Collection View Controller

The easiest way to incorporate a Collection View into your app is to use a Collection View Controller. A Collection View Controller is a screen that has only a Collection View on it and nothing else. The exception is that a Collection View Controller may also be embedded within a Tab Bar Controller, which will add tabs at the bottom; or it could be embedded within a Navigation Controller, which will add a Back button, title, and other items at the top.

In this app, you'll make a Collection View to show deaf sign language signs in a grid. You'll need to download the zip file that has the sign language signs.

Part 1: Setting Up the Storyboard

1. Make a new project and call it **SignLanguage**. iOS, single view app, Swift, Storyboard, etc.

2. Drag a new **UICollectionViewController** into your storyboard. Set the UICollectionViewController to be the **initial view controller** either by checking the box in the Attributes Inspector or my dragging the arrow from the other view controller. (The "initial view controller" is the starting point for your app.)

3. Use Cmd-N or "New File" to add a new file to your project. The file should be a **Cocoa Touch Class** that inherits from UICollectionViewController. Call the new class **SignCollectionViewController**.

4. Open **Main.storyboard** in Interface Builder. Note the Collection View Controller has a Collection View inside it, and that has a Collection View Cell inside it. We need to set up the cell.

5. Use the document outline to select the collection view inside the collection view controller, then go to the size inspector and set Cell Size to have the width 140 and height 180. Now set the section insets for top, bottom, left and right to all be 10.

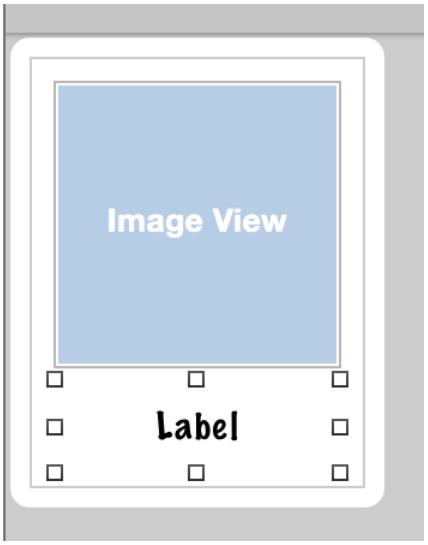
Collection views are extremely similar to table views, with the exception that they display as grids rather than as simple rows. But while the display is different, the underlying method calls are similar.

Our collection view already has one prototype cell, which is the empty square you'll see in the top-left corner. This works the same as with table.

6. Select that collection view cell now, then go to the attributes inspector: change its Background from "Default" (transparent) to white and give it the identifier "Sign" so that we can reference it

in code. Now place a **UIImageView** in there, with X:10, Y:10, width 120 and height 120. We'll be using this to show pictures of people's faces. You'll also need to set constraints for these. Set the width and height to 120. Set the top, right, and left I-beams to 10.

7. Place a **UILabel** in there too, with X:10, Y:134, width 120 and height 40. Also set constraints. Use width 120, height 40, and bottom, right, and left I-beams to 10.



We need to create a custom class for our cell. This is needed because our collection view cell has two views that we created – the image view and the label – and we need a way to manipulate this in code.

8. Go to the File menu and choose New > File, then select iOS > Source > **Cocoa Touch Class** and click Next. You'll be asked to fill in two text fields: where it says "Subclass of" you should enter **UICollectionViewCell**, and where it says "Class" enter **SignCell**. Click Next then Create, and Xcode will create a new class called SignCell that inherits from UICollectionViewCell.

```
class SignCell: UICollectionViewCell {
}
```

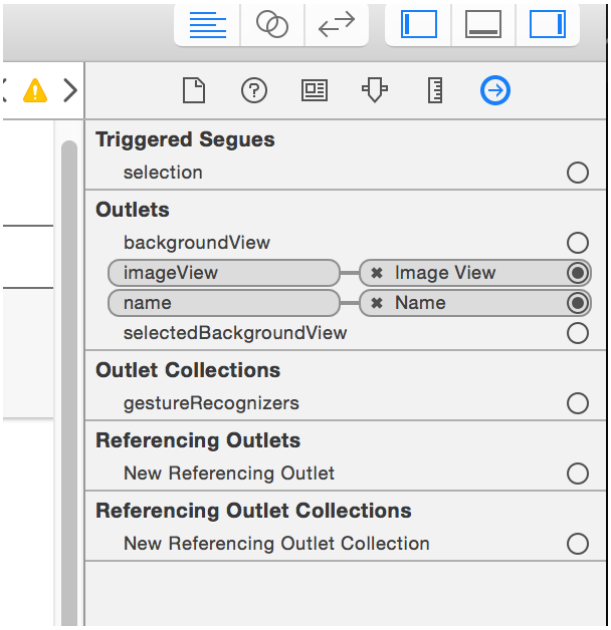
9. This new class needs to be able to represent the collection view layout we just defined in Interface Builder, so it just needs two outlets. Give the class these two properties. It's okay to just type these in. Let Xcode help you with the capitalization for words like IBOutlet.

```
@IBOutlet var imageView: UIImageView!
@IBOutlet var name: UILabel!
```

10. Now go back to Interface Builder and select the collection view cell in the document outline. Select the Identity Inspector and you'll see next to Class the word "UICollectionViewCell" in gray text. That's telling us that the cell is its default class type.

11. We want to use our custom class here, so drop down the menu and select "SignCell." You'll see that "SignCell" now appears in the document outline.

12. Now that Interface Builder knows that the cell is actually a SignCell, we can connect its outlets. Go to the **Connections Inspector** (it's the last one) with the cell selected and you'll see imageView and name in there, both with empty circles to their right. That empty circle has exactly the same meaning as when you saw it with outlets in code: there is no connection between the storyboard and code for this outlet.



13. To make a connection from the Connections Inspector, just click on the empty circle next to imageView and drag a line over the view you want to connect. In our case, that means dragging over the image view in our custom cell. Now connect name to the label, and you're done with the storyboard.

Part 2: Creating a Data Source

Like a Table View, a Collection View needs a data source that can answer three questions:

1. How many sections are there in the collection?

```
numberOfSections(in collectionView: UICollectionView) -> Int
```

2. How many items are there in this section?

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

3. What is this particular item?

```
collectionView(_ collectionView: UICollectionView, cellForItemAt
indexPath: IndexPath) -> UICollectionViewCell
```

We have fingerspelling signs for all 26 letters of the alphabet, and ASL signs for four more, for a total of 30 signs.

1. Open the Assets folder and drag these 30 images into the area below the Appicon.

2. For a data source, we will create an array of the available signs. Create a new Swift File and call it Signs.swift. Below the line import Foundation, add this array of all the signs we have. You should type this by hand because the quotation marks may not copy and paste properly from this document into your code.

```
let signs = [
    "A", "B", "C", “D”, "E", "F", "G", “H”,
    "I", "J", "K", "L", "M", "N", "O", “P”, “Q”,
    "R", "S", "T", "U", "V", "W", "X", "Y", “Z”,
    “Help”, “Mother”, “Stop”, “Water”
]
```

3. In the SignCollectionViewController code, add this code.

Number of sections:

```
return 1
```

Number of items in section:

```
return signs.count
```

Cell for item at index path: Here we have to put together the code that sets up the image and the label. There is a possibility of crashing if things go wrong. I used code something like this:

```
if let cell = collectionView.dequeueReusableCell(
    withReuseIdentifier: "Sign", for: indexPath)
        as? SignCell {
    cell.imageView.image = UIImage(
           named: signs[indexPath.row].lowercased())
    cell.name.text = signs[indexPath.row]
    return cell
}

return UICollectionViewCell()
```

indexPath.row is just the cell number, which is also the index into our array of signs.

Note that the signs all have all lower case letters, but we want to display upper case letters in our captions. So the list of signs is all upper case, but we convert them to lower case before trying to find the image.

Without that last line, the build will fail, because this function must return a value. But what if the "if let" fails? In that case, we have nothing to return, and the app might crash. This should never happen if our code is correct, because we have a valid list of signs and names. So I just return an empty UICollectionViewCell, which should prevent a crash if something strange happens.

Part 3: Collection View and Segues

What if you want to have a Collection View in your storyboard but you also want other items, such as a caption at the top or a search bar? Then you'd need to insert a Collection View into a regular View Controller. This requires only one extra step. We can explore this in another tutorial later.

Also, we might want to press a Collection View Cell to initiate a segue to another view controller that shows detail. We can explore this in another tutorial later.