

UITableView Overview

Brian Slick

contact@briterideas.com

What Do I Do?



- Founder of BriTer Ideas LLC
- Focused exclusively on iPhone development
- 1st app, SlickShopper, released on 5/28/09
- Available for contract development



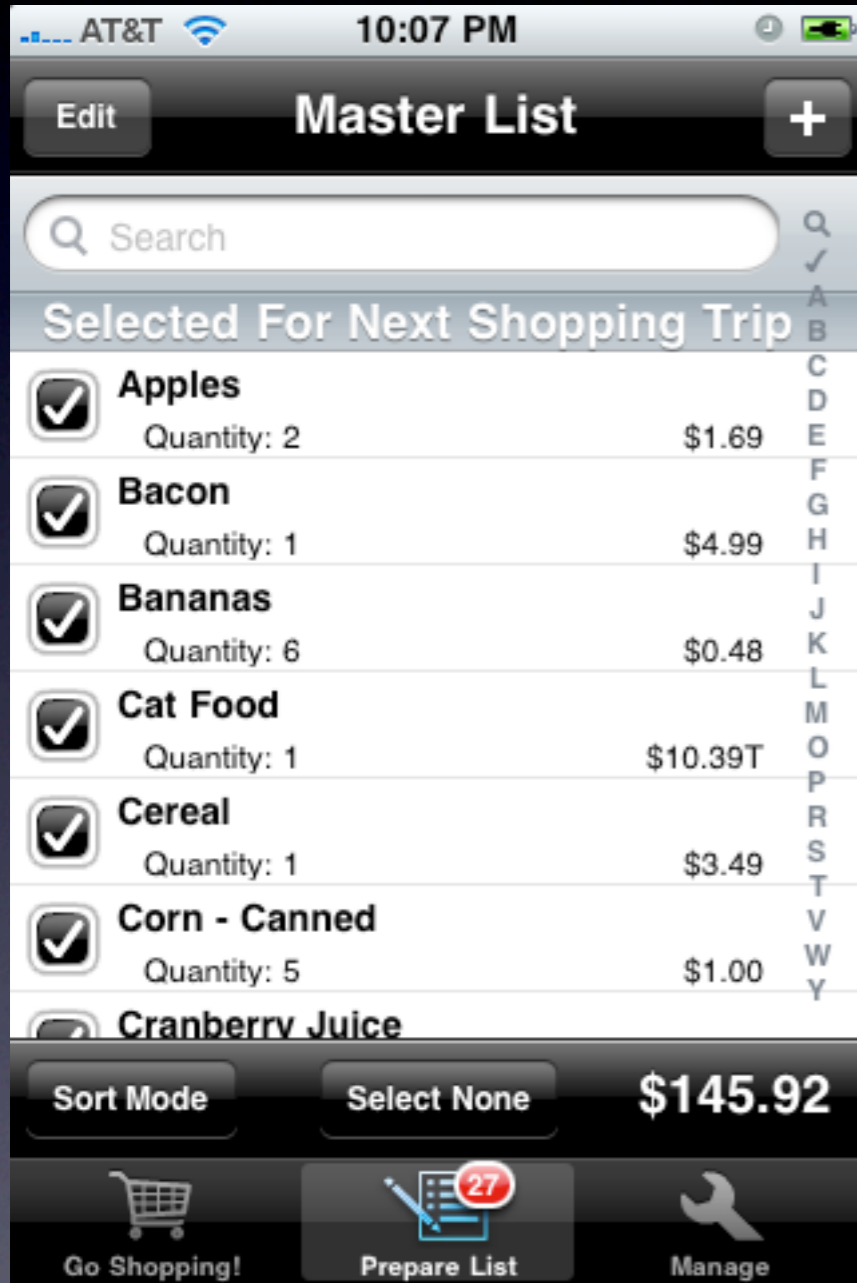
Topics

- Overview of UITableView
- Delegate and Data Source Methods
- UITableViewController
- Animation Gotcha

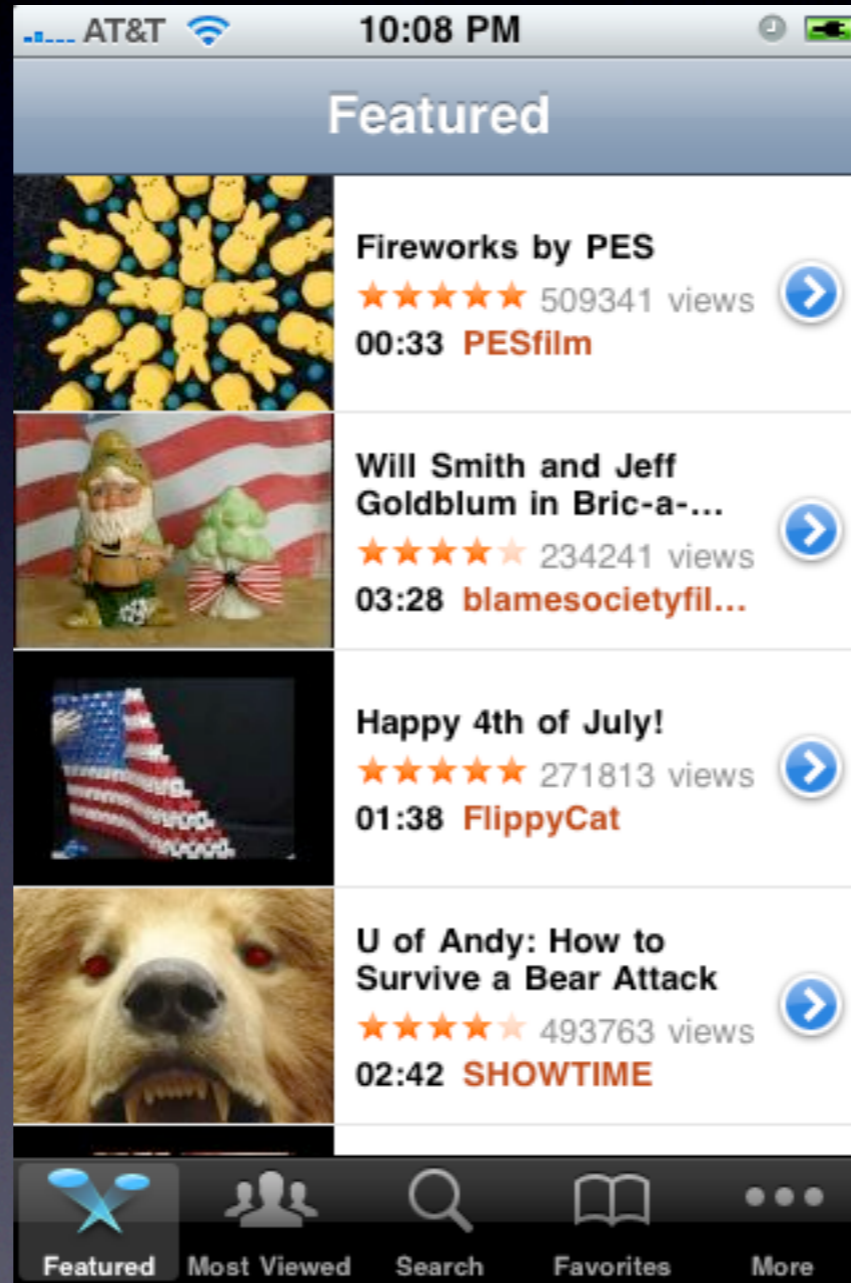
UITableView

- Same concept as NSTableView (Mac OS X)
- Display a large amount of data in a column
- Highly optimized for limited hardware
- Commonly used for navigation
- Can be placed in a UIViewController, or is automatically generated with a UITableViewController (discussed later)

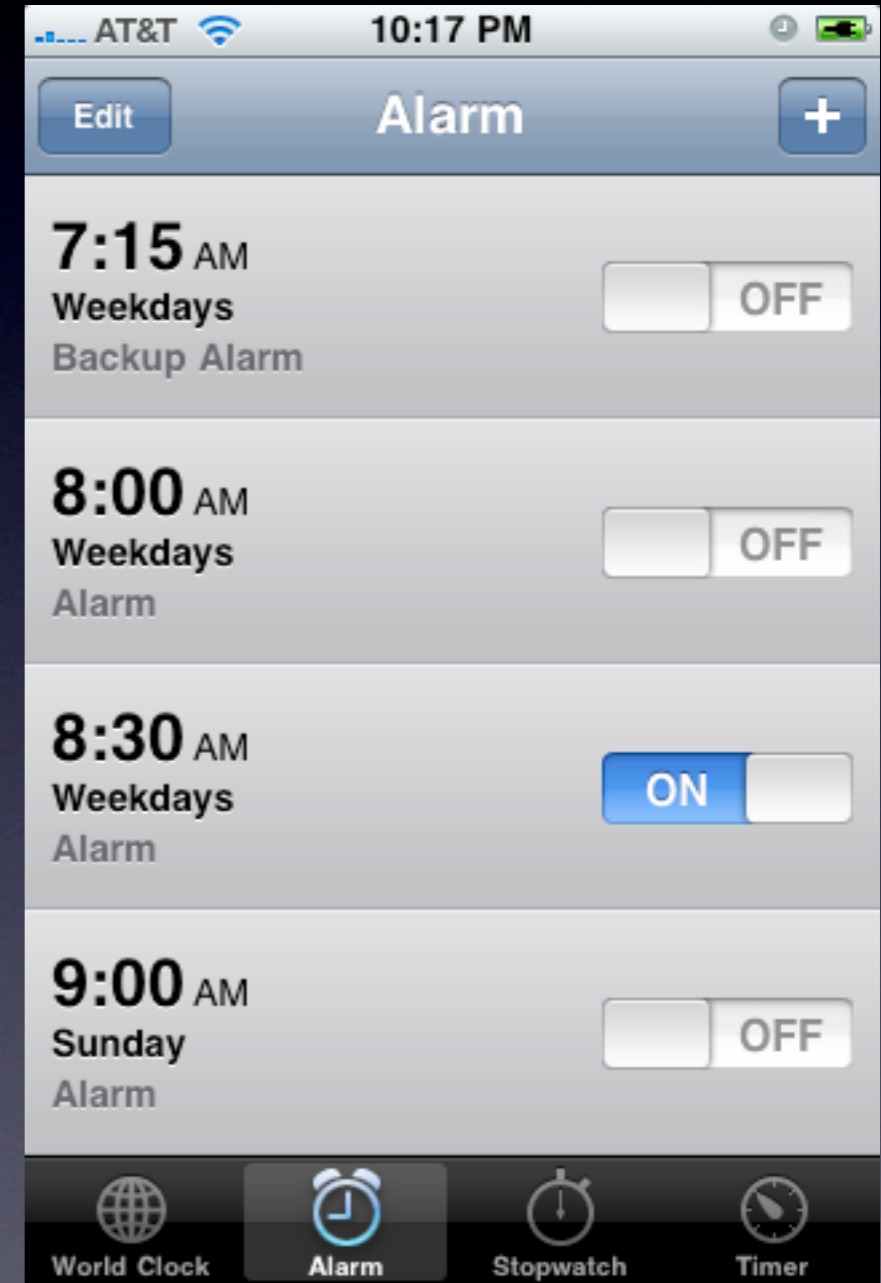
Examples



SlickShopper



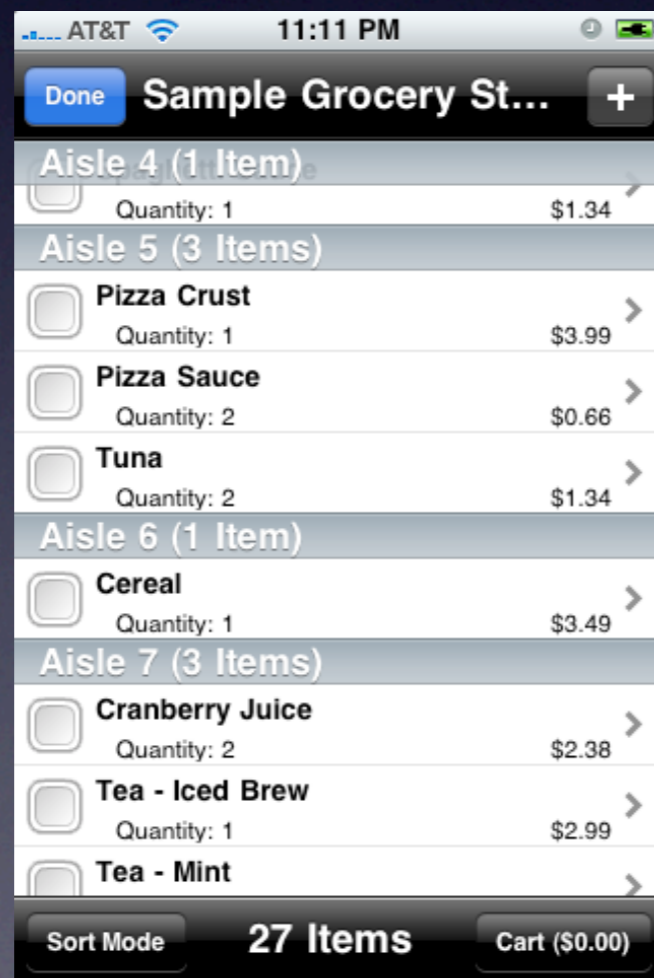
YouTube



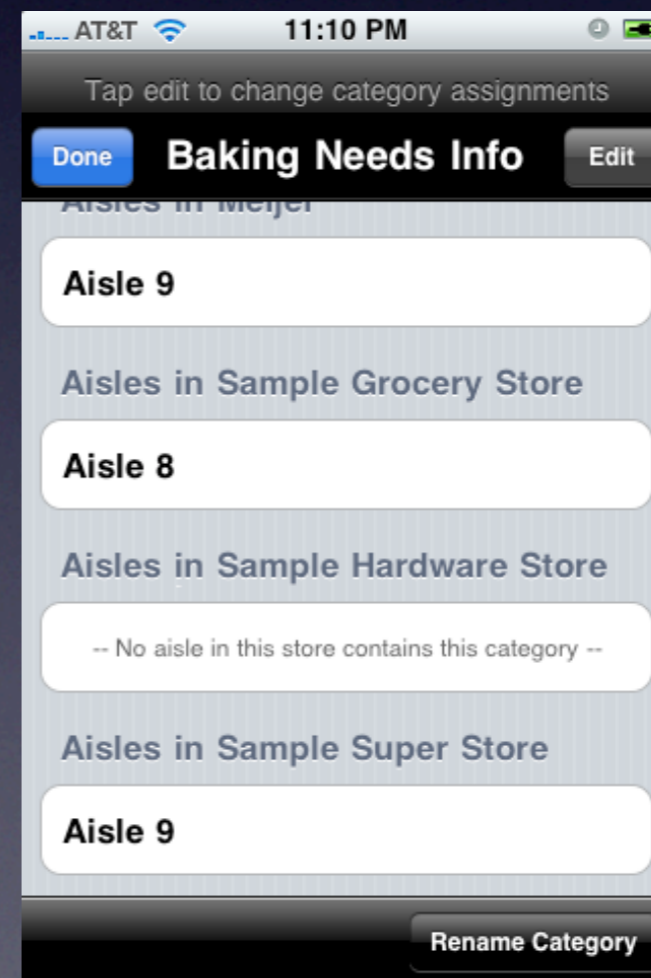
Clock

Style

- Two options: Plain, or Grouped
- Must be defined when creating instance (IB)
- Cannot be changed after initialization



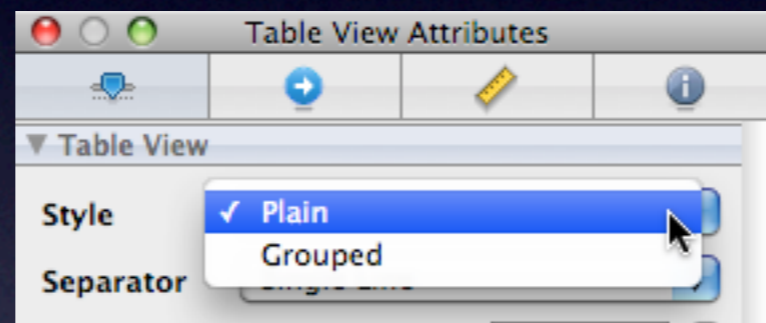
Plain



Grouped

Initializing

- In Interface Builder



- In code

```
UITableView *tableView = [[UITableView alloc]
    initWithFrame: CGRectZero // Provide a real frame
    style: UITableViewStylePlain];
    -- or --
    style: UITableViewStyleGrouped];
```

Populating

- Table Views do not store content
- They do ask (A LOT! of) questions of other object(s) to determine what to display
- They use OOP concept of delegation

Delegates

- UITableViewDataSource
 - Provides the content
 - Has required and optional methods
- UITableViewDelegate
 - Modifies behavior, appearance
 - Optional methods
- Typically the same object (view controller)

Protocol

- Your class must conform to the data source and/or delegate protocols:

```
// MyViewController.h
// SampleTable

#import <UIKit/UIKit.h>

@interface MyViewController : UIViewController <UITableViewDelegate, UITableViewDataSource>
{
}

}

@end
```

To Set Delegates In Code

```
// Implement viewDidLoad to do additional setup after loading the view

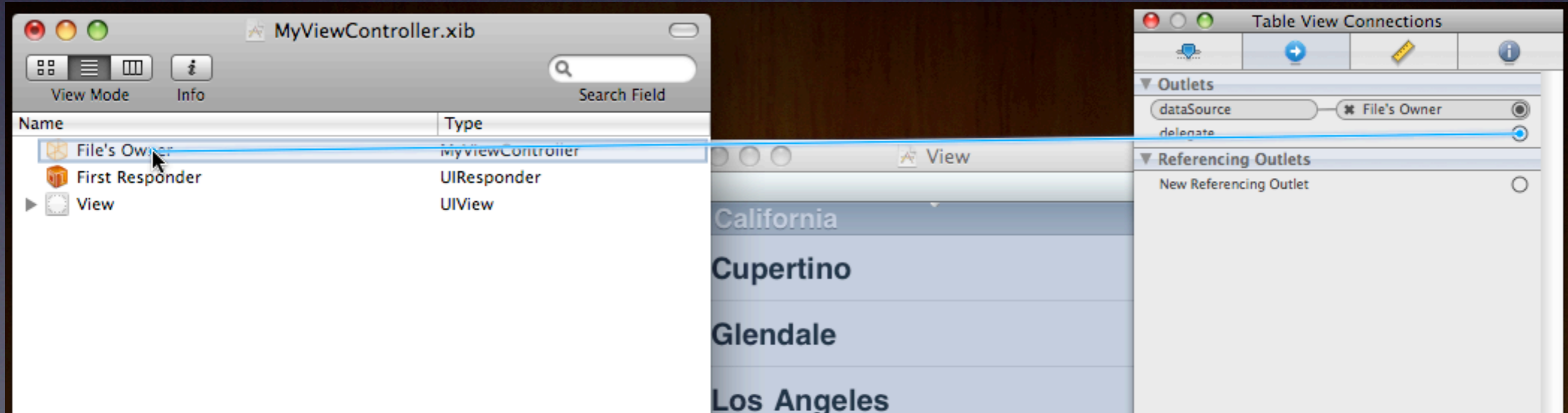
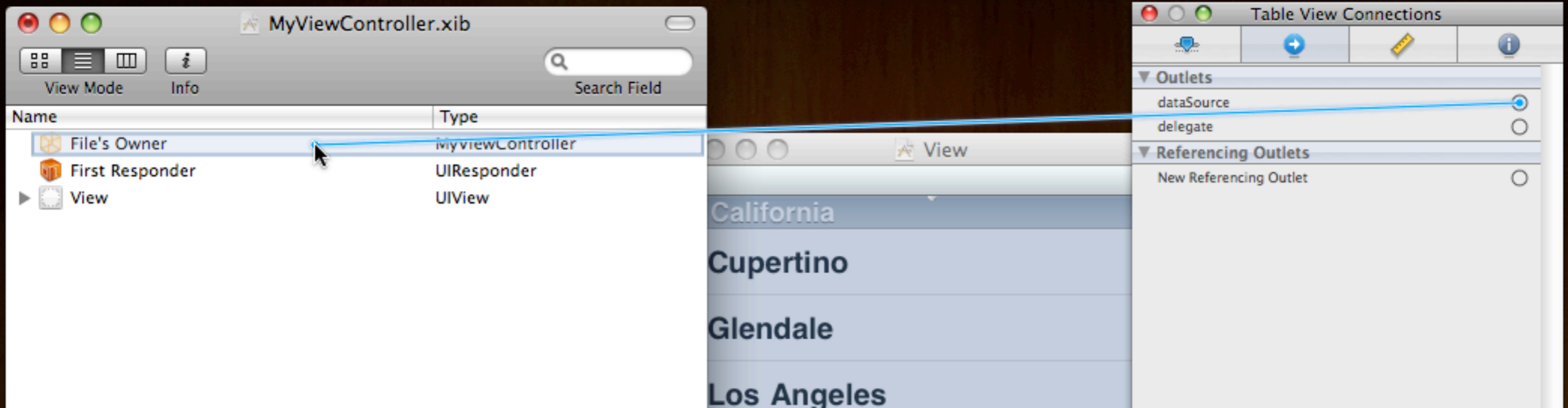
- (void)viewDidLoad
{
    UITableView *tableView = [[UITableView alloc] initWithFrame:CGRectZero
                                                                    style:UITableViewStylePlain];

    [tableView setDelegate:self];
    [tableView setDataSource:self];

    // Additional setup and assignment

    [super viewDidLoad];
}
```

To Set Delegates In IB

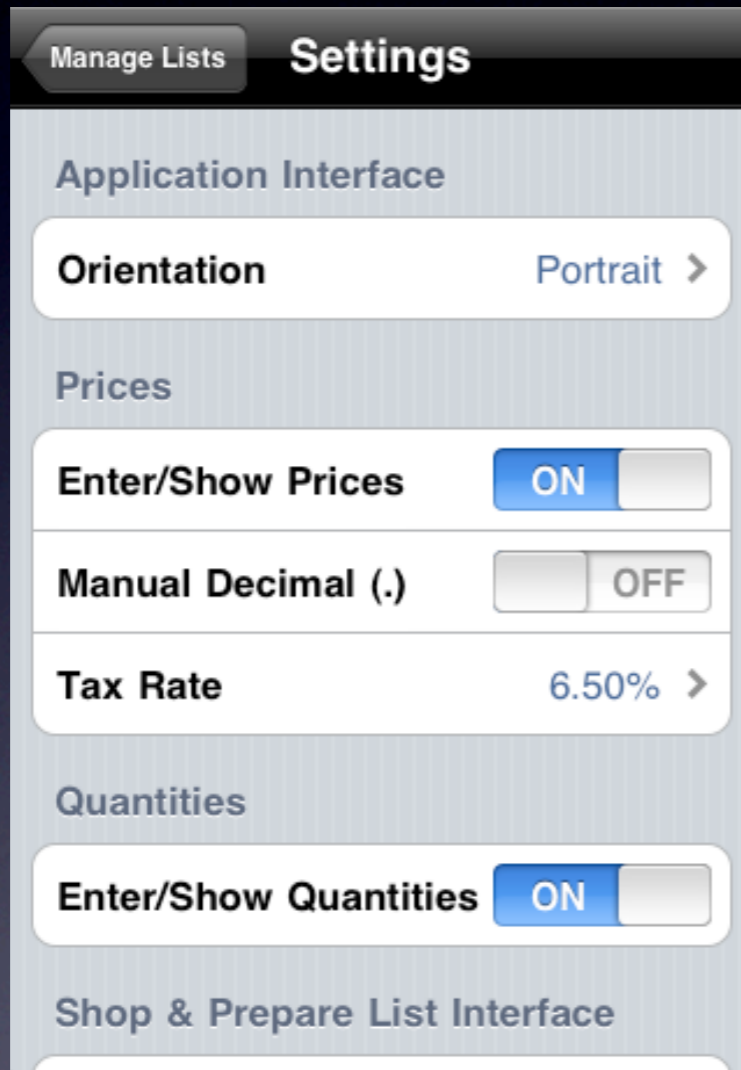


Data Source Methods

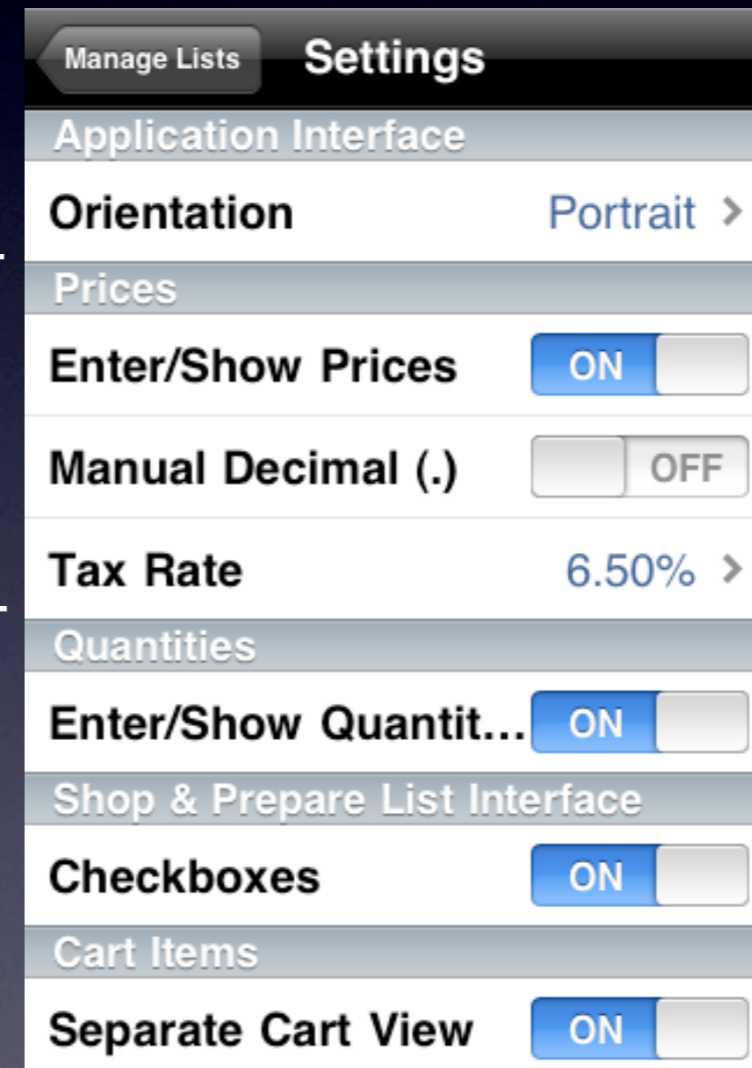
- Required
 - – `tableView:numberOfRowsInSection:`
 - – `tableView:cellForRowAtIndexPath:`
- Optional, but common
 - – `numberOfSectionsInTableView:`
 - – `tableView:titleForHeaderInSection:`
 - – `tableView:commitEditingStyle:forRowAtIndexPath:`
- See documentation for complete list

Terminology

Header
Row
Row
Row



Section



Header
Row
Row
Row

Number Of Sections

- MUST have at least one. (zero will crash)
- Hard code it, or tie it to data model

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 4;
}
```

```
-- or --
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    int sections = [[self myArray] count];

    if (sections == 0) // As a defensive measure
        sections = 1;

    return sections;
}
```

Number Of Rows

- Specify for each section, can be tied to data model

```
- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section {
    int rows;
    switch (section) {
        case 2:
            rows = 5;
            break;
        case 4:
            rows = 3;
            break;
        default:
            rows = 1;
            break;
    }
    return rows;
}
```

NSIndexPath

- Used to describe a location in the table (start at 0 (zero), just like arrays)
- Does more in Mac OS X, but for iPhone use it has two integer properties: section, row
- These are read-only properties. To retrieve:

```
int section = [indexPath section];  
int row = [indexPath row];
```
- Be sure to grab correct (for iPhone) document article:

Symbol Name	Class	Language	Type	Documentation Set
NSIndexPath		Objective-C	Class	iPhone OS 3.0 Library
NSIndexPath(UITableView)		Objective-C	Category	iPhone OS 3.0 Library

NSIndexPath UIKit Additions	
Table of Contents	Jump To...
Overview	NSIndexPath UIKit Additions
Tasks	

Providing a Table Cell

- The table view will ask the data source to provide a cell for every row in the table
- A no-frills method looks like this:

```
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    static NSString *CellIdentifier = @"Cell";

    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
                                         reuseIdentifier:CellIdentifier] autorelease];
    }

    // Configure the cell.

    return cell;
}
```

Position-based Content

- -tableView: cellForRowAtIndexPath: is called for each visible row in the table. A NSIndexPath is passed in so you can use that location to determine the needed data

```
int section = [indexPath section];
int row = [indexPath row];

MyDataObject *tempObject;

if (section == 0)
    tempObject = [[self myArray] objectAtIndex:row];
else
    tempObject = [[self myOtherArray] objectAtIndex:row];

// Create cell

[[cell.textLabel] setText: [tempObject name]];
```

Creating A Cell

- UITableViewCell
- Can create custom subclasses
- Can build in code or in IB
- Several styles built-in (see docs)

```
UITableViewCell *cell = [[[UITableViewCell alloc]
    initWithStyle: UITableViewCellStyleDefault
    reuseIdentifier: CellIdentifier] autorelease];
```

Reusing Cells - I

- Alloc/Init is expensive. Scrolling performance will suffer if each cell is newly created each time.
- A queueing mechanism is provided that allows cells to be re-used. Content can be changed in re-used cells. Don't build new cells if not necessary.
- Cells are identified by a string that enables reuse.

```
static NSString *CellIdentifier = @"Cell";
```

```
UITableViewCell *cell = [[[UITableViewCell alloc]  
    initWithStyle:UITableViewCellStyleDefault  
    reuseIdentifier:CellIdentifier] autorelease];
```

Reusing Cells - 2

- As cells scroll off the screen, they are placed into a queue (based on the identifier)
- Before building a new cell, find out if one is available in the queue for reuse.

```
UITableViewCell *cell = [tableView  
    dequeueReusableCellWithIdentifier: CellIdentifier];
```

Reusing Cells - 3

- Use the lazy-loading approach. If no cell is available in the queue, *then* create a new one.
- The table view will return nil if no cell is available for reuse. Test for nil, respond accordingly

```
UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
if (cell == nil)
{
    cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
                                     reuseIdentifier:CellIdentifier] autorelease];
}
```

Cells, All Together

- Combining each of the previous slides, this is how a typical method looks:

```
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    MyDataClass *tempObject = [[self myArray] objectAtIndex:[indexPath row]];

    static NSString *CellIdentifier = @"Cell";

    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
                                         reuseIdentifier:CellIdentifier] autorelease];
    }

    [[cell.textLabel] setText:[tempObject name]];

    return cell;
}
```

Delegate Methods

- Pretty much all methods are optional
- Used to augment appearance, and respond to user input
 - – `tableView:heightForRowAtIndexPath:`
 - – `tableView:accessoryButtonTappedForRowWithIndexPath:`
 - – `tableView:didSelectRowAtIndexPath:`
- See docs for full list

Responding To Touches

- Tapping a row will select it (blue background); you will need to manually deselect it in code
- In many programs, tapping a row causes a new screen to slide into view
- `didSelectRowAtIndexPath:` is a typical place to define how your program reacts

Navigation

- Assuming there is a UINavigationController, this is how to slide in a new view in response to a row touch

```
- (void)tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
    // Deselect the row
    [tableView deselectRowAtIndexPath:indexPath animated:YES];

    // Grab a data object to pass to the new view
    MyDataObject *tempObject = [[self myArray] objectAtIndex:indexPath.row];

    // Create the new view controller
    MyDetailViewController *childController = [[MyDetailViewController alloc] init];

    // Pass in the data
    [childController setImportantData: tempObject];

    // Push the new view controller onto the stack
    [[self navigationController] pushViewController:childController animated:YES];

    // Clean up
    [childController release], childController = nil;
}
```

UITableViewController

- Basically a convenience class, with a few benefits (see docs for full description):
 - ‘tableView’ property auto-generated
 - XIB file not needed, and table will be autosized
 - Automatically clears the selection whenever the table appears (returning from detail view, etc.)
 - Template file contains stubs for many delegate methods
 - <delegate> protocols assumed, not explicitly declared

Header Comparison

```
// MyViewController.h

#import <UIKit/UIKit.h>

@interface MyViewController : UIViewController <UITableViewDelegate, UITableViewDataSource>
{
    UITableView *mainTableView;
}
@property (nonatomic, retain) IBOutlet UITableView *mainTableView;
@end
```

```
// MyTableViewController.h

#import <UIKit/UIKit.h>

@interface MyTableViewController : UITableViewController
{
}

@end
```

Row Animation Gotcha

- There are several methods for row animation:
 - – `insertRowsAtIndexPaths:withRowAnimation:`
 - – `deleteRowsAtIndexPaths:withRowAnimation:`
 - – `insertSections:withRowAnimation:`
 - – `deleteSections:withRowAnimation:`
- You **MUST** manually make changes to the underlying data model at the same time, or else you will crash. Inserting/deleting rows does **NOT** manipulate data (remember: table view does not store content). Ex:

```
[[self myArray] removeObjectAtIndex:[indexPath row]];
```

```
[tableView deleteRowsAtIndexPaths:[NSArray arrayWithObject:indexPath]  
withRowAnimation:UITableViewRowAnimationRight];
```

Animation Approach

- If attempting multiple inserts/deletes, store them up then fire off at once. Ex:

```
// Prepare containers for index paths and sections
NSMutableArray *indexPathsToRemove = [[NSMutableArray alloc] init];
NSMutableArray *indexPathsToAdd = [[NSMutableArray alloc] init];
NSMutableIndexSet *sectionsToRemove = [[NSMutableIndexSet alloc] init];
NSMutableIndexSet *sectionsToAdd = [[NSMutableIndexSet alloc] init];

// --- Logic to determine what needs to be added or removed (see next slide)
// --- Don't forget to make data model changes, too!

[[self tableView] beginUpdates];

if ([indexPathsToRemove count] != 0)
    [[self tableView] deleteRowsAtIndexPaths:indexPathsToRemove withRowAnimation:kAnimateOut];
if ([sectionsToRemove count] != 0)
    [[self tableView] deleteSections:sectionsToRemove withRowAnimation:kAnimateOut];
if ([sectionsToAdd count] != 0)
    [[self tableView] insertSections:sectionsToAdd withRowAnimation:kAnimateIn];
if ([indexPathsToAdd count] != 0)
    [[self tableView] insertRowsAtIndexPaths:indexPathsToAdd withRowAnimation:kAnimateIn];

[[self tableView] endUpdates];
```

Logic Example

```
// Prepare containers for index paths and sections
NSMutableArray *indexPathsToRemove = [[NSMutableArray alloc] init];
NSMutableIndexSet *sectionsToRemove = [[NSMutableIndexSet alloc] init];

// Source data
NSString *sectionKey = [[self mySectionArray] objectAtIndex: section];
NSMutableArray *tempArray = [[self myContentDictionary] objectForKey: sectionKey];

if (deleteCondition)
{
    // If the array will become empty, animate out the section.  Otherwise, just the row
    if ([tempArray count] == 1)
    {
        [sectionsToRemove addIndex:section];
        [[self myContentDictionary] removeObjectForKey: sectionKey];
        [[self mySectionArray] removeObjectAtIndex: section];
    }
    else
    {
        [indexPathsToRemove addObject: indexPath];
        [tempArray removeObjectAtIndex: row];
    }
}
}
```

Questions?



Brian Slick

contact@briterideas.com

[@BrianSlick](#)

<http://clingingtoideas.blogspot.com/>