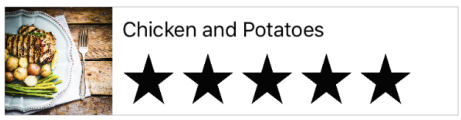
Create a Table View

On This Page

In this lesson, you create the main screen of the FoodTracker app. You'll create a second, table view-based scene, that lists the user's meals. You'll design custom table cells to display each meal, which look like this:



Learning Objectives

At the end of the lesson, you'll be able to:

- Create a second storyboard scene
- Understand the key components of a table view
- Create and design a custom table view cell
- Understand the roles of table view delegates and data sources
- Use an array to store and work with data
- Display dynamic data in a table view

Create the Opening Scene

So far, the FoodTracker app has a single scene managed by a view controller that represents a screen where a user can add and rate a new meal: the meal scene. Now it's time to create the scene that shows the entire list of meals. Fortunately, iOS comes with a powerful built-in class called a table view (`UITableView`) designed specifically to display a scrolling list of items.

A table view is managed by a table view controller (`UITableViewController`), a subclass of `UIViewController` that's specialized for handling table view-related logic. You'll create the new scene based on a table view controller.

To add a scene with a table view to your storyboard

1. Open your storyboard, `Main.storyboard`.
2. Open the Object library in the utility area. (Alternatively, choose View > Utilities > Show Object Library.)
3. In the Object library, find a Table View Controller object.
4. Drag a Table View Controller object from the list and drop it on the canvas to the left of the existing meal scene.

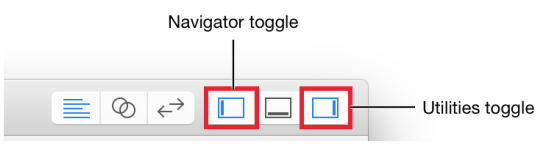
   If you see a table view with content and nothing happens when you try to drag it to the canvas, you're probably dragging a table view rather than a table view controller. A table view is one of the things managed by a table view controller, but you want the whole package, so find the table view controller and drag it to the canvas.

You now have two scenes, one for displaying the meal list and one for adding a new meal.

It makes sense to have the meal list be the first thing users see when they launch your app, so tell Xcode that's your intent by setting the table view controller as the first scene.

To set the table view controller as the initial scene

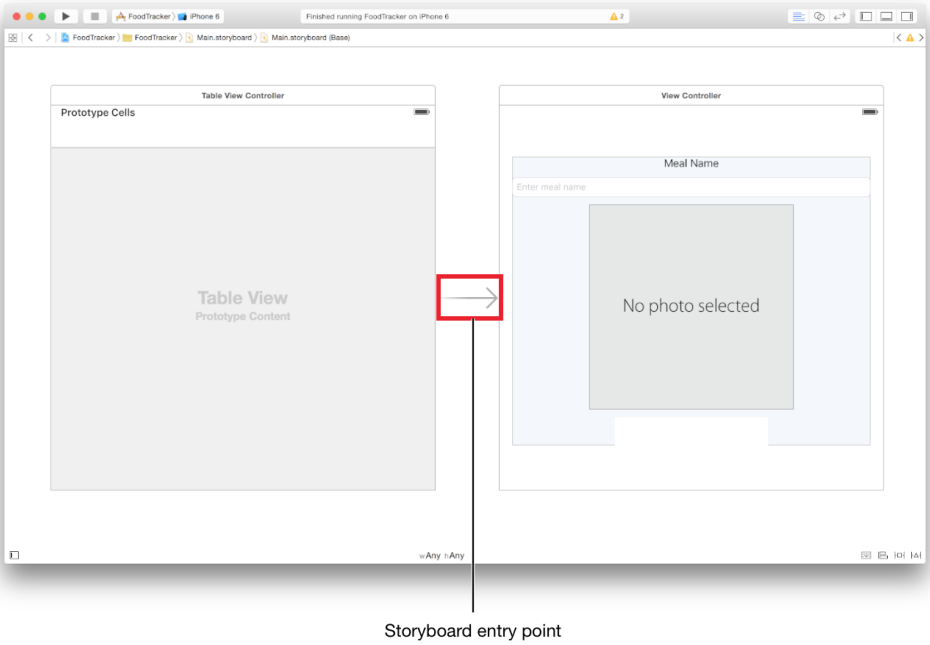
1. If you want more space to work, collapse the project navigator and utility area by clicking the Navigator and Utilities buttons in the Xcode toolbar.



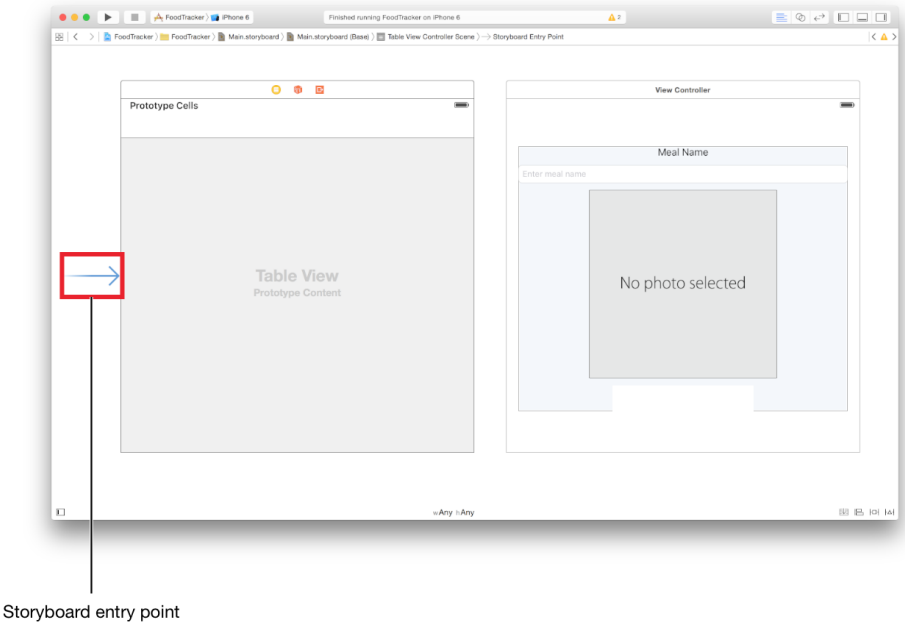
You can also collapse the outline view.

2. Drag the storyboard entry point from the meal scene to the table view controller.

On This Page



The table view controller is set as the initial view controller in your storyboard, making it the first scene that loads on app launch.



Checkpoint: Run your app. Instead of the meal scene with its text field, image view, and rating control, you should now see an empty table view—a screen with a number of horizontal dividers to separate it into rows, but with no content in each row.

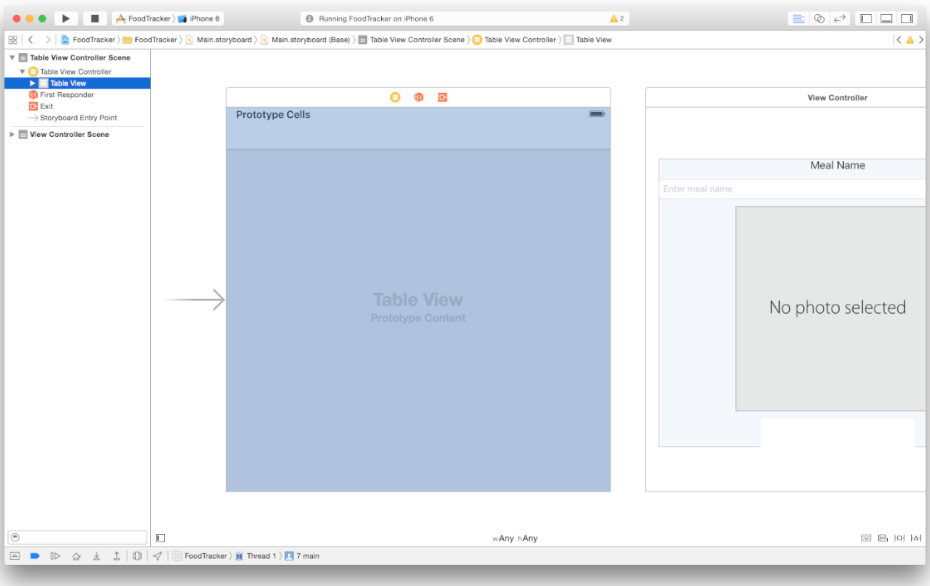
On This Page

You need to change a few settings on this table view so you can use it in your app.

To configure the table view

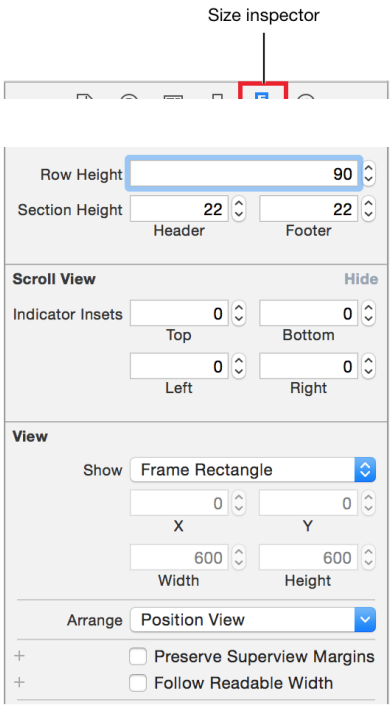
1. In your storyboard, open the outline view.
2. In the outline view, select Table View.

   The table view is nested under Table View Controller Scene > Table View Controller. You may have to click the disclosure triangles next to those objects to see the nested table view.



3. With the table view selected, open the Size inspector in the utility area.

   The Size inspector appears when you select the fifth button from the left in the inspector selector bar. It lets you edit the size and position of an object in your storyboard.



On This Page

4. In the Size inspector, find the field labeled Row Height and type `90`. Press Return.

You'll come back to working on the table view itself in a little while, after you design an interface for what the table view displays: its table view cells.

Design Custom Table Cells

The individual rows in a table view are managed by table view cells (`UITableViewCell`), which are responsible for drawing their contents. Table view cells come with a variety of predefined behavior and default cell styles. Default cell styles are great for a lot of situations, but you have more content to display in each of your cells than the default styles allow, so you'll need to define a custom cell style.

To create a subclass of UITableViewCell

1. Choose File > New > File (or press Command-N).
2. On the left of the dialog that appears, select Source under iOS.
3. Select Cocoa Touch Class, and click Next.
4. In the Class field, type `Meal`.
5. In the "Subclass of" field, select `UITableViewCell`.

   The class title changes to `MealTableViewCell`. Xcode makes it clear from the naming that you're creating a custom table view cell, so leave the new name as is.
6. Make sure the Language option is set to Swift.
7. Click Next.

   The save location defaults to your project directory.

   The Group option defaults to your app name, FoodTracker.

   In the Targets section, make sure your app is selected and the tests for your app are unselected.
8. Leave these defaults as they are, and click Create.

   Xcode creates a file that defines the `MealTableViewCell` class: `MealTableViewCell.swift`.

Now, open your storyboard.

If you look at the table view controller scene in your storyboard, you'll notice that it shows only a single cell.

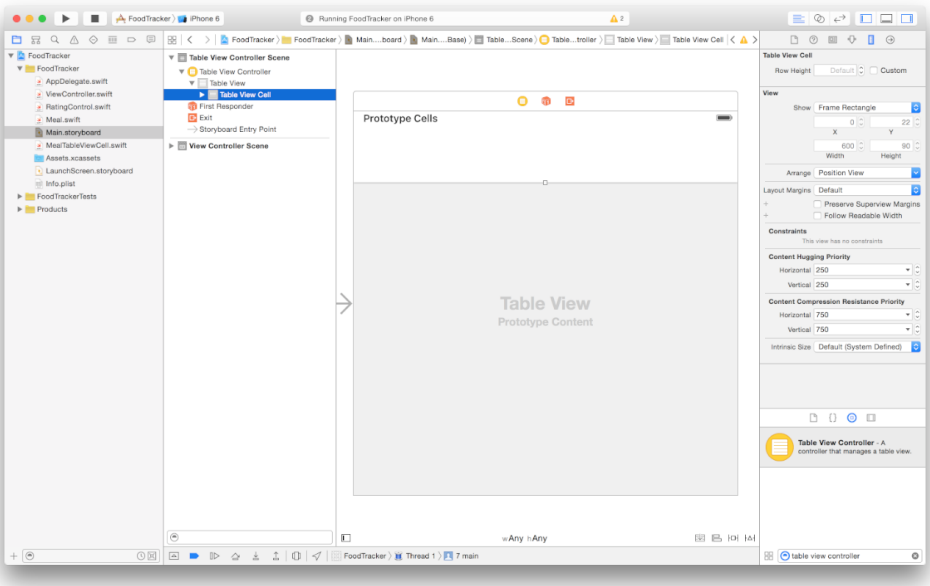
On This Page

This cell represents a prototype for the other cells; the design and behavior you give this cell will be used by the other cells in the table view. But first, you need to do a little bit of configuration. Connect the table view cell in your scene to the custom cell subclass you just created.

To configure a custom cell for your table view

1. In the outline view, select Table View Cell.

   The cell is nested under Table View Controller Scene > Table View Controller > Table View. You may have to disclose those objects to see the table view cell.

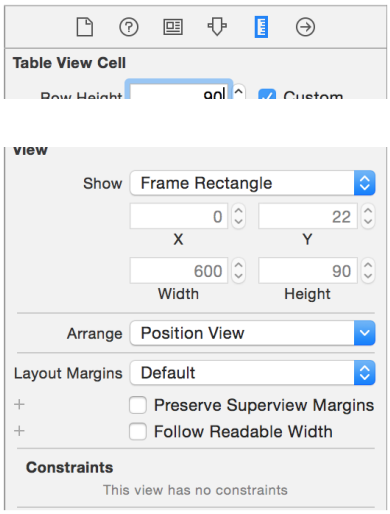


2. With the table view cell selected, open the Attributes inspector in the utility area.
3. In the Attributes inspector, find the field labeled Identifier and type `MealTableViewCell`. Press Return.

   This is an important step—you'll see why later.
4. In the Attributes inspector, find the field labeled Selection and select None.

   With this option, the cell won't get a visual highlight when a user taps it.
5. Open the Size inspector.
6. In the Size inspector, find the field labeled Row Height and type `90`.

   Make sure the Custom checkbox next to this field is selected:



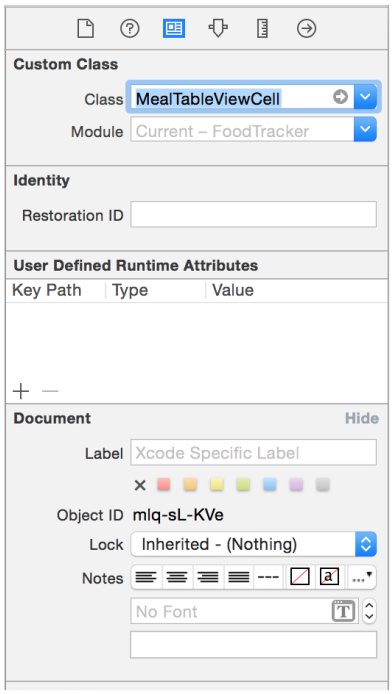
On This Page

Press Return to display the new cell height in your storyboard.

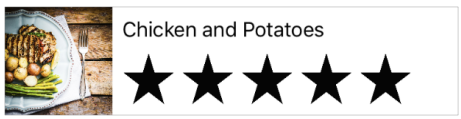
7. Open the Identity inspector.

   Recall that the Identity inspector lets you edit properties of an object in your storyboard related to that object's identity, such as what class the object belongs to.

8. In the Identity inspector, find the field labeled Class and select `MealTableViewCell`.



With the cell configured, you can design its custom UI directly in the storyboard. It'll contain the meal name, photo, rating, and look something like this:

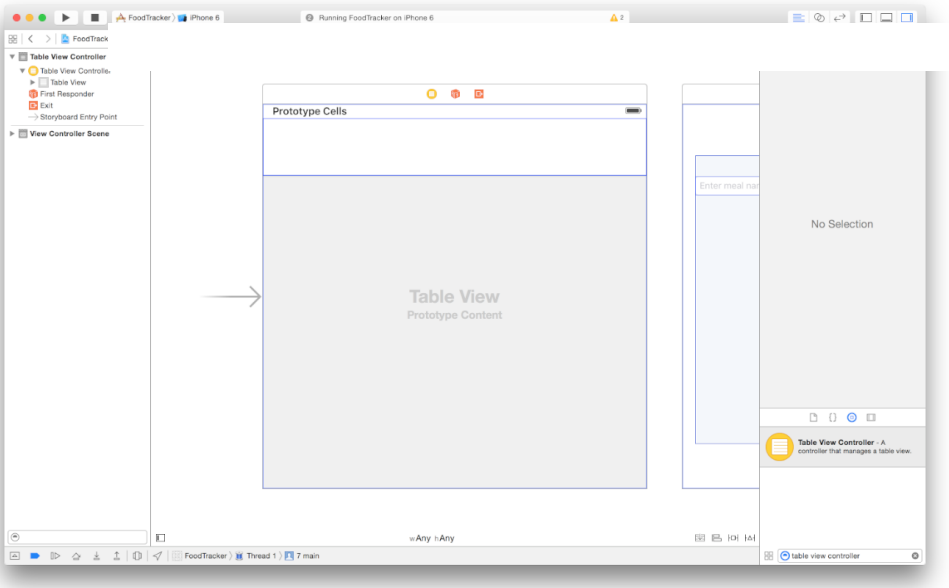


For this, you'll need a label, an image view, and a rating control. You can reuse the rating control UI you created in an earlier lesson.

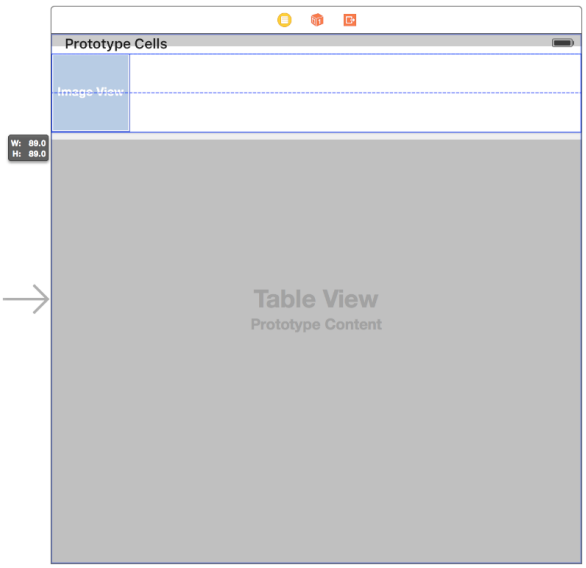
To design the interface of the custom table cell

1. Choose Editor > Canvas > Show Bounds Rectangles to show the bounds of the elements in the UI, making it easier to align elements in the table cell.

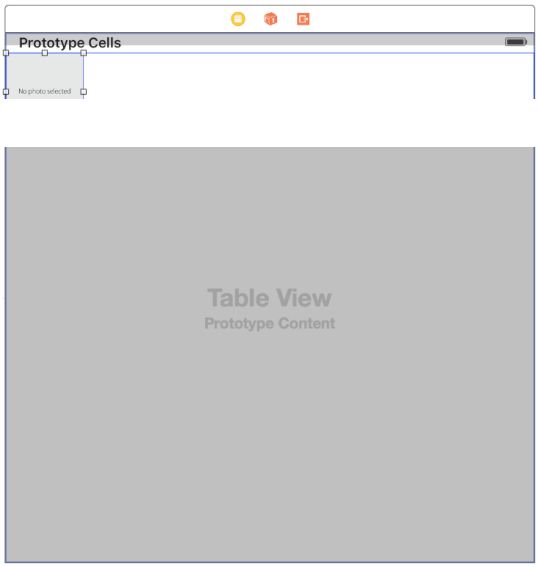
On This Page



2. Use the Object library to find an Image View object and drag it onto the table cell.
3. Drag and resize the image view so that it's square, flush against the left, top, and bottom of the cell.

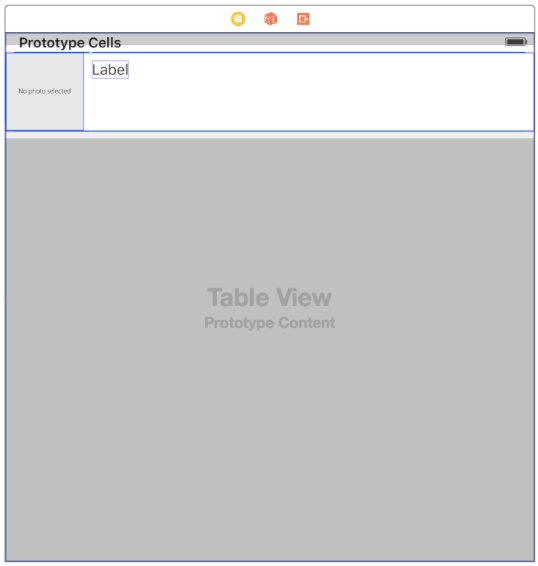


4. If you didn't add a default photo to your project in a previous lesson, add it now.
5. With the image view selected, open the Attributes inspector in the utility area.
6. In the Attributes inspector, find the field labeled Image and select `defaultPhoto`.

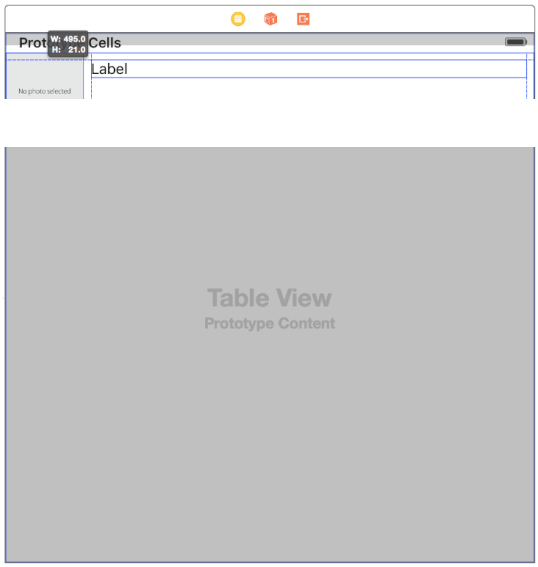


On This Page

7. Use the Object library to find a Label object and drag it onto the table cell.
8. Drag the label so that it's close to the right side of the image view and aligned with the top margin in the table cell.

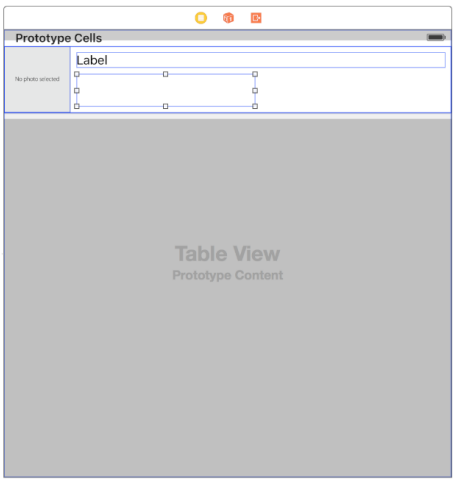


9. Resize the label so that its right edge stretches to the right margin of the table cell.

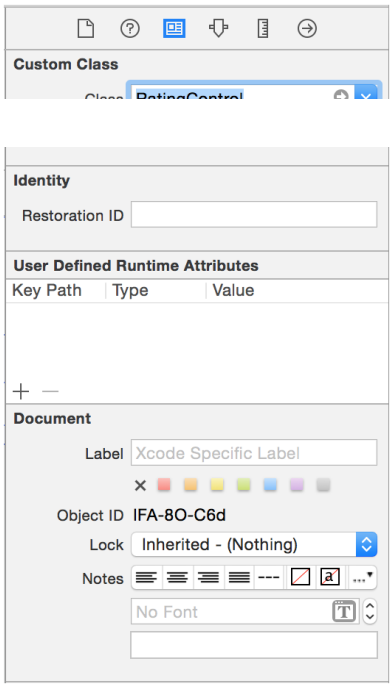


On This Page

10. Use the Object library to find a View object and drag it onto the table cell.
11. With the view selected, open the Size inspector in the utility area.
12. In the Size inspector, type `44` in the Height field and `240` in the Width field. Press Return.
13. Drag the view so that it's below the label and aligned with the label's left margin.



14. With the view selected, open the Identity inspector.
15. In the Identity inspector, find the field labeled Class and select `RatingControl`.



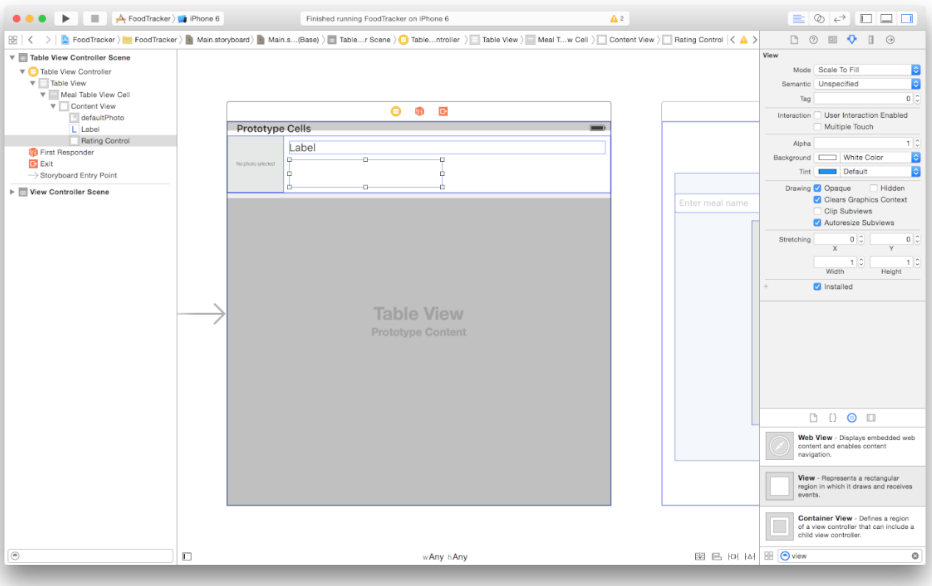
On This Page

If you don't see `RatingControl` as an option in the pop-up menu, make sure you have the correct UI element selected on the canvas (the one shown with resize handles in the previous image.)

16. With the view selected, open the Attributes inspector .
17. In the Attributes inspector, find the field labeled Interaction and deselect the User Interaction Enabled checkbox.

    You designed your custom rating control class to be interactive, but it doesn't need to be interactive when it's displayed in this table view cell. So it's important to disable that interaction when it's in this context.

Your UI should look something like this:



Checkpoint: Run your app. The table view cells now look taller. But even though you added all the necessary UI elements to your table view cells, they're showing up empty, just like before. Why's that?

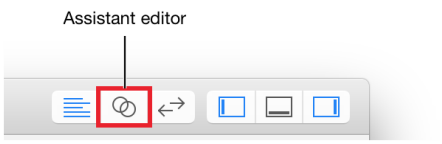
On This Page

In a storyboard, a table view can be configured to display static data (supplied in the storyboard UI) or dynamic data (supplied by the table view controller logic). The table view defaults to using dynamic data, and because you'll need to load data in code, this is what you want it to do—you just haven't implemented that behavior yet. This means that the static content you supplied in the storyboard doesn't show up at runtime, so you can't see it when you run the app—until you implement the data model behind it.

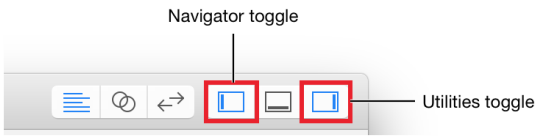
For now, preview your UI using the assistant editor.

To preview your interface

1. Click the Assistant button in the Xcode toolbar to open the assistant editor.

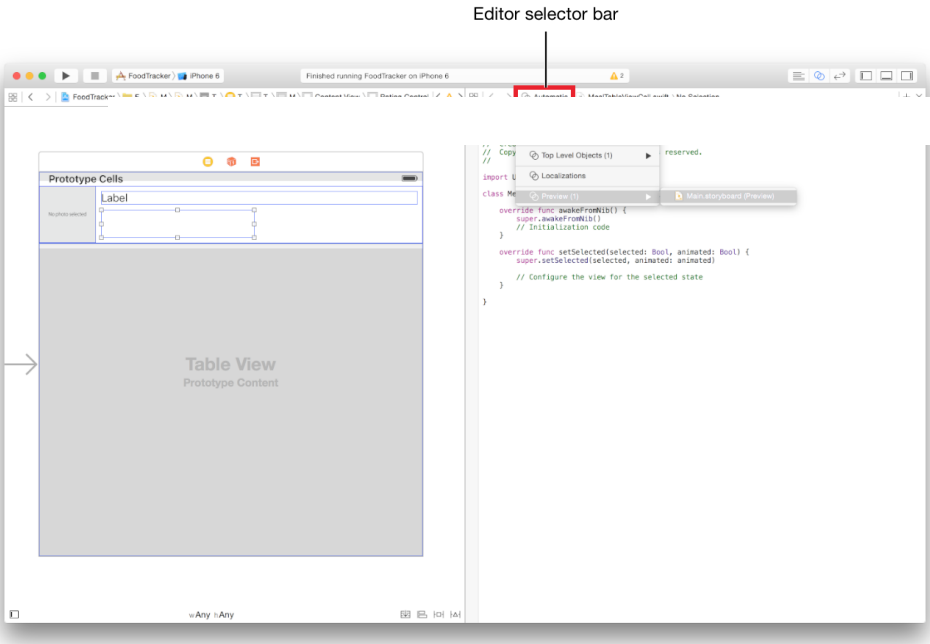


2. If you want more space to work, collapse the project navigator and utility area by clicking the Navigator and Utilities buttons in the Xcode toolbar.



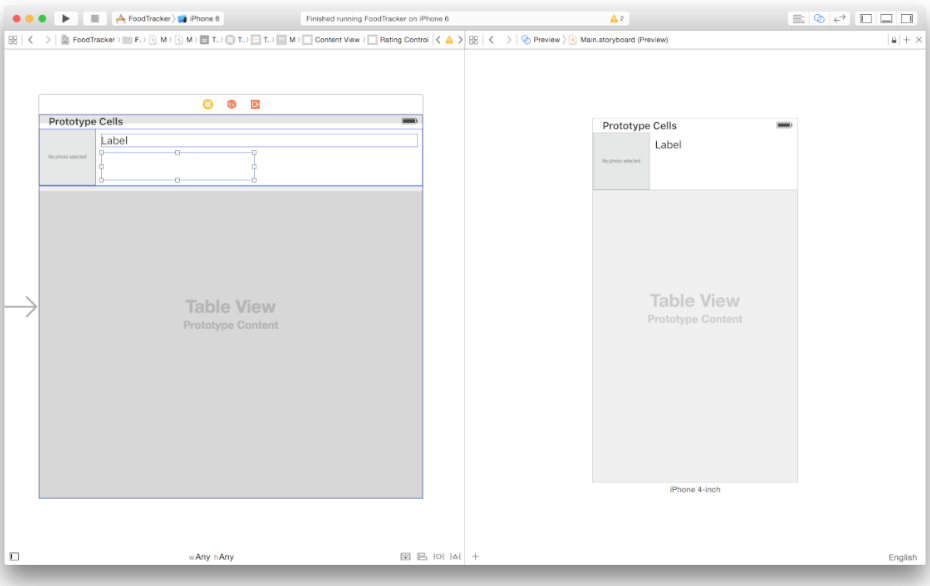
You can also collapse the outline view.

3. In the editor selector bar, which appears at the top of the assistant editor, switch the assistant editor from Automatic to Preview > Main.storyboard (Preview).



On This Page

Your Xcode window should look something like this:



The preview shows that the UI looks as expected. The prototype table view cell UI looks finished.

> NOTE
>
> If you see the wrong scene in the UI preview, make sure to select the table view scene by clicking its scene dock.

Add Images to Your Project

Next, you need to add a few sample images to your project. You'll use these images when you load initial meal data into your app.

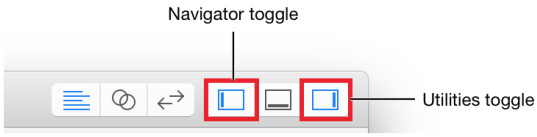
You can find sample images within the `Images/` folder of the downloadable file at the end of this lesson, or use your own images. (Just make sure the names of the images you use match the image names in the code later.)

To add images to your project

1. If the assistant editor is open, return to the standard editor by clicking the Standard button.

   Open the project navigator and utility area by clicking the Navigator and Utilities buttons in the Xcode toolbar.

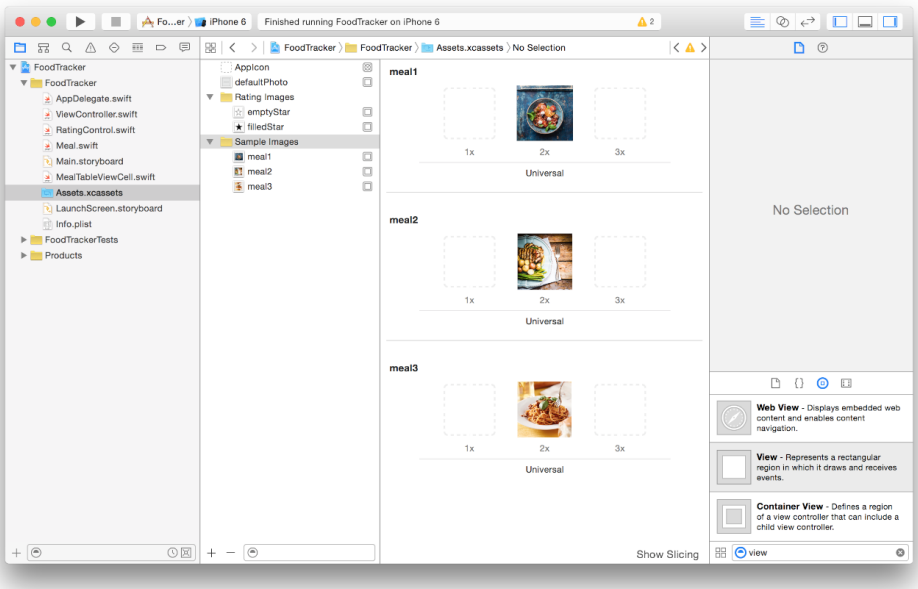
On This Page



2. In the project navigator, select `Assets.xcassets` to view the asset catalog.

   Recall that the asset catalog is a place to store and organize your image assets for an app.
3. In the bottom left corner, click the plus (+) button and select New Folder from the pop-up menu.
4. Double-click the folder name and rename it `Sample Images`.
5. With the folder selected, in the bottom left corner, click the plus (+) button and choose New Image Set from the pop-up menu.
6. Double-click the image set name and rename it to a name you'll remember when you're writing it in code.
7. On your computer, select the image you want to add.
8. Drag and drop the image into the 2x slot in the image set.

Repeat steps 5–8 for as many images as you like. The rest of this lesson assumes you have three different images.

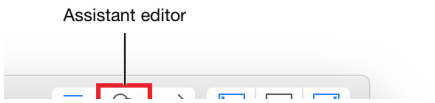


Connect the Table Cell UI to Code

Before you can display dynamic data in your table view cells, you need to create outlet connections between the views in your storyboard and the code that represents the table view cell in `MealTableViewCell.swift`.

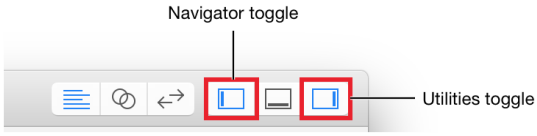
To connect the views to the MealTableViewCell.swift code

1. In your storyboard, select the label in the table view cell.
2. Click the Assistant button in the Xcode toolbar to open the assistant editor.

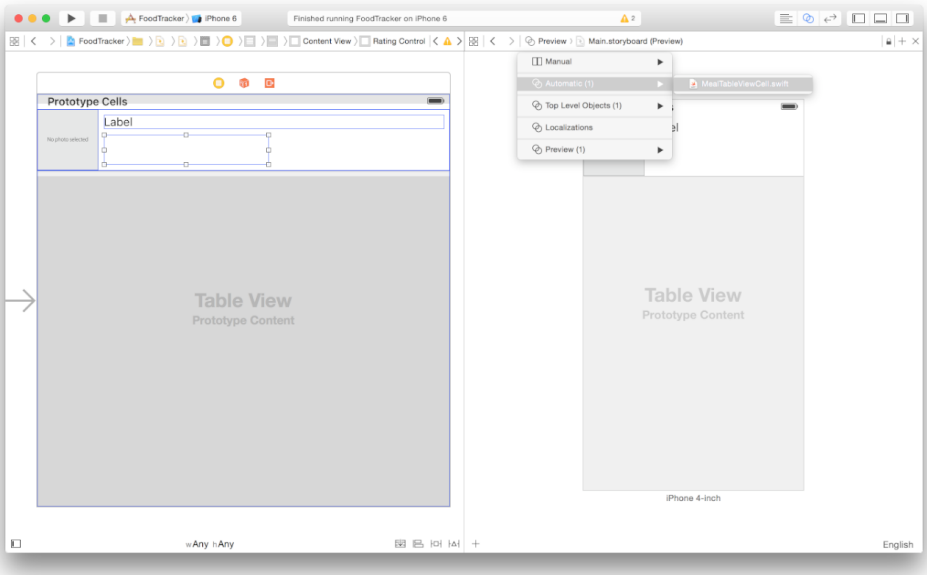


On This Page

3. If you want more space to work, collapse the project navigator and utility area by clicking the Navigator and Utilities buttons in the Xcode toolbar.



4. In the editor selector bar, which appears at the top of the assistant editor, switch the assistant editor from Preview to Automatic > `MealTableViewCell.swift`.



`MealTableViewCell.swift` displays in the editor on the right.

5. In `MealTableViewCell.swift`, find the `class` line, which should look like this:

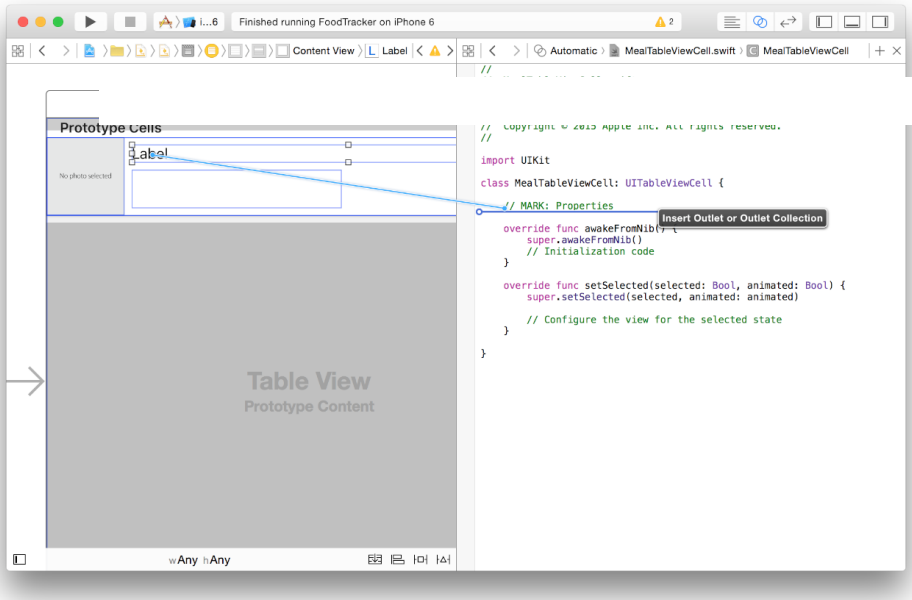
```
class MealTableViewCell: UITableViewCell {
```

6. Below the `class` line, add the following:

```
// MARK: Properties
```

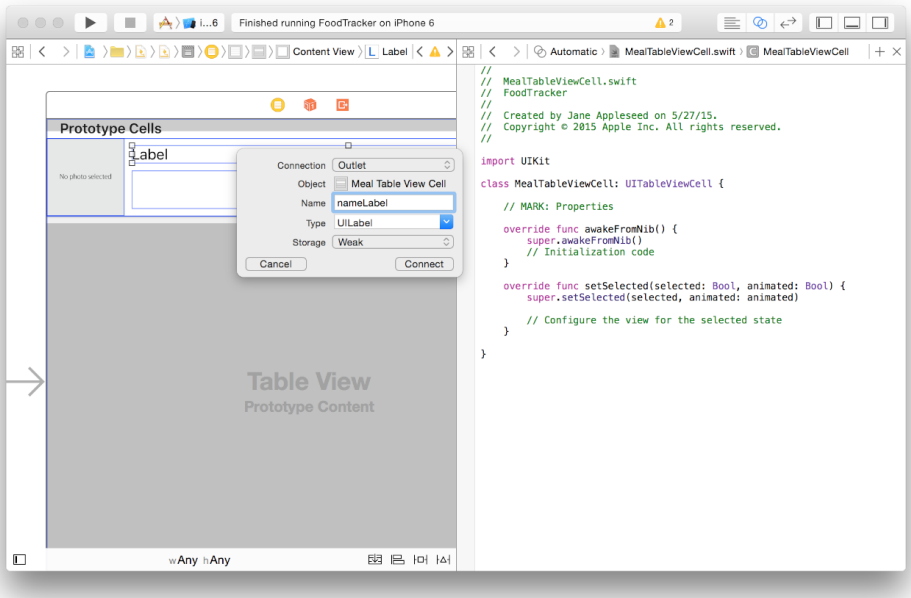
7. Control-drag from the label on your canvas to the code display in the editor on the right, stopping the drag at the line below the comment you just added in `MealTableViewCell.swift`.

On This Page



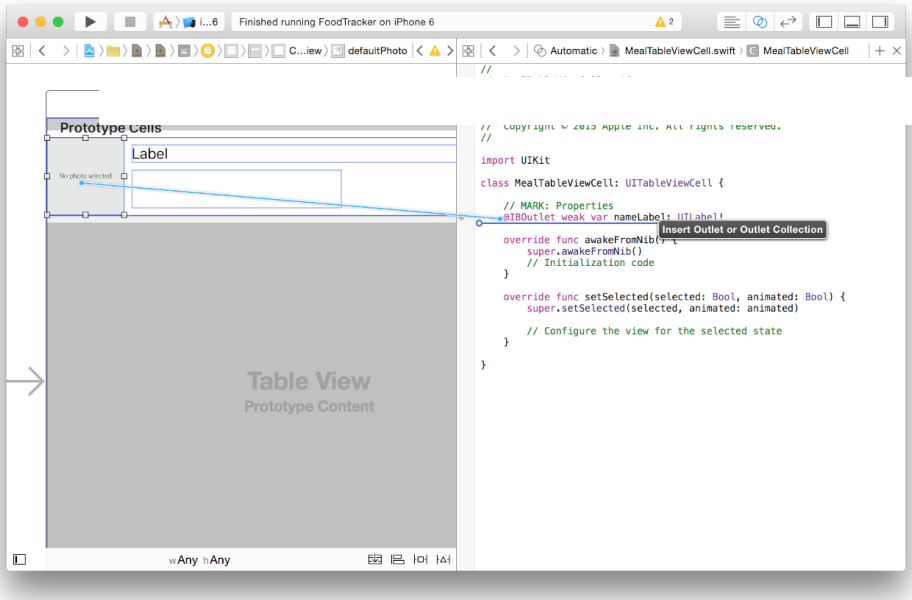
8. In the dialog that appears, for Name, type `nameLabel`.

   Leave the rest of the options as they are. Your dialog should look like this:



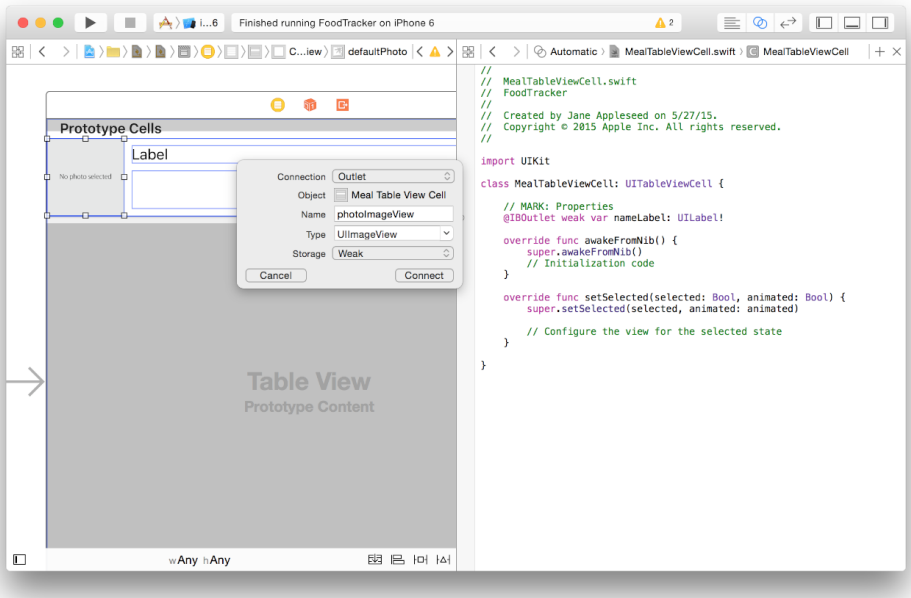
9. Click Connect.
10. In your storyboard, select the image view in the table view cell.
11. Control-drag from the image view on your canvas to the code display in the editor on the right, stopping the drag at the line just below the `nameLabel` property in `MealTableViewCell.swift`.

On This Page

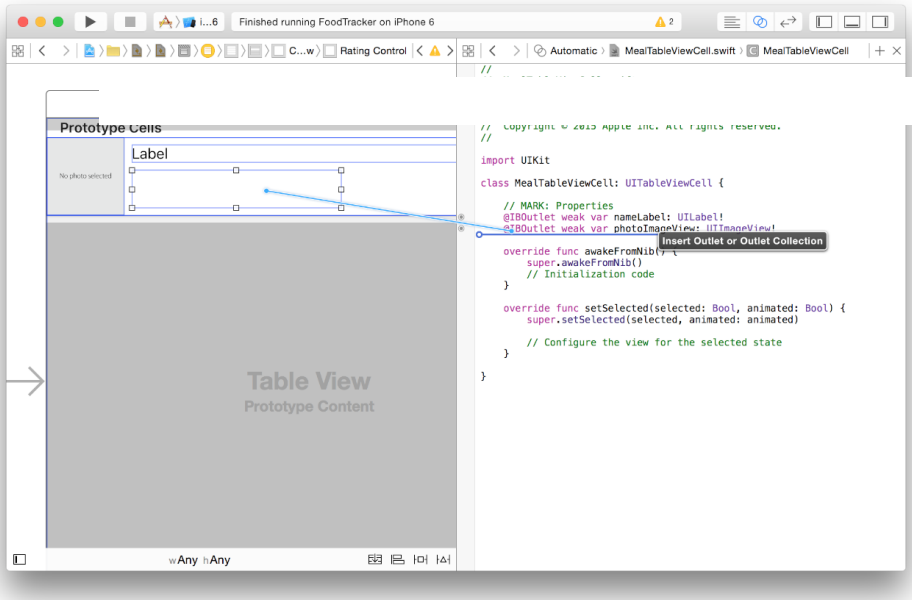


12. In the dialog that appears, for Name, type `photoImageView`.

    Leave the rest of the options as they are and click Connect.

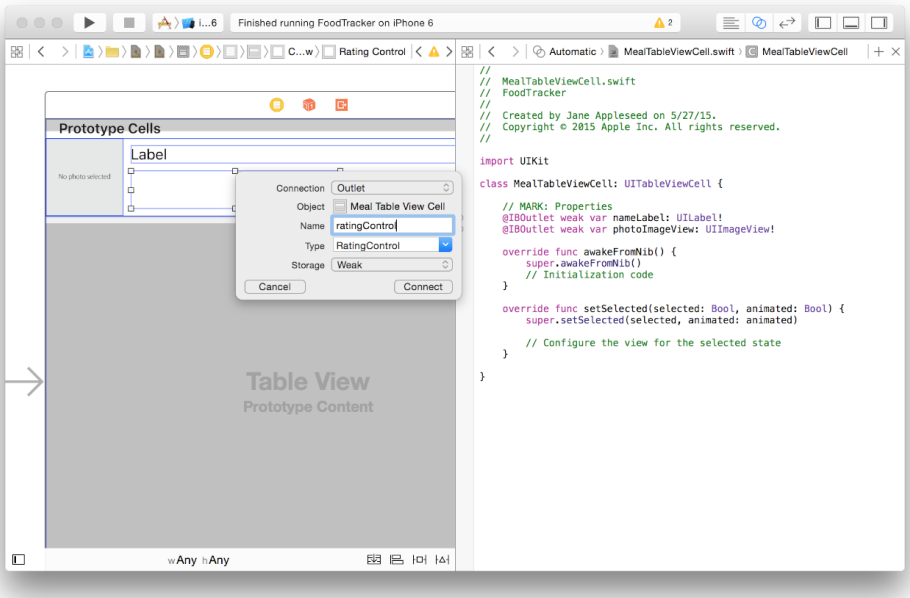


13. In your storyboard, select the rating control in the table view cell.
14. Control-drag from the rating control on your canvas to the code display in the editor on the right, stopping the drag at the line just below the `photoImageView` property in `MealTableViewCell.swift`.



15. In the dialog that appears, for Name, type `ratingControl`.

    Leave the rest of the options as they are and click Connect.



Your outlets in `MealTableViewCell.swift` should look like this:

```
@IBOutlet weak var nameLabel: UILabel!
@IBOutlet weak var photoImageView: UIImageView!
@IBOutlet weak var ratingControl: RatingControl!
```

Load Initial Data

To display any real data in your table cells, you need to write code to load that data. At this point, you have a data model for a meal: the `Meal` class. You also need to keep a list of those meals. The natural place to track this is in a custom view controller subclass that's connected to the meal list scene. This view controller will manage the view that displays the list of meals, and have a reference to the data model behind what's shown in the UI.

First, create a custom table view controller subclass to manage the meal list scene.

To create a subclass of UITableViewController

1. Choose F
2. On the left of the dialog that appears, select Source under iOS, then select Cocoa Touch Class.
3. Click Next.
4. In the Class field, type `Meal`.
5. In the “Subclass of” field, select `UITableViewController`.

   The class title changes to `MealTableViewController`. Leave that as is.
6. Make sure the “Also create XIB file” option is unselected.
7. Make sure the Language option is set to Swift.
8. Click Next.

   The save location defaults to your project directory.

   The Group option defaults to your app name, FoodTracker.

   In the Targets section, your app is selected and the tests for your app are unselected.
9. Leave these defaults as they are, and click Create.

   Xcode creates `MealTableViewController.swift`, a source code file that defines your custom table view controller subclass.

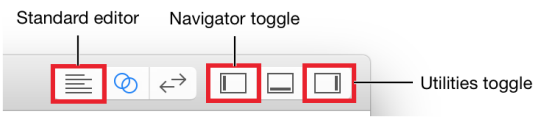
On This Page

In this custom subclass, you can now define a property to store a list of `Meal` objects. The Swift standard library includes a structure called `Array` that works well for tracking lists of items.

To load the initial data

1. If the assistant editor is open, return to the standard editor by clicking the Standard button.

   Expand the project navigator and utility area by clicking the Navigator and Utilities buttons in the Xcode toolbar.



2. Open `MealTableViewController.swift`.
3. Below the `class` line in `MealTableViewController.swift`, add the following code:

```
// MARK: Properties

var meals = [Meal]()
```

   This code declares a property on `MealTableViewController` and initializes it with a default value (an empty array of `Meal` objects). By making `meals` a variable (`var`) instead of a constant, you make the array mutable, which means you can add items to it after you initialize it.
4. In `MealTableViewController.swift`, after the `viewDidLoad()` method, add the following method:

```
func loadSampleMeals() {
}
```

   This is a helper method to load sample data into the app.
5. In the `loadSampleMeals()` method, add this code to create a few `Meal` objects. You can name and rate these sample meals whatever you like, of course, but here are some examples:

```
let photo1 = UIImage(named: "meal1")!
let meal1 = Meal(name: "Caprese Salad", photo: photo1, rating: 4)!

let photo2 = UIImage(named: "meal2")!
let meal2 = Meal(name: "Chicken and Potatoes", photo: photo2, rating: 5)!

let photo3 = UIImage(named: "meal3")!
let meal3 = Meal(name: "Pasta with Meatballs", photo: photo3, rating: 3)!
```

Make sure the names of the images in your project match the names you write in this code.

6. After creating the `Meal` objects, add them to the `meals` array using this code:

On This Page

```
meals += [meal1, meal2, meal3]
```

7. Find the `viewDidLoad()` method. The template implementation looks like this:

```
override func viewDidLoad() {
    super.viewDidLoad()

    // Uncomment the following line to preserve selection between presentations
    // self.clearsSelectionOnViewWillAppear = false

    // Uncomment the following line to display an Edit button in the navigation bar for this view controller.
    // self.navigationItem.rightBarButtonItem = self.editButtonItem()
}
```

The template implementation of this method includes comments that were inserted by Xcode when it created `MealTableViewController.swift`. Code comments like this provide helpful hints and contextual information in source code files, but you don't need them for this lesson.

8. In the `viewDidLoad()` method, delete the comments and instead add this code after the call to `super.viewDidLoad()` to load the sample meal data:

```
// Load the sample data.
loadSampleMeals()
```

The calls the helper method you just wrote to load the data when the view loads. You separated this into its own method to make your code more modular and readable.

Your `viewDidLoad()` method should look like this:

```
override func viewDidLoad() {
    super.viewDidLoad()

    // Load the sample data.
    loadSampleMeals()
}
```

And your `loadSampleMeals()` method should look something like this:

```
func loadSampleMeals() {
    let photo1 = UIImage(named: "meal1")!
    let meal1 = Meal(name: "Caprese Salad", photo: photo1, rating: 4)!

    let photo2 = UIImage(named: "meal2")!
    let meal2 = Meal(name: "Chicken and Potatoes", photo: photo2, rating: 5)!

    let photo3 = UIImage(named: "meal3")!
    let meal3 = Meal(name: "Pasta with Meatballs", photo: photo3, rating: 3)!

    meals += [meal1, meal2, meal3]
}
```

Checkpoint: Build your project by choosing Product > Build. It should build without errors.

> IMPORTANT
>
> If you're running into build issues, make sure the names of the images in your project exactly match the names you used in the code.

Display the Data

At this point, you
that's prepopulated with some sample meals. Now you need to display this data in the UI.

On This Page

To display dynamic data, a table view needs two important helpers: a data source and a delegate. A table view data source, as implied by its name, supplies the table view with the data it needs to display. A table view delegate helps the table view manage cell selection, row heights, and other aspects related to displaying the data. By default, `UITableViewController` and its subclasses adopt the necessary protocols to make the table view controller both a data source (`UITableViewDataSource` protocol) and a delegate (`UITableViewDelegate` protocol) for its associated table view. Your job is to implement the appropriate protocol methods in your table view controller subclass so that your table view has the correct behavior.

A functioning table view requires three table view data source methods.

```
func numberOfSectionsInTableView(tableView: UITableView) -> Int
func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int
func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath)
  -> UITableViewCell
```

The first of these is `numberOfSectionsInTableView(_:)`, which tells the table view how many sections to display. Sections are visual groupings of cells within table views, which is especially useful in table views with a lot of data. For a simple table view like the one in the FoodTracker app, you just need the table view to display a single section, so the implementation of the `numberOfSectionsInTableView(_:)` data source method is straightforward.

To display a section in your table view

1. In `MealTableViewController.swift`, find the `numberOfSectionsInTableView(_:)` data source method. The template implementation looks like this:

   ```
   override func numberOfSectionsInTableView(tableView: UITableView) -> Int {
       // #warning Incomplete implementation, return the number of sections
       return 0
   }
   ```

2. Change the return value from `0` to `1`, and remove the warning comment.

   ```
   override func numberOfSectionsInTableView(tableView: UITableView) -> Int {
       return 1
   }
   ```

   This code makes the table view show 1 section instead of 0. You removed the comment that says `#warning Incomplete implementation` because you've completed the implementation.

The next data source method, `tableView(_:numberOfRowsInSection:)`, tells the table view how many rows to display in a given section. A table view defaults to having a single section, which is just what you need for the FoodTracker app. Each meal should have its own row in that section. That means that the number of rows should be the number of `Meal` objects in your `meals` array.

To return the number of rows in your table view

1. In `MealTableViewController.swift`, find the `tableView(_:numberOfRowsInSection:)` data source method. The template implementation looks like this:

   ```
   override func tableView(tableView: UITableView, numberOfRowsInSection section:
     Int) -> Int {
       // #warning Incomplete implementation, return the number of rows
       return 0
   }
   ```

   You want to return the number of meals you have. `Array` has a property called `count` that returns the number of items in the array, so the number of rows is `meals.count`.

2. Change the `tableView(_:numberOfRowsInSection:)` data source method to return the appropriate number of rows, and remove the warning comment.

```
override func tableView(tableView: UITableView, numberOfRowsInSection section:
  Int) -> Int {
    return meals.count
}
```

On This Page

The last data source method, `tableView(_:cellForRowAtIndexPath:)`, configures and provides a cell to display for a given row. Each row in a table view has one cell, and that cell determines the content that appears in that row and how that content is laid out.

For table views with a small number of rows, all rows may be onscreen at once, so this method gets called for each row in your table. But table views with a large number of rows display only a small fraction of their total items at a given time. It's most efficient for table views to only ask for the cells for rows that are being displayed, and that's what `tableView(_:cellForRowAtIndexPath:)` allows the table view to do.

For any given row in the table view, you configure the cell by fetching the corresponding `Meal` in the `meals` array, and then setting the cell's properties to corresponding values from the `Meal` class.

To configure and display cells in your table view

1. In `MealTableViewController.swift`, find and uncomment the `tableView(_:cellForRowAtIndexPath:)` data source method. (To uncomment the method, remove the `/*` and `*/` characters surrounding it.)

   After you do that, the template implementation looks like this:

   ```
   override func tableView(tableView: UITableView, cellForRowAtIndexPath
     indexPath: NSIndexPath) -> UITableViewCell {
       let cell = tableView.dequeueReusableCellWithIdentifier("reuseIdentifier",
     forIndexPath: indexPath)

       // Configure the cell...

       return cell
   }
   ```

   The template performs several tasks. It asks the table view for a cell with a placeholder identifier, adds a comment about where code to configure the cell should go, and then returns the cell.

   To make this code work for your app, you'll need to change the placeholder identifier to the one you set earlier for the prototype cell in the storyboard (`MealTableViewCell`), and then add code to configure the cell.

2. Add code at the beginning of the method, before the rest of the template implementation:

   ```
   // Table view cells are reused and should be dequeued using a cell identifier.
   let cellIdentifier = "MealTableViewCell"
   ```

   This creates a constant with the identifier you set in the storyboard.

3. Update the placeholder identifier to the identifier you set in your storyboard. The second line of code in the method should now look like this:

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

4. Because you created a custom cell class that you want to use, downcast the type of the cell to your custom cell subclass, `MealTableViewCell`. The second line of code in the method should now look like this:

   ```
   let cell = tableView.dequeueReusableCellWithIdentifier(cellIdentifier,
     forIndexPath: indexPath) as! MealTableViewCell
   ```

5. Below the previous line, add the following code:

   ```
   // Fetches the appropriate meal for the data source layout.
   let meal = meals[indexPath.row]
   ```

   This code fetches the appropriate meal in the `meals` array.

6. Delete the `// Configure the cell` comment and add this code in its place:

```
cell.nameLabel.text = meal.name
c
c
```

On This Page

This sets each of the views in the table view cell to display the corresponding data from `meal`.

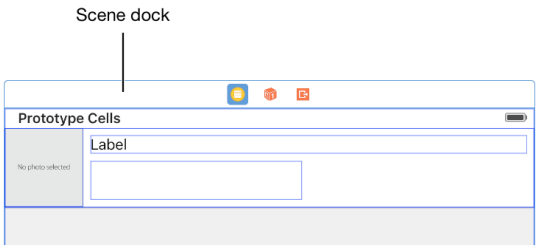
Your `tableView(_:cellForRowAtIndexPath:)` method should look like this:

```
override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath:
  NSIndexPath) -> UITableViewCell {
    // Table view cells are reused and should be dequeued using a cell identifier.
    let cellIdentifier = "MealTableViewCell"
    let cell = tableView.dequeueReusableCellWithIdentifier(cellIdentifier,
  forIndexPath: indexPath) as! MealTableViewCell

    // Fetches the appropriate meal for the data source layout.
    let meal = meals[indexPath.row]

    cell.nameLabel.text = meal.name
    cell.photoImageView.image = meal.photo
    cell.ratingControl.rating = meal.rating

    return cell
}
```

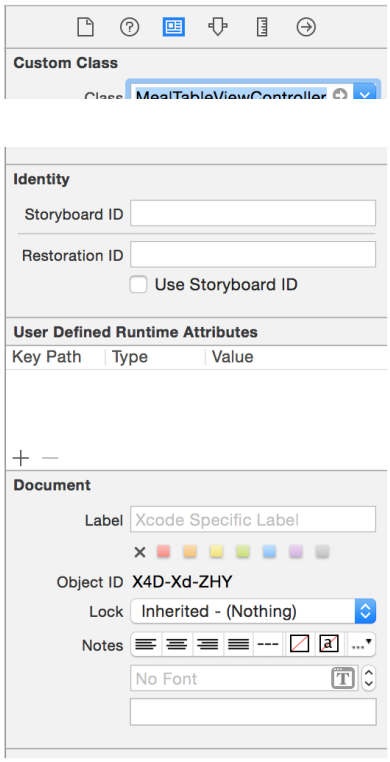
The final step to displaying data in the UI is to connect the code defined in `MealTableViewController.swift` to the meal list storyboard scene.

To point the table view controller scene to MealTableViewController.swift

1. Open your storyboard.
2. Select the table view controller by clicking on its scene dock until the entire scene has a blue outline around it.

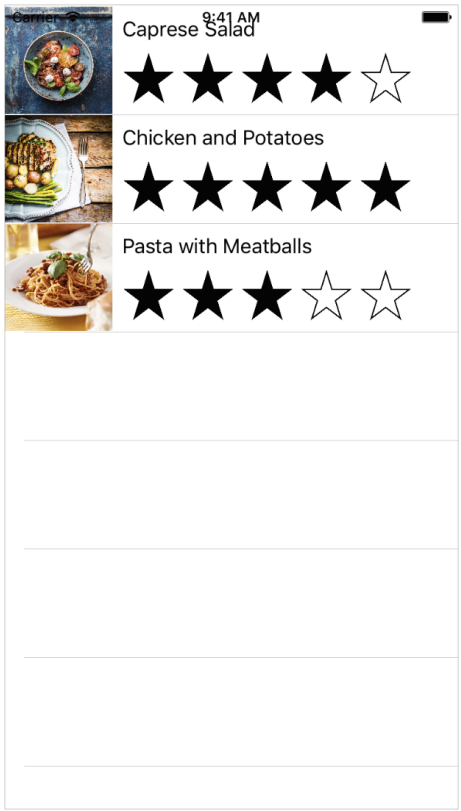


3. Open the Identity inspector.
4. In the Identity inspector, find the field labeled Class, and select `MealTableViewController`.



On This Page

Checkpoint: Run your app. The list of items you added in the `viewDidLoad()` method should show up as cells in your table view. You'll notice there's a little bit of overlap between the table view cells and the status bar—you'll fix that in the next lesson.



Prepare the Meal Scene for Navigation

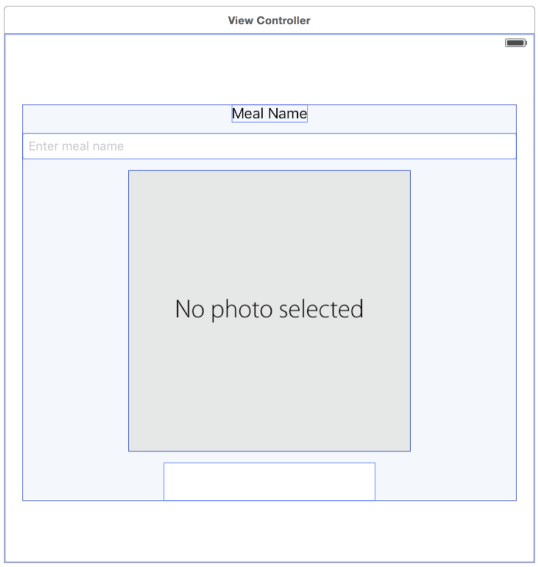
As you prepare to implement navigation in the FoodTracker app, you need to delete and replace a few pieces of code and UI that you won't need anymore.

To clean up unused pieces of the project

1. Open your storyboard and look at the meal scene.

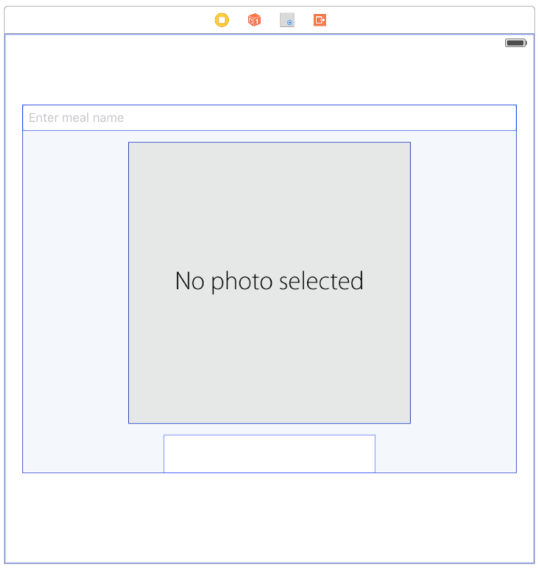
   Your meal scene UI should look something like this:

On This Page



2. In the meal scene, select the `Meal Name` label, and press the Delete key to delete it.

   The rest of the elements in the stack view reposition themselves appropriately.



3. Open `ViewController.swift`.
4. In `ViewController.swift`, find the `textFieldDidEndEditing(_:)` method.

```
func textFieldDidEndEditing(textField: UITextField) {
    mealNameLabel.text = textField.text
}
```

5. Delete the line that sets the text property of the label.

```
mealNameLabel.text = textField.text
```

   You'll replace this with a new implementation soon.

6. In `ViewController.swift`, find the `mealNameLabel` outlet and delete it.

```
@IBOutlet weak var mealNameLabel: UILabel!
```

Because you no
more meaningfu

On This Page

To rename the ViewController.swift file

1. In the project navigator, click the `ViewController.swift` file once and press the Return key.

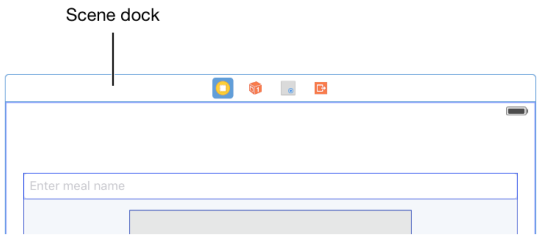
   Xcode lets you type in a new name for the file.
2. Rename the file `MealViewController.swift`. Press Return.
3. In `MealViewController.swift`, find the class declaration line:

```
class ViewController: UIViewController, UITextFieldDelegate,
  UIImagePickerControllerDelegate, UINavigationControllerDelegate {
```

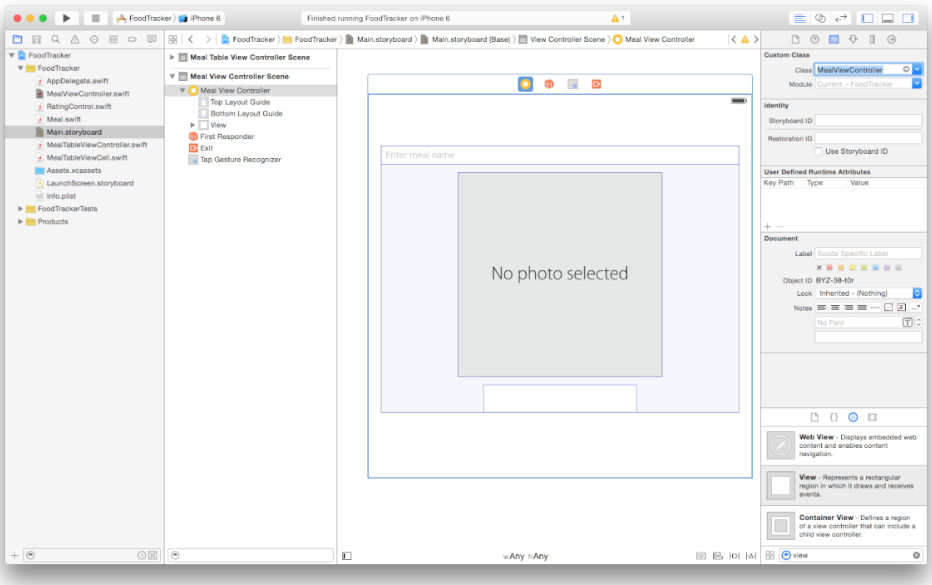
4. Change the class name to `MealViewController`.

```
class MealViewController: UIViewController, UITextFieldDelegate,
  UIImagePickerControllerDelegate, UINavigationControllerDelegate {
```

5. In the comment at the top of the file, also change the name from `ViewController.swift` to `MealViewController.swift`.
6. Open your storyboard.
7. Select the meal scene by clicking on its scene dock.



8. With the meal scene selected, open the Identity inspector .
9. In the Identity inspector, change the Class field from `ViewController` to `MealViewController`.



Checkpoint: Build or run your app. Everything should work as before.

At this point, you may see an Xcode warning related to the fact that there's no way to reach the meal scene in the app. Don't worry; you'll add navigation behavior in the next lesson.

> NOTE
>
> To see the completed sample project for this lesson, download the file and view it in Xcode.
>
> Download File

On This Page