

Announcements

- **Lab 3 is due next Monday at 11:59 PM**
- **Discuss Software Engineering Topics in the next two lectures**
- **Midterm is on Monday March 2nd**

Today's Topics

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

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

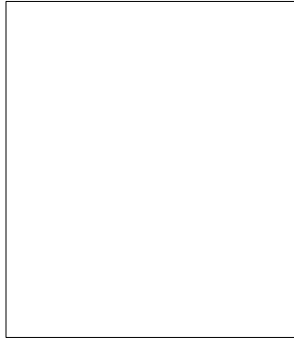


Using a Scroll View

- **Create with the desired frame**
 - `CGRect frame = CGRectMake(0, 0, 200, 200);`
 - `scrollView = [[UIScrollView alloc] initWithFrame:frame];`
- **Add subviews (frames may extend beyond scroll view frame)**
 - `frame = CGRectMake(0, 0, 500, 500);`
 - `myImageView = [[UIImageView alloc] initWithFrame:frame];`
 - `[scrollView addSubview:myImageView];`
- **Set the content**
 - `scrollView.contentSize = CGSizeMake(500, 500);`

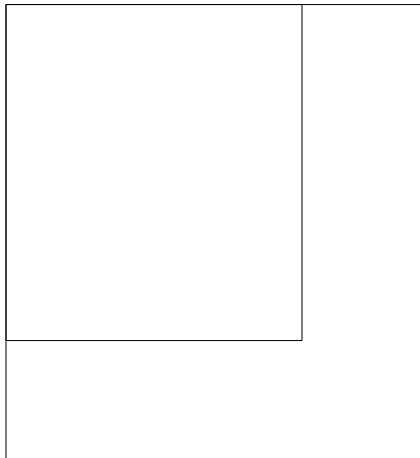
Frame and Content

scrollView.frame



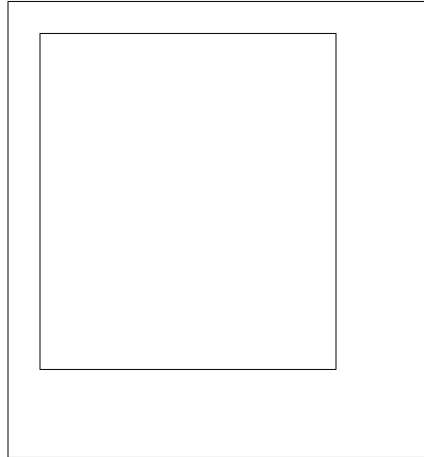
Frame and Content

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

```
@protocol UIScrollViewDelegate<NSObject>

@optional

// Respond to interesting events
- (void)scrollViewDidScroll:(UIScrollView *)scrollView;

...

// Influence behavior
- (BOOL)scrollViewShouldScrollToTop:(UIScrollView *)scrollView;

@end
```

Implementing a Delegate

- Conform to the delegate protocol

```
@interface MyController : NSObject <UIScrollViewDelegate>
```

- Implement all required methods and any optional methods

```
-(void)scrollViewDidScroll:(UIScrollView *)scrollView  
{  
    // 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 = 2.0;  
scrollView.minimumZoomScale = scrollView.size.width/  
    myImage.size.width;
```

- Implement delegate method for zooming

```
- (UIView *)viewForZoomingInScrollView:(UIView *)view {  
    return someViewThatWillBeScaled;  
}
```


Demo

Zooming with a UIScrollView

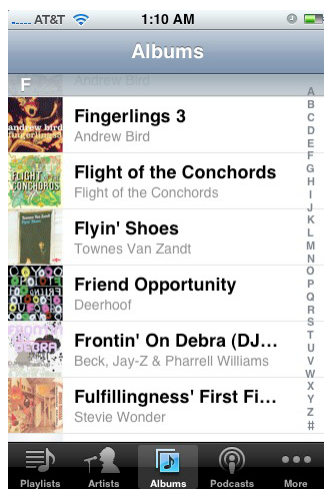
Table Views

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 ->

The diagram illustrates the internal structure of a UITableView in Plain Style. It shows a vertical stack of sections. Each section begins with a 'Header' row, followed by a series of 'Row' entries (labeled 'Row 0' and 'Row 1'), and concludes with a 'Footer' row. The rows are visually grouped by thin horizontal lines. The header and footer rows of each section are highlighted with a light gray background, while the data rows have a white background. The entire structure is contained within a frame that mimics an iPhone screen, complete with a status bar at the top showing 'AT&T' and '9:42 AM'.

Header
Header 0
Row 0
Row 1
Footer 0
Header 1
Row 0
Row 1
Footer 1
Footer

Table View Anatomy – Plain Style

Section Header ->

This diagram is identical to the one on the previous slide, showing the internal structure of a UITableView in Plain Style. It depicts a vertical stack of sections, each containing a header, data rows (labeled 'Row 0' and 'Row 1'), and a footer. The header and footer rows are highlighted in light gray, and the data rows are white. The entire structure is framed within an iPhone-like screen with a status bar at the top.

Header
Header 0
Row 0
Row 1
Footer 0
Header 1
Row 0
Row 1
Footer 1
Footer

Table View Anatomy – Plain Style

Table Cell->

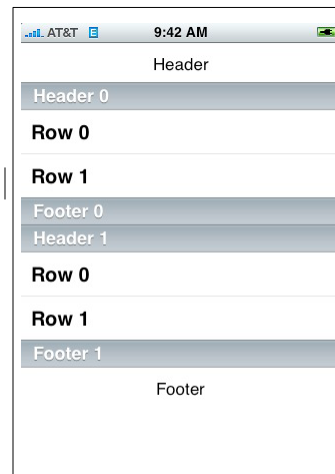


Table View Anatomy – Plain Style

Section Footer ->

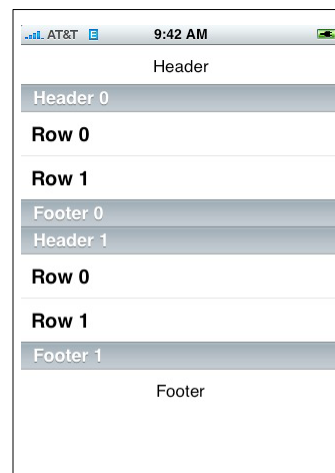


Table View Anatomy – Plain Style

Section ->

Header
Header 0
Row 0
Row 1
Footer 0
Header 1
Row 0
Row 1
Footer 1
Footer

Table View Anatomy – Plain Style

Table Footer->

Header
Header 0
Row 0
Row 1
Footer 0
Header 1
Row 0
Row 1
Footer 1
Footer

Table View Anatomy – Plain Style

Table Header ->
Section Header ->

Table Cell->
Section Footer ->

Section ->

Table Footer->

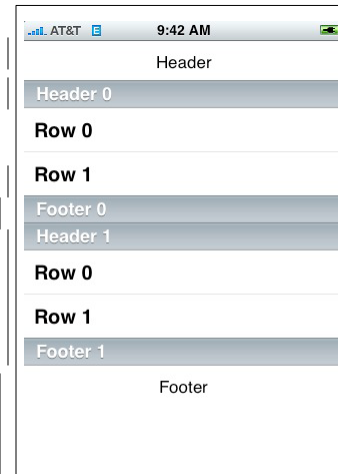


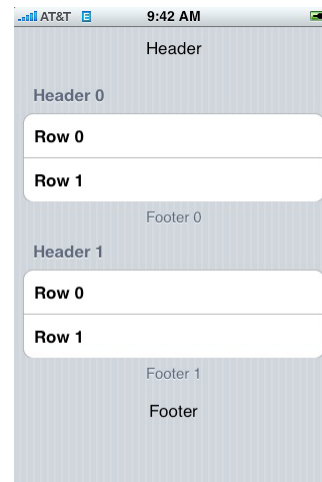
Table View Anatomy – Grouped Style

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
-(NSInteger)numberOfSectionsInTableView:(UITableView *)table;
```

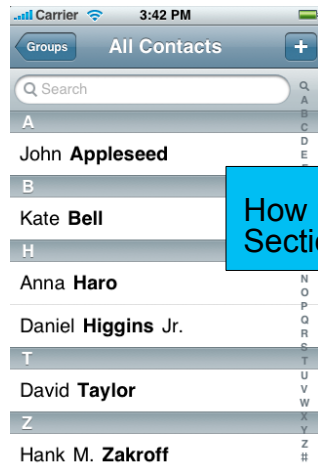
```
// Required method
-(NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section;
```

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

```
// Required method
-(UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath;
```


Datasource Message Flow

numberOfSectionsInTableView:



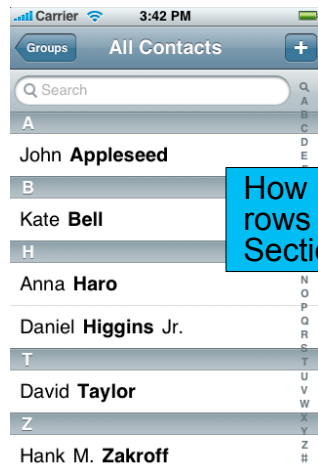
How many
Sections?

5

Datasource

Datasource Message Flow

tableView:numberOfRowsInSection:



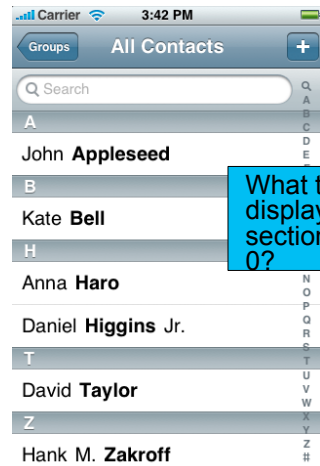
How many
rows in
Section 0?

1

Datasource

Datasource Message Flow

tableView:cellForRowAtIndexPath:



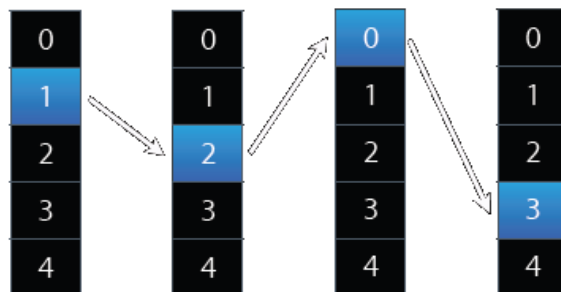
What to display at section 0, row 0?

"John Appleseed"

Datasource

NSIndexPath

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



NSIndexPath and Table Views

- Cell location described with an index path
 - Section index + row index
- Category on NSIndexPath with helper methods

```
@interface NSIndexPath (UITableView)
+ (NSIndexPath *)indexPathForRow:(NSInteger)row
    inSection:(NSInteger)section;

@property(nonatomic,readonly) NSInteger section;
@property(nonatomic,readonly) NSInteger row;

@end
```

Single Section Table View

Return the number of rows

```
- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section
{
    return [myStrings count];
}
```

Provide a cell when requested

```
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = ...;
    cell.textLabel.text = [myStrings objectAtIndex:indexPath.row];
    return [cell autorelease];
}
```

Cell Reuse

- When asked for a cell, it would be expensive to create a new cell each time.

```
- (UITableViewCell *)tableView:(UITableView *)tableView  
cellForRowAtIndexPath:(NSIndexPath *)indexPath  
{  
    UITableViewCell *cell = [tableView  
        dequeueReusableCellWithIdentifier:@"MyIdentifier"];  
  
    if (cell == nil) {  
        cell = [[UITableViewCell alloc]  
            initWithFrame:... reuseIdentifier:@"MyIdentifier"];  
    }  
    cell.textLabel.text = [myStrings objectAtIndex: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

```
- (void)viewWillAppear:(BOOL)animated  
{  
    [super viewWillAppear:animated];  
    [self.tableView reloadData];  
}
```

Simple Demo and Flickr Demo

Basic properties

- **UITableViewCell** has image and text properties

```
cell.image = [UIImage imageNamed:@"obama.png"];  
cell.textLabel.text = @"Barack Obama";
```



Barack Obama

Accessory Types

```
// UITableView delegate method
(UITableViewCellAccessoryType)tableView:(UITableView *)table
    accessoryTypeForRowAtIndexPath:(NSIndexPath *)indexPath

return UITableViewCellAccessoryDisclosureIndicator

return UITableViewCellAccessoryDetailDisclosureButton

return UITableViewCellAccessoryCheckmark

- (void)tableView:(UITableView *)tableView
    accessoryButtonTappedForRowWithIndexPath:(NSIndexPath *)indexPath
{
    // Only for the blue disclosure button
    NSUInteger row = indexPath.row;
    ...
}
```



Customizing the Content View

- For cases where a simple image + text cell doesn't suffice
- UITableViewCell has a content view property
 - Add additional views to the content view

```
-(UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = ...;
    CGRect frame = cell.contentView.bounds;

    UILabel *myLabel = [[UILabel alloc] initWithFrame:frame];
    myLabel.text = ...;
    [cell.contentView addSubview:myLabel];

    [myLabel release];
}
```

Custom Row Heights

- Rows in a table view may have variable heights
- `NSString` category in `UIStringDrawing.h` is very useful for computing text sizes

```
- (CGFloat)tableView:(UITableView *)tableView
heightForRowAtIndexPath:(NSIndexPath *)indexPath
{
    NSString *text = ...;
    UIFont *font = [UIFont systemFontOfSize:...];

    CGSize withinSize = CGSizeMake(tableView.width, 1000);

    CGSize size = [text sizeWithFont:font
                        constrainedToSize:withinSize
                        lineBreakMode:UILineBreakModeWordWrap];

    return size.height + somePadding;
}
```

Custom Cells Demo