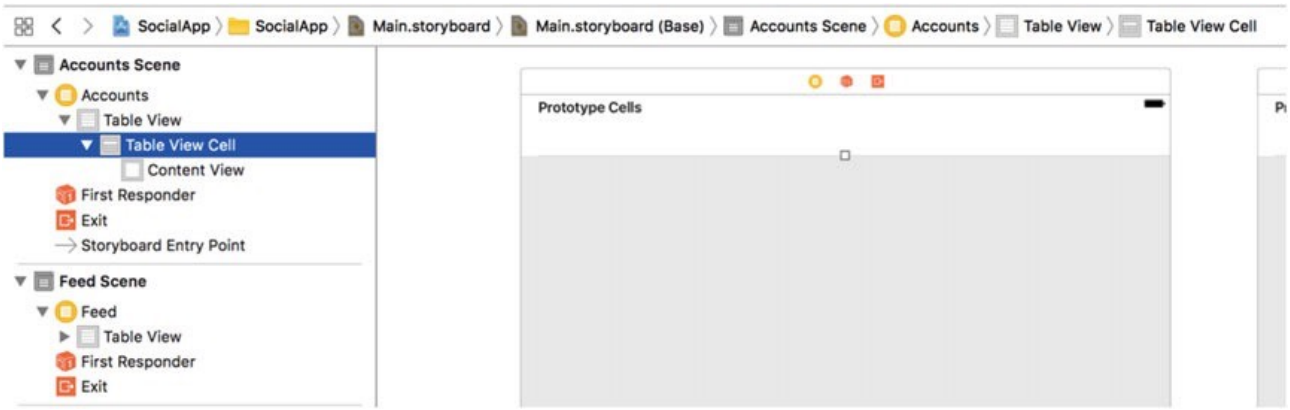

***Figure 24.** The Accounts view controller table cell selected*

Note It's not immediately obvious that you've selected the table cell, but if you look at the jump bar above the design area, or in the Document Outline shown in Figure 24, you see that it is indeed selected.

3. Hold down the Control key, click the table cell, and drag a connecting line from the cell to the Feed view controller, as shown in Figure 25.

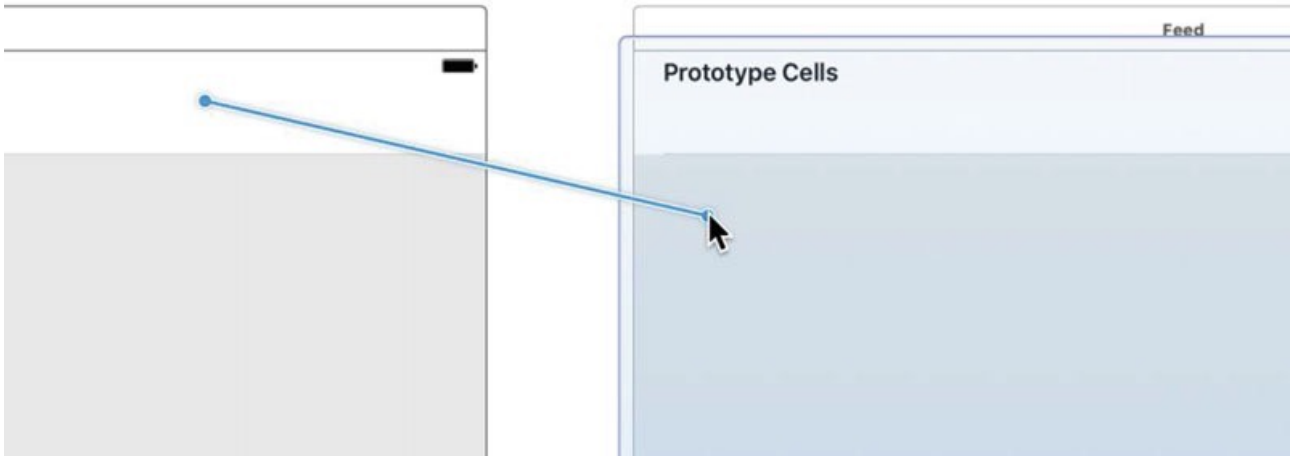

Figure 25. *Dragging a connecting line between the table cell and the Feed view controller*

4. When you release the mouse button, you're presented with a contextual dialog asking about the type of segue you want to create, as shown in Figure 26. In this instance, you want to create a Show segue from the Selection Segue list of types.

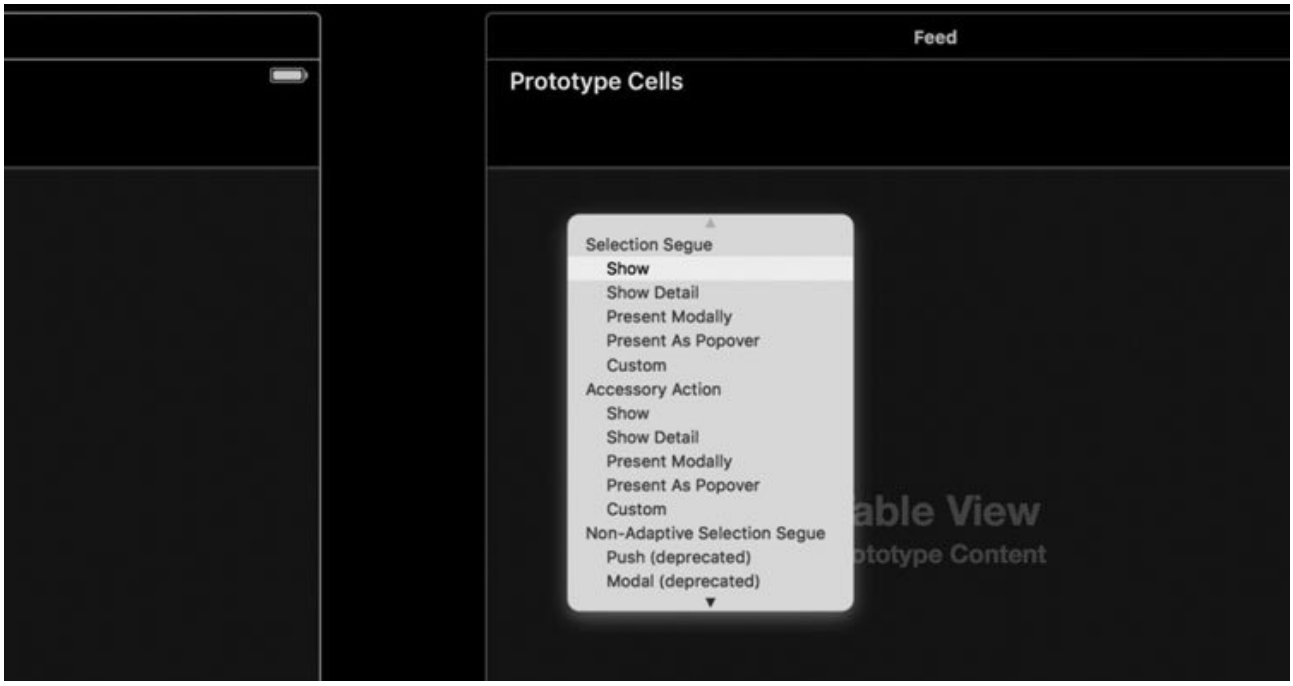

Figure 26. *The contextual menu presented when you create a segue from a table cell*

Note The dialog that appears when you create a segue is contextual because depending on your starting point, the options for creating a segue are different. The starting point for this segue is a table cell, so the options you're presented with are specific to this scenario. As you can see, there are three headings: Selection Segue, Accessory Action, and Non-Adaptive Selection Segue. This is because a table cell has two elements that support user interaction: the cell itself triggers the Selection segue, whereas the cell accessory triggers the Accessory Action segue, the Non-Adaptive heading encapsulates the segues that were deprecated with Xcode 6 and iOS 8.

5. A segue is created between your two view controllers. Select it and then open the Identity Inspector, as shown in Figure 27. In the Identifier field, type ShowTweets.

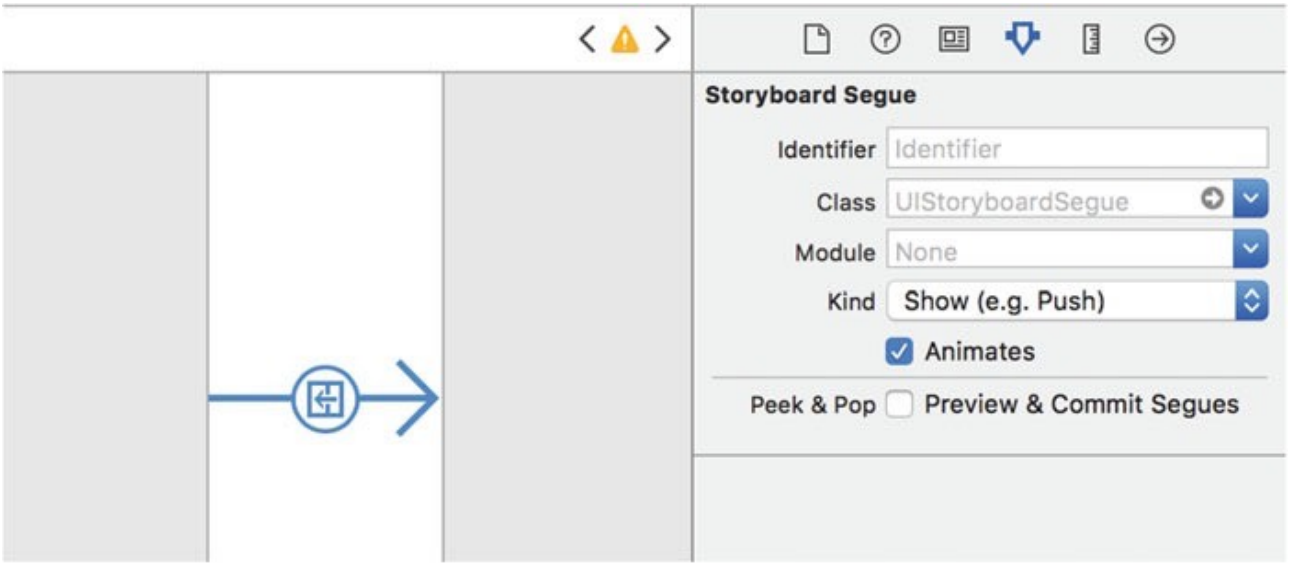

Figure 27. *Inspecting the segue*

This segue is triggered when the users tap the table cell for the Twitter account they want to use. When this happens, you need to pass the selected account details to the Feed view controller. Previously, you did this by writing code that passed an object to the next view controller; but when working with segues, you use the prepareForSegue method, which is called every time a segue on the view controller is triggered. Later you will write the code that performs different actions depending on the segue that is being triggered. But to identify which segue is being triggered, you must give each an identifier.

Note Even when you have only one segue coming from a view controller, as in this case, it's still a good practice to give it an identifier so that if you add more segues to the view controller in the future, you won't have a situation where you're passing information to the wrong target view controller.

Before we go any further, let's have a look at the different segue styles available and in what situations we might use each one:

Show: Prior to Xcode 6 and iOS 8, this was referred to as a Push segue. This segue dismisses the current view and pushes the target of the segue onto the screen. Behind the scenes, the Show segue adds the target view controller onto the navigation stack, which is why you always need a navigation controller to be present when using a Show segue. Xcode does all the work of managing the navigation stack for you: when your Show segue is triggered, the view controller that is presented automatically has a button to go back to the previous view controller.

Present Modally: Modal segues are definitely the most interesting and varied type you can use, especially when working with iPad applications. A modal segue presents another view controller without the need of a navigation controller. You can slide these over the top of the view that calls the segue. A number of transition animations and presentation styles are available, some of which we will see later in this practice. Figure 28 shows a modal segue in action, presenting the Tweet view controller from SocialApp modally with the Form Sheet presentation style.

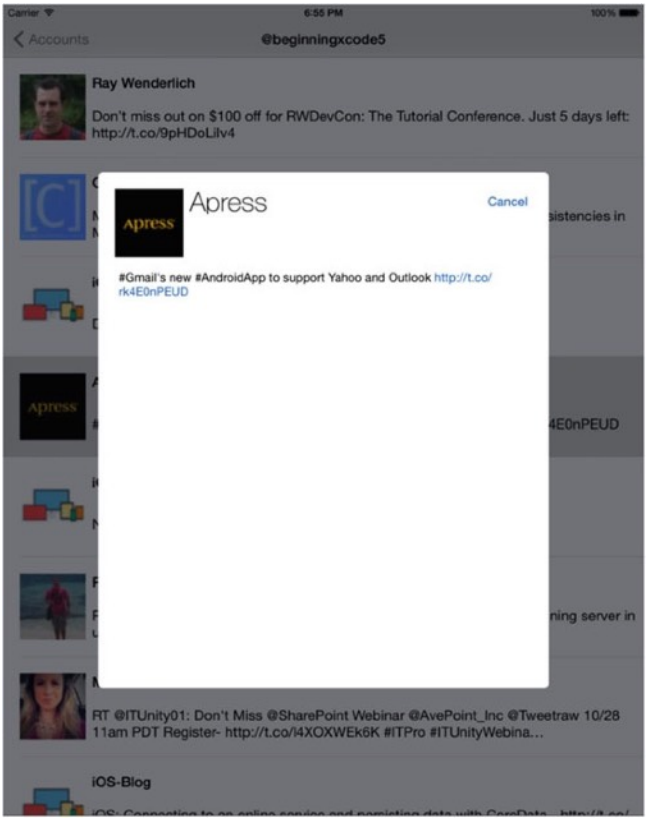

Figure 28. *A modal segue in action*

Popover Presentation: Popovers are visually similar to some modal segues in that they cause a view controller to appear above the view controller that originated the segue. These are useful for displaying contextual information; for example, if you created an application for an online store, you could use popover segues to provide a quick view of your products. You can see an example of a popover in Figure 29; note that the arrow at the top of the view controller is generated by Xcode and can be configured.

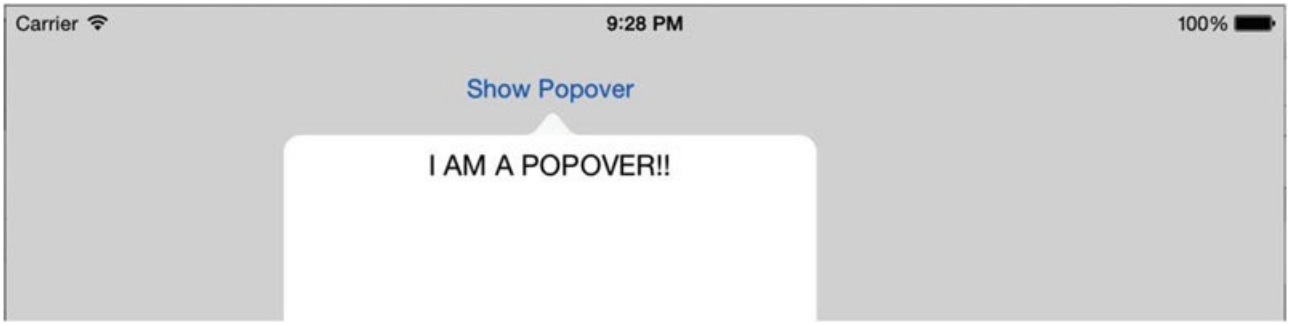

Figure 29. *An example of a popover segue presenting another view controller on top of the view*

Show Detail: Known as a Replace segue prior to Xcode 6, this style was previously available only when working with an iPad storyboard. It's mainly used with master-detail applications. A Show Detail segue replaces the originating view controller with the target one in the navigation stack.

Custom: As you might expect, a custom segue can be anything you tell it to be. With a custom segue, you specify a custom UIStoryboardSegue class in a way similar to how you would set custom classes for the view controllers.

Now that you understand the different segue styles, perhaps you've noticed from the description of the Show segue that your application is missing something essential: a navigation controller.

Adding a Navigation Controller

A navigation controller, or UINavigationController, is used to manage navigation through the various view controllers in your application. It keeps track of where the user has been, adding each successive view controller to the navigation stack, and provides a mechanism for the user to navigate backward through the navigation stack, all without you having to write any code.

Because the Accounts view controller is the originator of the Show segue, and because it's the initial scene on the storyboard, the navigation controller needs to be added here:

1. Select Accounts from the Document Outline, as shown in Figure 30, because you're applying the navigation controller explicitly to the view controller.

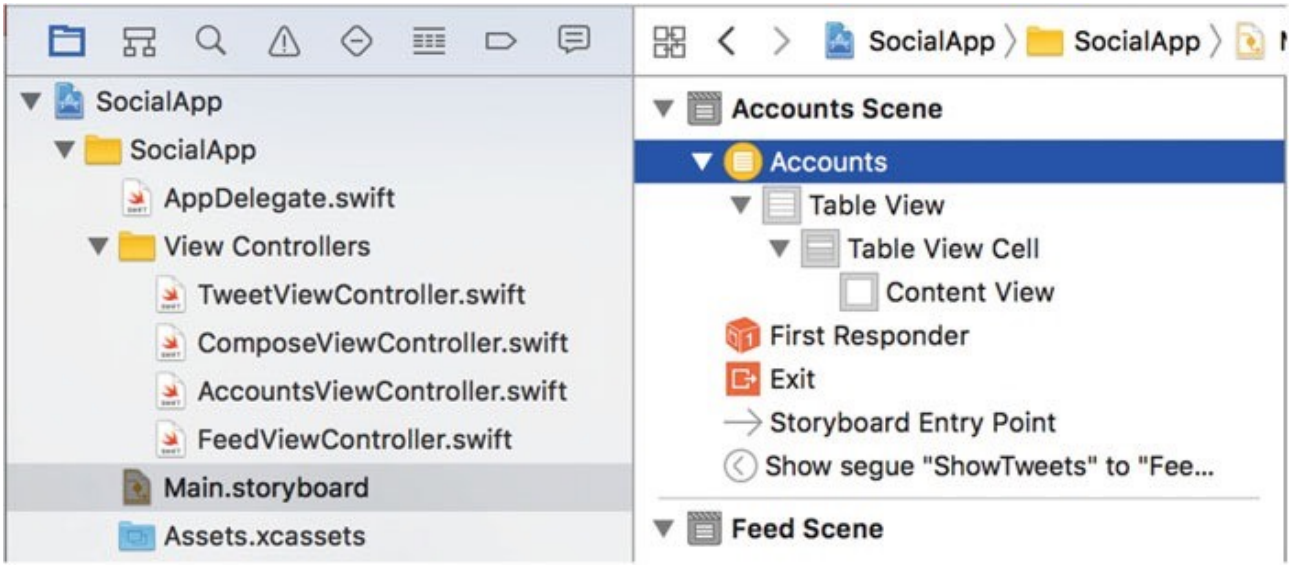

Figure 30. *Selecting the Accounts view controller from the Document Outline*

2. To add a navigation controller to this view controller, from the menu bar select Editor ➤ Embed In ➤ Navigation Controller. Xcode adds a navigation controller to the storyboard and attaches it to the Accounts view controller for you, as shown in Figure 31.

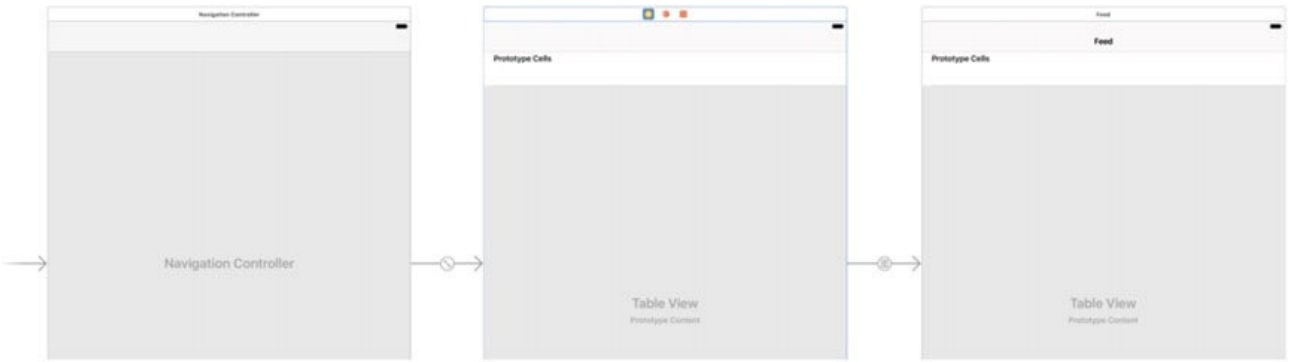

Figure 31. *The navigation controller added to the storyboard and linked to the Accounts view controller*

3. You can set a title for the view that is visible to the user and provides meaningful text for the Back button. Zoom back in to the storyboard. In Accounts, highlight the navigation bar at the top of the view or select Navigation Item from the Document Outline, as shown in Figure 32.

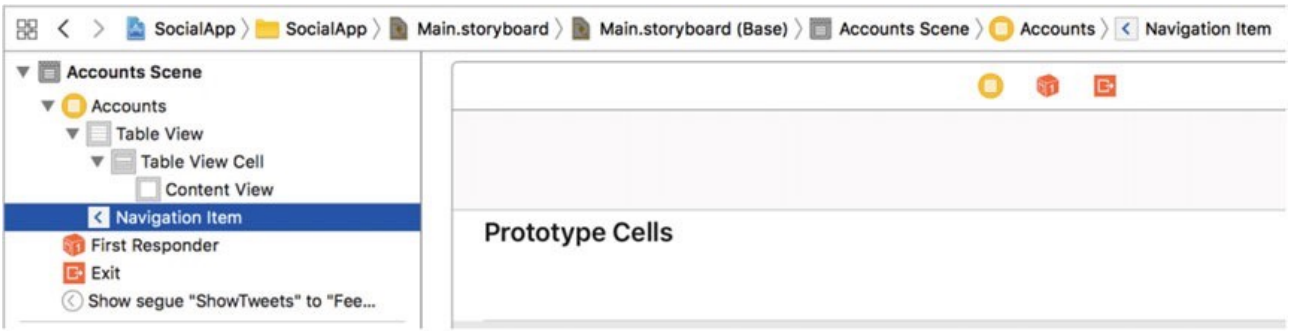

Figure 32. *Selecting the Navigation Item from the Document Outline for the Accounts view controller*

4. Open the Attributes Inspector and set the Title attribute to Accounts. Notice how the title appears at the top of the view controller, as shown in Figure 33. What's also neat is that when you select an account and segue to the Feed view controller, the Back button is automatically labeled Accounts.

Figure 33. *The Title attribute for the navigation bar in place*

With the navigation controller in place and the Accounts view controller complete for now, you're ready to start building the interface on the remaining three scenes. Next up: the Feed view controller.

Creating an Interface for the Feed View Controller

The Feed view controller is responsible for showing all the tweets in the user's Twitter timeline. Later in this practice, you will create a custom table cell to display the actual tweet. But for now you need to add a button to the navigation bar so you can compose new tweets, and then create modal segues to both the Compose scene and the Tweet scene. Zoom to 100% and ensure that the Feed view controller is front and center:

1. Use the mouse or track pad to scroll and pan through the design area until the Feed view controller is centered, ready for configuration. Alternatively selecting a scene from the Document Outline will shift the focus to that item.

2. Locate Navigation Item in the Object Library, as shown in Figure 34, and drag it onto the Feed view controller. Remember to use the filter and search for *navigation* to make your search easier. When you do this, the title will change from Feed to Title.

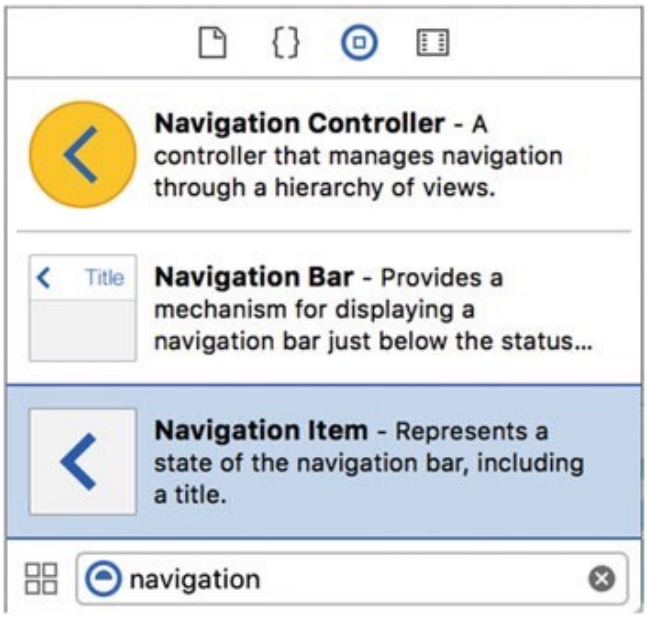

Figure 34. *Searching for Navigation Item in the Object Library*

3. You need to add a button to the navigation bar so that you can create a modal segue to the Compose view controller. Search for *button* in the Object Library. You're looking specifically for Bar Button Item, which should be the second item in the Object Library. Drag it onto the right side of the navigation bar, positioning it as shown in Figure 35.

Figure 35. *Adding a bar button item to the navigation bar*

4. If you still have the Attributes Inspector open, when you drop the bar button item on the navigation bar, its attributes are displayed. If not, open the Attributes Inspector and select the new button.

5. You could change the Title attribute of this button to read Compose, because that is the scene it will link to. However, Apple provides a standard set of icons you can use for this button by choosing one of the predefined Identifiers. Click the System Item attribute list and select Compose, as shown in Figure 36. Your button changes from text to an icon with a pencil in a box.

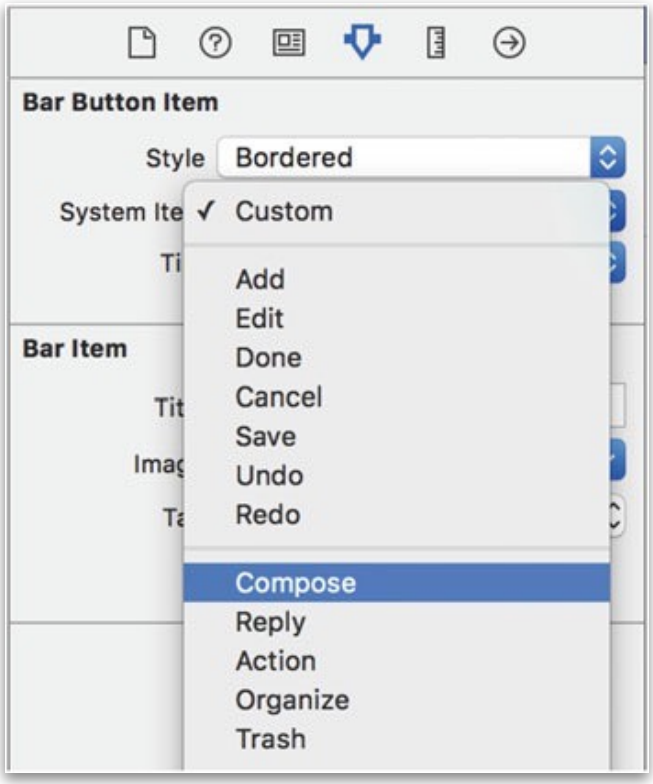

Figure 36. *Selecting the Compose identifier for the bar button item*

Note Apple provides a number of identifiers that can be used for many common tasks. You should use these whenever possible, to provide users with icons they're familiar with and also to future-proof your application. If Apple updates that identifier in the future, your applications will be easy to update to the new design.

6. Finally, select the navigation bar and, in the Attributes Inspector, change the Title attribute from Title to Feed.

7. You've created the interface for your second scene. Now you need to create modal segues to the Compose view controller and the Tweet view controller. This time, because the scenes are spread out in the storyboard, you can use the Document Outline to add an element of simple precision to the task of creating segues. Compress all the scenes except the Feed, Compose, and Tweet view controllers by clicking the triangles to the left of each scene's title. Then, expand Feed, and beneath that, expand Table View, so your Document Outline resembles that shown in Figure 37.

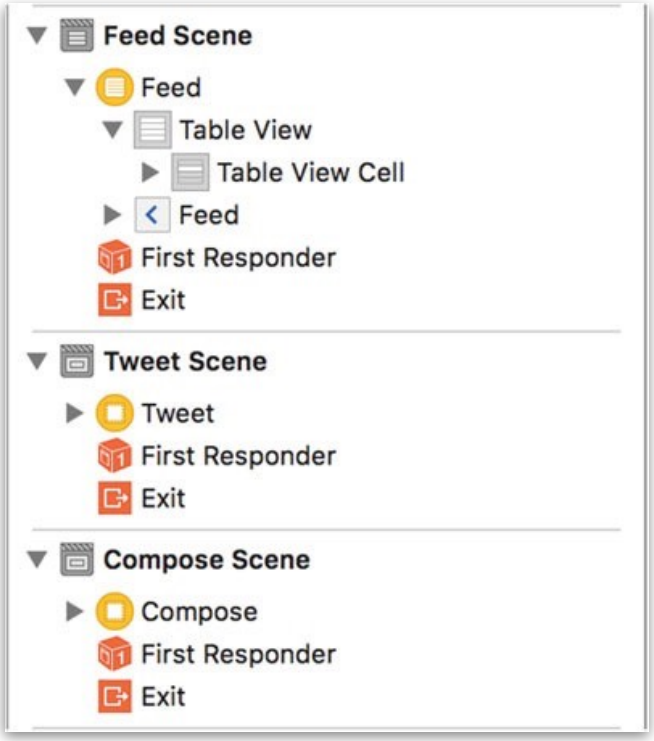

Figure 37. *Preparing to create segues using the Document Outline*

8. The first segue you create is to the Tweet view controller. You get to this scene by selecting one of the table cells, so highlight Table View Cell and Control+drag a connection from there to the Tweet view controller, as shown in Figure 38.

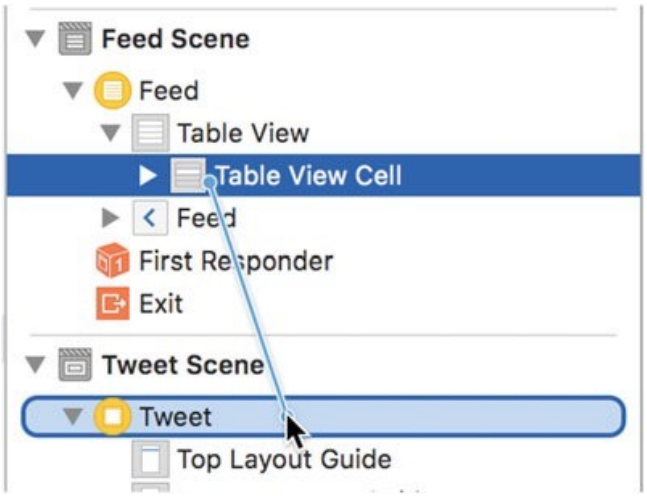

Figure 38. *Creating a segue using the Document Outline*

9. When you release the mouse button, the contextual dialog appears, just as it did in Figure 26. This time, select Present Modally under the Selection Segue heading. You now have a modal segue connecting the table cell in your scene to the Tweet view controller, but you need to customize it slightly.

10. In the Feed Scene section of the Document Outline, notice the item Present Modally Segue to Tweet. This is the segue: select it to highlight it, and then open the Attributes Inspector.

11. Set the Identifier attribute to ShowTweet, the Presentation attribute to Form Sheet, and the Transition attribute to Cover Vertical.

12. You need to create a modal segue from the Compose bar button item to the Compose view controller. Expand Feed in the Document Outline to reveal Compose, as shown in Figure 39.

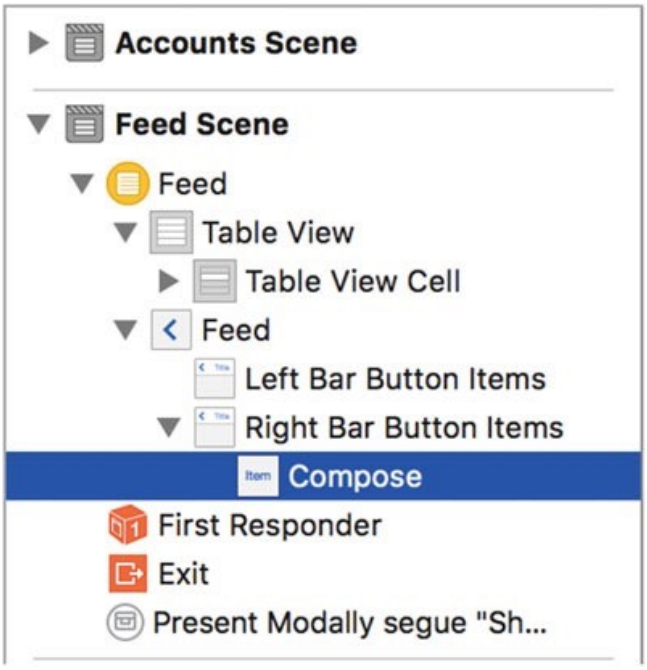

Figure 39. *Exposing the bar button item in the Document Outline*

13. In the Document Outline, Control+drag a connection from the Compose button to the Compose view controller, as shown in Figure 40. When you release the button, select Present Modally. Notice this time that the menu is different because you're dragging from a button, not a table cell.

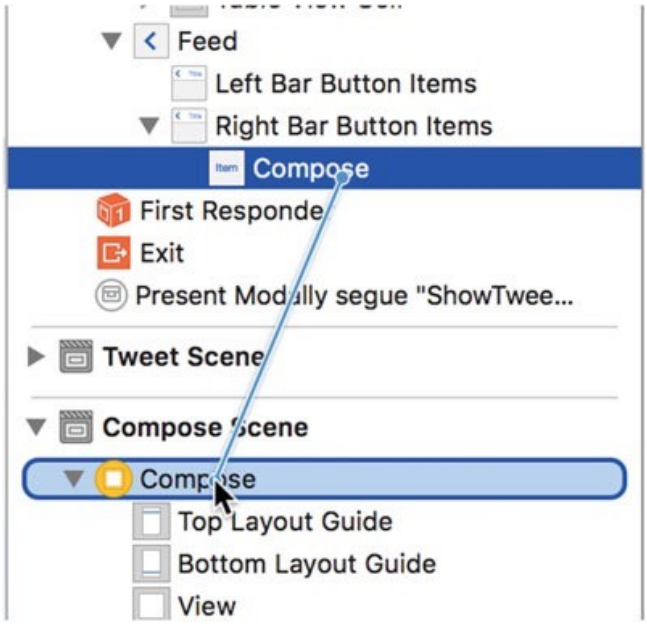

Figure 40. *Making a connection between the button and the Compose view controller*

14. Two segues are listed in the Document Outline for your scene. Select the one named Present Modally Segue to Compose. Open the Attributes Inspector and set the Identifier attribute to ComposeTweet, the Presentation attribute to Form Sheet, and Transition to Cover Vertical.

Let's review where you are with the project. Zoom out so you can see more of the storyboard, as shown in Figure 41. You're ready to move on to the third scene, the Tweet view controller.

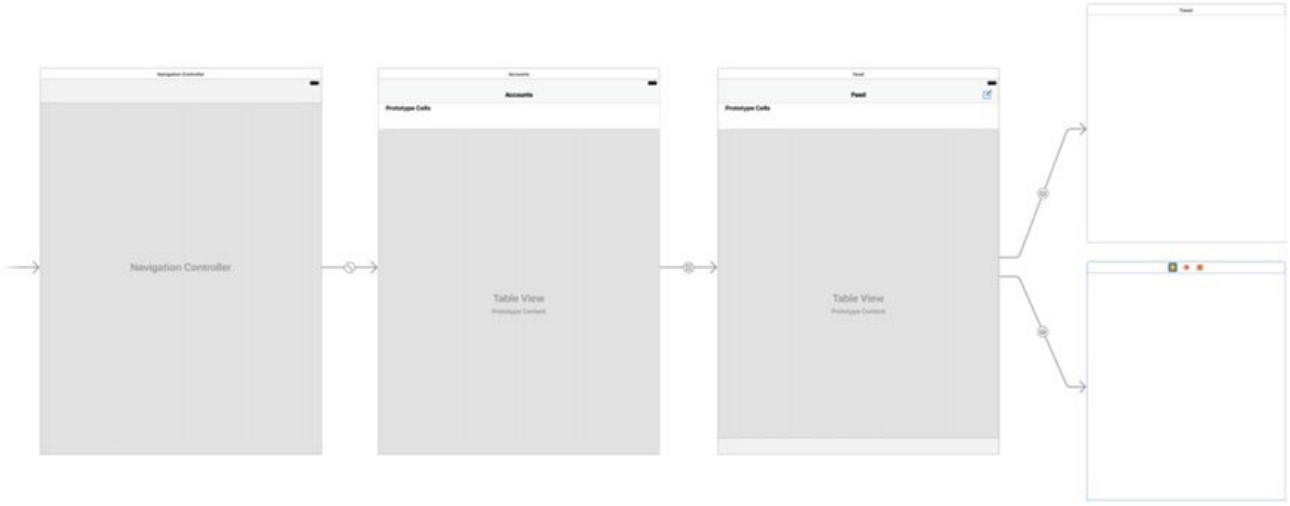

Figure 41. *The storyboard changes in appearance after you set up two modal scenes*

Creating an Interface for Tweet View Controller

The purpose of the third scene is to display details about the tweet the user selects from the Twitter feed. You could display all types of information, but for this application you simply display the tweet author's name, the tweet content, and the tweet author's avatar image. Because you've done these steps a few times already in this practice, we use screenshots at key points in the process so you can verify that you haven't missed anything:

1. Because this is a modal scene, it has no Back button. You need to be able to dismiss the view controller when the user wants to return to the Twitter feed. Add a button at upper right in the view by dragging one from the Object Library. In the Attributes Inspector, set the Title attribute to Cancel.

2. You may need to resize the button slightly to make the text visible. When you're happy with the button's appearance, position it as shown in Figure 42. You'll set the auto layout constraints once the view is built.

Figure 42. *Placing the button on the Tweet view controller*

3. You need to add an image view from the Object Library to the view controller. Position it somewhere in the view, and then open the Size Inspector. This is a handy tool for gaining pinpoint precision over an object's size and position. Set the X axis value to 20 and the Y axis value to 20; these control where the top-left corner of the image view is positioned relative to the parent view. Set the Width and Height values to 82, as shown in Figure 43, to create a square image view.

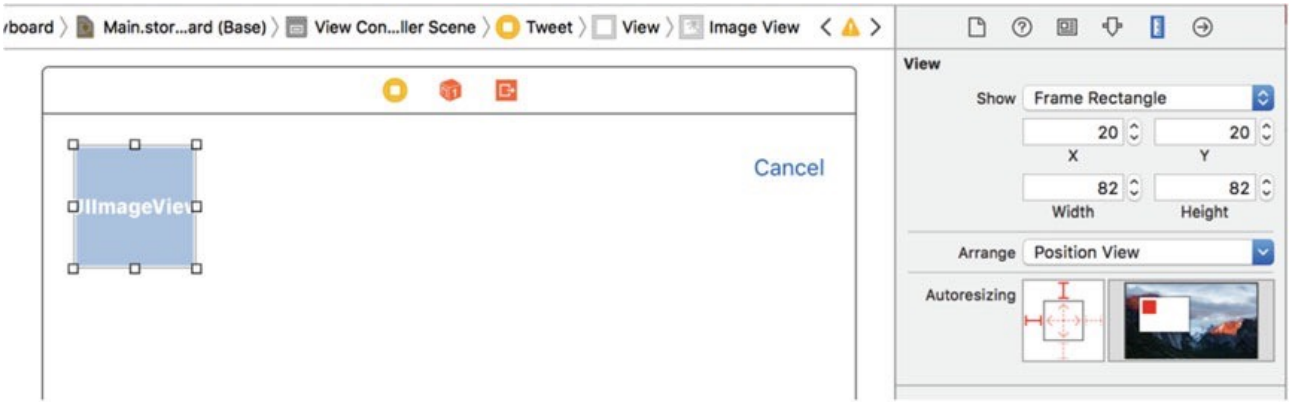

Figure 43. *Setting the width, height, and x and y axis positions of the image view*

4. From the Object Library, drag in a label to contain the tweet author's name. Position it loosely between the image view and the Cancel button. You'll adjust the size and the font, so it makes sense not to finalize the position until after you've done that.

5. Open the Attributes Inspector with the label selected. Select the T icon within the Font attribute. Set Font to Custom, Family to Helvetica Neue, Style to Thin, and Size to 34, as shown in Figure 44.

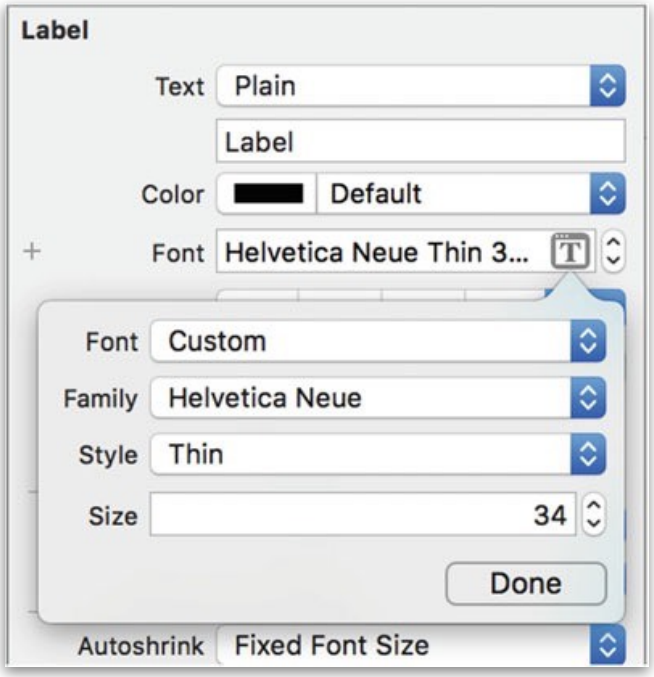

Figure 44. *Adjusting the Font attribute for the label that shows the tweet author's name*

6. Notice that you can't see any text in the label, due to its size and shape. Position the left side and top of the label so it snaps into alignment with the top of the image view. Then resize it so that it uses the rest of the available width in the view controller and the height is sufficient to show the text clearly, as shown in Figure 45.

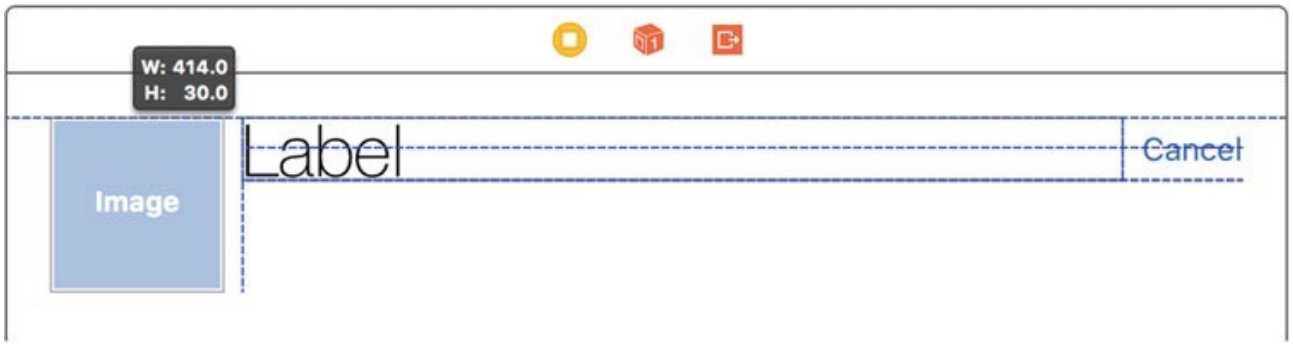

Figure 45. *Setting the label to fill the remaining width*

7. You need something to show the main content of the selected tweet. Tweets vary in size, so the best choice is a text view. Drag one from the Object Library, snapping it into place below the image view. Resize it to give it a generous size, as shown in Figure 46.

Figure 46. *The text view added to the viewcontroller*

8. You don't want this text view to behave like a regular text view, because it shouldn't be editable. Open the Attributes Editor and uncheck the Editable attribute. Leave Selectable checked so the user can copy the tweet text.

9. While the attributes of the text view are open, look at the Data Detectors section. Twitter posts often contain links, so check the Link attribute, which tells Xcode to detect URLs and provide a code-free way of opening the link in Safari.

10. Click the Resolve Auto Layout Issues button. Under the heading All Views in Tweet View Controller, click Add Missing Constraints. This should ensure that your layout adapts correctly.

11. Even though you're using storyboards, you still need to create actions and outlets for the interface objects. Open the Assistant Editor, and be sure it displays TweetViewController.swift in the right pane. If it doesn't, you need to go back and set the class for this view controller to TweetViewController.

12. As you've done numerous times previously, Control+drag the following outlets: tweetAuthorAvatar for the image view, tweetAuthorName for the author's name label, and tweetText for the text view. Create an action for the Cancel button called dismissView. Your header should contain the following highlighted code:

```
import UIKit
class TweetViewController: UIViewController {
    @IBOutlet weak var tweetAuthorAvatar: UIImageView!
    @IBOutlet weak var tweetAuthorName: UILabel!
    @IBOutlet weak var tweetText: UITextView!
    @IBAction func dismissView(_ sender: AnyObject) {
}
```

You're finished setting up the third scene for viewing tweets in detail. Next you need to set up the fourth and final scene: the Compose scene.

Creating an Interface for the Compose View Controller

The Compose view controller provides an interface for the user to create a tweet and post it to Twitter. You use a text view for the composition of the tweet, and you give the user two buttons—one to dismiss the view and one to post the tweet—and an activity indicator that triggers when posting the tweet. Let's get started:

1. Switch back to the Standard editor and focus on the Compose Scene.

2. To make the text view stand out more when you add it, let's change the view's background color. Click a blank area of the view and open the Attributes Inspector. Choose a background color; we changed the Background attribute to one of the predefined colors (Group Table View Background Color), but you can select whichever color you feel works for you.

3. Before you add the other interface objects, add a label to act as a title for the view. Drag in a label from the Object Library and position it in the upper-left corner of the view.

4. In the Attributes Inspector, change the label's Text attribute to Compose a Tweet. You also want to make it stand out, so click the arrow inside the Font attribute and set Font to Custom, Family to Helvetica Neue, Style to Thin, and Size to 32. Resize the Label so that all the text is visible, but don't resize it to fill the width of the view. Reposition the label in the upper-left corner so the guidelines for the margin appear.

5. Drag a button to the upper-right corner of the view; this button will dismiss the view controller. Open the Attributes Inspector and change the button's Title attribute to Cancel. Before you move on, the top of your scene should resemble the one shown in Figure 47.

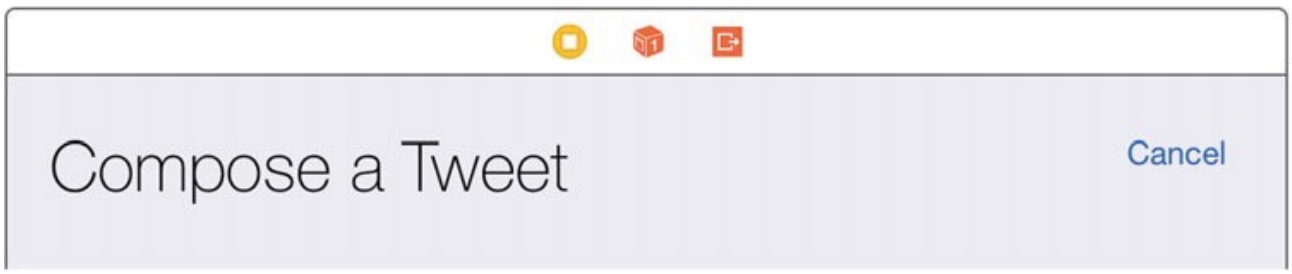

Figure 47. *The label and button in position on the Compose scene*

6. Drag in a text view and position it below the label and button. This is where the user composes posts to Twitter, so make it a decent size. Be sure to expand the text view left and right until the margin appears.

7. One downside is that by default, the text view is populated with Lorem Ipsum placeholder text. Open the Attributes Inspector and remove all the default text from the Text attribute. Your scene should be progressing nicely and resemble that shown in Figure 48.

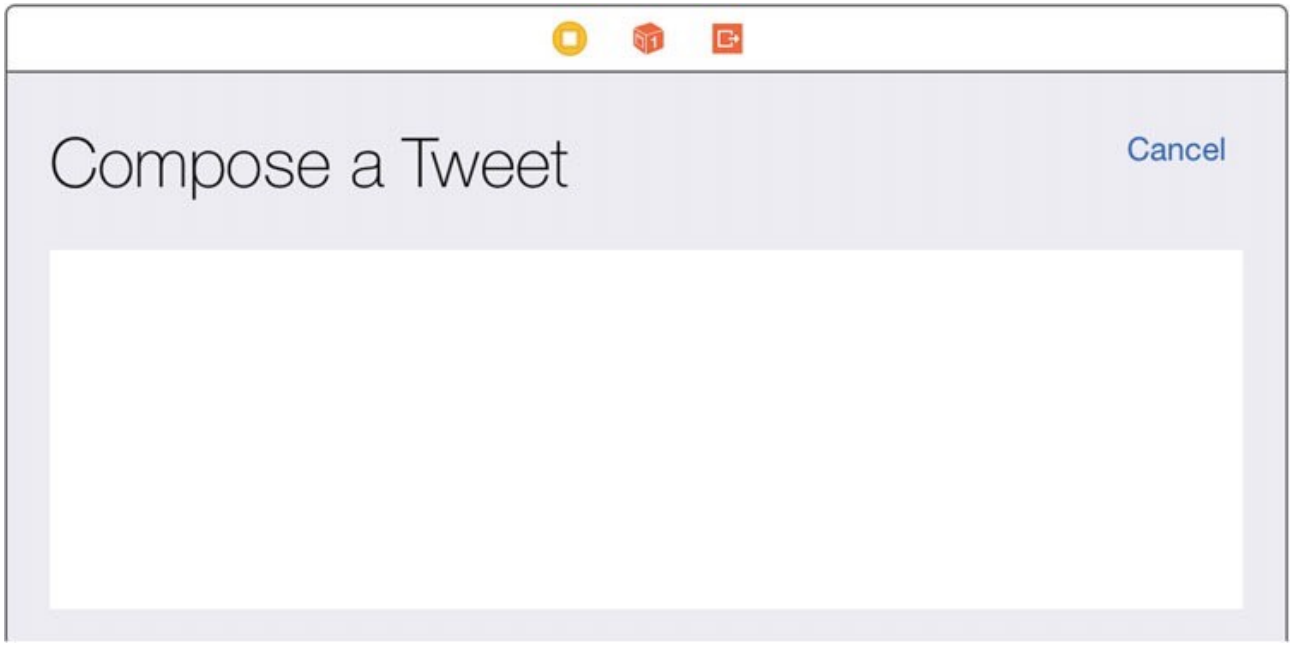

Figure 48. *The text view in position below the button and label*

8. You need to add a second button to allow the user to post the content to Twitter. Drag in a button and position it below to the text view. In the Attributes Inspector, change the Title attribute to Post.

9. Set the Background attribute of the button to White Color. Then make the button a little larger and move it toward the center of the view until the blue guideline appears and it snaps into place, centered horizontally and positioned just below the text view.

10. This is the first time you've dealt with the next object: an activity indicator. Activity indicators are very common in applications that rely on sending or receiving data from the Internet. The control produces the familiar spinning wheel that has become synonymous with data transfer over the past decade;
it has long been used with AJAX-based applications on the Web and in iOS applications since iOS 2.0. Drag one in from the Object Library and position it to the left of the Post button.

11. By default, the activity indicator is visible and static; you want it to be hidden until it's told to start animating. Xcode provides a simple attribute to achieve this: open the Attributes Inspector with the activity indicator selected and check the Hides When Stopped attribute. Figure 49 shows the finished scene.

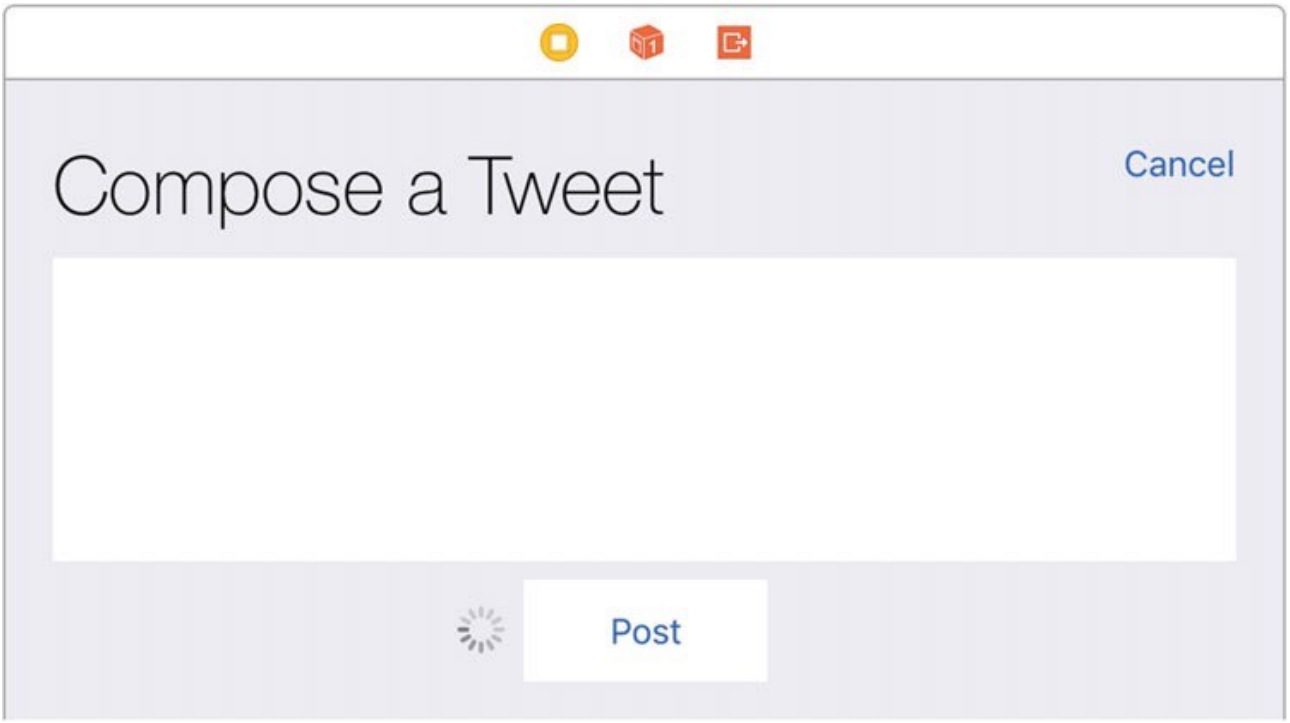

Figure 49. *The finished Compose scene*

12. With the interface elements laid out, let's fix them in place. Click Resolve Auto Layout Issues. Then, under the All Views in Compose View Controller heading, click Add Missing Constraints.

13. The text view needs to remain at a constant height, so click the text view and then click Pin. Click the Height check box and be sure the value is around the 170 mark. If you change the value, remember to use the Update Frames function to resolve any issues.

14. If you click the Post button, you see a constraint travelling from it to the bottom of the view. Click the constraint, and then, in the Attributes Editor, change Priority to 250, as shown in Figure 50. This change means iOS won't try to stretch your button down to the bottom of the view.

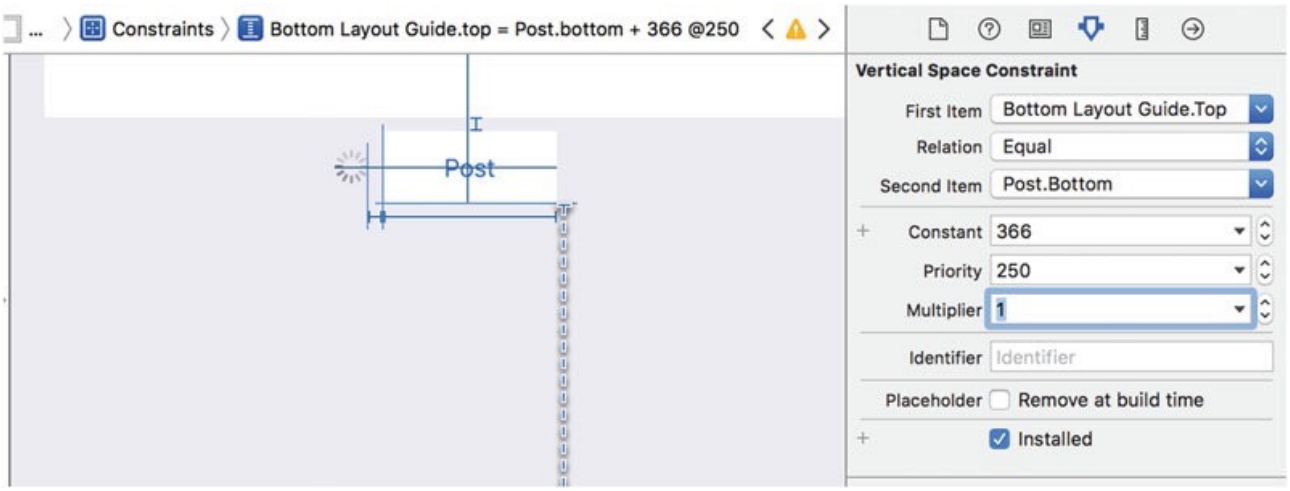

Figure 50. *Adjusting the vertical space constraint beneath the Post button*

15. There are no segues from this scene, so all that remains is to create the outlets and actions. Open the Assistant Editor and ensure that ComposeViewController. swift is showing on the

right.

16. Create an outlet for the text view called tweetContent, one for the Post button called postButton, and one for the activity indicator called postActivity. Create an action for the Cancel button called dismissView and one for the Post button called postToTwitter.

You should have the following highlighted outlets and actions in your header file:

```
class ComposeViewController: UIViewController {
    @IBOutlet weak var tweetContent: UITextView!
    @IBOutlet weak var postButton: UIButton!
    @IBOutlet weak var postActivity: UIActivityIndicatorView!
    @IBAction func dismissView(_ sender: AnyObject) {
    }
    @IBAction func postToTwitter(_ sender: AnyObject) {
    }
}
```

That's it for the fourth and final scene—you have all the elements in place for your users to compose messages and post them to Twitter.

Summary

In any project, you have to do preparation work, which is what you've done here. Further you will see the application come to life when you configure table views and take the first steps in using the Social and Accounts frameworks.

In this practice, specifically you've learned about the following:

- Organizing files in the Project Navigator using groups
- Applying custom view controller classes to view controllers in the storyboard
- Different ways of creating segues
- Types of segues
- Specifying identifiers for segues
- Embedding navigation controllers
- Using the Size Inspector to precisely position elements in the view

***