17 Collection Views

In the previous lesson, you learned about table views. Collection views are similar to table views in many respects; the primary difference between them is that collection views are not restricted to single column layouts. A collection views layout can be customized programmatically, allowing collection views to present data in grid layouts, circular layouts, and cover-flow layouts (see Figure 17-1).

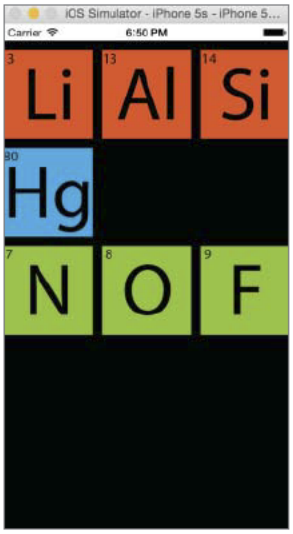


FIGURE 17-1

Collection views are instances of the `UICollectionView` class and are part of the UIKit framework. Data in a collection view is referred to as *Items* that are grouped into sections. Each section can have an optional header and a footer view.

CREATING A COLLECTION VIEW WITH INTERFACE BUILDER

To add a new scene in your storyboard that contains a collection view, simply drag and drop a Collection View Controller object into your scene. A collection view controller is an object that manages a collection view (see Figure 17-2).

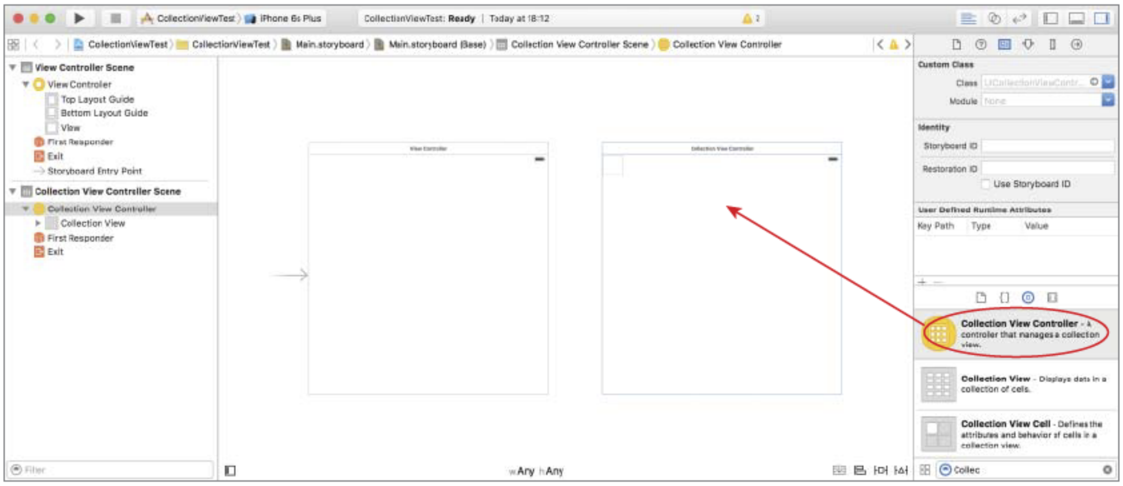
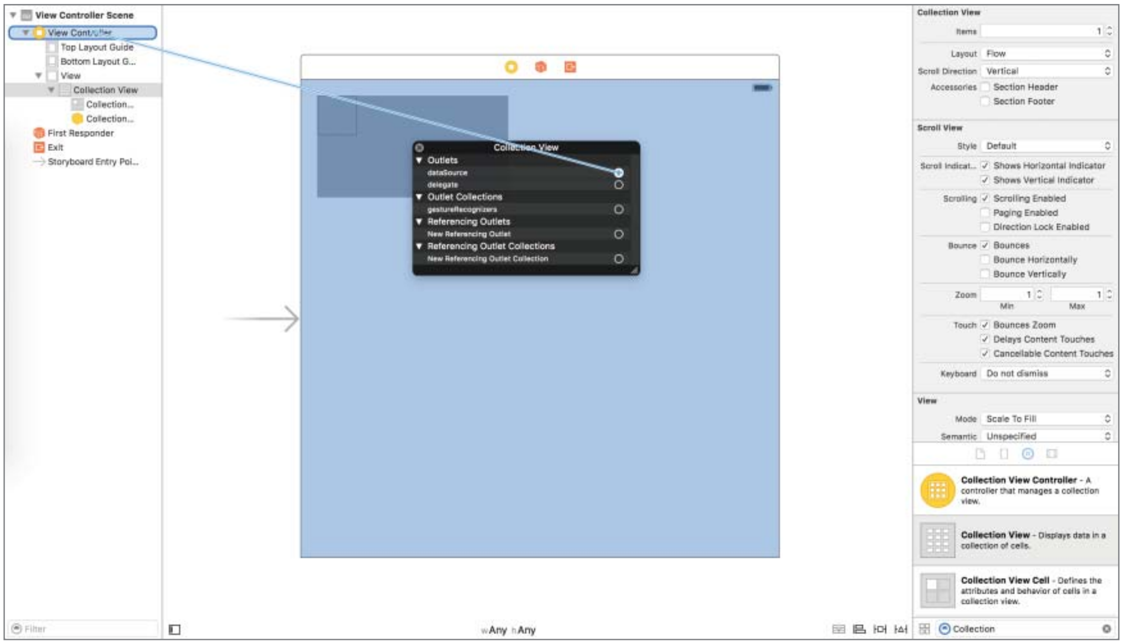

View Controller Scene
View Controller
Top Layout Guide
Bottom Layout Guide
View
First Responder
Exit
Storyboard Entry Point
Collection View Controller Scene
Collection View Controller
Collection View
First Responder
Exit
Custom Class
Class
Module
Identity
Storyboard ID
Restoration ID
Use Storyboard ID
User Defined Runtime Attributes
Key Path
Type
Value
Collection View Controller - A controller that manages a collection view.
Collection View - Displays data in a collection of cells.
Collection View Cell - Defines the attributes and behavior of cells in a collection view.
wAny hAny

FIGURE 17-2

If, however, you want to add a collection view to an existing scene, drag and drop a collection view object onto the scene. When using a collection view your view controller class will be responsible for managing the collection view and will need to implement relevant methods from the `UICollectionViewDataSource` and `UICollectionViewDelegate` protocols. You will also need to set the view controller to act as both the data source and delegate object for the collection view (see Figure 17-3). The data source and delegate will be discussed in more detail later in this lesson.



Collection View
Items 1
Layout Flow
Scroll Direction Vertical
Accessories ☐ Section Header
☐ Section Footer

Scroll View
Style Default
Scroll Indicat... ✓ Shows Horizontal Indicator
✓ Shows Vertical Indicator
Scrolling ✓ Scrolling Enabled
☐ Paging Enabled
☐ Direction Lock Enabled
Bounce ✓ Bounces
☐ Bounce Horizontally
☐ Bounce Vertically
Zoom 1 Min 1 Max
Touch ✓ Bounces Zoom
✓ Delays Content Touches
✓ Cancellable Content Touches
Keyboard Do not dismiss

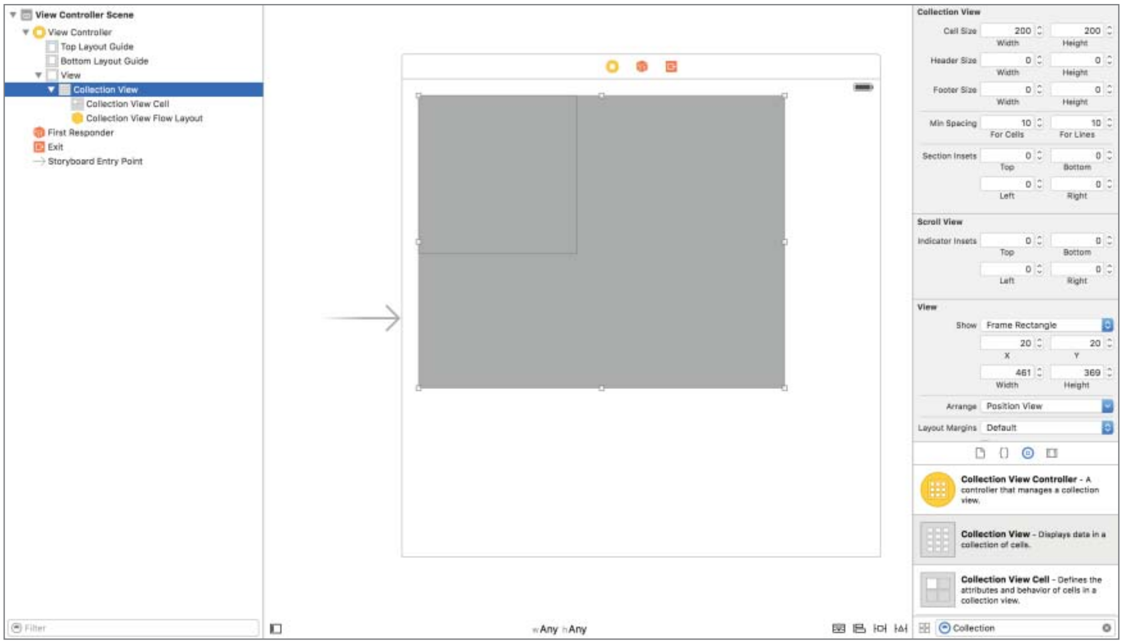
View
Mode Scale To Fill
Semantic Unspecified

FIGURE 17-3

COLLECTION VIEW CELLS

Each item in a collection view is an instance of another `UIKit` class called `UICollectionViewCell`. The collection view has a mechanism in place that allows you to reuse collection view cells instead of creating one for each item.

When you add a collection view to your storyboard, the collection view has a default cell of dimensions 50 x 50 units. Select the collection view in the document outline, and use the Size Inspector to edit the dimensions of the cell and the spacing between cells (see Figure 17-4).



View Controller Scene
View Controller
Top Layout Guide
Bottom Layout Guide
View
Collection View
Collection View Cell
Collection View Flow Layout
First Responder
Exit
Storyboard Entry Point

Collection View
Cell Size 200 Width, 200 Height
Header Size 0 Width, 0 Height
Footer Size 0 Width, 0 Height
Min Spacing 10 For Cells, 10 For Lines
Section Insets 0 Top, 0 Bottom, 0 Left, 0 Right

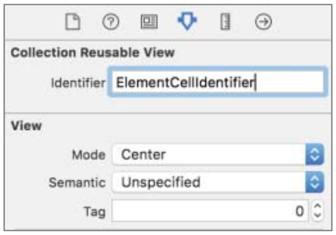
Scroll View
Indicator Insets 0 Top, 0 Bottom, 0 Left, 0 Right

View
Show Frame Rectangle
20 X, 20 Y
461 Width, 369 Height
Arrange Position View
Layout Margins Default

Collection View Controller - A controller that manages a collection view.
Collection View - Displays data in a collection of cells.
Collection View Cell - Defines the attributes and behavior of cells in a collection view.

FIGURE 17-4

Select the collection view cell (not the collection view) in the document outline and switch to the Attribute inspector to set up a unique identifier that will be used to access this cell programmatically (see Figure 17-5).

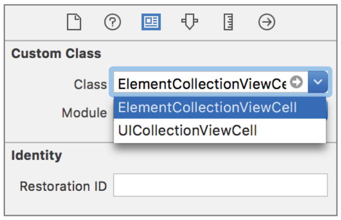


Collection Reusable View
Identifier ElementCellIdentifier

View
Mode Center
Semantic Unspecified
Tag 0

FIGURE 17-5

You will also need to create a custom `UICollectionViewCell` subclass and associate it with the collection view cell using the Identity Inspector (see Figure 17-6).



Custom Class
Class ElementCollectionViewCe
Module

ElementCollectionViewCell
UICollectionViewCell

Identity
Restoration ID

FIGURE 17-6

Unlike table view cells, the default collection view cell is empty. You will need to build a layout using standard `UIKit` elements such as labels and image views, and set up layout constraints within the cell.

Once a custom `UICollectionViewCell` subclass is associated with the collection view cell in the storyboard, you can create outlets and actions for the elements within the cell in this class.

COLLECTION VIEW DELEGATE AND DATA SOURCE

The `UICollectionViewDataSource` and `UICollectionViewDelegate` protocols define several methods, most of which are optional. The most common `UICollectionViewDataSource` methods you are likely to implement are:

```
func collectionView(collectionView: UICollectionView,
numberOfItemsInSection section: Int) -> Int

func collectionView(collectionView: UICollectionView,
cellForItemAtIndexPath indexPath: NSIndexPath) ->
UICollectionViewCell

func numberOfSectionsInCollectionView(collectionView: UICollectionView) -> Int
```

You should return the number of sections in your collection view from the `numberOfSectionsInCollectionView` method. If this method is not implemented, the collection view assumes a single section. The number of items in each section are to be returned from `collectionView(collectionView, numberOfItemsInSection)`, and a `UICollectionViewCell` instance for each row within each section is to be returned from `collectionView(collectionView, cellForItemAtIndexPath)`.

The most common `UICollectionViewDelegate` method that you are likely to implement is:

```
func collectionView(collectionView: UICollectionView,
didSelectItemAtIndexPath indexPath: NSIndexPath)
```

This method is called when the user selects an item in your collection view. If, however, you have created segues in your storyboard from your table view to another scene, then `prepareForSegue` is called instead.

If, on the other hand, you are using a collection view controller in your storyboard, then you will need to associate a custom `UICollectionViewController` subclass with the collection view controller in the Identity Inspector and implement the data source and delegate methods in this subclass.

The following code snippet shows how these data source and delegate methods are implemented in a view controller; this snippet assumes that `ElementCollectionViewCell` is a custom `UICollectionViewCell` subclass:

```
class ViewController: UIViewController,
                      UICollectionViewDataSource,
                      UICollectionViewDelegate {

    var elements:Array<String> = ["Hi", "He", "Li", "Be", "B", "C",
                                  "N", "O", "F" ,"Ne", "Na", "Mg",
                                  "Al", "Si", "P", "S", "Cl", "Ar",
                                  "K", "Ca"]

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

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }
    func collectionView(collectionView: UICollectionView,
         numberOfItemsInSection section: Int) -> Int
    {
        return elements.count;
    }


    func collectionView(collectionView: UICollectionView,
                        cellForItemAtIndexPath indexPath: NSIndexPath) ->
                        UICollectionViewCell
    {
        var cell: ElementCollectionViewCell =
collectionView.dequeueReusableCellWithReuseIdentifier("ElementCell",
forIndexPath:indexPath) as! ElementCollectionViewCell

        var elementName:String = elements[indexPath.row]
        cell.elementImage.image = UIImage(named: elementName)

        return cell
    }

    func numberOfSectionsInCollectionView(collectionView: UICollectionView) -> Int
    {
        return 1;
    }

}
```

TRY IT

In this Try It, you create a new Xcode project based on the Single View Application template called `CollectionViewTest` that adds a collection view with three sections to an existing scene of the default storyboard. The sections display a few elements from the periodic table sorted into solids, liquids, and gases.

Lesson Requirements

- Launch Xcode.
- Create a new project based on the Single View Application template.
- Edit the storyboard with Interface Builder.
- Add a collection view to the default scene.
- Set up a collection view cell.
- Set up collection view attributes with the Attribute inspector.
- Set up the data source and delegate property of the table view.
- Implement `UICollectionViewDataSource` and `UICollectionViewDelegate` methods.

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

Hints

- To show the Object library, select View ➪ Utilities ➪ Show Object Library.
- To show the assistant editor, select View ➪ Assistant Editor ➪ Show Assistant Editor.

Step-by-Step

- Create a Single View Application in Xcode called `CollectionViewTest`.
    1. Launch Xcode and create a new application by selecting File ➪ New ➪ Project.
    2. Select the Single View Application template from the list of iOS project templates.
    3. In the project options screen, use the following values:
        - Product Name: CollectionViewTest
        - Organization Name: your company
        - Organization Identifier: com.yourcompany
        - Language: Swift
        - Devices: iPhone
        - Use Core Data: Unchecked
    4. Save the project onto your hard disk.
- Add image resources to your project.
    1. Ensure the project navigator is visible. To show it, select View ➪ Navigators ➪ Show Project Navigator.
    2. Open the `Assets.xcassets` file by clicking it in the project navigator.
    3. Navigate to the `Images` folder in this chapter's resources from the website.
    4. Create a new Image set by selecting Editor ➪ New Image Set, and name this new image set `Al`.
    5. Drag the `Al_1x.png`, `Al_2x.png`, and `Al_3x.png` images from this chapter's resources into the appropriate placeholders in the image set.
    6. Similarly, create new image sets called `F`, `Hg`, `Li`, `N`, `O`, and `Si`, and use the appropriate images from this chapter's resources folder.

- Add a `UICollectionView` instance to the default scene.
  1. From the Object library, drag and drop a Collection View object onto the scene.
  2. Ensure the collection 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.
  3. 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.
  4. Set up the data source and delegate properties.
     - Right-click the collection view to bring up a context menu. Drag from the item labeled “dataSource” in the context menu to the item labeled “View Controller” in the document outline.
     - Right-click the collection view to bring up a context menu. Drag from the item labeled “delegate” in the context menu to the item labeled “View Controller” in the document outline.
- Set up the collection view’s appearance.
  1. Select the collection view and ensure the Attribute inspector is visible.
  2. Ensure the Layout attribute is set to Flow.
  3. Ensure the Scroll Direction attribute is set to Vertical.
  4. Ensure the Section Header check box is unchecked.
  5. Ensure the collection view is selected, and switch to the Size Inspector to set the height and width of the collection view cell to 150 units each.
  6. Set the Top Section Inset to 10 units.
- Add a `UICollectionViewCell` subclass.
  1. Option-click the `CollectionViewTest` group in the project explorer and select New File from the context menu.
  2. Select Swift File under the iOS Templates section.
  3. Name the file `ElementCollectionViewCell.swift` and click Create.

4. Modify the contents of the `ElementCollectionViewCell.swift` file to subclass `UICollectionViewCell`. The modified contents of this file should resemble the following:

```
import UIKit

class ElementCollectionViewCell: UICollectionViewCell {

}
```

- Set up the collection view cell.

1. Expand the collection view in the document outline; this will reveal the collection view cell.
2. Select the collection view cell.
3. Use the attribute editor to ensure that the value of the identifier attribute is `ElementCellIdentifier`.
4. Drag and drop an Image view onto the collection view cell.
5. Ensure the image 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 0.
    - Click the Add 4 Constraints button to dismiss the constraints editor popup.
6. Update the frames to match the constraints you have set.
    - Select the collection view cell in the document outline. This will now be listed as `ElementCellIdentifier`.
    - Select Editor ➪ Resolve Auto Layout Issues ➪ Update Frames.
7. Select the collection view cell, and use the Identity Inspector to change the custom class of the cell to `ElementCollectionViewCell`.
8. Ensure the assistant editor is visible and the `ElementCollectionViewCell.Swift` file is open in it.
9. Select the image view in the document outline and create an outlet for the image view in the `ElementCollectionViewCell.Swift` file. Name the outlet `imageView`.

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

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

```
var statesOfMatter:Array<String> = ["Solid", "Liquid", "Gas"]
var solids:Array<String> = ["Li", "Al", "Si"]

var liquids:Array<String> = ["Hg"]

var gasses:Array<String> = ["N", "O", "F"]
```

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

```
func numberOfSectionsInCollectionView(collectionView: UICollectionView)
     -> Int
{
    return statesOfMatter.count;
}
```

3. Implement the `collectionView(collectionView, numberOfItemsInSection)` data source method as follows:

```
func collectionView(collectionView: UICollectionView,
numberOfItemsInSection section: Int) -> Int
{
    if section == 0
    {
        return solids.count
    }
    else if section == 1
    {
        return liquids.count
    }
    else if section == 2
    {
        return gasses.count
    }

    return 0
}
```

4. Implement the `collectionView(collectionView, cellForItemAtIndexPath)` data source method as follows:

```
func collectionView(collectionView: UICollectionView,
cellForItemAtIndexPath indexPath: NSIndexPath) ->
UICollectionViewCell
{
    var section =  indexPath.section
    var row = indexPath.row

    var cell: ElementCollectionViewCell =
collectionView.dequeueReusableCellWithReuseIdentifier
("ElementCellIdentifier", forIndexPath:indexPath)
as! ElementCollectionViewCell

    if section == 0
    {
        var elementName:String = solids[indexPath.row]
        cell.imageView.image = UIImage(named: elementName)
    }
```

```
        else if section == 1
        {
            var elementName:String = liquids[indexPath.row]
            cell.imageView.image = UIImage(named: elementName)
        }
        else if section == 2
        {
            var elementName:String = gasses[indexPath.row]
            cell.imageView.image = UIImage(named: elementName)
        }

        return cell
    }
```

5. Modify the declaration of the `ViewController` class to inherit from `UIViewController`, `UICollectionViewDataSource`, and `UICollectionViewDelegate`:

```
class ViewController: UIViewController,
                      UICollectionViewDataSource,
                      UICollectionViewDelegate {
```

➤ 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 17 video online at* `www.wrox.com/go/swiftiosvid`.