Lab 01: Tables in iOS

Lab Goals

The goal in this lab will be to take a comprehensive look at iOS tables and learn how to work with them. Topics covered include:

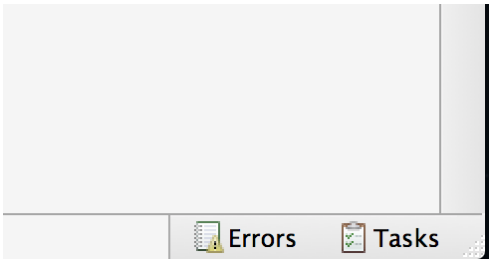
- **Table parts** – Introducing and explaining the visual elements of the `UITableView` control.
- **Displaying data in tables** – Demonstrating how to create and populate a table, how to use different table and cell styles, and how to avoid memory issues by recycling cell objects.
- **Advanced Usage** – Building custom cells and using the editing features of the `UITableView` class.

The code samples are structured to incrementally add functionality as you progress through the chapter. There are seven sample projects included in the solution:

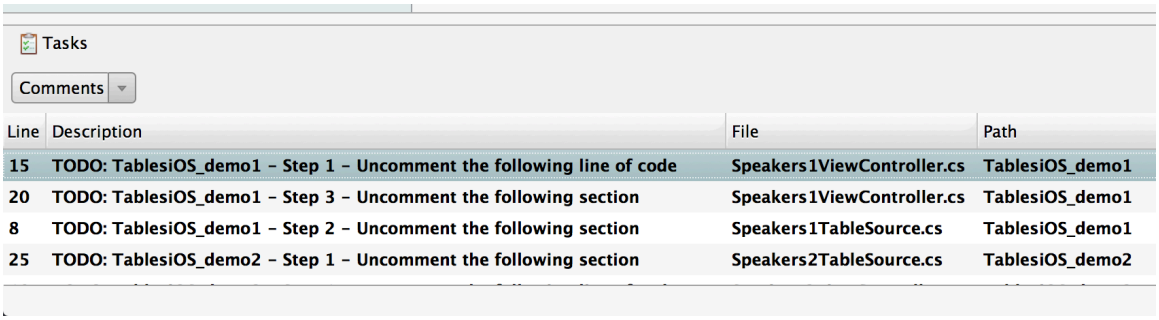
- **TablesiOS_demo1** – Implement a basic table.
- **TablesiOS_demo2** – Add an index.
- **TablesiOS_demo3** – Using Plain or Grouped style with a header.
- **TablesiOS_demo4** – Using alternate built-in cell layouts.
- **TablesiOS_demo5** – Using custom cell layouts.
- **TablesiOS_demo6** – Using tables for navigation.
- **TablesiOS_demo7** – Using table-editing features.

The code introduced in this chapter is already present in the demo projects, but it's commented out. Remember you can use *cmd + /* hot key to comment and uncomment whole sections of code at a time by selecting the lines of code and then using the hot key.

The tasks for this lab are also marked with a `// TODO:` task identifier and can be found quickly using the **Tasks** pane in Xamarin Studio without having to navigate to individual files. You may open the Tasks pane by clicking on the Tasks icon in the lower right corner of Xamarin Studio as shown below:



The Task pane will open and resemble the following:



Steps

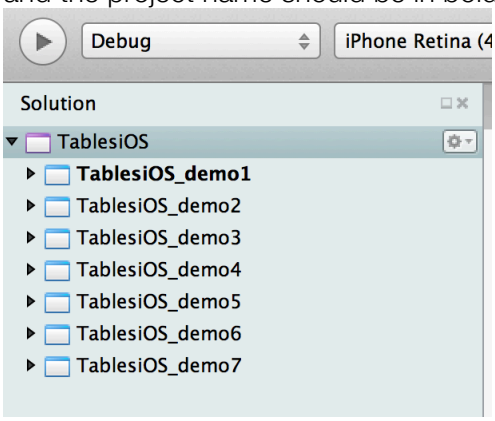
Open the Lab Environment

1. Connect to the remote environment for this lab using the instructions provided to you. Please contact your instructor, if there are any issues.
2. Launch Xamarin Studio.
3. File > Open...
4. Navigate to `/Desktop/Xamarin Studio Labs/ Tables and Collection Views in iOS/Lab 01 resources/Lab 01 Start/` and double-click on the TablesiOS solution.

This solution has 7 demo projects we'll be using to learn about Table Views in iOS. All of the code is already present in the solution, but some of the key parts have been commented out so we can try different options during the labs. In this lab we'll take the approach of doing the lab steps first so we can see the result in the iOS Simulator and then we'll go over the code together.

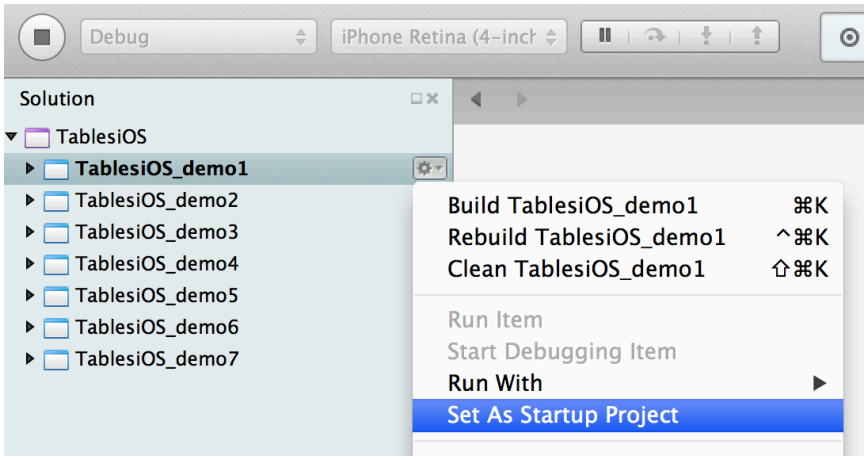
TablesiOS_demo1 – Implement a basic table

1. The project TablesiOS_demo1 should be setup as the *Startup Project* and the project name should be in bold type as shown below:

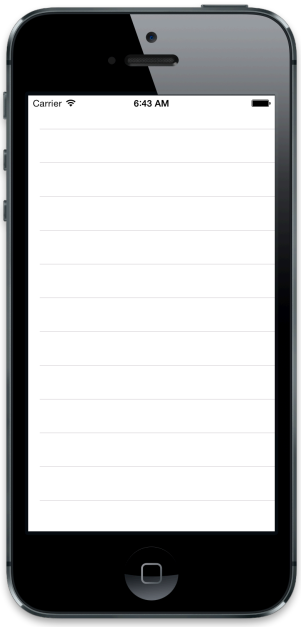


If the TablesiOS_demo1 project does not appear in bold type:

a. Select **TablesiOS_demo1** and choose **Set As Startup Project** from the **Action Button** which will appear to the right of the **TablesiOS_demo1** when selected as shown below:



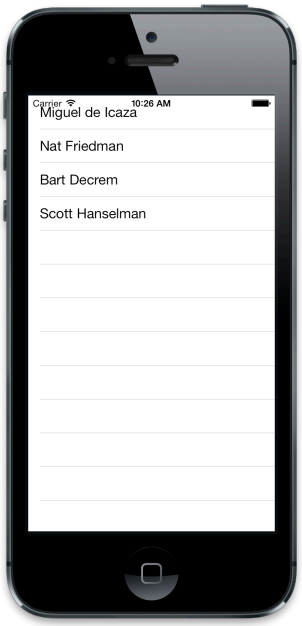
2. Expand the **TablesiOS_demo1** project.
3. Click the **Play** button to build and run the project. The iOS Simulator will launch and display a basic table with empty rows. The `UITableViewCellStyle` is set to `Default` as shown below:



4. Expand the **TablesiOS_demo1** project.
5. We'll start wiring up the data source by creating a new instance of `SpeakersTableSource` and passing in the lists of speakers. Double-click on **Speakers1ViewController.cs** and uncomment the line of the code under the comment `//TODO: Step 1a: uncomment to set SpeakersTableSource as the TableView Source` using the cmd+/ hot key:

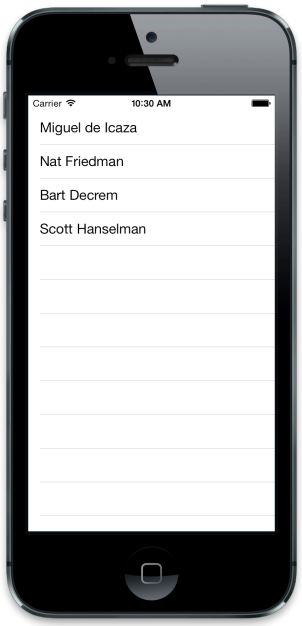
```
//  TableView.Source = new speakersTableSource(items);
```

6. Next, we'll uncomment the code implementing the SpeakersTableSource class. Double-click on **Speakers1TableSource.cs** and uncomment all of the code under the comment `//TODO: Step 1b: - Uncomment to implement SpeakersTableSource` by selecting the lines of code you wish to uncomment and using the cmd+/ hot key.
7. File > Save All
8. Click the **Play** button to build and run the project. The iOS Simulator will launch and display a basic table with 4 rows of data using the **Default** table style as shown below:



Everything should look good except the iPhone's **Status Bar** is overlapping the first row of our table. The table view in iOS 7extends under the **Status Bar** so that we can see rows passing underneath it. We need to adjust our table so the first row displays correctly.

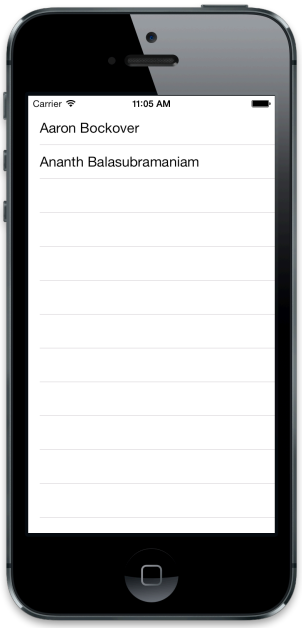
9. Press the **Stop** button to halt the iOS Simulator.
10. We'll fix this issue by having the table view display the first row with an offset from the top of the view using the code below. Uncomment the `ViewDidLayoutSubviews` method under the comment `//TODO: Step 1c: uncomment to fix overlap of first row by status bar for iOS 7.`
11. File > Save All
12. Click the **Play** button to build and run the project. The iOS Simulator will launch and display a basic table with 4 rows of data using the **Default** table style as shown below:



13. Let's test out the new iOS 7 scrolling behavior. Clicking somewhere inside the table area, drag upwards and the speaker rows should slide up under the transparent **Status Bar**.
14. Press the **Stop** button to halt the iOS Simulator.
15. Close all tabs displaying code for **TablesiOS_demo1** and collapse the project.

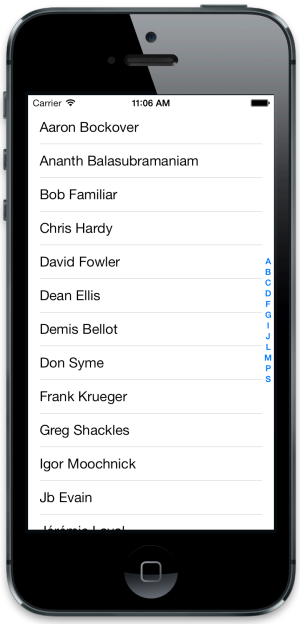
TablesiOS_demo2 – Add an index

1. Select **TablesiOS_demo2** and choose **Set As Startup Project** from the **Action Button** that will appear to the right of the **TablesiOS_demo2** when selected.
2. Expand the **TablesiOS_demo2** project.
3. Click the **Play** button to build and run the project. The iOS Simulator will launch and display a basic table with 2 rows of data representing the speakers which will be in our first index as shown below:



Only data from the first index is showing in this example.

4. The data we need for the rows in each section and to create the index is already setup for us in the lab. We need to uncomment two overrides to configure the table to use that information to display the table. Double-click on **Speakers2TableSource.cs** and uncomment all of the code under `//TODO: Step 2a: uncomment to implement NumberOfSections and SectionIndexTitles.`
5. File > Save All
6. Click the **Play** button to build and run the project. The iOS Simulator will launch and display a basic table with an index using the **Default** table style as shown below:

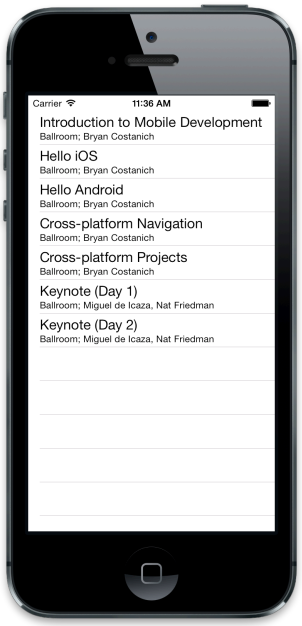


We added the code to organize the data rows alphabetically and to create an index that can be seen on the right side of the table view.

7. Press the **Stop** button to halt the iOS Simulator.
8. Close all tabs displaying code for **TablesiOS_demo2** and collapse the project.

TablesiOS_demo3 – Using Plain or Grouped style with a Header

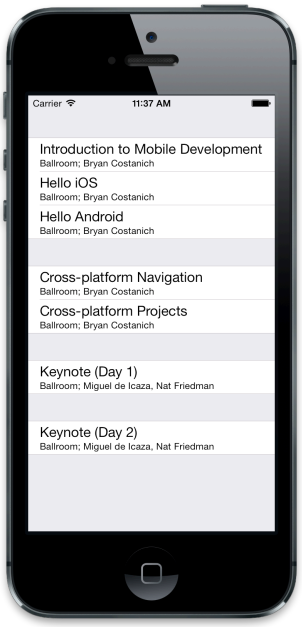
1. Select **TablesiOS_demo3** and choose **Set As Startup Project** from the **Action Button** that will appear to the right of the **TablesiOS_demo3** when selected.
2. Expand the **TablesiOS_demo3** project.
3. Click the **Play** button to build and run the project. The iOS Simulator will launch and display a **Plain Style** table with 7 rows of data representing sessions from Xamarin's Evolve event. Each row of data has a **Title** holding the session name and a **Subtitle** with the event location and the speaker's name. The `UITableViewCellStyle` is set to `Subtilte`. We'll cover the 4 available cell styles in the next part of the lab. The table should resemble the following:



4. Let's configure our table to use a **Grouped Style**. Double-click on **Sessions3ViewController.cs** and uncomment the following line of code under the comment `// TODO: Step 3a: uncomment to set the table style to Grouped.`

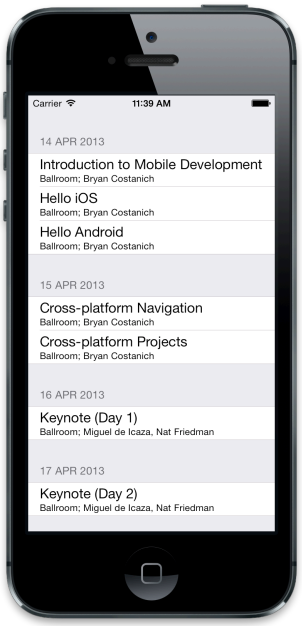
```
// TableView = new UITableView(Rectangle.Empty, UITableViewStyle.Grouped);
```

5. File > Save All
6. Click the **Play** button to build and run the project. The iOS Simulator will launch and display a **Grouped Style** table with the `UITableViewCellStyle` set to `Subtitle` as shown below:



7. Press the **Stop** button to halt the iOS Simulator.

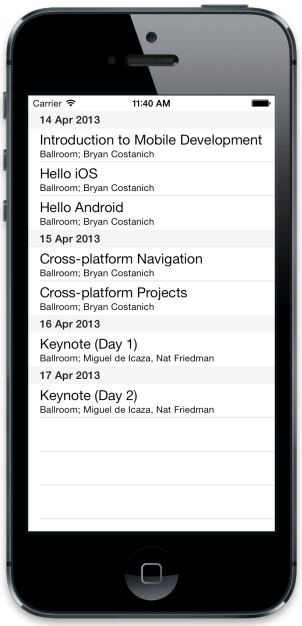
8. Let's add a header to each section of our table. Double-click on **Sessions3TableSource.cs** and uncomment the code for the `TitleForHeader` method under the comment `//TODO: Step 3b: uncomment to add a title to the header over each section.`
9. File > Save All
10. Click the **Play** button to build and run the project. The iOS Simulator will launch and display a **Grouped Style** table with the `UITableViewCellStyle` set to `Subtitle` and a **Header** for each section containing the date for the events being held on that day as shown below:



11. Press the **Stop** button to halt the iOS Simulator.
12. Let's set our table to display a header using the **Plain Style**. Double-click on **Sessions3ViewController.cs** and *comment out* the following line of code to default back to the Plain table view style under the comment `//TODO: Step 3a: uncomment to set the table style to Grouped:`

```
//      TableView = new UITableView(Rectangle.Empty, UITableViewStyle.Grouped);
```

1. File > Save All
2. Click the **Play** button to build and run the project. The iOS Simulator will launch and display a **Plain Style** table with the `UITableViewCellStyle` set to `Subtitle` and a **Header** for each section containing the date for the events being held on that day as shown below:

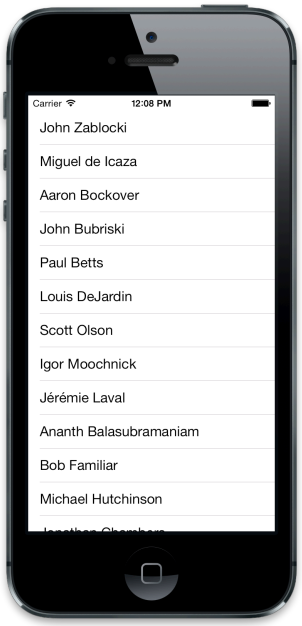


3. Press the **Stop** button to halt the iOS Simulator.
4. Close all tabs displaying code for **TablesiOS_demo3** and collapse the project.

TablesiOS_demo4 – Using alternate built-in cell layouts

Options for the Default Table View Cell Style

1. Select **TablesiOS_demo4** and choose **Set As Startup Project** from the **Action Button** that will appear to the right of the **TablesiOS_demo4** when selected.
2. Expand the **TablesiOS_demo4** project.
3. Click the **Play** button to build and run the project. The iOS Simulator will launch and display a basic table using with many rows of data representing the speakers from Xamarin's Evolve event. The screen should resemble the following:



4. Test the scrolling behavior of the table in the iOS Simulator by holding your finger down while dragging up or down.
5. Press the **Stop** button to halt the iOS Simulator.
6. Let's explore the built-in cell styles. Double-click on **Speakers4TableSource.cs** and uncomment the line of code under the comment that sets the `DetailedTextLabel` property. `//TODO: Step 4a: uncomment to set the cell's DetailTextLabel.`

```
//   cell.DetailTextLabel.Text = speaker.Company;
```

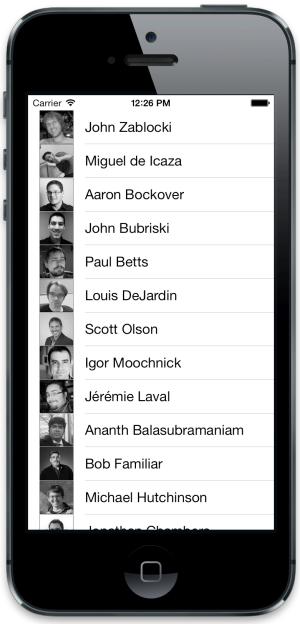
7. **File** > **Save All**
8. Click the **Play** button to build and run the project. The iOS Simulator will launch and it will crash since the Default `UITableViewCellStyle` does not support the `DetailedTextLabel` property.
9. Let's fix it. Comment out:

```
cell.DetailTextLabel.Text = speaker.Company;
```

10. Let's add an image to our Default cell style. Double-click on **Speakers4TableSource.cs** and uncomment the line of code under the comment `//TODO: Step 4b: uncomment to set the cell's ImageView` to set ImageView for the cell.

```
//   cell.ImageView.Image = UIImage.FromBundle(speaker.HeadshotUrl);
```

11. **File > Save All**
12. Click the **Play** button to build and run the project. The iOS Simulator will launch and display a basic table representing the speakers from Xamarin's Evolve event. Each row will have the speaker's name and image. The screen should resemble the following:



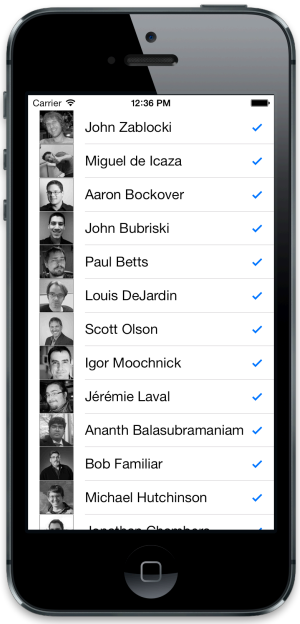
13. Press the **Stop** button to halt the iOS Simulator.

Options for the Default Table View Cell Style - Accessories

1. Let's add a **Checkmark** accessory to our rows. Uncomment the following line of code under the comment `//TODO: Step 4c: change the cell's Accessory according to the lab instructions.`

```
// cell.Accessory = UITableViewCellAccessory.Checkmark;
```

2. **File** > **Save All**
3. Click the **Play** button to build and run the project. The **Checkmark** accessory should now be displayed as shown below:



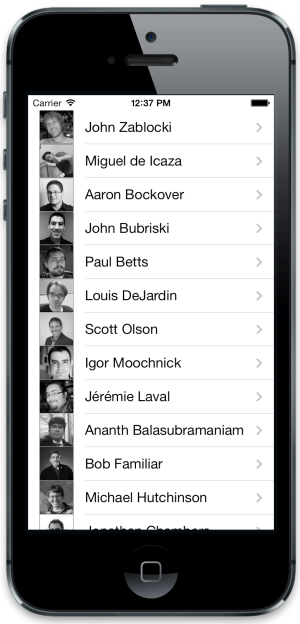
4. Press the **Stop** button to halt the iOS Simulator.
5. Let's change the code to display the **DisclosureIndicator** accessory.

   Comment out the following line to remove the **Checkmark** accessory:

   ```
   // cell.Accessory = UITableViewCellAccessory.Checkmark;
   ```

   Uncomment the following line to display the **DisclosureIndicator** accessory:

   ```
   // cell.Accessory = UITableViewCellAccessory.DisclosureIndicator;
   ```
6. **File** > **Save All**
7. Click the **Play** button to build and run the project and the **DisclosureIndicator** accessory should now be displayed as shown below:



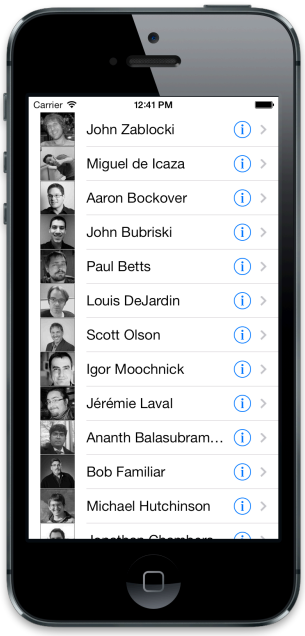
8. Press the **Stop** button to halt the iOS Simulator.
9. Let's change the code to display the **DetailDisclosureButton** accessory.

   Comment out the following line to remove the **DisclosureIndicator** accessory:

   ```
   // cell.Accessory = UITableViewCellAccessory.DisclosureIndicator;
   ```

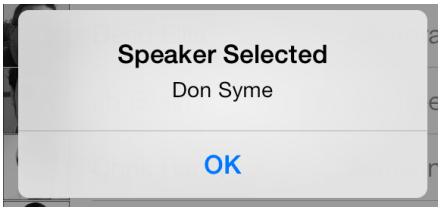
   Uncomment the following line to display the **DetailDisclosureButton** accessory:

   ```
   // cell.Accessory = UITableViewCellAccessory.DetailDisclosureButton;
   ```
10. **File** > **Save All**
11. Click the **Play** button to build and run the project. The accessory **DetailDisclosureButton** should now be displayed as shown below:

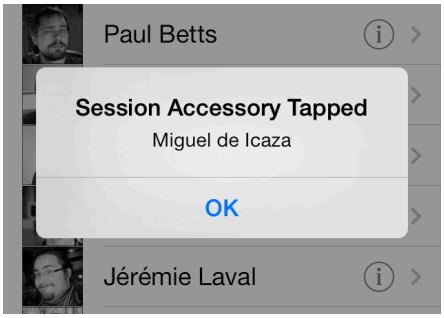


The row already has a 'tap' event wired up to it in this lab. The **DetailDisclosureButton** allows us to wire up an additional 'tap' event.

12. Click on a row and a an alert dialog will appear with the name of the speaker from the row you selected and will resemble the following:



13. Try clicking on a **DetailDisclosureButton** , it will "**flash**" but nothing else happens since we have not yet wired it up.
14. Press the **Stop** button to halt the iOS Simulator.
15. Let's wire up the **DetailDisclosureButton** to add an alert dialog to demonstrate the difference between a row tap and a **DetailDisclosureButton** tap. Uncomment the section of code under the comment `//TODO: Step 4d: uncomment to wire up action to DetailDisclosureButton.`
16. **File** > **Save All**
17. Click the **Play** button to build and run the project. The accessory **DetailDisclosureButton** should be displayed as before. Press the **DetailDisclosureButton** and an **Alert Dialog** will pop up which will resemble the following:



18. Click the **OK** button to dismiss the alert dialog.
19. Press the **Stop** button to halt the iOS Simulator.

Table View Cell Styles

1. Let's explore the other 3 built-in cell styles. We need to uncomment the line setting `DetailTextLabel` since the 3 other built-in table view cell styles use it. This line of code caused the crash earlier in lab 4 since the **Default** table view cell style does not support this property. Uncomment the following line of code under the comment `//TODO: Step 4a: uncomment to set the cell's DetailTextLabel.`

```
//   cell.DetailTextLabel.Text = speaker.Company;
```

2. Locate the comment `//TODO: Step 4e: change the UITableViewCellStyle according to the lab instructions` and follow the instructions below to explore the 3 other built-in table style cell styles.

Options for the Subtitle Table View Cell Style

3. Let's change the table view cell style to `Subtitle`.

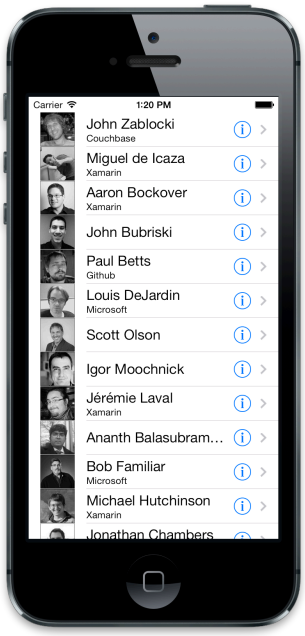
   Comment out the following line of code to remove the **Default** table view cell style:

```
cell = new UITableViewCell (UITableViewCellStyle.Default, speakerCellId);
```

   Uncomment the following line of code to set the **Subtitle** table view cell style:

```
// cell = new UITableViewCell (UITableViewCellStyle.Subtitle, speakerCellId);
```

4. **File** > **Save All**
5. Click the **Play** button to build and run the project. The `UITableViewCellStyle` should display a table using the **Subtitle** style for each row in the table with the speaker's company name added as the **DetailTextLabel** as shown below:



The **image** and **DetailDisclosureButton** are optional and were enabled in prior steps of the lab.

6. Press the **Stop** button to halt the iOS Simulator.

Options for the Value1 Table View Cell Style

1. Let's change the table view cell style to `Value1`.

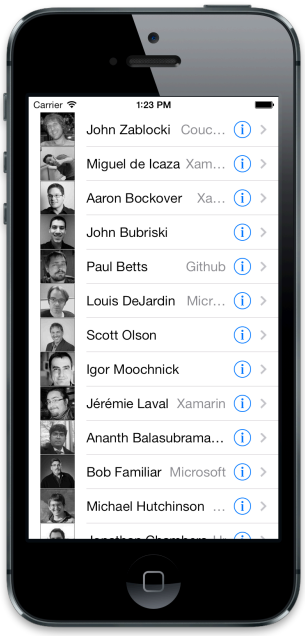
   Comment out the following line of code to remove the **Subtilte** table view cell style:

```
cell = new UITableViewCell (UITableViewCellStyle.Subtitle, speakerCellId);
```

2. Uncomment the following line of code to set the **Value1** table view cell style:

```
// cell = new UITableViewCell (UITableViewCellStyle.Value1, speakerCellId);
```

3. **File** > **Save All**
4. Click the **Play** button to build and run the project. The `UITableViewCellStyle` should display a table using **Value1** style row with the speaker's company name as shown below:

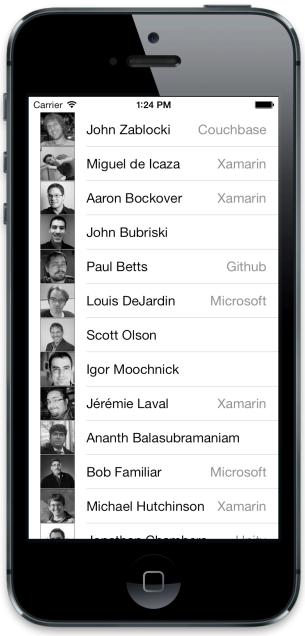


Notice the company names are truncated because there is not enough room to display them in the row represented by the table cell.

5. Press the **Stop** button to halt the iOS Simulator.
6. Let's try commenting out the DetailDisclosureButton. Locate the comment `//TODO: Step 4c: change the cell's Accessory according to the lab instructions` and comment out the following line of code:

```
//   cell.Accessory = UITableViewCellAccessory.DetailDisclosureButton; //...
```

7. File > **Save All**
8. Click the **Play** button to build and run the project. The `UITableViewCellStyle` should display a table using **Value1** style rows with the speaker's company name as shown below:



Notice the company names are NOT truncated because deleting the DetailDisclosureButton left enough room to display them.

Options for the Value2 Table View Cell Style

1. Let's change the table view cell style to `Value2`.

   Comment out the following line of code to remove the **Value1** table view cell style:

```
cell = new UITableViewCell (UITableViewCellStyle.Value1, speakerCellId);
```

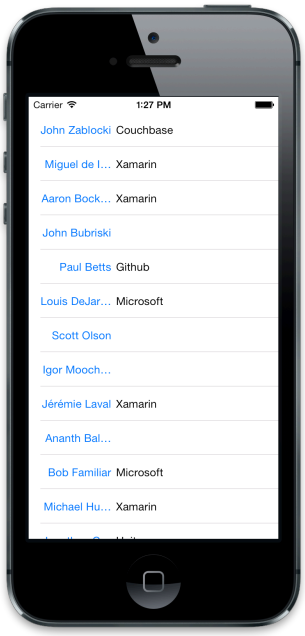
   Uncomment the following line of code to set the **Value2** table view cell style:

```
// cell = new UITableViewCell (UITableViewCellStyle.Value2, speakerCellId);
```

2. We'll also have to comment out the code setting the **ImageView** for the cell since **Value2** doesn't support an image. Comment out the following line of code under the comment `//TODO: Step 4b: uncomment to set the cell's ImageView.`

```
cell.ImageView.Image = UIImage.FromBundle(speaker.HeadshotUrl);
```

1. **File > Save All**
2. Click the **Play** button to build and run the project. The `UITableViewCellStyle` should display a table using **Value2** style row with the speaker's company name as shown below:



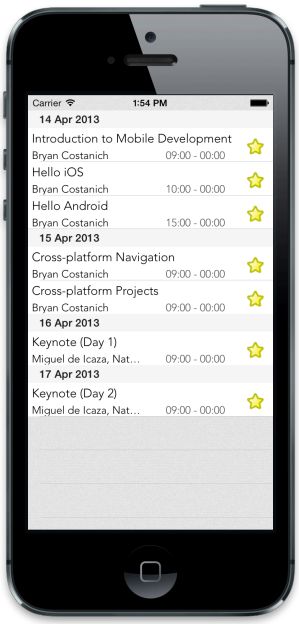
The speaker names are truncated using this style in this example.

3. Press the **Stop** button to halt the iOS Simulator.
4. Close all tabs displaying code for **TablesiOS_demo4** and collapse the project.

TablesiOS_demo5 – Using custom cell layouts

1. Select **TablesiOS_demo5** and choose **Set As Startup Project** from the **Action Button** that will appear to the right of the **TablesiOS_demo5** when selected.
2. Expand the **TablesiOS_demo5** project.
3. Click the **Play** button to build and run the project. The iOS Simulator will launch and display a table representing sessions from Xamarin's Evolve event organized using **Headers** with the date for each day's sessions.
4. Press the **Stop** button to halt the iOS Simulator.
5. Let's implement a table with custom a custom cell layout. First we need to build a custom cell – open the **SessionCell.cs** file or navigate to `//TODO: Step5a: This custom cell has been prebuilt` to review a sample custom cell.
    a. The constructor of this class creates all the user interface controls that will appear in the cell.
    b. The LayoutSubviews method assigns data to the controls and sets their position.
    c. The Session property is used to tell the class what data needs to be displayed.

6. Then we need to remove the code defining the **Default** table view cell style. Double-click on **Sessions5TableSource.cs** and delete the code above the comment `//TODO: Step 5b: uncomment to use our custom table cell for the table (delete above)`
7. Next, uncomment the code below this comment `//TODO: Step 5b: uncomment to use our custom table cell for the table (delete above`. This code will now create a new instance of our custom cell class instead of instantiating one of the built-in cell styles.
8. **File > Save All**
9. Click the **Play** button to build and run the project. The iOS Simulator will launch and should display a table with a custom cell layout which will resemble the following screen:

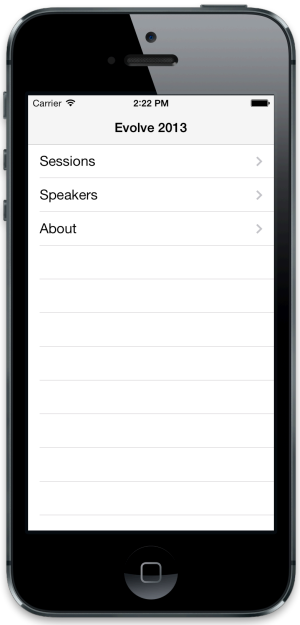


10. Press the **Stop** button to halt the iOS Simulator.
11. Close all tabs displaying code for **TablesiOS_demo5** and collapse the project.

TablesiOS_demo6 – Using tables for navigation

1. Select **TablesiOS_demo6** and choose **Set As Startup Project** from the **Action Button** which will appear to the right of the **TablesiOS_demo6** when selected.
2. Expand the **TablesiOS_demo6** project.
3. Click the **Play** button to build and run the project. The iOS Simulator will launch and display a basic table embedded inside a `UINavigationController` with 3 rows of data representing the **Sessions**, **Speakers**, and the **About** screen from Xamarin's Evolve event.

4. Try clicking on any of the rows. We haven't wired up any of the rows to navigate to another view controller, so we don't expect anything to happen yet.
5. Before we do any coding, review AppDelegate.cs and examine the code under `//TODO: Step 6a: see how the NavigationController is used`. We have introduced a new class – the UINavigationController – as a container for our other view controllers.
6. Let's wire up each of these table rows to provide us with navigation to a view controller with more detailed information associated with the row that has been selected. Double-click on **MenuTableViewController.cs** and uncomment the `RowSelected ()` method under the comment `//TODO: Step 6b: uncomment to enable navigation to the session detail screen.`
7. Click the **Play** button to build and run the project. The iOS Simulator will launch and display our root view controller as shown below:

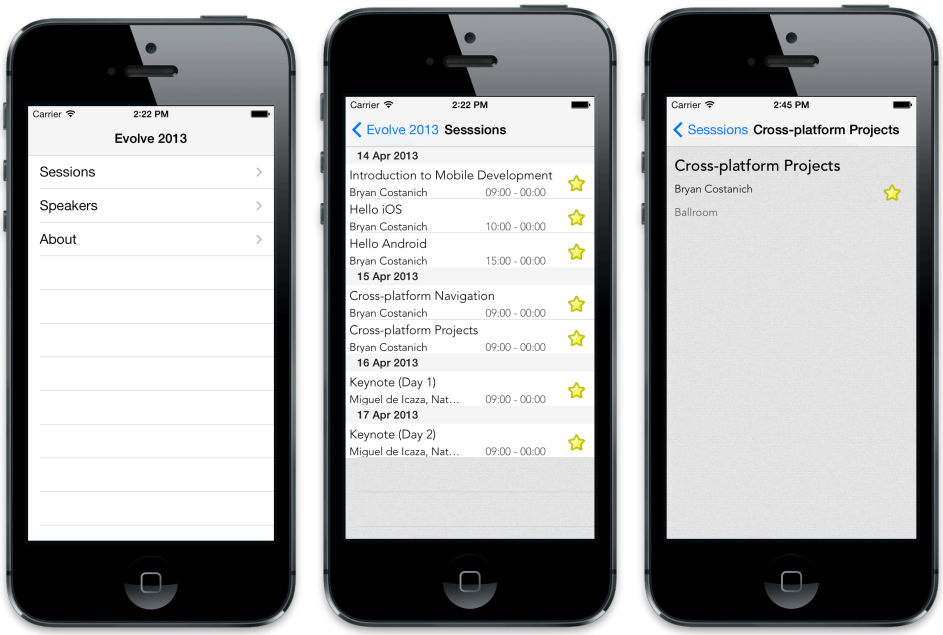


8. Each row of the table is now wired to launch a view controller with more detailed information associated with the topic specified in the row. Try clicking on each row (**Sessions**, **Speakers**, and **About**) to verify everything is working properly.
9. When the **Sessions row** is tapped a new instance of the **SessionsViewController** is instantiated and pushed on to the **Navigation Controller's** stack. Let's take that one level deeper by wiring up each row within the **SessionsViewController** to create a new instance of **SessionViewController** that displays information specific to the session row that was tapped in the **SessionsViewController**. We'll do that by wiring up each row in our **SessionsViewController** to push a new instance of **SessionViewController** on to the **Navigation Controller's** stack, passing it detailed information about the session

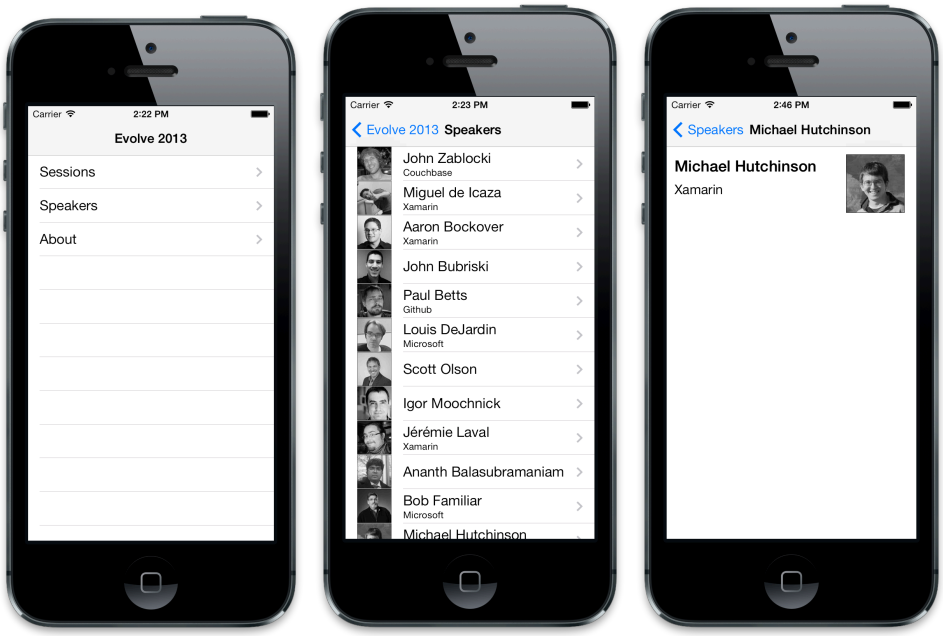
selected. Double-click on **Sessions6TableSource.cs** and uncomment the `RowSelected ()` method under the comment `//TODO: Step 6c: uncomment to enable navigation to the session detail screen.`

10. We'll do the something very similar for the SpeakersViewController. Double-click on **Speakers6TableSource.cs** and uncomment the `RowSelected ()` method under the comment `//TODO: Step 6c: uncomment to enable navigation to the speaker detail screen.`
11. The **About** row in our root view controller launches an html page and does not require any additional navigation.
12. The setup for our navigation is now complete. Click the **Play** button to build and run the project. The iOS Simulator will launch and display our 3 rows of data representing the **Sessions**, **Speakers**, and the **About** screen.
13. Explore the navigation in the app using the following screen shots as a guide:

Sessions Screen Shots



Speakers Screen Shots

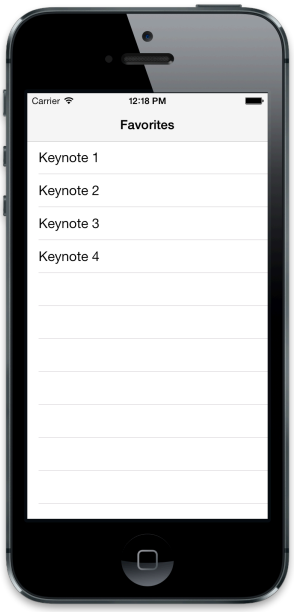


About Screen Shots

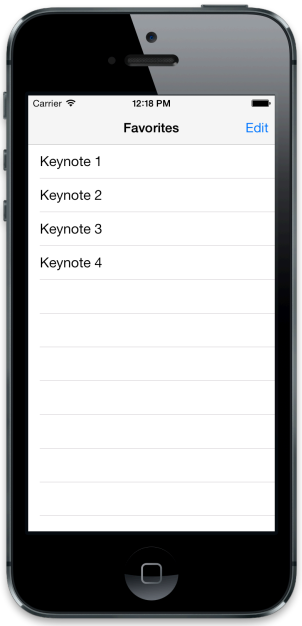
14. Press the **Stop** button to halt the iOS Simulator.
15. Close all tabs displaying code for **TablesiOS_demo6** and collapse the project.

TablesiOS_demo7 – Using table-editing features

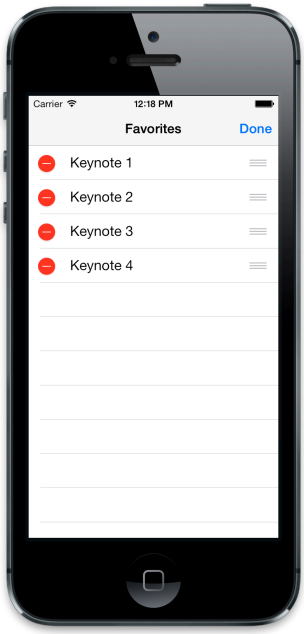
1. Select **TablesiOS_demo7** and choose **Set As Startup Project** from the **Action Button** that will appear to the right of the **TablesiOS_demo7** when selected.
2. Expand the **TablesiOS_demo7** project.
3. Click the **Play** button to build and run the project. The iOS Simulator will launch and display a basic table embedded inside a `UINavigationController` with 4 rows of string data as shown below:



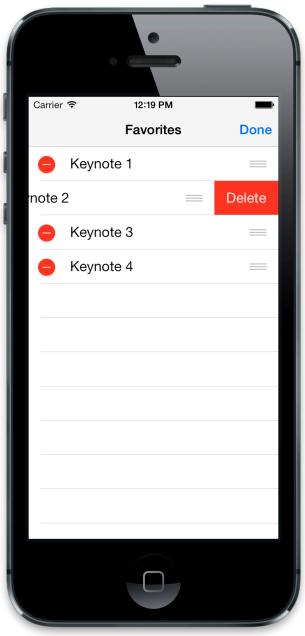
4. Let's setup the editing controls for this table. First, we'll add the **Edit** button to the **Navigation Bar**. Double-click on **FavoritesViewController.cs** and uncomment the line of code under under the comment `//TODO: Step 7a: uncomment to create the Edit button in the Navigation Bar.`
5. Next, we'll need to add the code to adjust the model when we delete or insert code. Uncomment the `commitEditingStyle ()` method under the comment `//TODO: Step 7b: uncomment to enable editing within the table.`
6. Click the **Play** button to build and run the project. The iOS Simulator will launch and display our new **Edit** button in the **Navigation Bar** as shown below:



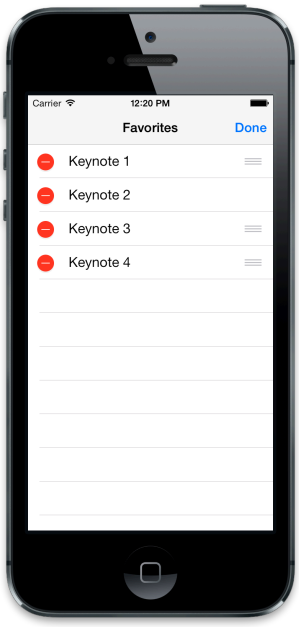
7. Click the **Edit** button in the upper right corner of our app. Our app has been configured to allow moving rows within the table as well as deleting individual rows as shown below:



8. Drag one of the **Row Reorder** controls on the right side of a row and drag it up or down to move a row.
9. Click on the **Delete Button Control** next to one of the rows. The resulting screen displays a **Delete Button** that allows you to delete the row as shown below:



10. Press the **Delete** button to delete the row.
11. Press the **Stop** button to halt the iOS Simulator.
12. Finally, we'll cover how to control if a row can be moved or not. Uncomment the `section of code` under the comment `/TODO: Step 7c: uncomment to disable Row Reorder "grab bars" control in 2nd row.`
13. Click the **Play** button to build and run the project. The iOS Simulator will launch and display the view controller as before except that the Row Reorder control does not appear in row 2 as shown below:



Congratulations!

We covered all of the key concepts involved in working with Tables in iOS apps. In the next lab we'll cover Collection Views.