18

Tab Bars and Toolbars

Lesson 15 discussed navigation controllers, which allowed your application to present a hierarchy of views one at a time. A tab bar controller, on the other hand, allows you to display multiple view controllers at the same time (see Figure 18-1).

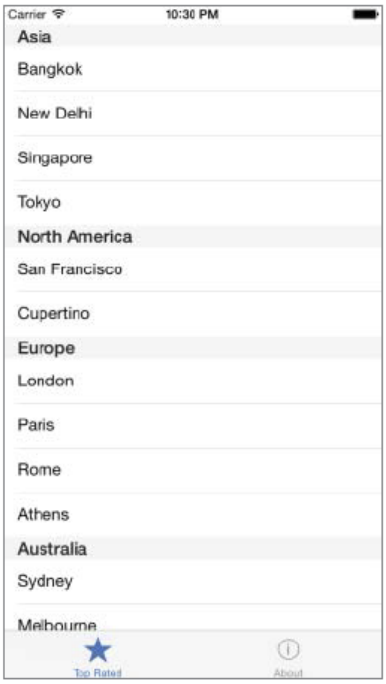
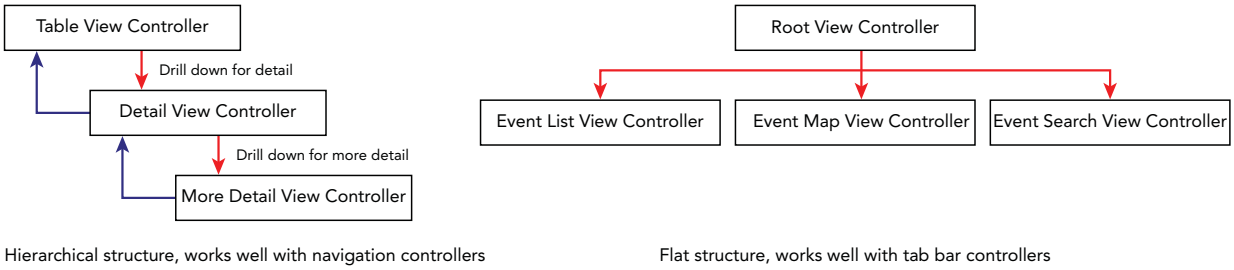


FIGURE 18-1

Navigation controllers are well suited to a hierarchical app structure, where users navigate one screen at a time to reach their destination. A tab bar controller is handy when it comes to creating a flat app structure where users can navigate directly from one primary category to another (see Figure 18-2). Examples of such apps include the Clock and the App store apps on iOS devices.



Hierarchical structure, works well with navigation controllers

Flat structure, works well with tab bar controllers

FIGURE 18-2

View controllers provide the content of tabs within a tab bar controller. It is quite common to combine tab bar controllers and navigation controllers to create an app that provides the best of both. The tab bar sits at the top of the view controller hierarchy and some of the tabs could have a navigation controller within them that provides a drill-down interface for the content within that tab. The Phone app on the iPhone is an example of such an app, with the Contacts tab containing a navigation controller that allows you to drill down into a contact's details.

A tab bar represents a single tab within a tab bar controller. Tab bars are located at the bottom of the screen and consist of an icon and text to describe the content it represents. Each tab can also have a badge, which is a red oval with a number in it (see Figure 18-3).

FIGURE 18-3

On an iPhone, a tab bar controller can only display five tabs at a time. If there are more than five, then the first four are displayed and the tab bar controller adds a More tab that reveals a list of additional tabs. The iPad can display more than five tabs because it has a larger screen.

CREATING A TAB BAR CONTROLLER

Xcode contains a template specifically for applications that want to present a tabbed interface. This template is called the Tabbed Application template and can be selected when creating a new project (see Figure 18-4).

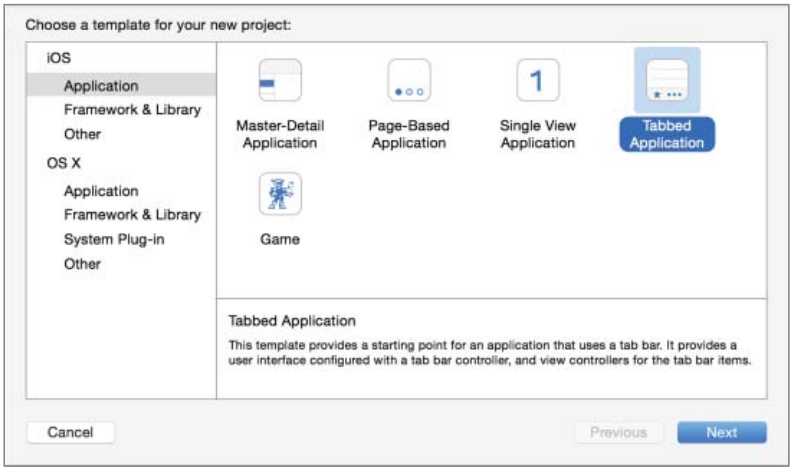


FIGURE 18-4

The template adds a tab bar controller to the default storyboard scene and configures the tab bar to present two tabs, the contents of which are provided by two view controllers, called `FirstViewController` and `SecondViewController` respectively (see Figure 18-5).

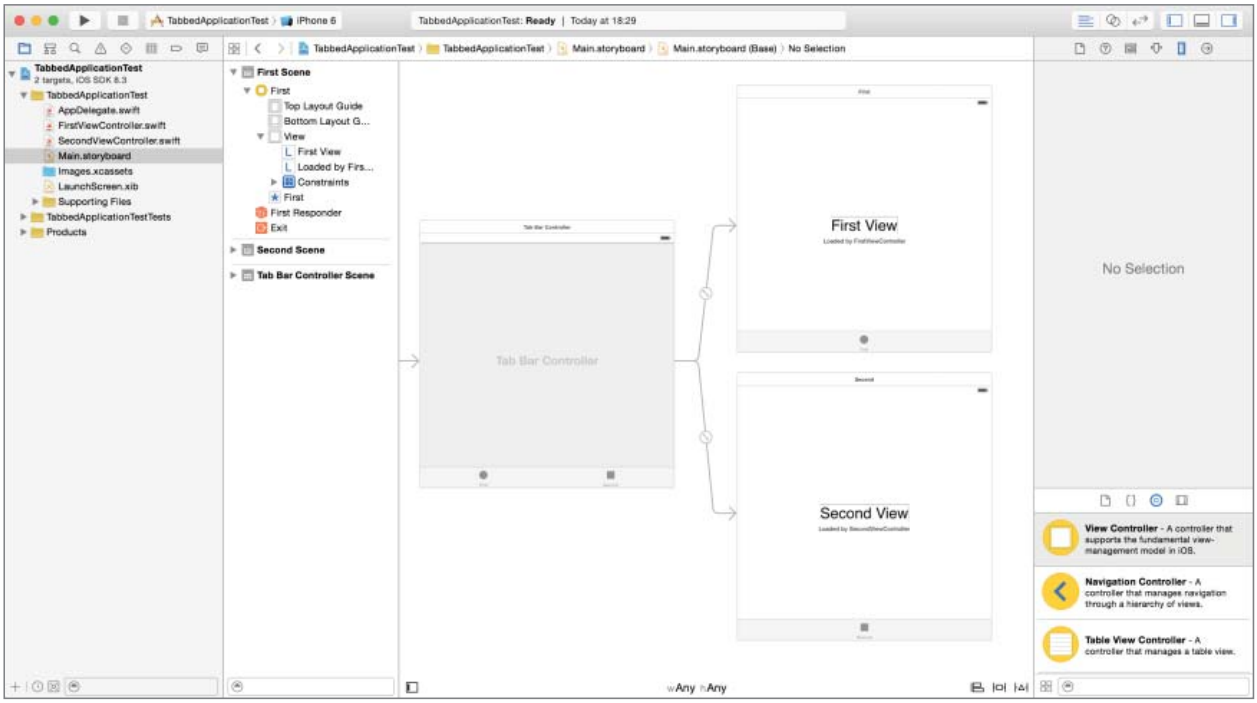


FIGURE 18-5

Inspecting the storyboard in the document outline reveals that each view controller contains a view and a tab bar item. The tab bar item is used to represent the view controller within the parent tab bar controller (see Figure 18-6).

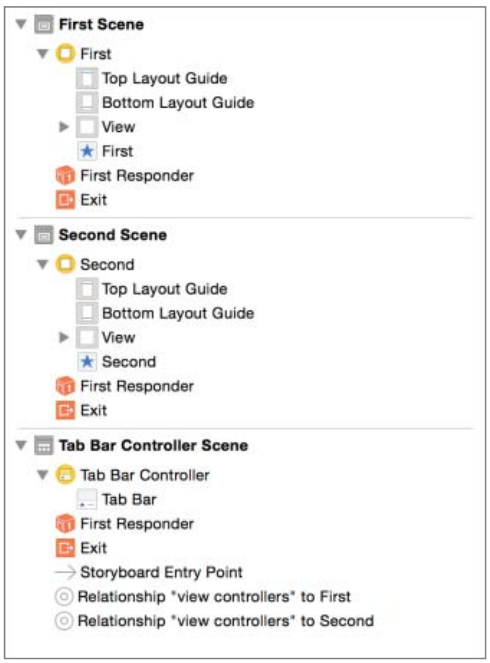


FIGURE 18-6

You can use the Attribute inspector to change the contents of the tab bar item. Apple provides a list of standard tab bar items:

- More
- Favorites
- Featured
- Top rated
- Recents
- Contacts
- History
- Bookmarks
- Search
- Downloads
- Most recent
- Most visited

You can choose one of these using the System Item drop-down combo in the Attribute inspector (see Figure 18-7).

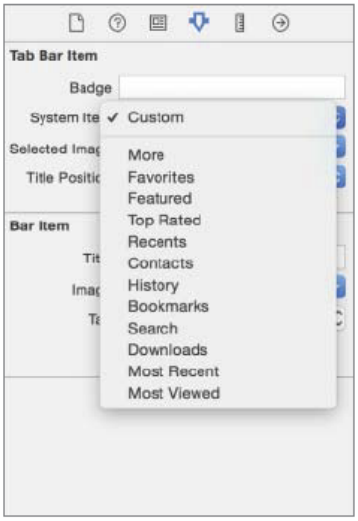


FIGURE 18-7

If you choose one of the standard tab bar items, then Xcode will provide a suitable icon and caption for you. If, however, you wish to use your own icon and caption, set the System Item to Custom. Astute readers will note that the default setting for the tab bar items created by the Tabbed Application template is Custom, with appropriate icons in the project's asset bundle.

To add a new tab to the tab bar controller, you must first add a new view controller scene to the storyboard. To do this, drag and drop a View Controller from the Object library onto the storyboard scene (see Figure 18-8).

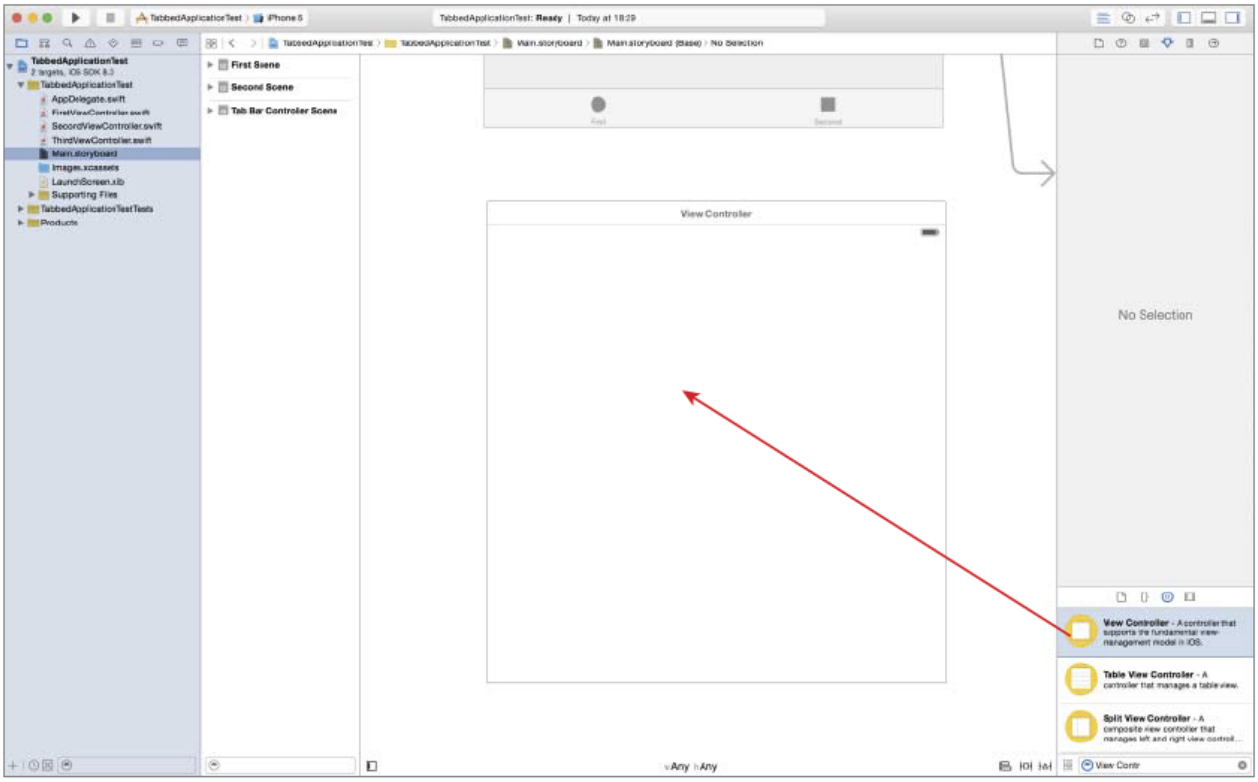


FIGURE 18-8

Next, create a new Swift class that subclasses `UIViewController` and use the Identity Inspector to associate this class with the view controller you have just dropped onto the storyboard.

To add a tab bar item to the new scene, ensure the scene is selected, and then drag and drop a Tab Bar Item from the Object library anywhere onto the scene. The tab bar item will automatically snap to the bottom of the scene regardless of where you drop it. To configure the tab bar item, simply select it and use the attribute editor.

Finally, to add the new view controller to the tab bar, simply hold down the Ctrl key on your keyboard and drag from the tab bar onto the new view controller scene (you are creating a segue). When you release the mouse button, select Relationship Segue from the popup menu (see Figure 18-9).

FIGURE 18-9

TOOLBARS

Toolbars look similar to tab bars in that both of them appear at the bottom of the screen, but the similarity ends there. A tab bar is used when you want to present multiple view controllers on the screen simultaneously. A toolbar is used to present a menu of options related to the content presented in a view controller. The two are not usually used together. The Maps application uses a toolbar to present options related to the map being displayed (see Figure 18-10). Tapping the info button brings up a modal view with options that will change the way in which data is displayed on the map.

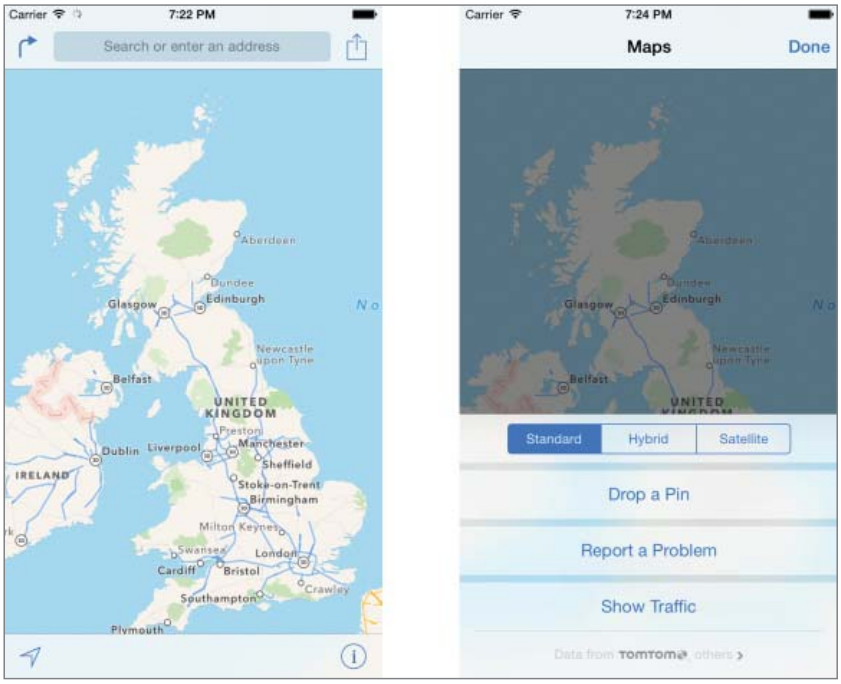


FIGURE 18-10

Typically, the buttons on a tab bar represent command functions that would be used on the current view. To add a toolbar to a view controller scene, simply drag and drop a Toolbar from the Object library. Unlike tab bar items, toolbars do not automatically snap to the bottom of the screen. You will need to provide appropriate constraints to anchor the toolbar to the bottom of the screen (see Figure 18-11).

Options within a toolbar are instances of the `UIBarButtonItem` class. You can add to the options displayed in a toolbar by dragging and dropping a Bar Button Item from the Object library (see Figure 18-12).

Configuring a bar button item is similar to configuring a tab bar item. You simply select it and bring up the Attribute inspector to change appropriate properties. Xcode provides a set of standard bar button item styles that can be selected using the Identifier drop-down combo box in the Attribute inspector (see Figure 18-13).

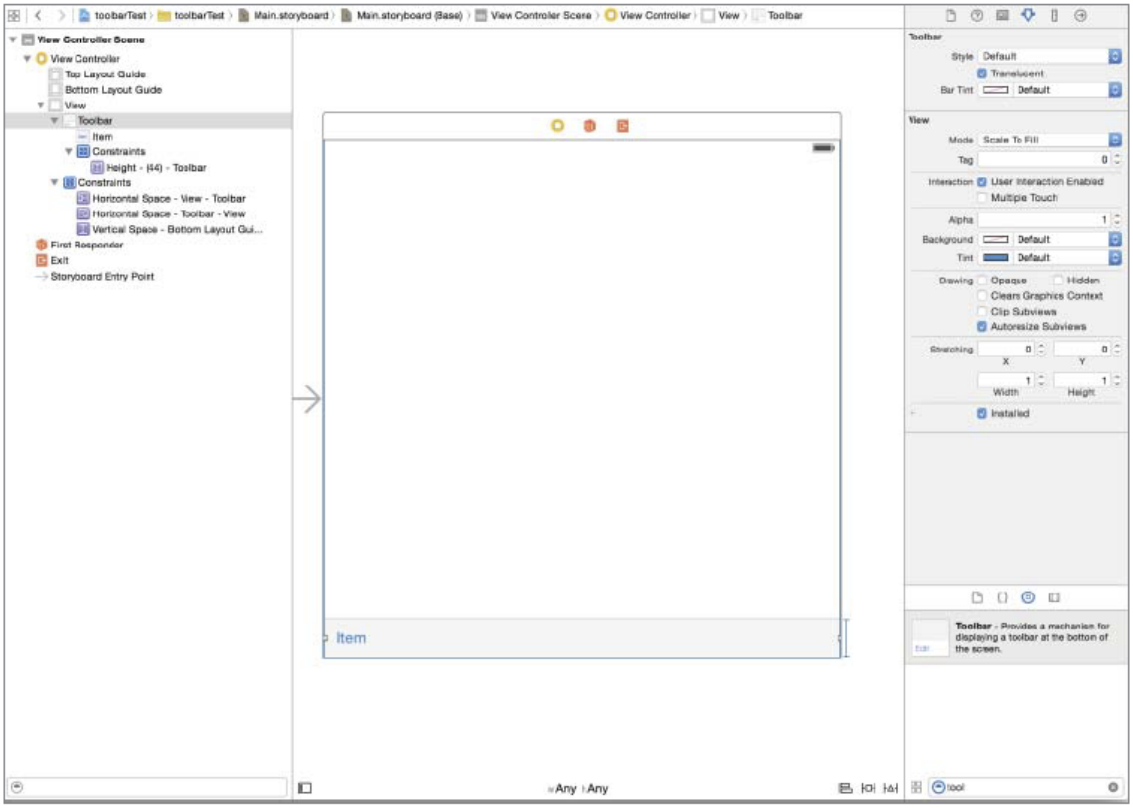


FIGURE 18-11

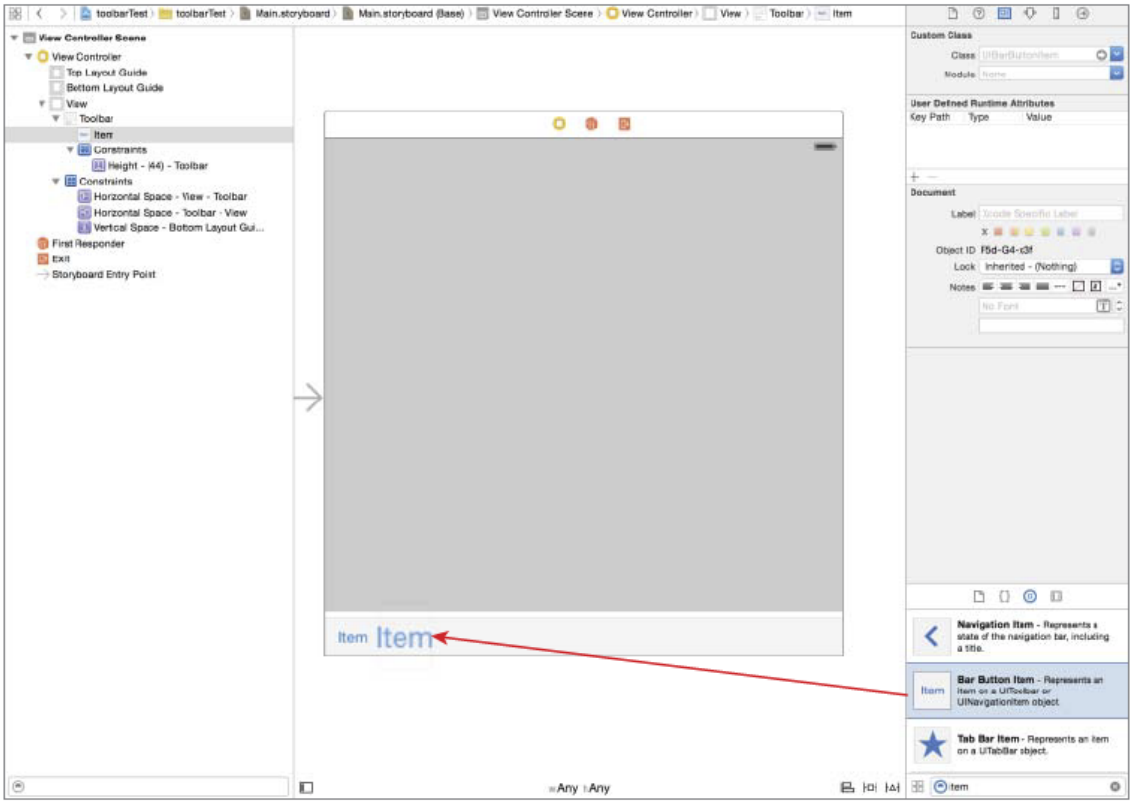


FIGURE 18-12

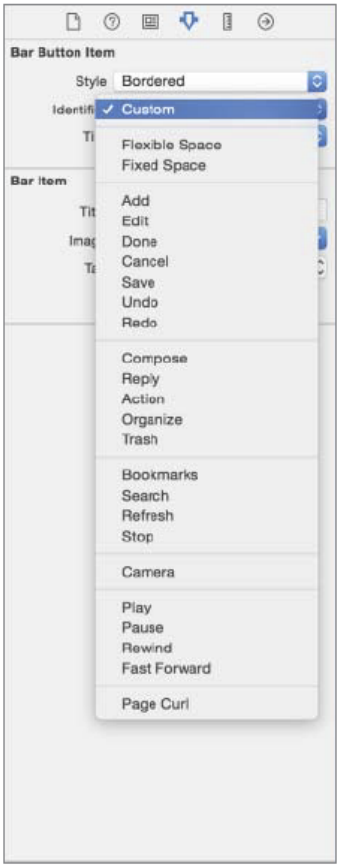


FIGURE 18-13

Two styles are worth special mention. The first is the Fixed Space style. When applied, the bar button item renders as empty space (of fixed width) between its neighboring bar button items. Figure 18-14 shows a toolbar that starts with three bar button items. The Fixed Space style is then applied to the one in the middle. Note how it changes to represent whitespace. You can edit the width of a bar button item (even fixed spaces) by using the Size inspector.

FIGURE 18-14

The second style is the Flexible Space style. The toolbar distributes the available free space across all bar button items that have this style applied to them. In a toolbar with three items, if this style were to be applied to the middle item, the neighboring items would be placed at ends of the toolbar because the width of the flexible spacebar button item would equal all the free space in the toolbar (see Figure 18-15).

FIGURE 18-15

For a toolbar with five items in which the second and fourth items are given the Flexible Space style, the remaining three items would be spaced apart evenly (see Figure 18-16) because the free space in the toolbar would be split equally between the two flexible space items.

FIGURE 18-16

Last but not least, you will need to associate action methods in your view controller class with each bar button item. To do this simply use the assistant editor to create a method and associate it with the bar button item (see Figure 18-17).

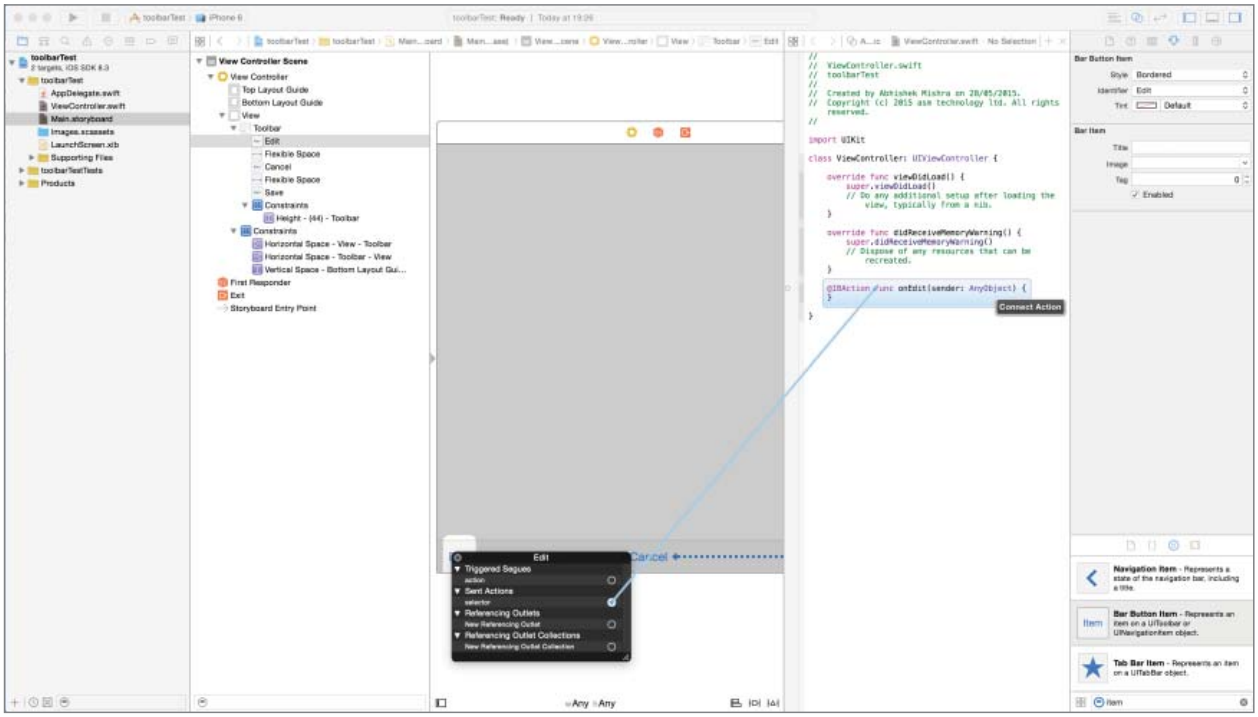


FIGURE 18-17

TRY IT

In this Try It, you create a simple application based on the Tabbed Application template, called TabbedApplication, that contains two tabs. The first tab contains a list of cities; the second tab serves as an About page for the app.

Lesson Requirements

- Launch Xcode.
- Create a new project based on the Tabbed Application template, with two tabs.
- Add a Table View to the first tab.
- Use Interface Builder to add several user interface elements to the second tab.
- Add code to the view controller class for the first tab to populate the table view.

> **REFERENCE** *The code for this Try It is available at* `www.wrox.com/go/swiftios`.

Hints

- When creating a new project, you can use your website's domain name as the Company Identifier in the Project Options dialog box.
- To show the Object library, select View ➪ Utilities ➪ Show Object Library menu item.
- To show the assistant editor, select View ➪ Assistant Editor ➪ Show Assistant Editor.

Step-by-Step

- Create a Tabbed Application in Xcode called `TabbedApplicationTest`.
  1. Launch Xcode and create a new application by selecting File ➪ New ➪ Project.
  2. Select the Tabbed Application template from the list of iOS project templates.
  3. In the project options screen use the following values:
     - Product Name: TabbedApplicationTest
     - Organization Name: your company
     - Organization Identifier: com.yourcompany
     - Language: Swift
     - Devices: iPhone
     - Include UI Tests: Unchecked
     - Include Unit Tests: Unchecked
  4. Save the project onto your hard disk.

- Add image resources to the project.
  1. Open the `Assets.xcassets` asset bundle by clicking it in the project explorer.
  2. Add a new image set called `aboutImage` by selecting Editor ➪ Add Assets ➪ New Image Set menu item.
  3. Select the new image set and use the Attribute inspector to change the value of the Scale Factors property to Single Vector.
  4. Drag and drop the `about.pdf` file from this lesson's resources folder onto the placeholder in the image set.
- Add user interface elements to the first tab.
  1. Open the `Main.storyboard` file and locate the scene called First Scene.
  2. Delete the two labels that are present on the scene. These should have the captions "First View" and "Loaded by FirstViewController" respectively.
  3. Add a `UITableView` instance to the scene using the Object library.
  4. Ensure the table view is selected and use the Pin button to display the constraints editor popup.
     - Ensure the Constrain to margins option is unchecked.
     - Pin the distance between the left edge of the view and the table view to 0.
     - Pin the distance between the right edge of the view and the table view to 0.
     - Pin the distance between the bottom of the view and the table view to 0.
     - Pin the distance between the top of the view and the table view to 20.
     - Click the Add 4 Constraints button to dismiss the constraints editor popup.
  5. Update the frames to match the constraints you have set.
     - Click on the View controller item in the dock above the storyboard scene. This is the first of the three icons located directly above the selected storyboard scene.
     - Select Editor ➪ Resolve Auto Layout Issues ➪ Update Frames.
  6. Using the assistant editor, create an outlet for the table view in the view controller class, and call the outlet `tableView`.
  7. Set up the data source and delegate properties.
     - Right-click the table view to bring up a context menu. Drag from the item labeled "dataSource" in the context menu to the item labeled "First" in the document outline.
     - Right-click the table view to bring up a context menu. Drag from the item labeled "delegate" in the context menu to the item labeled "First" in the document outline.

8. Set up the table view's appearance.
    - Select the table view and ensure the Attribute inspector is visible.
    - Ensure the Content attribute is set to Dynamic Prototypes.
    - Ensure the value of the Prototype Cells attribute is 1.
    - Ensure the Style attribute is set to Grouped.
9. Set up the prototype cell.
    - Expand the table view in the document outline; this will reveal the table view cell.
    - Select the table view cell.
    - Use the attribute editor to ensure that the value of the identifier attribute is `prototypeCell1`.
    - Ensure the Style attribute is set to Basic.

- Update the tab bar item for the first tab.
    1. Select the Tab bar item on the scene called First.
    2. Use the Attribute inspector to set the value of the System Item property to Top Rated.
- Add user interface elements to the second tab.
    1. Open the `Main.storyboard` file and locate the scene called Second Scene.
    2. Edit the contents of the "Second View" label to "City Index."
    3. Edit the contents of the "Loaded by SecondViewController" label to "Cities listed by continent."
- Update the tab bar item for the second tab.
    1. Select the Tab bar item on the scene called Second.
    2. Use the Attribute inspector to set the value of the Title property to About.
    3. Set the value of the Image attribute to `aboutImage`.
- Ensure the `FirstViewController` class implements the `UITableViewDataSource` and `UITableViewDelegate` protocols.

  Modify the declaration of the `FirstViewController` class from

```
class FirstViewController: UIViewController
```

  to

```
class FirstViewController: UIViewController,
                           UITableViewDataSource,
                           UITableViewDelegate
```

- Implement the data source and delegate methods in the view controller.

1. Add the following code snippet to the First`ViewController.swift` file to declare five arrays of strings:

```
let continents:Array<String> = ["Asia", "North America",
                                "Europe", "Australia"]

let citiesInAsia:Array<String> = ["Bangkok", "New Delhi",
                                  "Singapore", "Tokyo"]
let citiesInNorthAmerica:Array<String> = ["San Francisco","Cupertino"]
let citiesInEurope:Array<String> = ["London", "Paris", "Rome", "Athens"]
let citiesInAustralia:Array<String> = ["Sydney", "Melbourne", "Cairns"]
```

2. Implement the `numberOfSectionsInTableView` data source method as follows:

```
func numberOfSectionsInTableView(tableView: UITableView) -> Int
{
    return continents.count;
}
```

3. Implement the `numberOfRowsInSection` data source method as follows:

```
func tableView(tableView: UITableView,
numberOfRowsInSection section: Int) -> Int
{
    if section == 0
    {
        return citiesInAsia.count
    }
    else if section == 1
    {
        return citiesInNorthAmerica.count
    }
    else if section == 2
    {
        return citiesInEurope.count
    }
    else if section == 3
    {
        return citiesInAustralia.count
    }

    return 0
}
```

4. Implement the `titleForHeaderInSection` data source method as follows:

```
func tableView(tableView: UITableView,
titleForHeaderInSection section: Int) -> String?
{
    return continents[section];
}
```

5. Implement the `cellforRowAtIndexPath` data source method as follows:

```
func tableView(tableView: UITableView,
cellForRowAtIndexPath indexPath: NSIndexPath) ->
UITableViewCell
{
```

```
        let cell =
        tableView.dequeueReusableCellWithIdentifier("prototypeCell1",
        forIndexPath: indexPath)

        if indexPath.section == 0
        {
            cell.textLabel?.text = citiesInAsia[indexPath.row]
        }
        else if indexPath.section == 1
        {
            cell.textLabel?.text = citiesInNorthAmerica[indexPath.row]
        }
        else if indexPath.section == 2
        {
            cell.textLabel?.text = citiesInEurope[indexPath.row]
        }
        else if indexPath.section == 3
        {
            cell.textLabel?.text = citiesInAustralia[indexPath.row]
        }

        return cell
    }
```

- Test your app in the iOS Simulator.

  Click the Run button in the Xcode toolbar. Alternatively, you can select Project ➪ Run.

> **REFERENCE** *To see some of the examples from this lesson, watch the Lesson 18 video online at* `www.wrox.com/go/swiftiosvid`.