

## Announcement

- **Lab 2 is due tonight by 11:59 PM**
- **Lab 3 is posted**
  - Due Monday July 9<sup>th</sup>

## Today's Topics

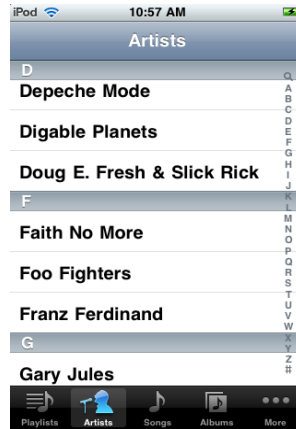
- **Scroll views**
- **Table views**
  - Displaying data
  - Controlling appearance & behavior
- **UITableViewController**
- **Table view cells**
- **Collection views**

# Scroll Views

## UIScrollView

- **For displaying more content than can fit on the screen**
- **Handles gestures for panning and zooming**
- **Noteworthy subclasses: UITableView and UITextView**

## Scrolling Examples

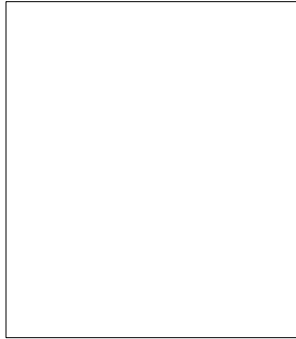


## Using a Scroll View

- **Create with the desired frame**  
let theFrame = CGRect(x:0, y:0, width:200, height:200)  
let scrollView = UIScrollView(frame: theFrame)
- **Add subviews (frames may extend beyond scroll view frame)**  
let bigFrame = CGRect(x:0, y:0, width:500, height:500)  
let myImageView = UIImageView(frame: bigFrame)  
scrollView.addSubview(myImageView)
- **Set the content size**
  - scrollView.contentSize = CGSize(width: 500, height: 500)

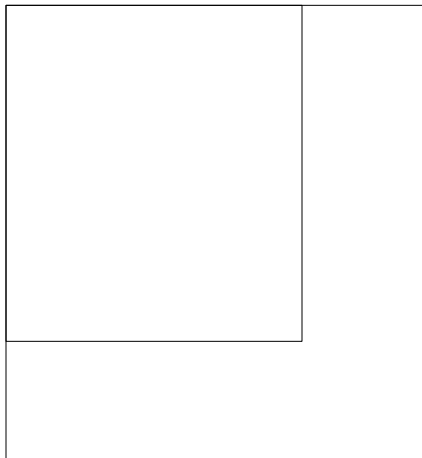
## Frame and Content

`scrollView.frame`



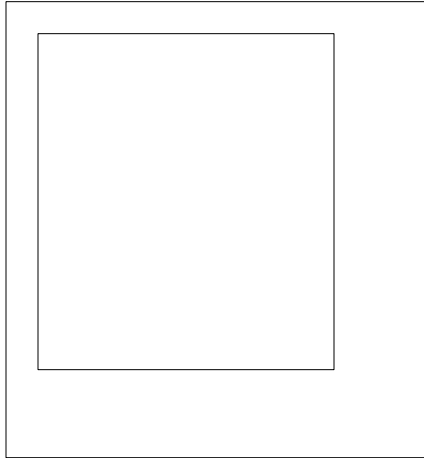
## Frame and Content

`scrollView.contentSize`



## Frame and Content

`scrollView.contentOffset`



## Demo

Using a UIScrollView

## Extending Scroll View Behavior

- **Applications often want to know about scroll events**
  - When the scroll offset is changed
  - When dragging begins & ends
  - When deceleration begins & ends

## Extending with a Subclass

- **Create a subclass**
- **Override methods to customize behavior**
- **Issues with this approach**
  - Application logic and behavior is now part of a View class
  - Tedious to write a one-off subclass for every scroll view instance
  - Your code becomes tightly coupled with superclass

## Extending with Delegation

- **Delegate is a separate object**
- **Clearly defined points of responsibility**
  - Change behavior
  - Customize appearance
- **Loosely coupled with the object being extended**

## UIScrollView Delegate

```
public protocol UIScrollViewDelegate: NSObjectProtocol

// Respond to interesting events
optional public func scrollViewDidScroll(_ scrollView: UIScrollView)
optional public func scrollViewDidZoom(_ scrollView: UIScrollView)
...
// Influence behavior
// return a yes if you want to scroll to the top. if not defined, assumes YES
optional public func scrollViewShouldScrollToTop(_ scrollView: UIScrollView) -> Bool
```

## Implementing a Delegate

- **Conform to the delegate protocol**

```
class ViewController: UIViewController, UIScrollViewDelegate {
```

- **Implement all required methods and any optional methods**

```
func scrollViewDidScroll(_ scrollView: UIScrollView){  
// Do something in response to the new scroll position  
if (scrollView.contentOffset ...) {  
  
    }  
}
```

## Zooming with a Scroll View

- **Set the minimum, maximum, initial zoom scales**

```
scrollView.maximumZoomScale = 5.0
```

```
scrollView.minimumZoomScale = 1.0
```

- **Implement delegate method for zooming**

```
func viewForZooming(in scrollView: UIScrollView) -> UIView? {  
    return someViewThatWillBeScaled  
}
```



## Demo

Zooming with a UIScrollView

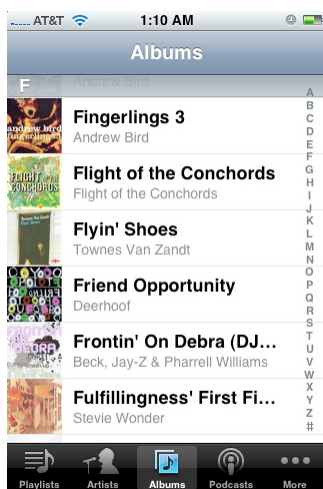
## Table Views

## Table Views

- **Display lists of content**
  - Single column, multiple rows
  - Vertical scrolling
  - Large data sets
  
- **Powerful and ubiquitous in iPhone applications**

## Table View Styles

### UITableViewStylePlain

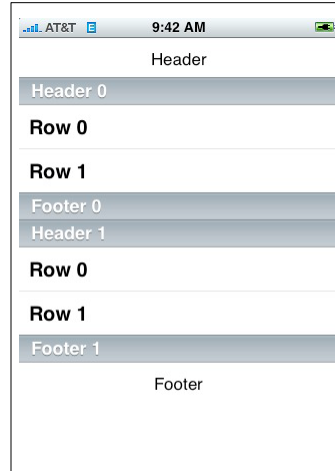


### UITableViewStyleGrouped



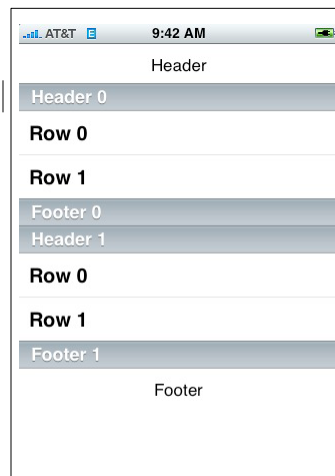
## Table View Anatomy – Plain Style

Table Header ->



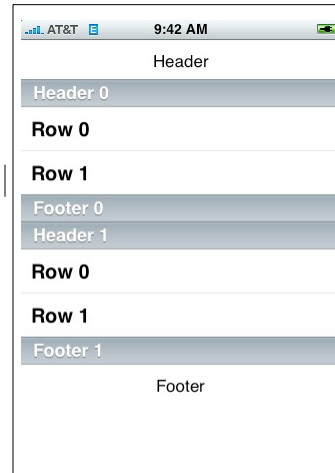
## Table View Anatomy – Plain Style

Section Header ->



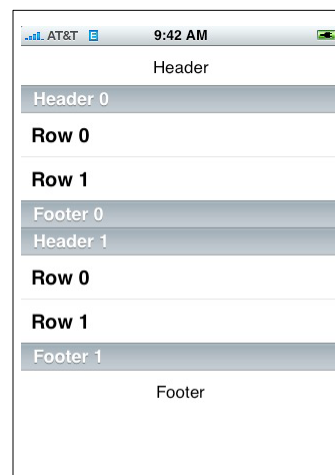
## Table View Anatomy – Plain Style

Table Cell->



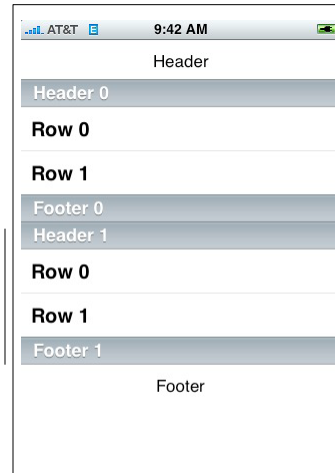
## Table View Anatomy – Plain Style

Section Footer ->



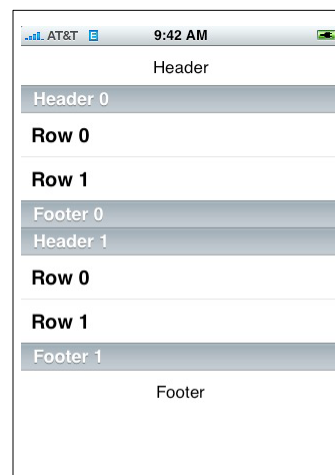
## Table View Anatomy – Plain Style

Section ->



## Table View Anatomy – Plain Style

Table Footer->



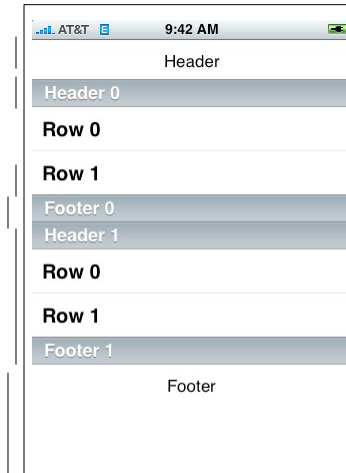
## Table View Anatomy – Plain Style

Table Header ->  
Section Header ->

Table Cell->  
Section Footer ->

Section ->

Table Footer->



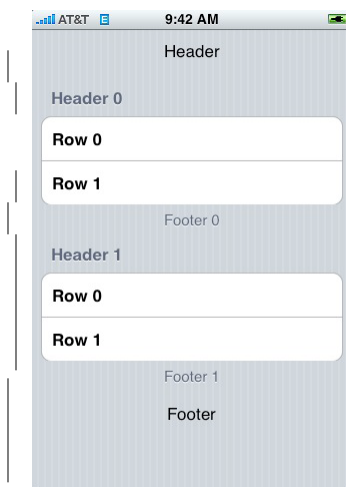
## Table View Anatomy – Grouped Style

Table Header ->  
Section Header ->

Table Cell->  
Section Footer ->

Section ->

Table Footer->



# Displaying Data in a Table View

## A Naïve Solution

- **Table views display a list of data, so use an array**  
myTableView.setList(myListOfStuff)
- **Issues with this approach**
  - All data is loaded upfront
  - All data stays in memory

## A More Flexible Solution

- **Another object provides data to the table view**
  - Not all at once
  - Just as it's needed for display
- **Like a delegate, but purely data-oriented**

## UITableViewDataSource

- **Provide number of sections and rows**

```
// Optional method, defaults to 1 if not implemented  
func numberOfSections(in tableView: UITableView) -> Int
```

```
// Required method  
func tableView(_ tableView: UITableView, numberOfRowsInSection:  
section: Int) -> Int
```

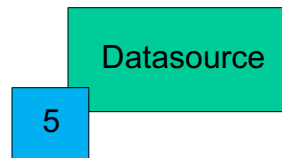
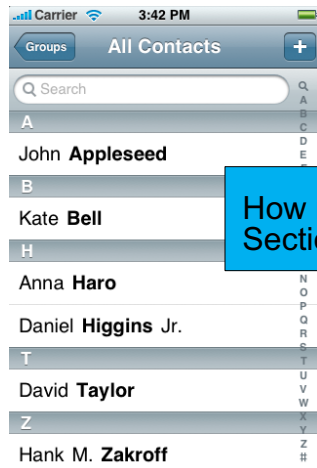
- **Provide cells for table view as needed**

```
// Required method  
func tableView(_ tableView: UITableView, cellForRowAt indexPath:  
IndexPath) -> UITableViewCell
```



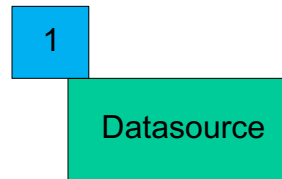
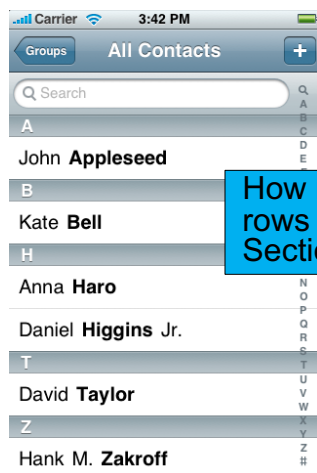
## Datasource Message Flow

`numberOfSections(in tableView: UITableView) -> Int`



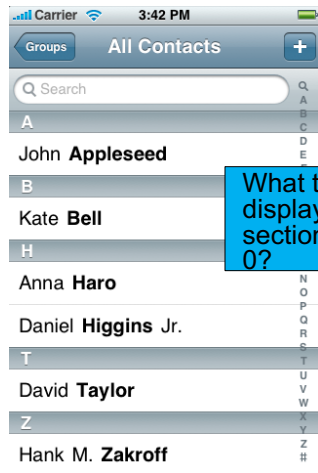
## Update Datasource Message Flow

`tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int`



## Datasource Message Flow

`tableView(_ tableView: UITableView, cellForRowAtIndexPath indexPath: IndexPath) -> UITableViewCell`



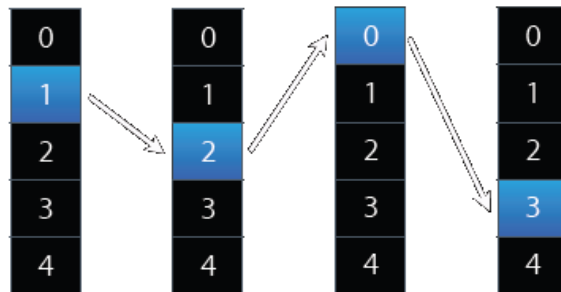
What to display at section 0, row 0?

“John Appleseed”

Datasource

## IndexPath

- Generic class in Foundation
- Path to a specific node in a tree of nested arrays



## Single Section Table View

### Return the number of rows

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

### Provide a cell when requested

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {  
    let cell = ...  
    cell.textLabel.text = myStrings[indexPath.row]  
    return cell  
}
```

## Creating Cells without reuse

```
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {  
    UITableViewCell {  
  
        let myCell = UITableViewCell(style: .default, reuseIdentifier: nil)  
  
        myCell.textLabel.text = myStrings[indexPath.row]  
  
        return myCell  
    }  
}
```

## Cell Reuse

- **When asked for a cell, it would be expensive to create a new cell each time.**
  - First register the class for use in creating new cells

```
override func viewDidLoad(){
...
    tableView.register (UITableViewCell.self, forCellReuseIdentifier: "theCell")
}

func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
    UITableViewCell {

let myCell = tableView.dequeueReusableCellWithIdentifier("theCell")! as UITableViewCell
    cell.textLabel.text = myStrings[indexPath.row]
    return cell
}
```

## Triggering Updates

- **When is the datasource asked for its data?**
  - When a row becomes visible
  - When an update is explicitly requested by calling - reloadData

```
override func viewWillAppear(animated: Bool) {
    super.viewWillAppear(animated)
    tableView.reloadData()
}
```

## Basic properties

- UITableViewCell has image and text properties

```
cell.imageView?.image = UIImage.init(named: "obama.png")  
cell.textLabel?.text = "Barack Obama"
```



**Barack Obama**

**Demo**

## UICollectionView

- **Convenient way to show information in a grid format**
- **Similar to a UITableView**
  - But with multiple columns
- **Flow based layout makes displaying rows and columns of data very easy**

## UICollectionView

```
func numberOfSections (in collectionView: UICollectionView) -> Int {
    //return number of sections
}

func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section:
Int) -> Int {
    //return number of items per section
}

func collectionView(_ collectionView: UICollectionView, cellForItemAtIndexPath indexPath:
IndexPath) -> UICollectionViewCell {

let cell = collectionView.dequeueReusableCell(withReuseIdentifier: "someCellNameId" for:
indexPath)
    // configure the cell
    // return UICollectionViewCell cell for item at IndexPath
    return cell
}
```

# UICollectionView Demo