

# Mobile Application Development

Lecture 18

Scroll View, Table View, Collection View, Web View

# Lecture Summary

- Scroll views
- Table views and table view cells
- Collection views
- Web views
- DEMO



# Scroll views

- **Scroll views** allow users to see content that is larger than the view's boundaries
- Vertical or horizontal scroll indicators show users that there is more content
- Scroll views are implemented in the `UIScrollView` class
- `UIScrollView` also enables users:
  - to scroll within that content by making swiping gestures
  - to zoom in and back from portions of the content by making pinching gestures
- A `UIScrollView` is a view whose origin is adjustable over the content view
- A scroll view clips the content to its frame
- The content of a scroll view is one or more views of any kind

# Adding content to scroll views

– The steps to work with a scroll view are the following:

1. Add a view to a scroll view:

```
UIImageView *imageView = [[UIImageView alloc] initWithImage:image];  
[scrollView addSubview:imageView];
```

2. Set the size of the scroll view's content:

```
scrollView.contentSize = CGSizeMake(3000, 3000);
```

```
// or ...
```

```
scrollView.contentSize = imageView.bounds.size;
```

# Adding content to scroll views

- The `contentOffset` property returns the upper left corner of the currently portion of the view that is being displayed relatively to the scroll view

```
scrollView.contentOffset.x, scrollView.contentOffset.y
```

- The `bounds` property returns the rectangle which is currently visible

```
scrollView.bounds.origin.x, scrollView.bounds.origin.y
```

```
scrollView.bounds.size.width, scrollView.bounds.size.height
```

- The visible area in the subview's coordinates may be different from that returned by the `bounds` property because of scaling (zooming), different subview's frame origin

```
CGRect visibleRect = [scrollView convertRect:scrollView.bounds toView:subview];
```

# Adding scroll views

- The recommended way to use scroll views is to add to a view by dragging it from the object palette and then add the subviews programmatically
- The scroll view can also be inserted in code with alloc/init
- It is also possible to embed an existing view from (Editor → Embed in... → Scroll View), but you should not use this option
- All the subviews added to a scroll view are automatically located at the origin (upper left corner) of the scroll view; to change the location, a new frame for the view should be set

```
subview.frame = CGRectMake(200, 200, 100, 100);
```

- It is essential to set the `contentSize` property of the scroll view, or it will not work!
- To have a scroll view to be of exactly of the same size of its subview

```
scrollView.contentSize = subview.bounds.size;
```

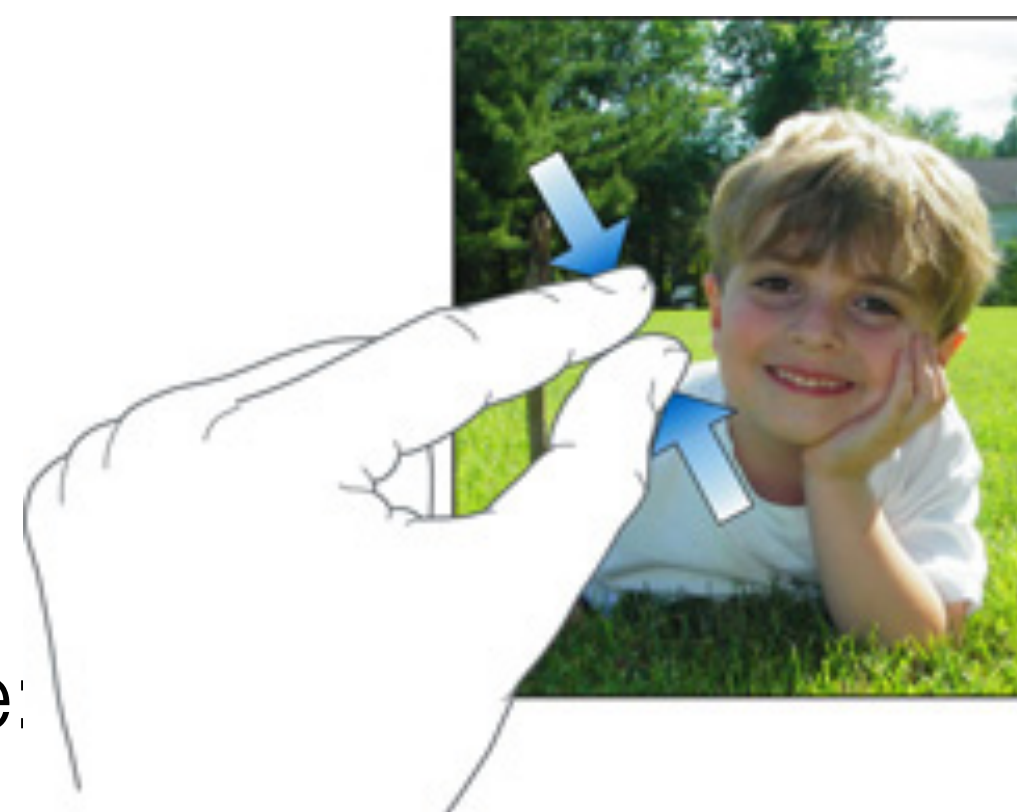
# Working with scroll views

- Scroll views natively support the scrolling made by the user but it is also possible to scroll programmatically:
  - `(void)scrollRectToVisible:(CGRect)rect animated:(BOOL)animated`
- It is possible to enable and disable scrolling using the `scrollEnabled` property
- The `directionalLockEnabled` property can be used to lock the scrolling direction to the user's first move (for the current drag)

# Zooming in scroll views

- Scroll views support zooming of the content by pinching-in or pinching-out
- To enable zooming in a scroll view some preliminary actions must be taken:
  1. set the minimum and maximum zoom scale:

```
self.scrollView.minimumZoomScale = 0.5; // 50%  
self.scrollView.maximumZoomScale = 3.0; // 300%
```



# Zooming in scroll views

- Scroll views support zooming of the content by pinching-in or pinching-out
- To enable zooming in a scroll view some preliminary actions must be taken:
  1. set the minimum and maximum zoom scale:

```
self.scrollView.minimumZoomScale = 0.5; // 50%  
self.scrollView.maximumZoomScale = 3.0; // 300%
```

2. declare the view controller as `UIScrollViewDelegate` and set it as `delegate` of the scroll view

```
@interface MyViewController : UIViewController<UIScrollViewDelegate>
```

```
// or ...
```

```
@interface MyViewController ()<UIScrollViewDelegate>
```

```
self.scrollView.delegate = self;
```



# Zooming in scroll views

- Scroll views support zooming of the content by pinching-in or pinching-out
- To enable zooming in a scroll view some preliminary actions must be taken:

1. set the minimum and maximum zoom scale:

```
self.scrollView.minimumZoomScale = 0.5; // 50%  
self.scrollView.maximumZoomScale = 3.0; // 300%
```

2. declare the view controller as `UIScrollViewDelegate` and set it as `delegate` of the scroll view

```
@interface MyViewController : UIViewController<UIScrollViewDelegate>  
  
    // or ...
```

```
@interface MyViewController ()<UIScrollViewDelegate>  
  
self.scrollView.delegate = self;
```

3. implement the delegate method to specify the subview that must be zoomed:

```
- (UIView *)viewForZoomingInScrollView:(UIScrollView *)sender
```



# UIScrollViewDelegate

- The `UIScrollViewDelegate` protocol provides several methods to handle scrolling, zooming, and dragging events in the scroll view:
  - `(void)scrollViewDidScroll:(UIScrollView *)scrollView`
  - `(void)scrollViewDidZoom:(UIScrollView *)scrollView`
  - `(void)scrollViewWillBeginZooming:(UIScrollView *)scrollView  
withView:(UIView *)view`
  - `(void)scrollViewDidEndZooming:(UIScrollView *)scrollView  
withView:(UIView *)view  
atScale:(CGFloat)scale`
  - `(void)scrollViewWillBeginDragging:(UIScrollView *)scrollView`
  - `(void)scrollViewDidEndDragging:(UIScrollView *)scrollView  
willDecelerate:(BOOL)decelerate`

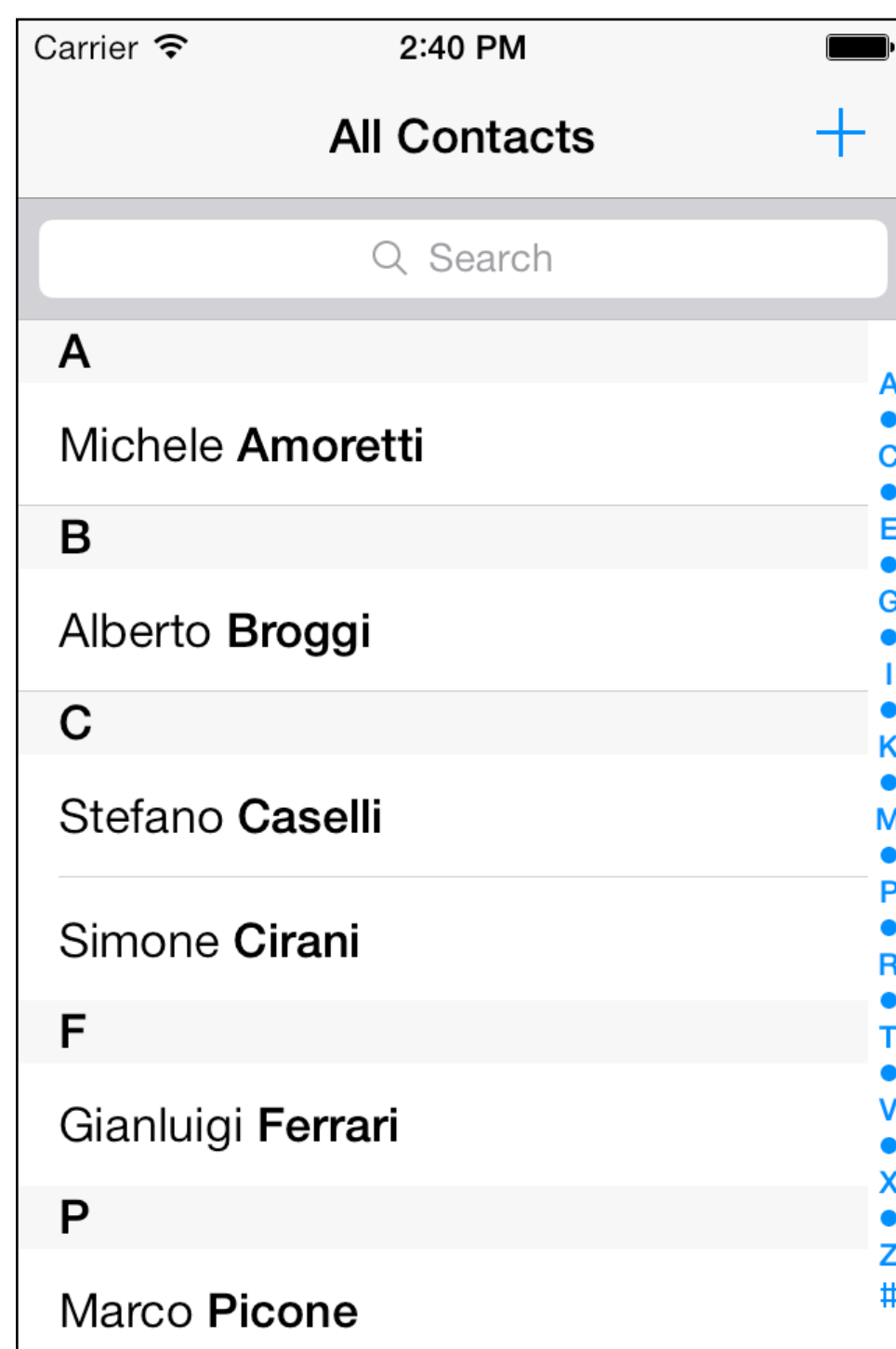
# Table views

- **Table views** are used to display and edit hierarchical lists of information
- A table view displays a list of items in a single column (one dimensional: just rows, no columns)
- Multiple rows may be grouped into sections
- Multi-dimensional data are usually displayed combining table views into a navigation controller
- **UITableView** is the class that implements a table view
  - it is a subclass of **UIScrollView**
  - only vertical scrolling is permitted
- **UITableView** can efficiently handle large amounts of data

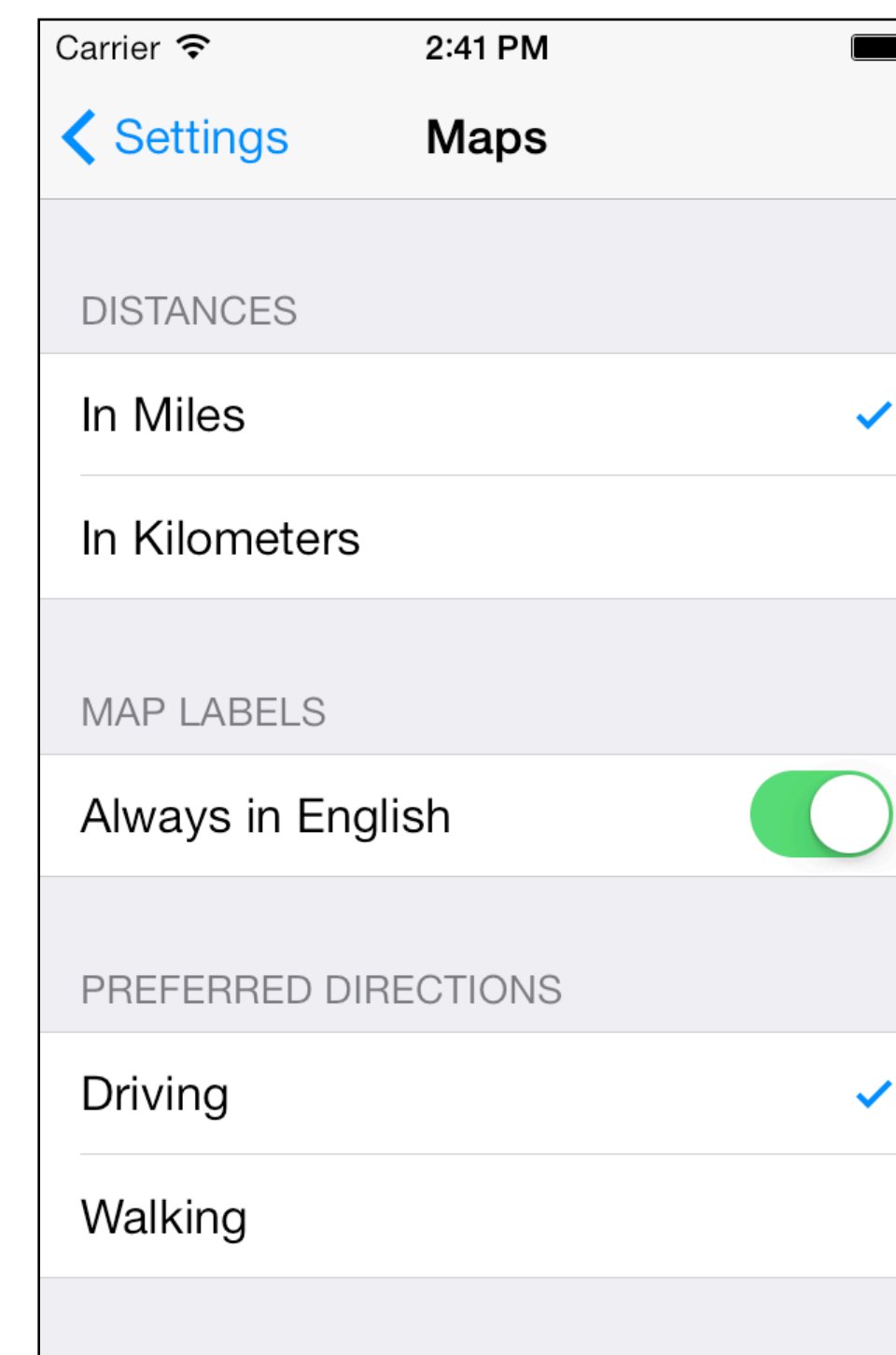
# Table view styles

- A table view can have a plain style or a grouped style

## UITableViewStylePlain



## UITableViewStyleGrouped



# Table view styles

- A table view can be dynamic or static
- Dynamic table views have content that is mutable and unpredictable (e.g. database entries, RSS feeds, ...)
- Static table views have fixed content (defined by the developer), e.g. for settings

# UITableView

## UITableViewStylePlain

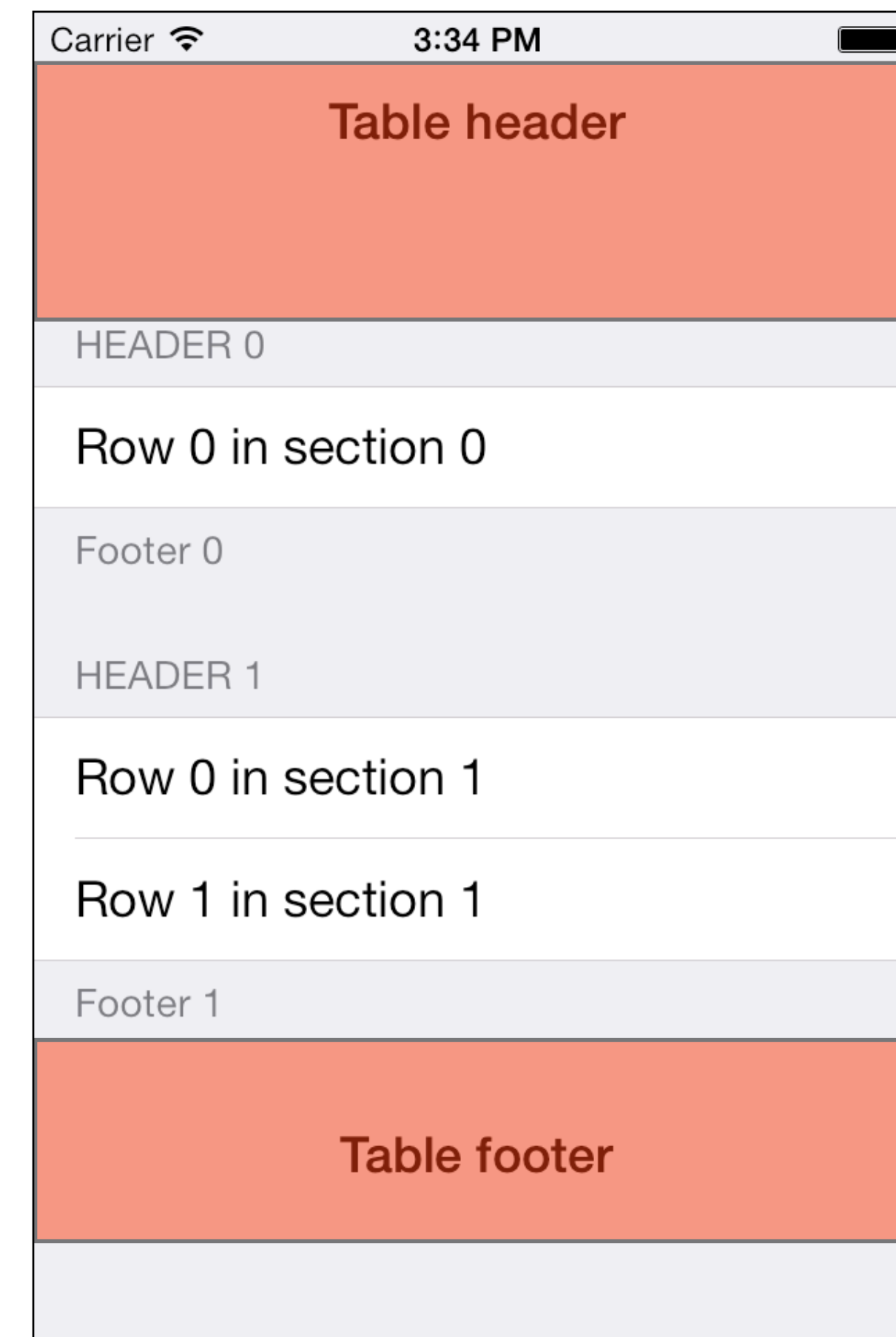
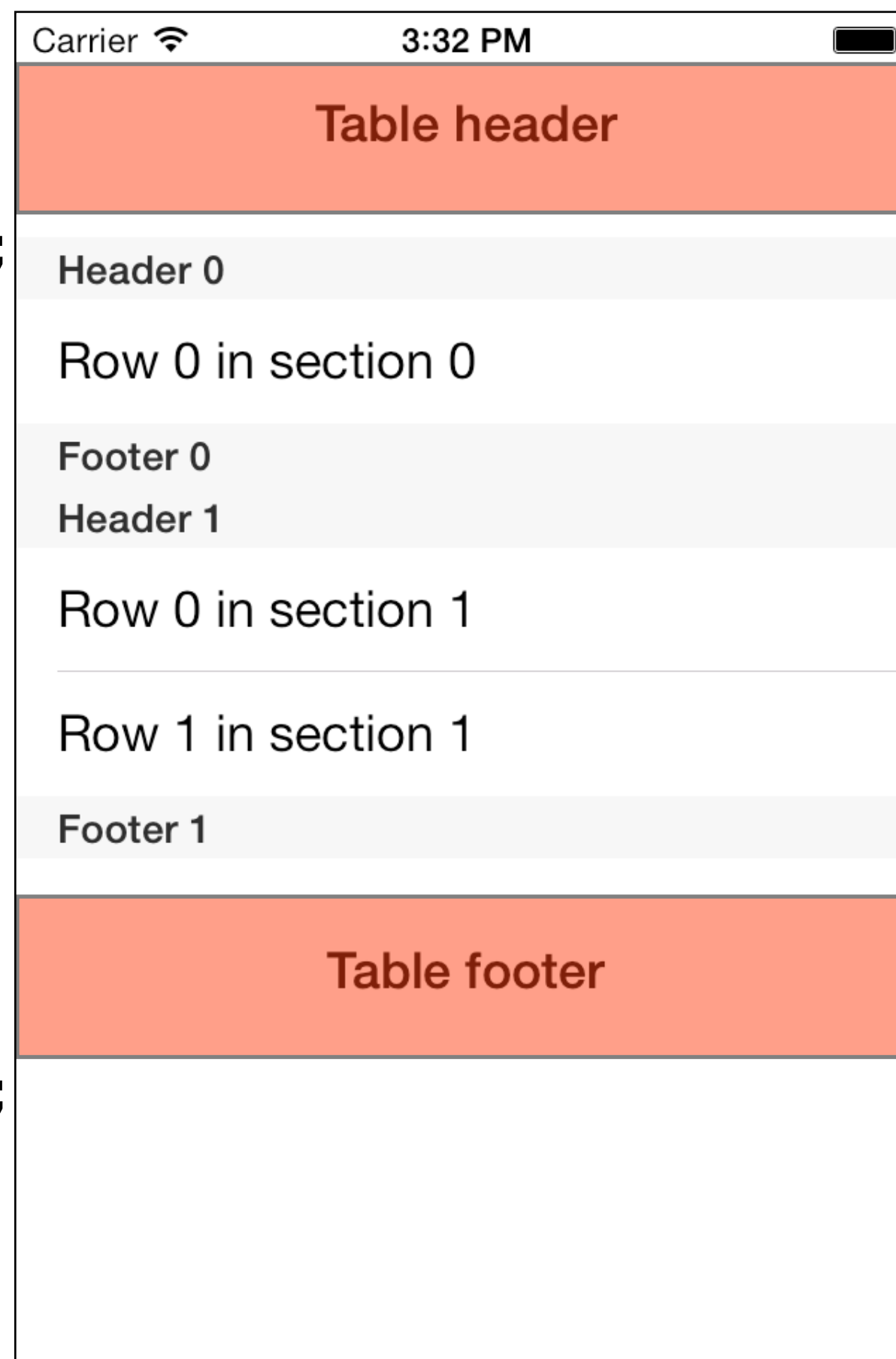
## UITableViewStyleGrouped

Table header →

```
@property UIView *tableHeaderView;
```

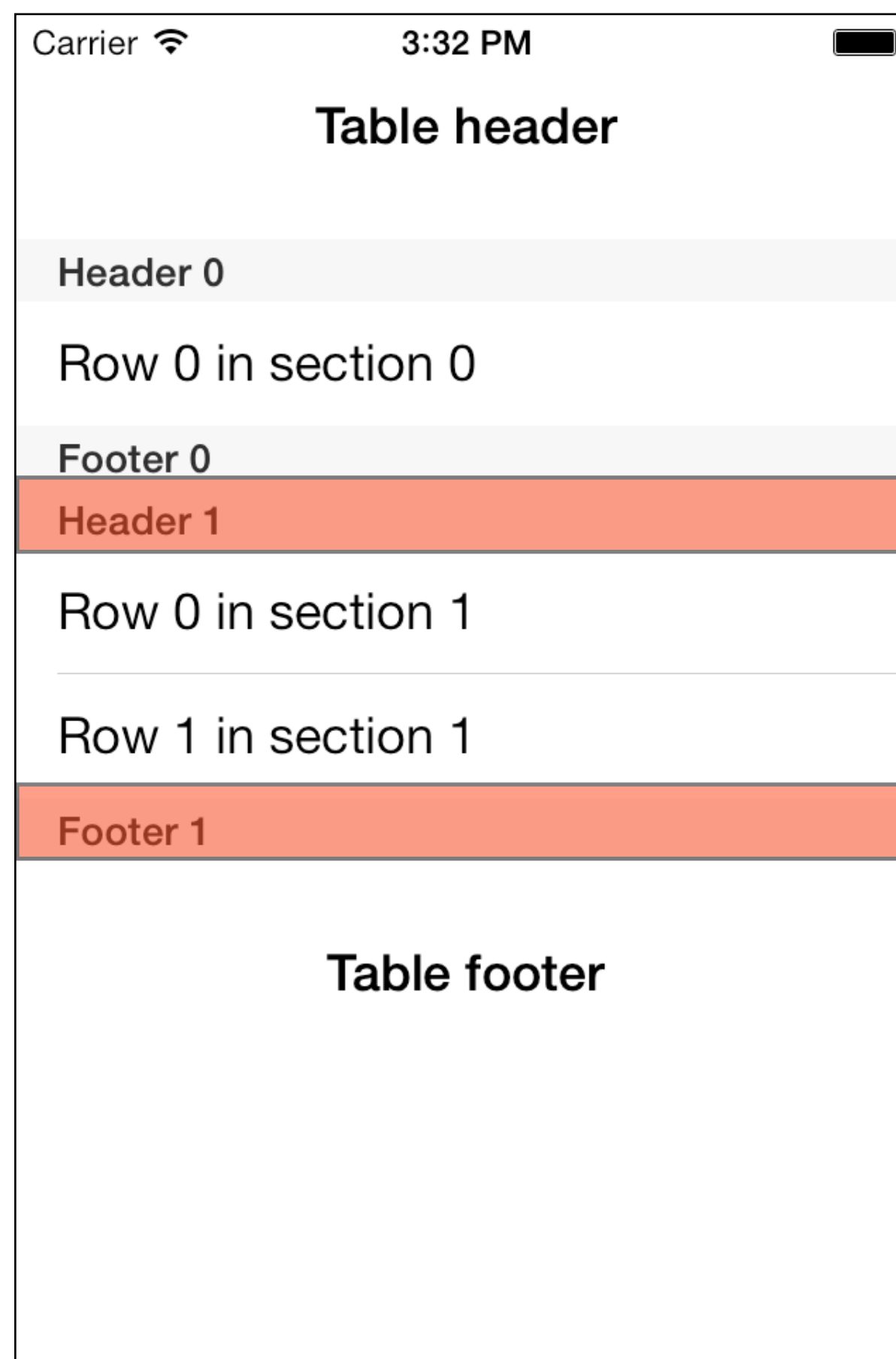
Table footer →

```
@property UIView *tableFooterView;
```



# UITableView

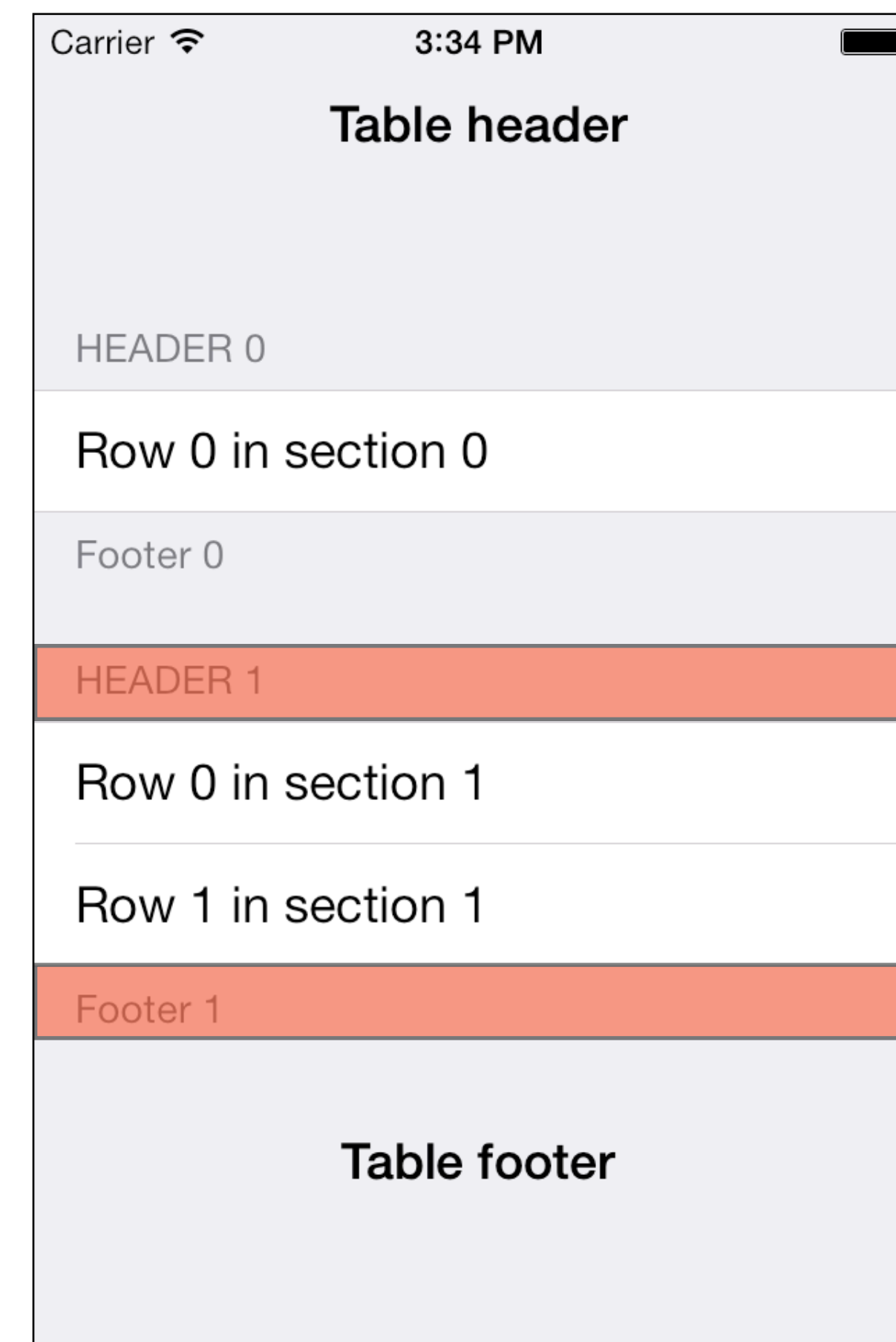
## UITableViewStylePlain



Section header →

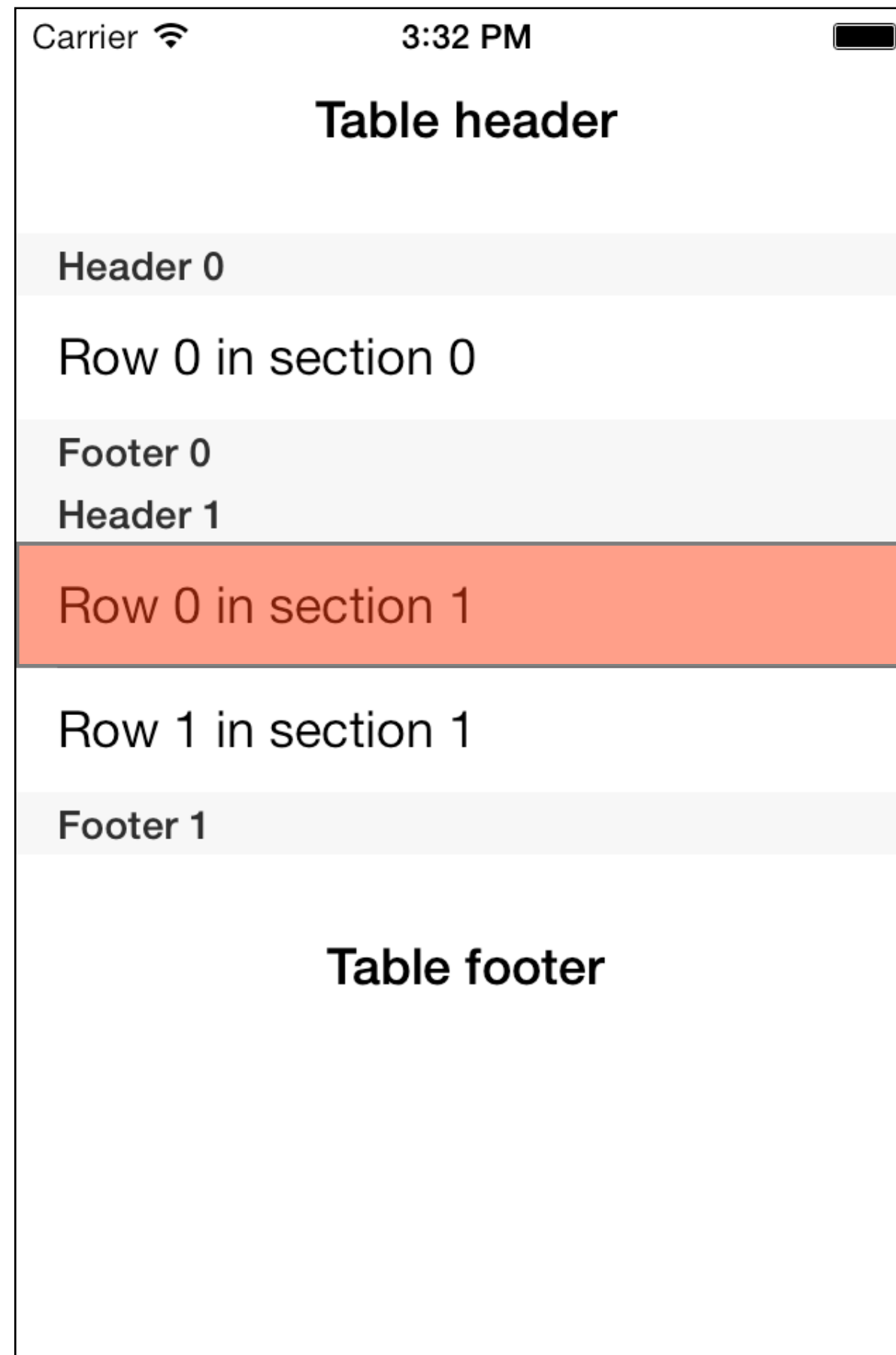
Section footer →

## UITableViewStyleGrouped



# UITableView

## UITableViewStylePlain



## UITableViewStyleGrouped

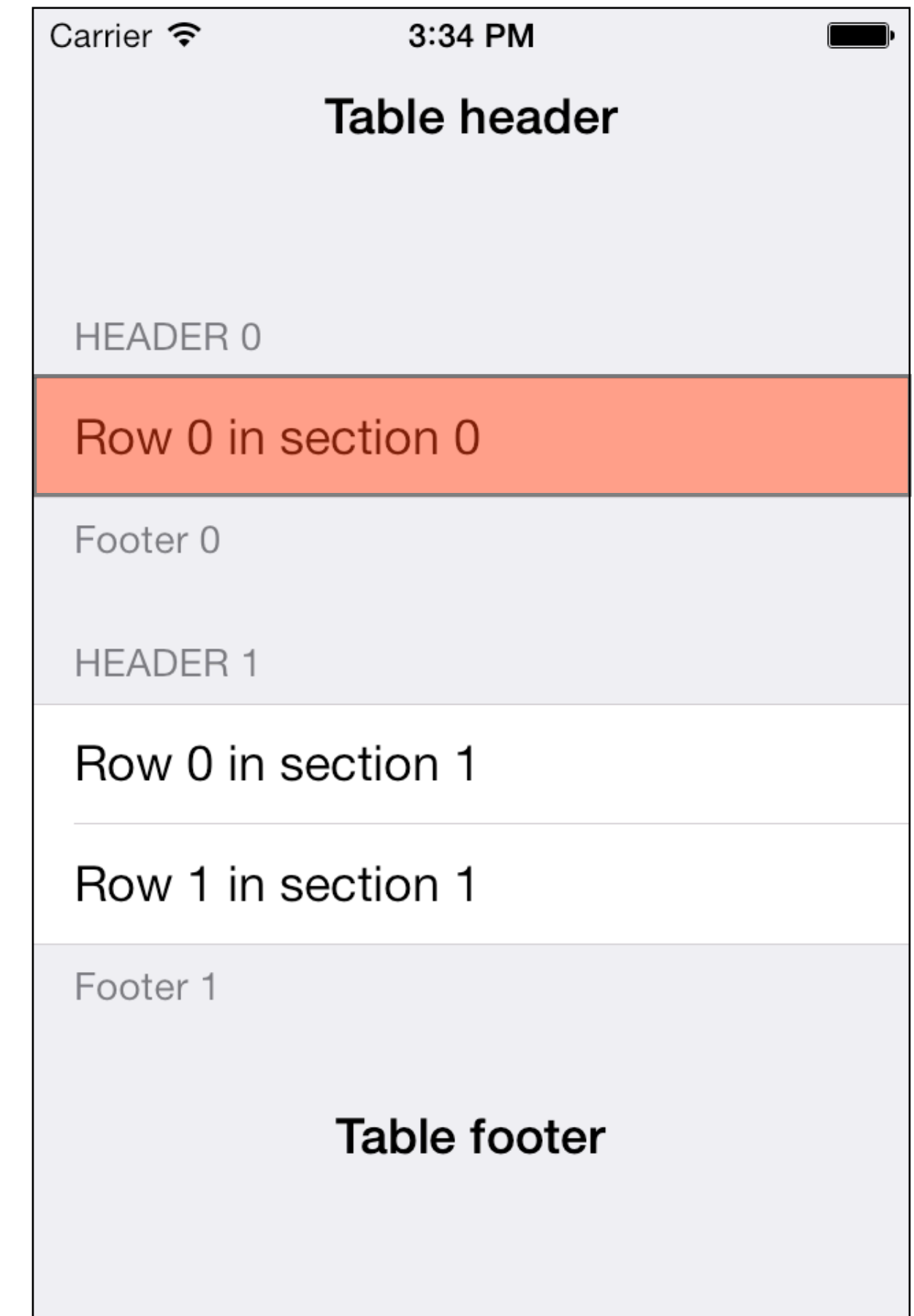
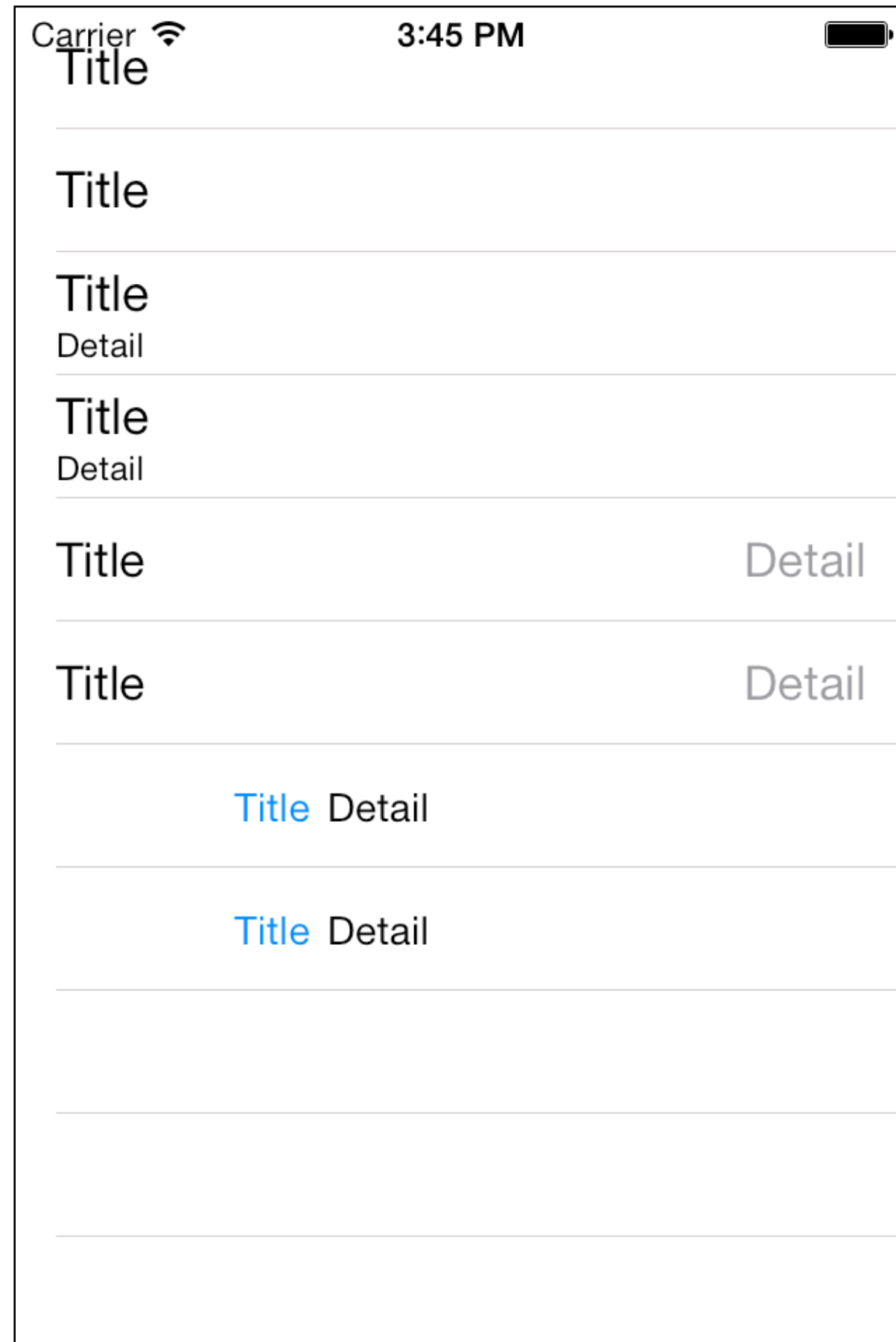


Table cell  
(UITableViewCell)



Row 0 in section 1

# Table cell styles

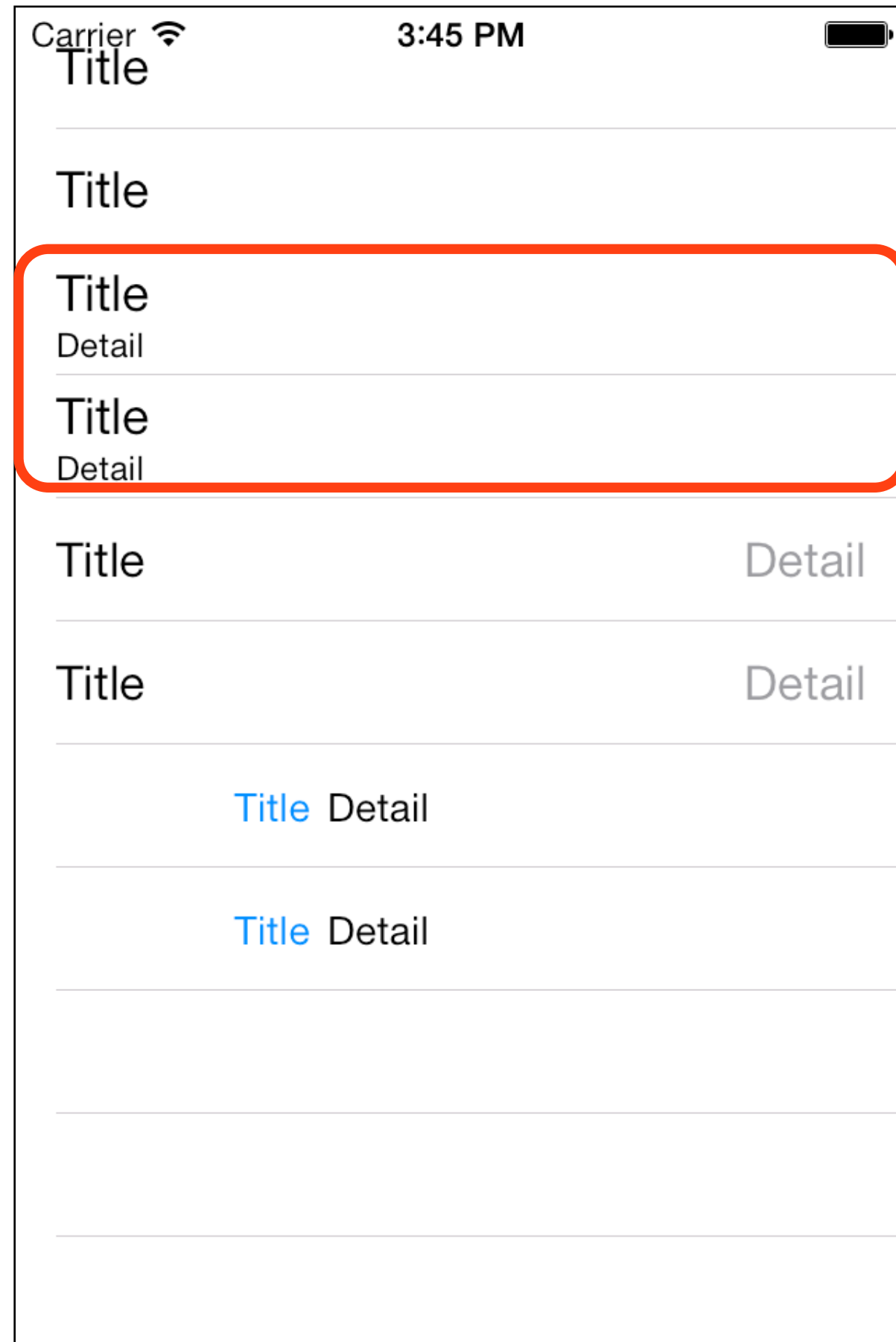


System-defined table cells can have different styles, depending on which content they are intended to display.

Table cells can be customized to have a different look in order to display custom content.

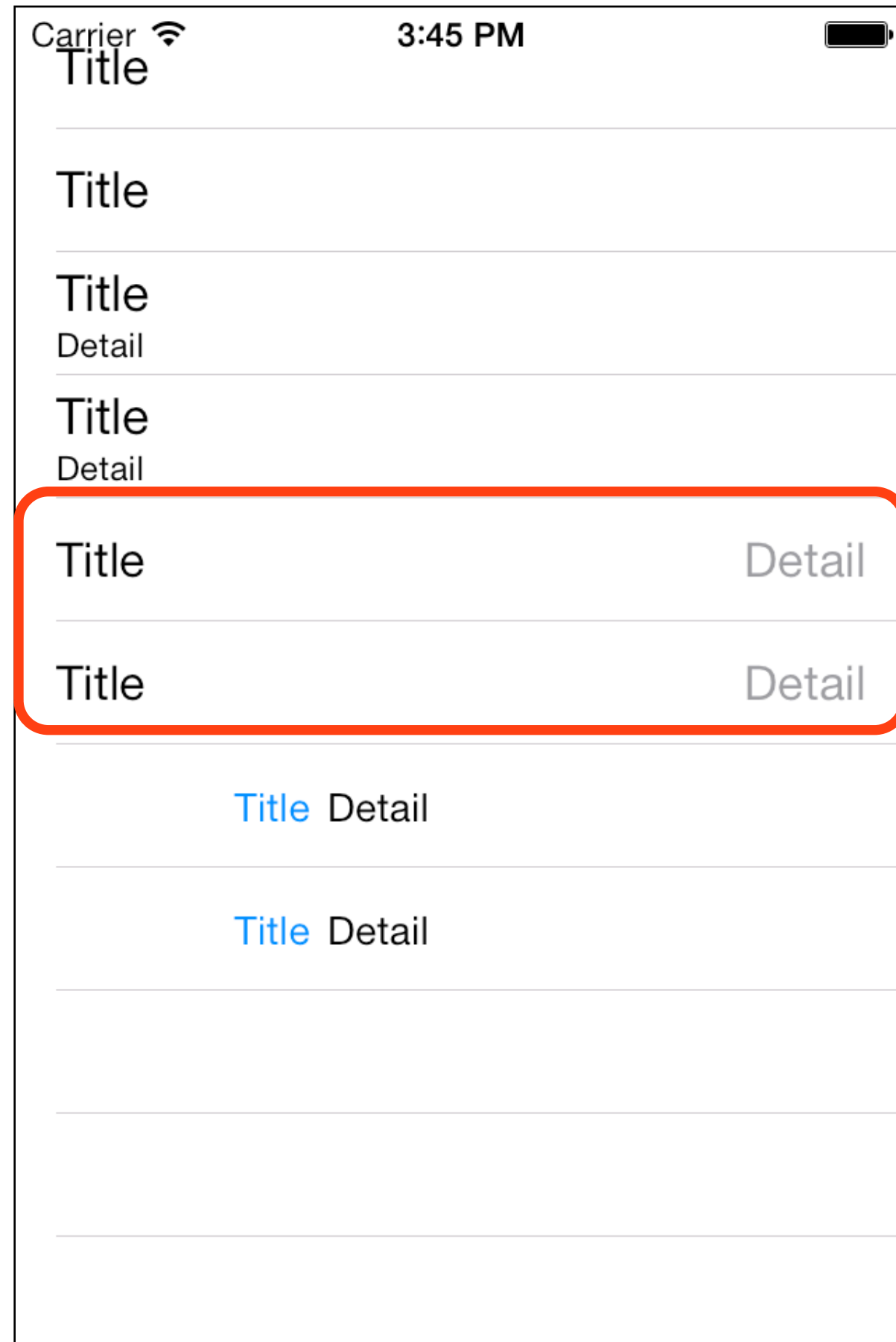


# Table cell styles



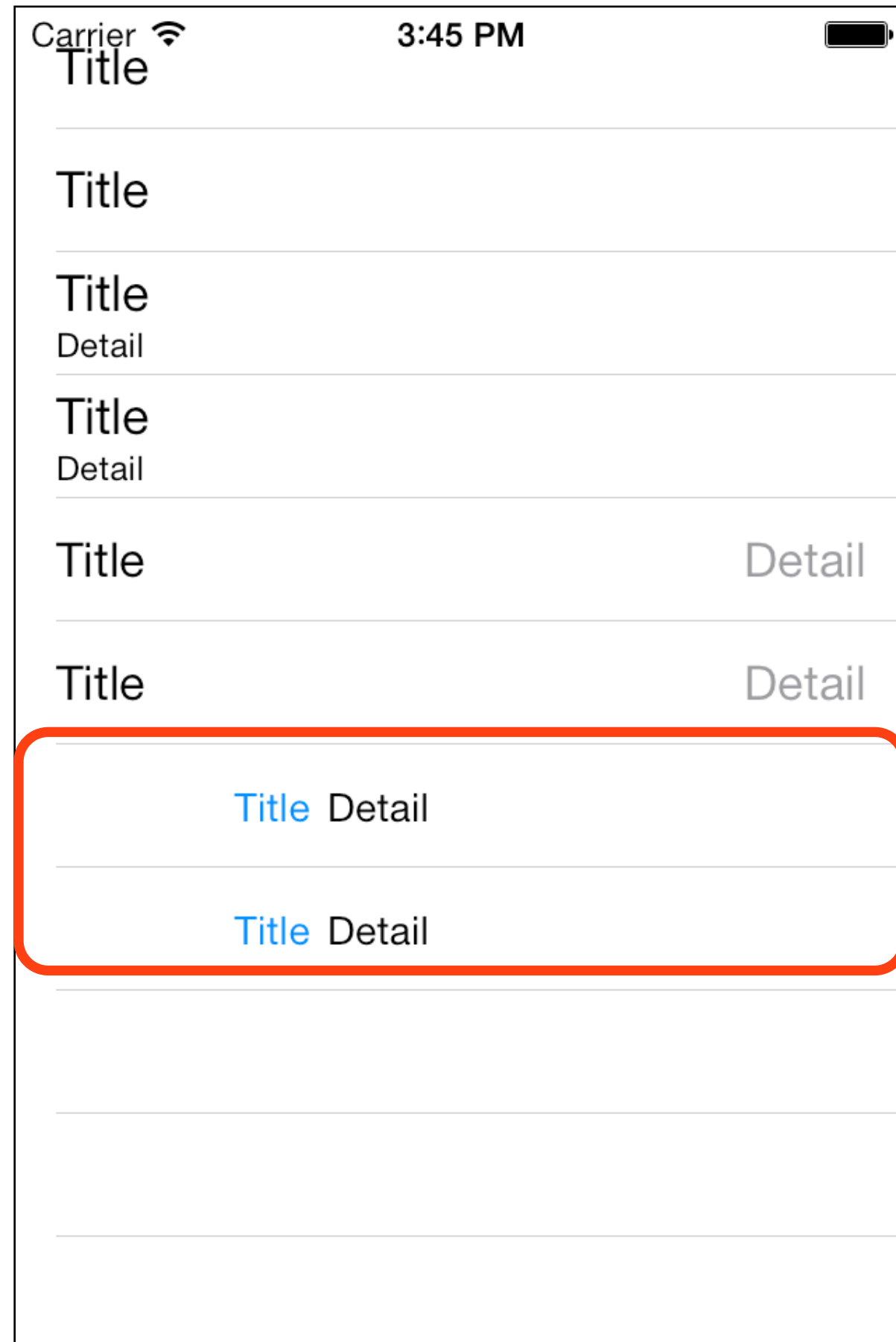
`UITableViewCellStyleSubtitle`

# Table cell styles



`UITableViewCellStyleValue1`

# Table cell styles

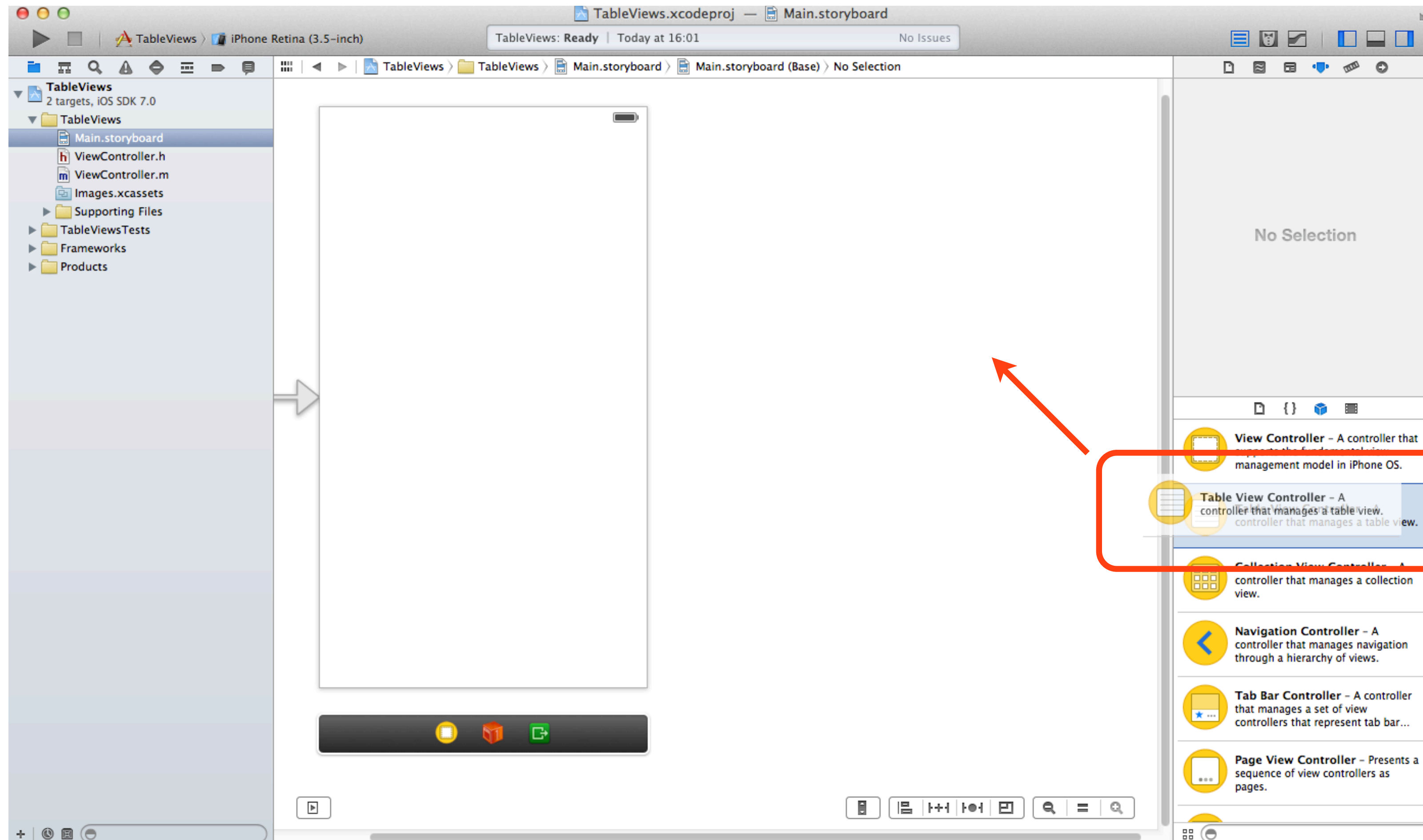


`UITableViewCellStyleValue2`

# Table View Controller

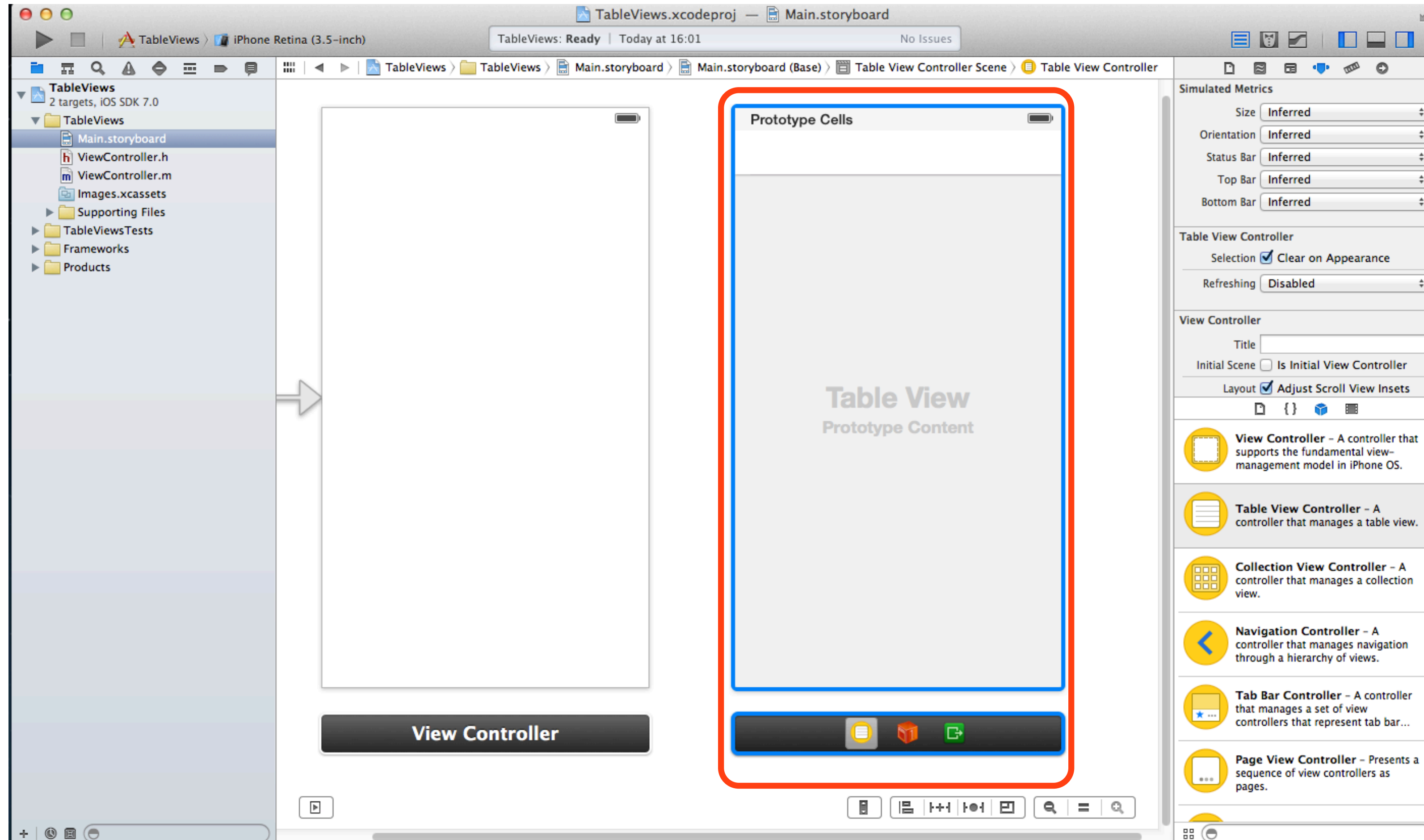
- A table view is usually controlled by a specialized view controller, called **UITableViewController**
- Table views normally take the whole view
- To add a table view controller to storyboard, just drag a Table View Controller item from the object palette
- When adding a Table View Controller, the controller is a subclass of **UITableViewController** and the view is a **UITableView**
- After a Table View Controller has been dragged, you need to create a new Objective-C class that is a subclass of **UITableViewController** and set it as the class of the view in the Identity inspector in Storyboard
- The table view can be configured to set its style (plain or grouped) and type of content (static or dynamic)

# Table View Controller



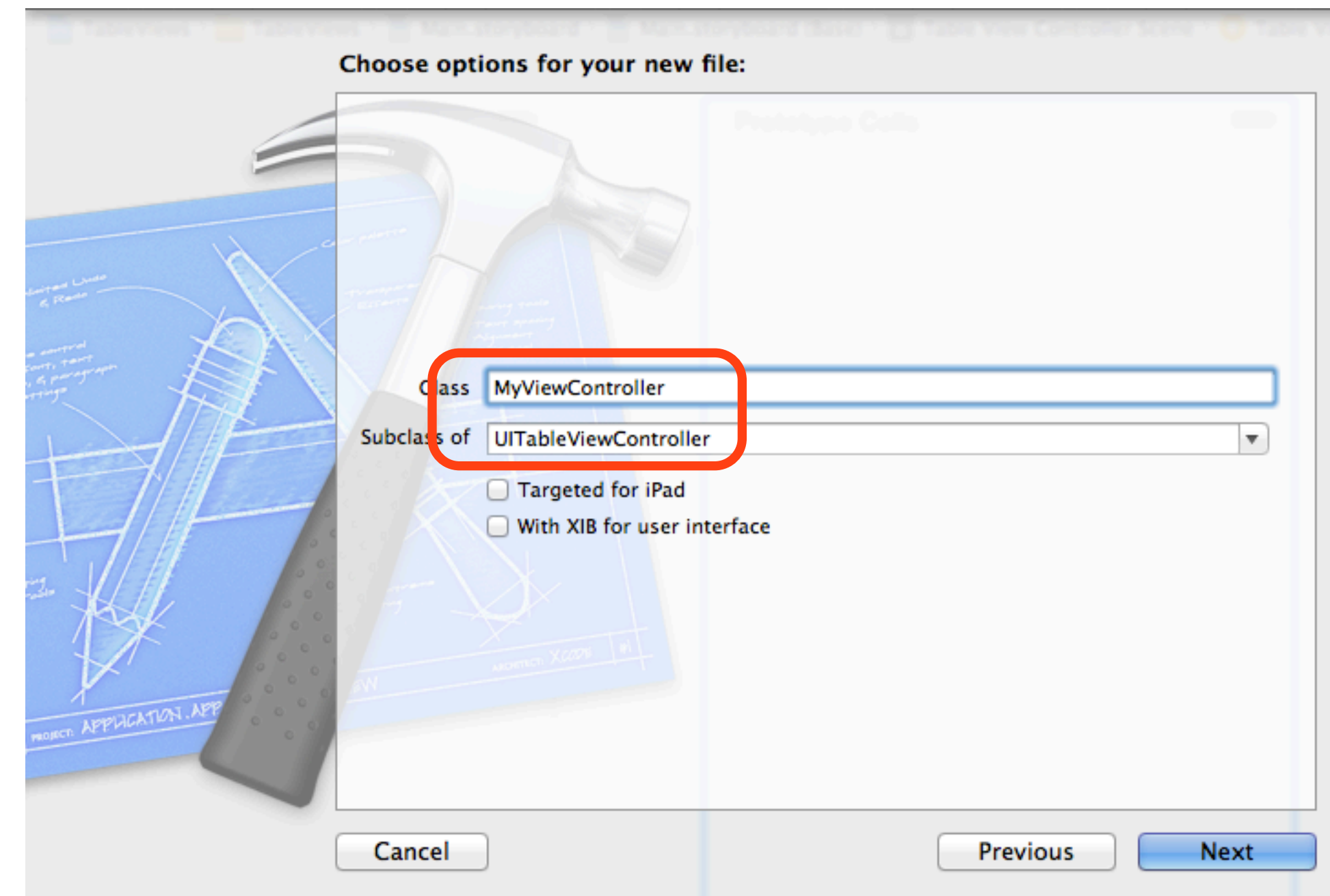
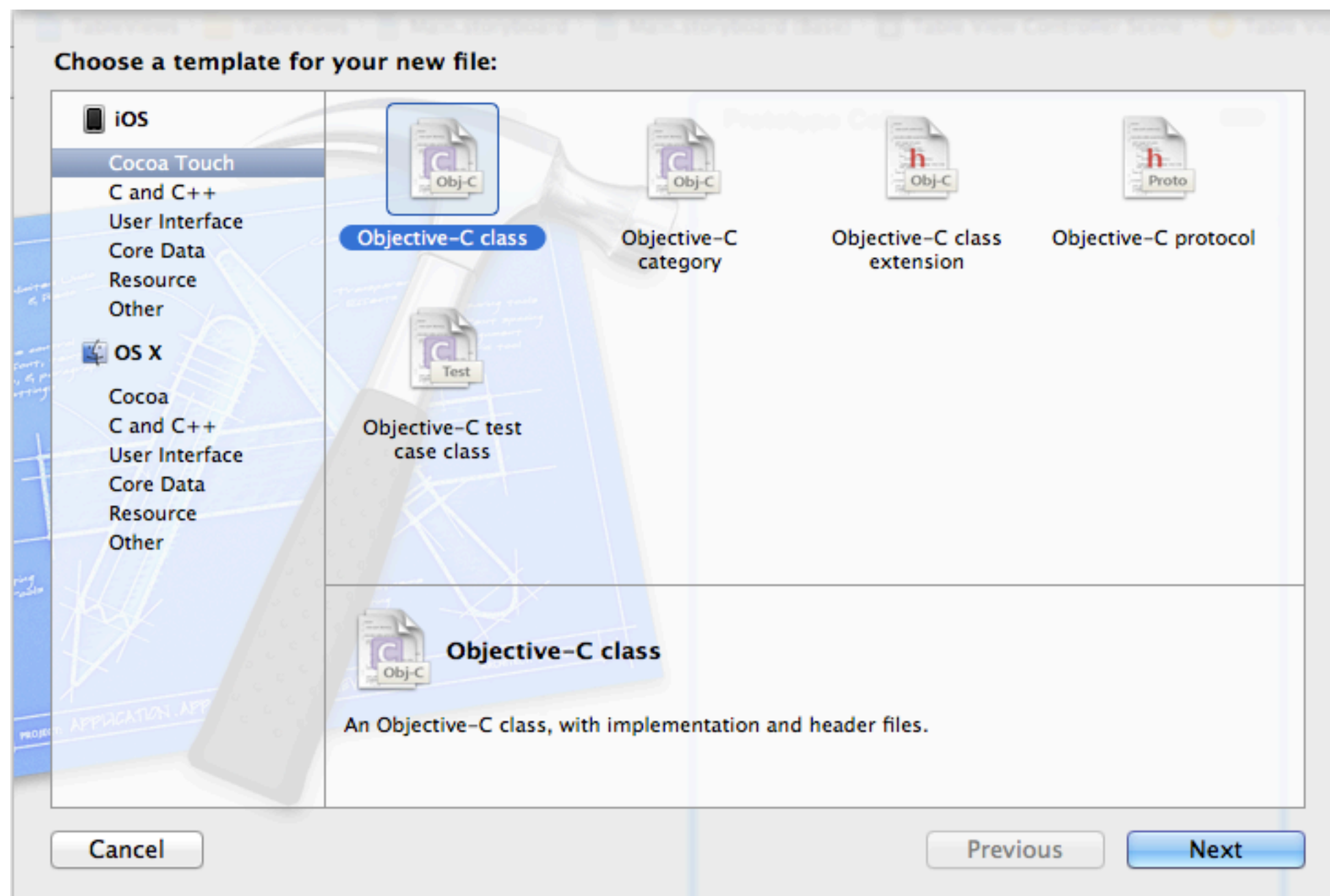
**Drag a Table View Controller item from the object palette**

# Table View Controller



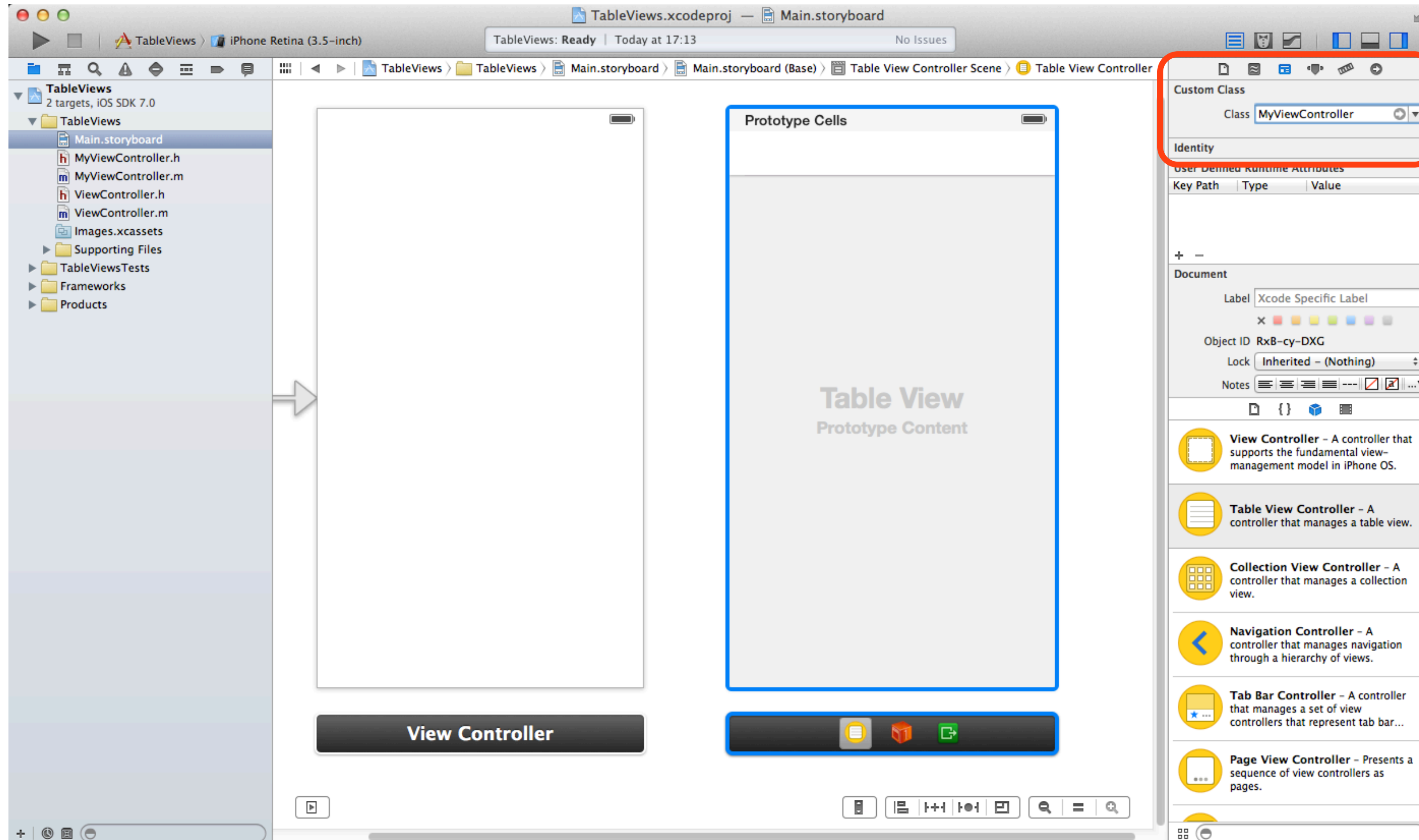
**Drag a Table View Controller item from the object palette**

# Table View Controller



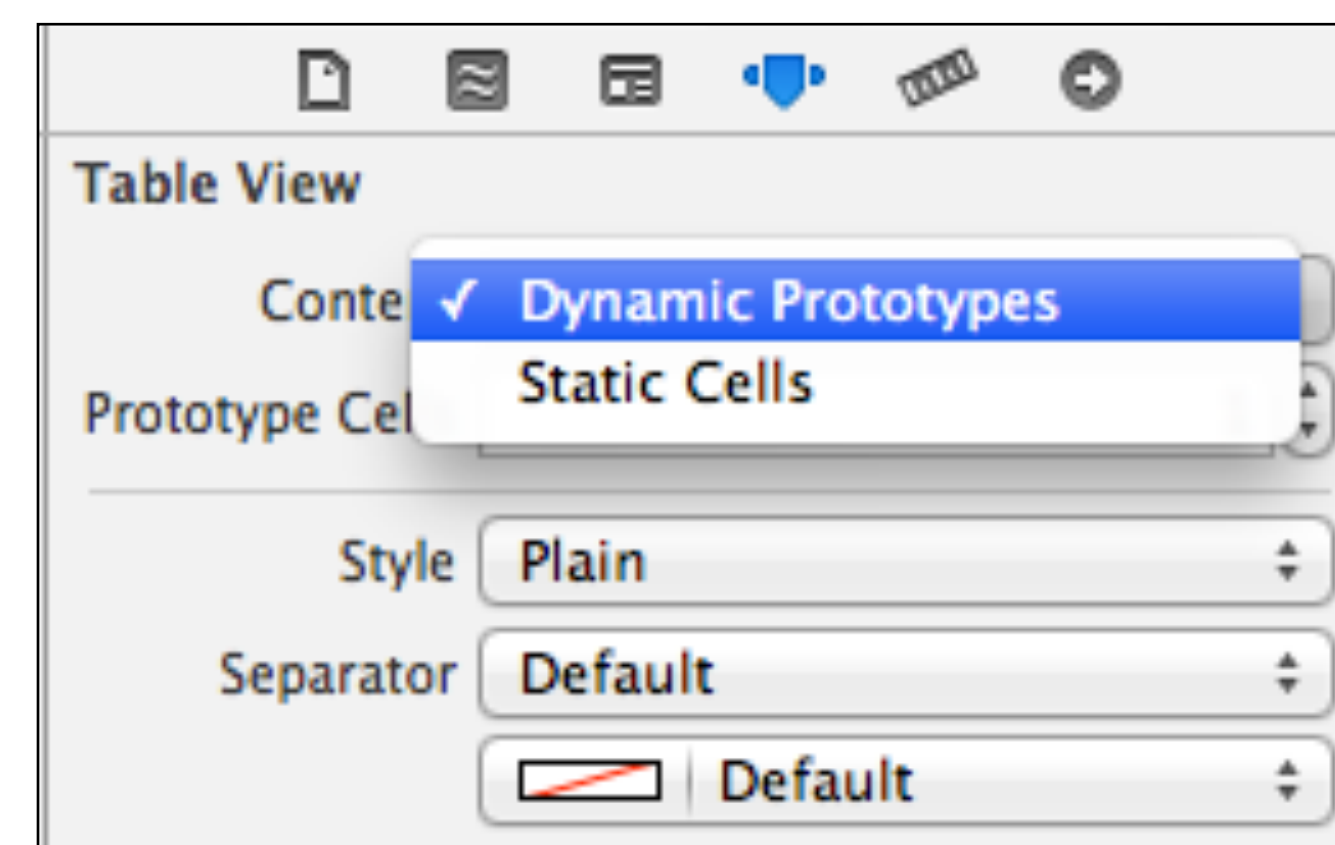
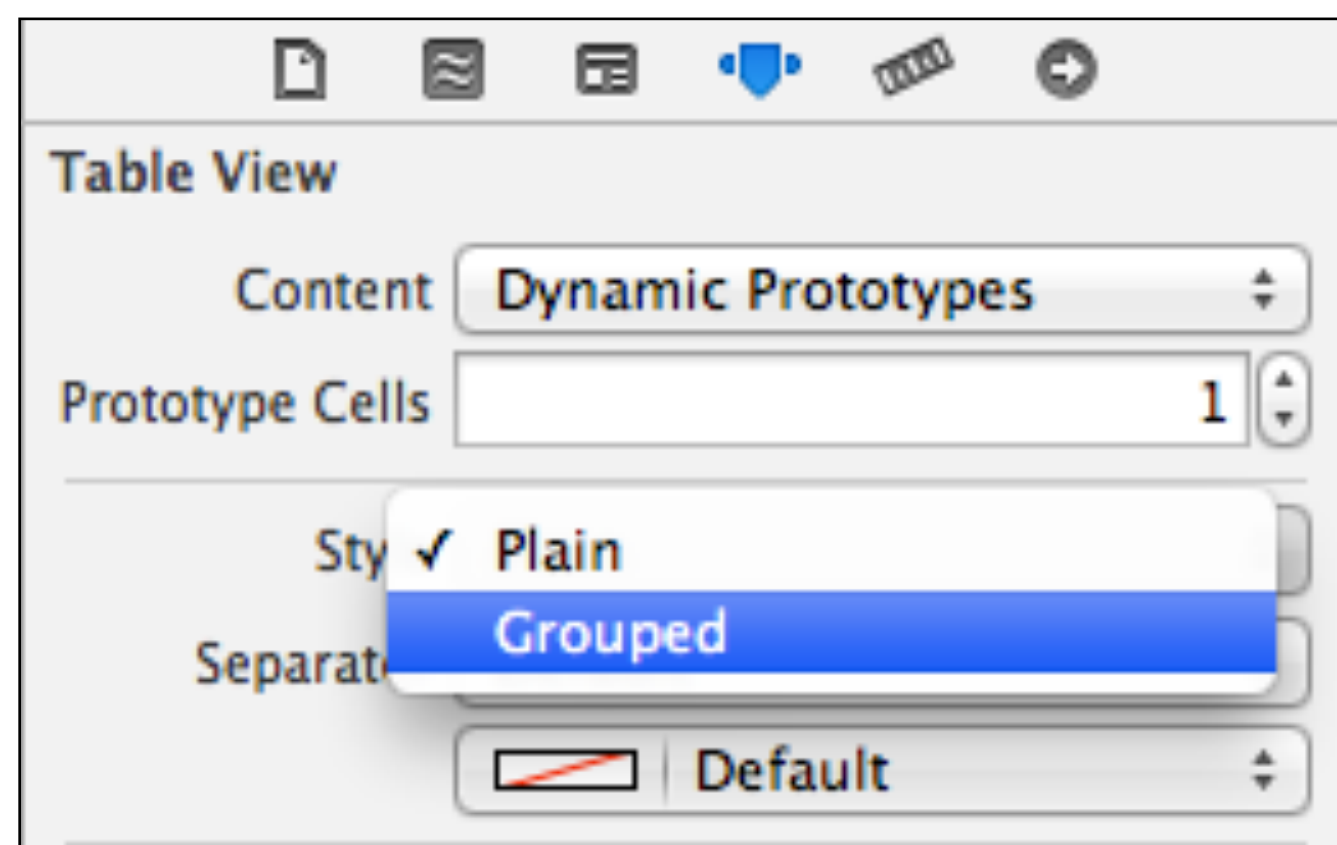
**Create a new Objective-C class (⌘-N) that is a subclass of UITableViewController**

# Table View Controller



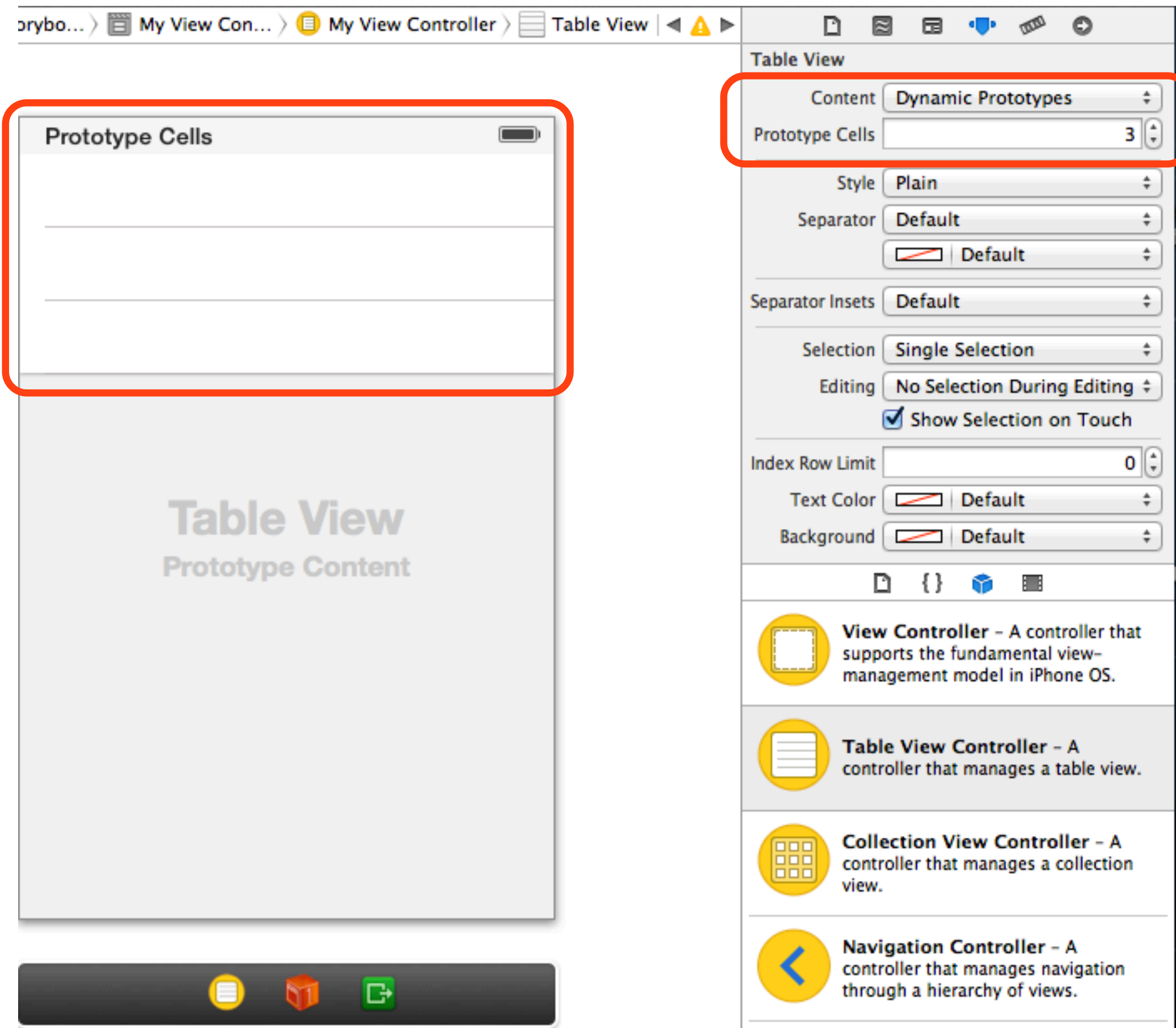
**Set the new class as the class of the view in the Identity inspector**

# Table View Controller



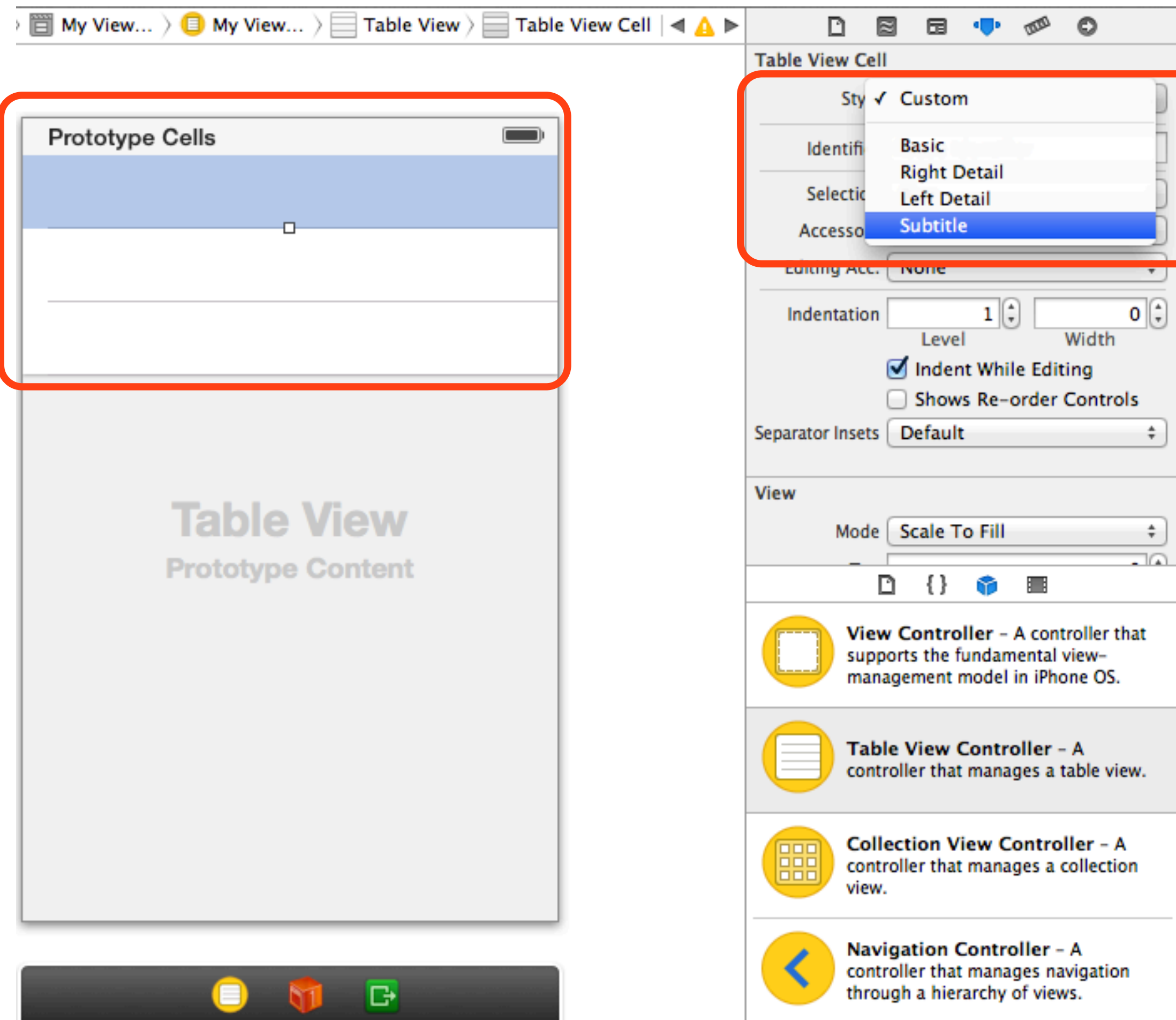
**Configure style (plain or grouped) and type of content (dynamic or static) of the table view**

# Table View Controller

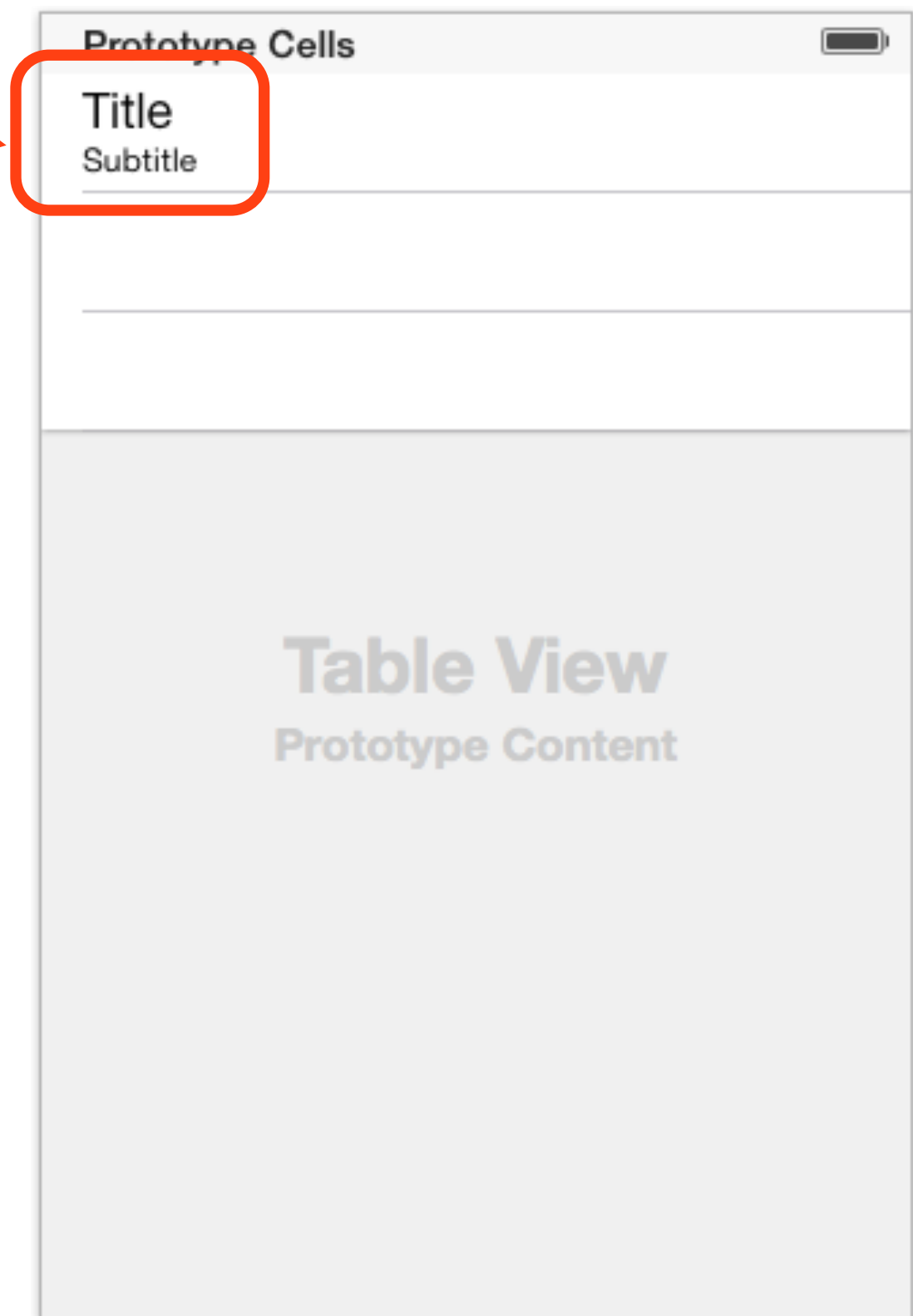


**When the dynamic prototypes option is selected, it is possible to define the prototypes (templates) for the table cells that will be displayed can be configured**

# Table View Controller



**For each prototype cell, it is possible to set its style...**

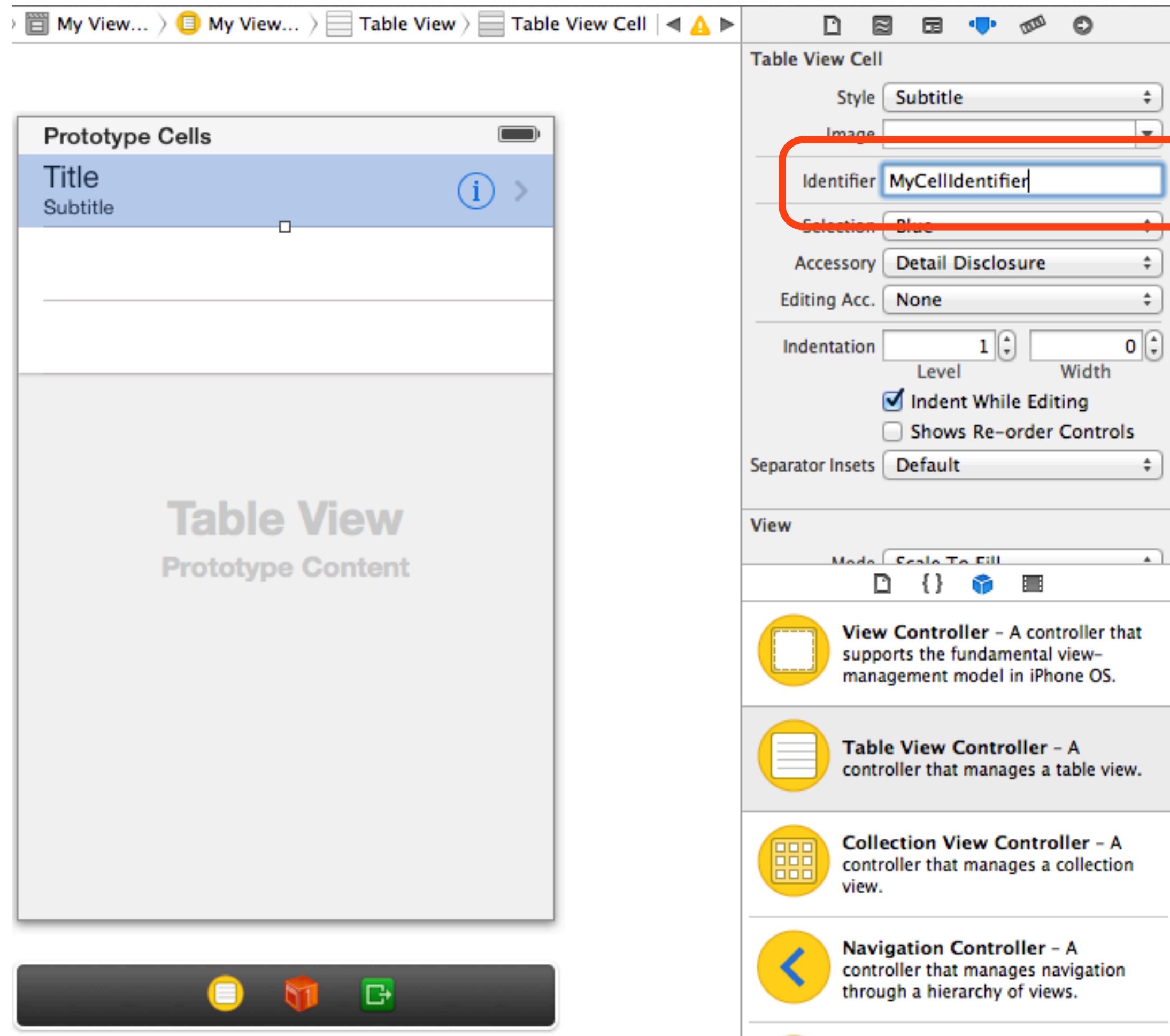


# Table View Controller

... and accessories (which provide a hint of what will happen by selecting a row)

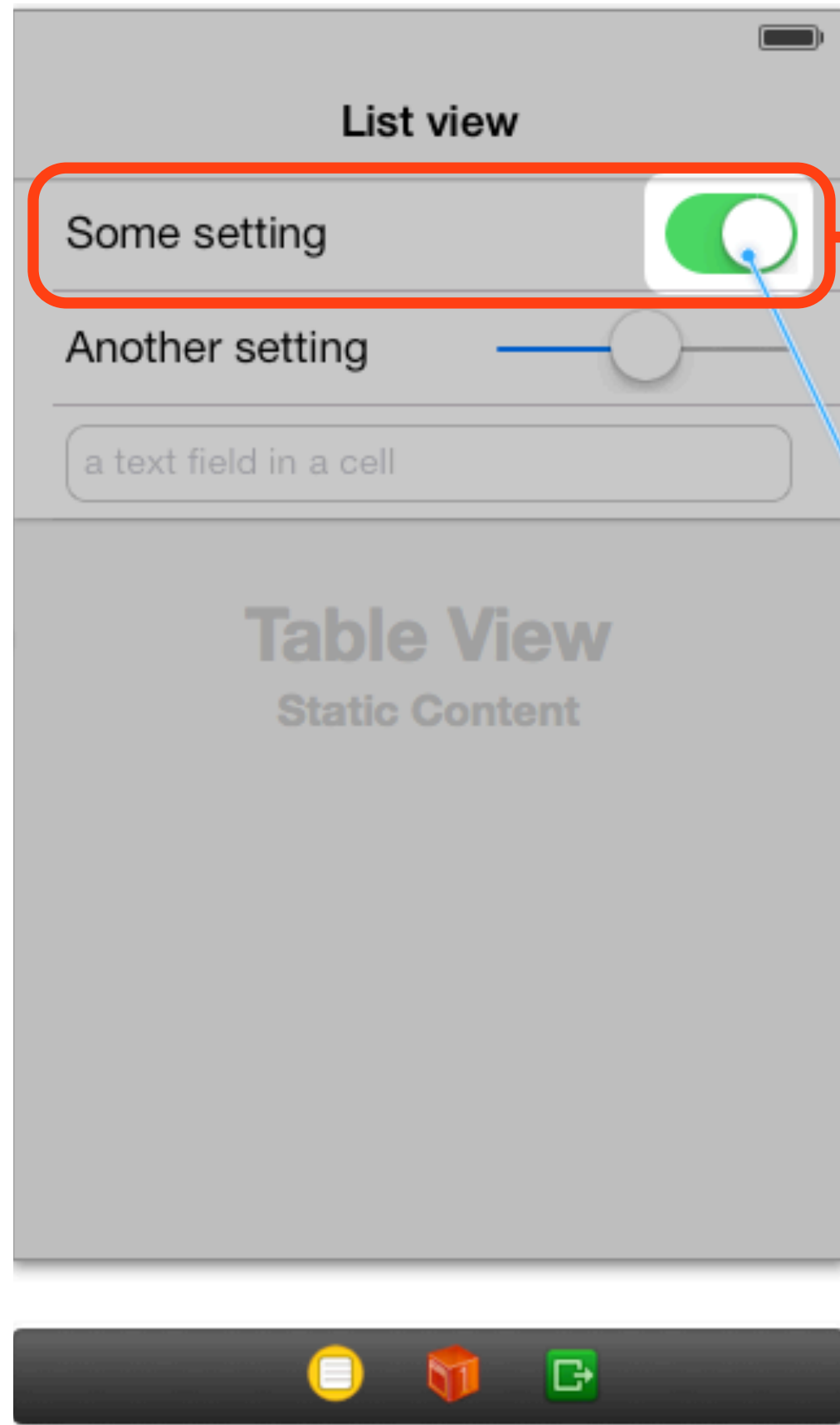


# Table View Controller



**An identifier for the cell must be specified so that it is possible to reference to the cell prototype in code (similar to the identifier of a segue)**

# Table View Controller



```
// Copyright (c) 2013 Università degli Studi di Parma. All rights reserved.
//

#import "MyViewController.h"

@interface MyViewController ()
@end

@implementation MyViewController

- (id)initWithStyle:(UITableViewStyle)style
{
    self = [super initWithStyle:style];
    if (self) {
        // Custom initialization
    }
    return self;
}

- (IBAction)refreshTableView:(UIRefreshControl *)sender {
    [sender beginRefreshing];

    // ... reload data
    [self.tableView reloadData];

    [sender endRefreshing];
}

- (void)viewDidLoad
{
    [super viewDidLoad];

    // Uncomment the following line to preserve selection
    // presentations.
    // self.clearsSelectionOnViewWillAppear = NO;

    // Uncomment the following line to display an Edit but
```

**Static cells can be configured directly in storyboard**

**Views and controls (such as labels, switches, and sliders) can be added to the cells**

Insert Action

# Table View Controller

```
// Copyright (c) 2013 Università degli Studi di Parma. All rights reserved.
//

#import "MyViewController.h"

@interface MyViewController ()
@end

@implementation MyViewController

- (id)initWithStyle:(UITableViewStyle)style
{
    self = [super initWithStyle:style];
    if (self) {
        // Custom initialization
    }
    return self;
}

- (IBAction)refreshTableView:(UIRefreshControl *)sender {
    [sender beginRefreshing];

    // ... reload data
    [self.tableView reloadData];

    [sender endRefreshing];
}

- (void)viewDidLoad
{
    [super viewDidLoad];

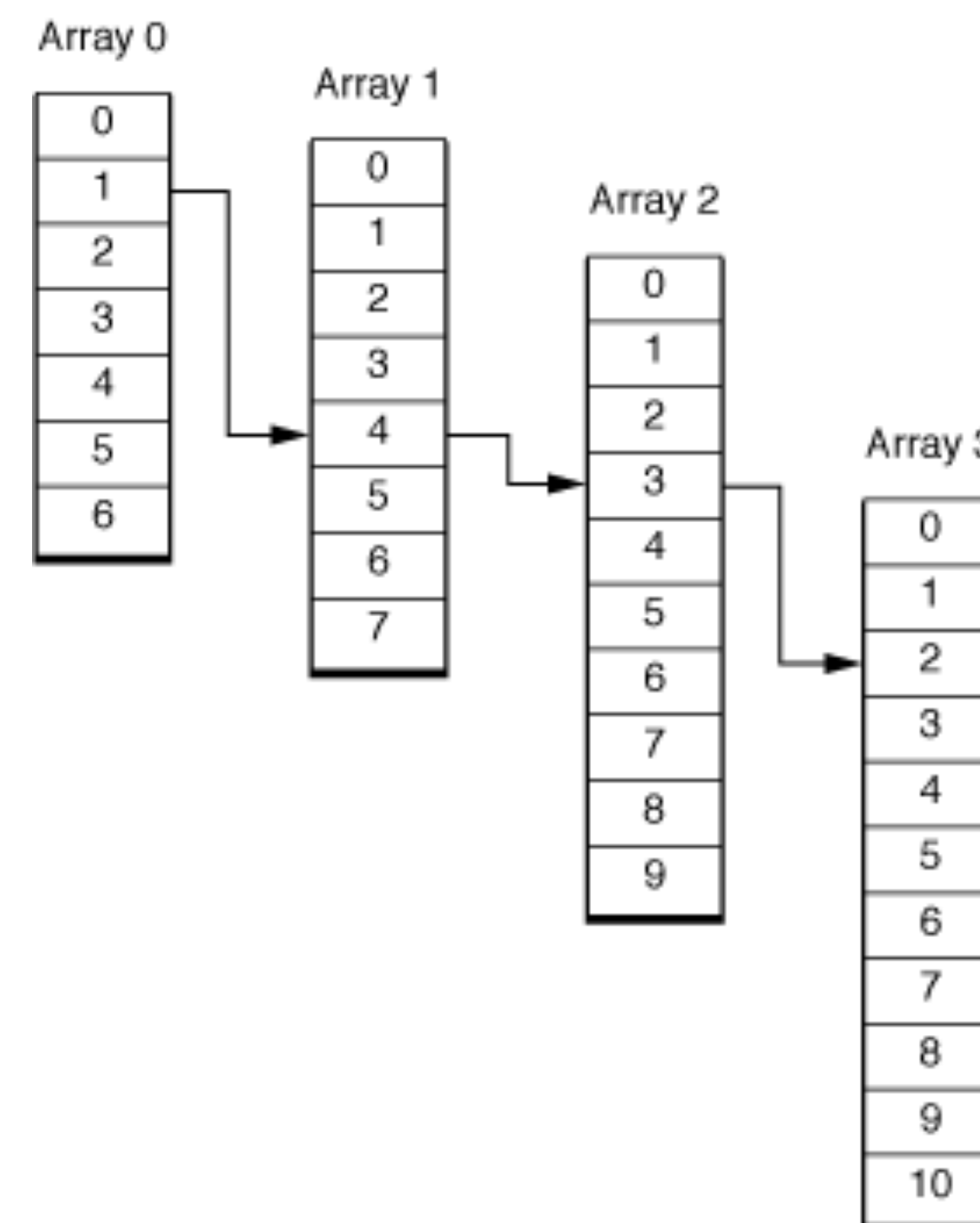
    // Uncomment the following line to preserve selection presentations.
    // self.clearsSelectionOnViewWillAppear = NO;

    // Uncomment the following line to display an Edit button in each cell.
    // self.editingMode = UIEditingModeAlways;
}
```

**Target-actions can be added to the view controller by ctrl-dragging from the control to the implementation file**

# NSIndexPath

- The **NSIndexPath** class is used to represent the path to a specific node in a tree of nested array collections
- In iOS, UIKit adds some functionalities to **NSIndexPath** instances so that they can be used to refer to the position of a cell in a table view
- There are two properties that are used for this purpose:
  - **section**: an index number identifying a section in a table view (or collection view)
  - **row**: an index number identifying a row in a section of a table view
- An instance of **NSIndexPath** is passed in as argument to the methods that refer to a specific cell in the table view in order to identify the cell univocally



# Table view content: UITableViewDataSource

- A table view relies on an external object to inject content into rows, called **data source**
- The data source stands between the model and the table view, so it is common (but not necessary) that the controller is the data source for a table view that it manages
- Typically a data source must provide content to the table by answering to the following questions:
  - ▶ How many sections will the table view have?
  - ▶ How many rows are there in section  $i$ ?
  - ▶ What content (`UITableViewCell`) should be displayed in the cell at section  $i$  and row  $j$ ?
  - ▶ Which title should be displayed for the header and footer of section  $i$ ?
  - ▶ Is a table cell at section  $i$  and row  $j$  editable?
- A table view's data source must conform to the **UITableViewDataSource** protocol
- A data source must be provided for table views that display dynamic contents: it is the **WHAT** that you are displaying
- Static tables do not need to implement **UITableViewDataSource** protocol methods (all automatic)

# UITableViewDataSource

- The `UITableViewDataSource` protocol defines the following methods, which must be implemented by a data source to give enough information to the `UITableView` to draw its content:
  - `(NSInteger)numberOfSectionsInTableView:(UITableView *)tableView`
    - `(NSInteger)tableView:(UITableView *)tableView  
numberOfRowsInSection:(NSInteger)section`
  - `(UITableViewCell *)tableView:(UITableView *)tableView  
cellForRowAtIndexPath:(NSIndexPath *)indexPath`
  - `(NSString *)tableView:(UITableView *)tableView  
titleForHeaderInSection:(NSInteger)section`
  - `(NSString *)tableView:(UITableView *)tableView  
titleForFooterInSection:(NSInteger)section`
    - `(BOOL)tableView:(UITableView *)tableView  
canEditRowAtIndexPath:(NSIndexPath *)indexPath`
    - `(void)tableView:(UITableView *)tableView  
commitEditingStyle:(UITableViewCellEditingStyle)editingStyle  
forRowAtIndexPath:(NSIndexPath *)indexPath`

# UITableViewDataSource

- (`NSInteger`)numberOfSectionsInTableView:(`UITableView *`)tableView

How many sections will the given table view have? (defaults to 1)

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView{  
    // Return the number of sections.  
    return <NUMBER OF SECTIONS>;  
}
```

# UITableViewDataSource

- `(NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section`

How many rows will there be in the given section?

- `(NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section{  
 // Return the number of rows in the section.  
 return <NUMBER OF ROWS FOR SECTION section>;  
}`

# UITableViewDataSource

- (UITableViewCell \*)tableView:(UITableView \*)tableView  
cellForRowAtIndexPath:(NSIndexPath \*)indexPath

What content (UITableViewCell) should be displayed for the cell at the given index path?

```
- (UITableViewCell *)tableView:(UITableView *)tableView  
    cellForRowAtIndexPath:(NSIndexPath *)indexPath{  
  
    static NSString *CellIdentifier = @"MyCellIdentifier";  
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier  
                            forIndexPath:indexPath];  
  
    // Configure the cell...  
  
    return cell;  
}
```

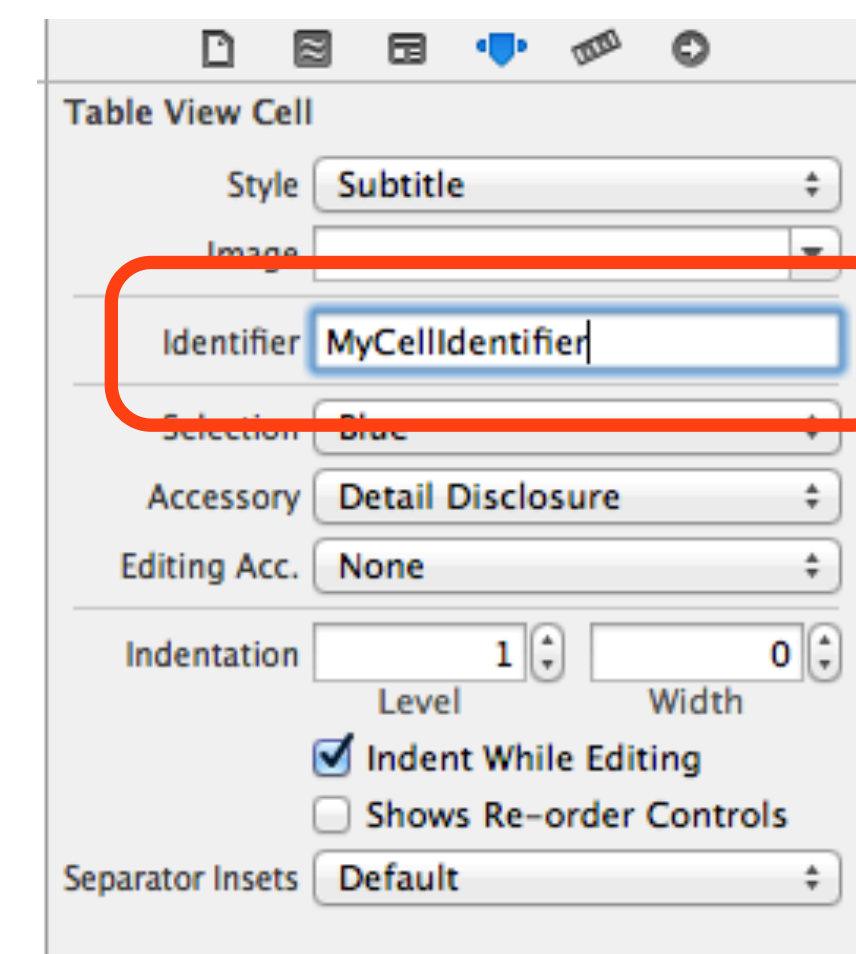
# UITableViewDataSource

- (UITableViewCell \*)tableView:(UITableView \*)tableView  
cellForRowAtIndexPath:(NSIndexPath \*)indexPath

What content (UITableViewCell) should be displayed for the cell at the given index path?

```
- (UITableViewCell *)tableView:(UITableView *)tableView  
cellForRowAtIndexPath:(NSIndexPath *)indexPath{  
  
    static NSString *CellIdentifier = @"MyCellIdentifier";  
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier  
                             forIndexPath:indexPath];  
  
    // Configure the cell...  
  
    return cell;  
}
```

Dynamic prototype identifier specified in the Attribute inspector in storyboard; very important to determine which prototype should be used if there are many



# UITableViewDataSource

- (UITableViewCell \*)tableView:(UITableView \*)tableView  
cellForRowAtIndexPath:(NSIndexPath \*)indexPath

What content (UITableViewCell) should be displayed for the cell at the given index path?

```
- (UITableViewCell *)tableView:(UITableView *)tableView  
cellForRowAtIndexPath:(NSIndexPath *)indexPath{  
  
    static NSString *CellIdentifier = @"MyCellIdentifier";  
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier  
                            forIndexPath:indexPath];  
  
    // Configure the cell...  
  
    return cell;  
}
```

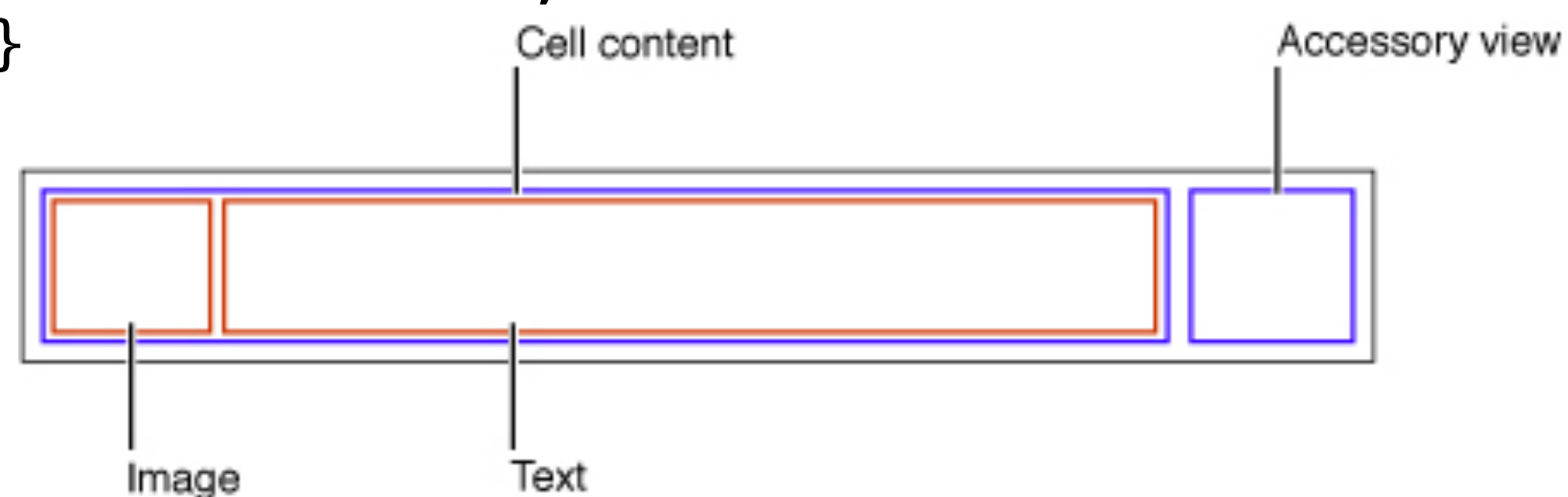
A table view's data source should reuse cell objects for performance reasons (it would be extremely inefficient to always create new cells for each row in the table view: only those on screen should be kept in memory)

# UITableViewDataSource

- (UITableViewCell \*)tableView:(UITableView \*)tableView  
cellForRowAtIndexPath:(NSIndexPath \*)indexPath

What content (UITableViewCell) should be displayed for the cell at the given index path?

- (UITableViewCell \*)tableView:(UITableView \*)tableView  
cellForRowAtIndexPath:(NSIndexPath \*)indexPath{  
  
static NSString \*CellIdentifier = @"MyCellIdentifier";  
UITableViewCell \*cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier  
forIndexPath:indexPath];  
  
// Configure the cell...  
cell.textLabel.text = [NSString stringWithFormat:@"cell at row [%d]", indexPath.row];  
return cell;  
}



Here is where you can set the cell's `textLabel`, `detailLabel`, and `imageView` properties of the cell

# UITableViewDataSource

- (UITableViewCell \*)tableView:(UITableView \*)tableView  
cellForRowAtIndexPath:(NSIndexPath \*)indexPath

What content (UITableViewCell) should be displayed for the cell at the given index path?

- (UITableViewCell \*)tableView:(UITableView \*)tableView  
cellForRowAtIndexPath:(NSIndexPath \*)indexPath{  
  
static NSString \*CellIdentifier = @"MyCellIdentifier";  
UITableViewCell \*cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier  
forIndexPath:indexPath];  
  
// Configure the cell...  
cell.textLabel.text = [NSString stringWithFormat:@"cell at row [%d]", indexPath.row];  
return cell;  
}

Custom table view cells can also be used to display special content that does not fit into the system-provided cell types

# UITableViewDataSource

- (NSString \*)tableView:(UITableView \*)tableView titleForHeaderInSection:(NSInteger)section
- (NSString \*)tableView:(UITableView \*)tableView titleForFooterInSection:(NSInteger)section

What is the title for the given section's header/footer?

```
- (NSString *)tableView:(UITableView *)tableView titleForHeaderInSection:(NSInteger)section{  
    return [NSString stringWithFormat:@"Header %d",section];  
}
```

```
- (NSString *)tableView:(UITableView *)tableView titleForFooterInSection:(NSInteger)section{  
    return [NSString stringWithFormat:@"Footer %d",section];  
}
```

# Table view behavior: UITableViewDelegate

- A table view relies on an external object to specify its appearance and behavior, called **delegate**
- The delegate is needed to manage selections, configure section headings and footers, help to delete and reorder cells
- It is common (but not necessary) that the controller is the delegate for a table view that it manages
- Typically a delegate must implement methods that answer to the following questions:
  - ▶ What should be done if a cell is about to be/was selected/deselected? (instead of just following a segue)
  - ▶ What should be done if a cell's accessory button was tapped? (a special behavior can be supplied)
  - ▶ Which height should a cell have?
  - ▶ Which view should be displayed for the header/footer of a given section?
- A table view's delegate must conform to the **UITableViewDelegate** protocol
- A delegate must be provided for table views that must manage selections and can have particular behavior and/or appearance: it is the HOW the table view is being displayed

# UITableViewDelegate

- (NSIndexPath \*)tableView:(UITableView \*)tableView willSelectRowAtIndexPath:(NSIndexPath \*)indexPath
- (NSIndexPath \*)tableView:(UITableView \*)tableView didSelectRowAtIndexPath:(NSIndexPath \*)indexPath
- (NSIndexPath \*)tableView:(UITableView \*)tableView willDeselectRowAtIndexPath:(NSIndexPath \*)indexPath
- (NSIndexPath \*)tableView:(UITableView \*)tableView didDeselectRowAtIndexPath:(NSIndexPath \*)indexPath
  - (void)tableView:(UITableView \*)tableView accessoryButtonTappedForRowWithIndexPath:(NSIndexPath \*)indexPath
  - (CGFloat)tableView:(UITableView \*)tableView heightForRowAtIndexPath:(NSIndexPath \*)indexPath
  - (UIView \*)tableView:(UITableView \*)tableView viewForHeaderInSection:(NSInteger)section
  - (UIView \*)tableView:(UITableView \*)tableView viewForFooterInSection:(NSInteger)section

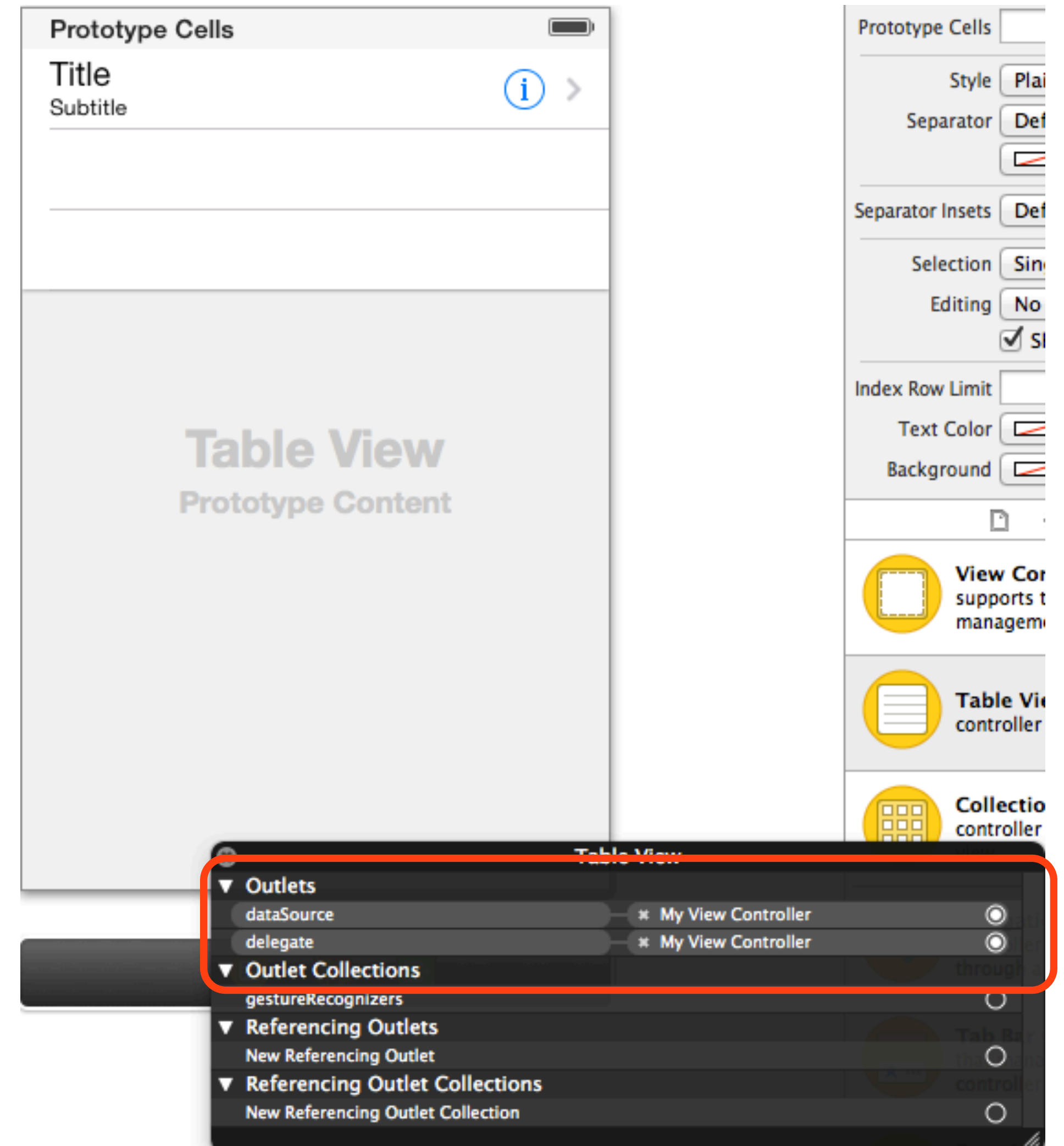
# UITableViewController

- A `UITableViewController` sets itself as the data source and delegate for the table view it controls
- A `UITableViewController` has a `tableView` property which can be used to refer to the table view

```
@property UITableView *tableView;
```

- For a `UITableViewController` the following identity holds:

```
self.view == self.tableView;
```



# UITableViewController

- When a subclass of `UITableViewController` is created, Xcode provides a skeleton implementation of the most important methods that must be implemented to conform to the `UITableViewDataSource` protocol

```
#pragma mark - Table view data source

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView{
    // Return the number of sections.
    return 1;
}

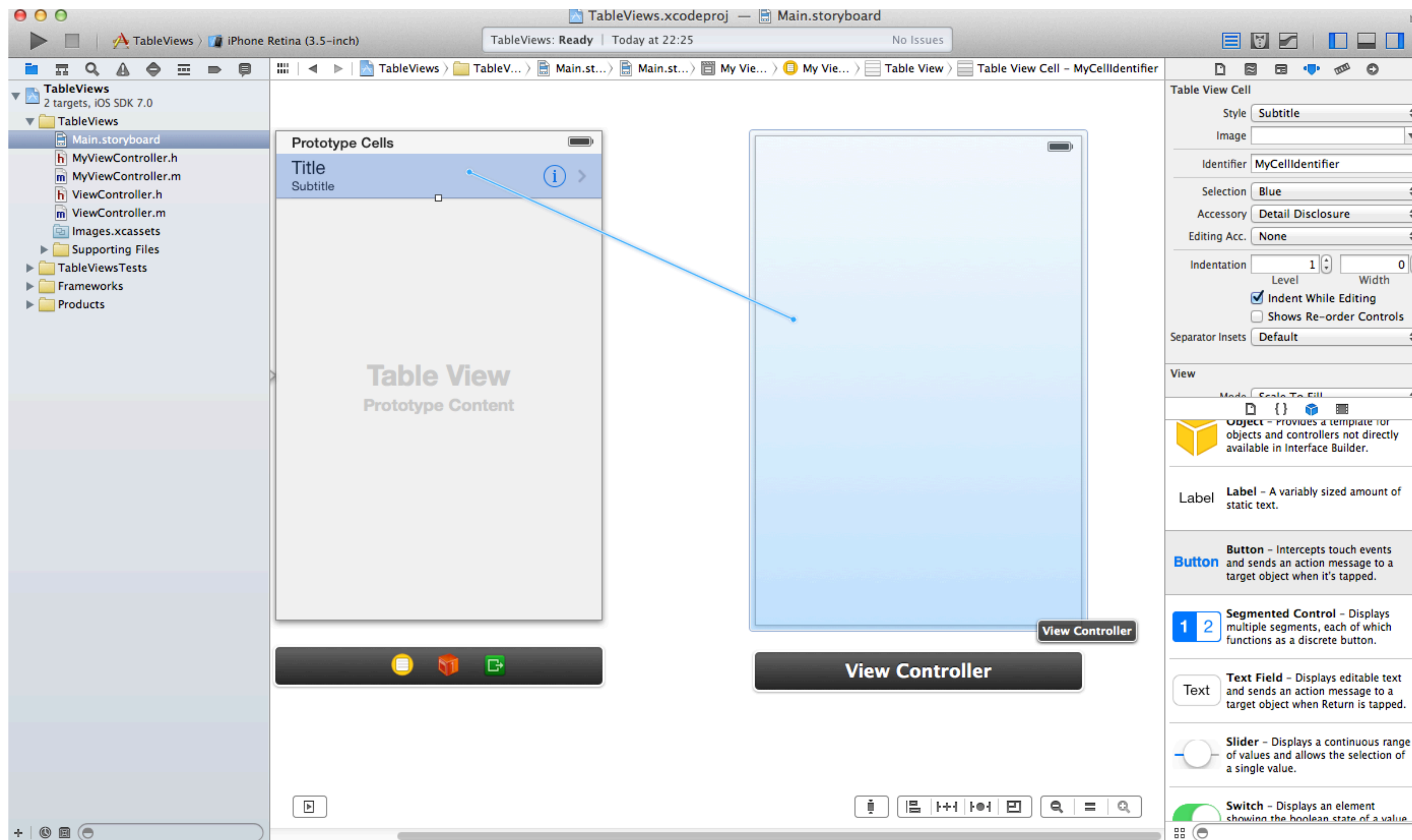
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section{
    // Return the number of rows in the section.
    return 0;
}

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath{
    static NSString *CellIdentifier = @"MyCellIdentifier";
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier forIndexPath:indexPath];

    // Configure the cell...

    return cell;
}
```

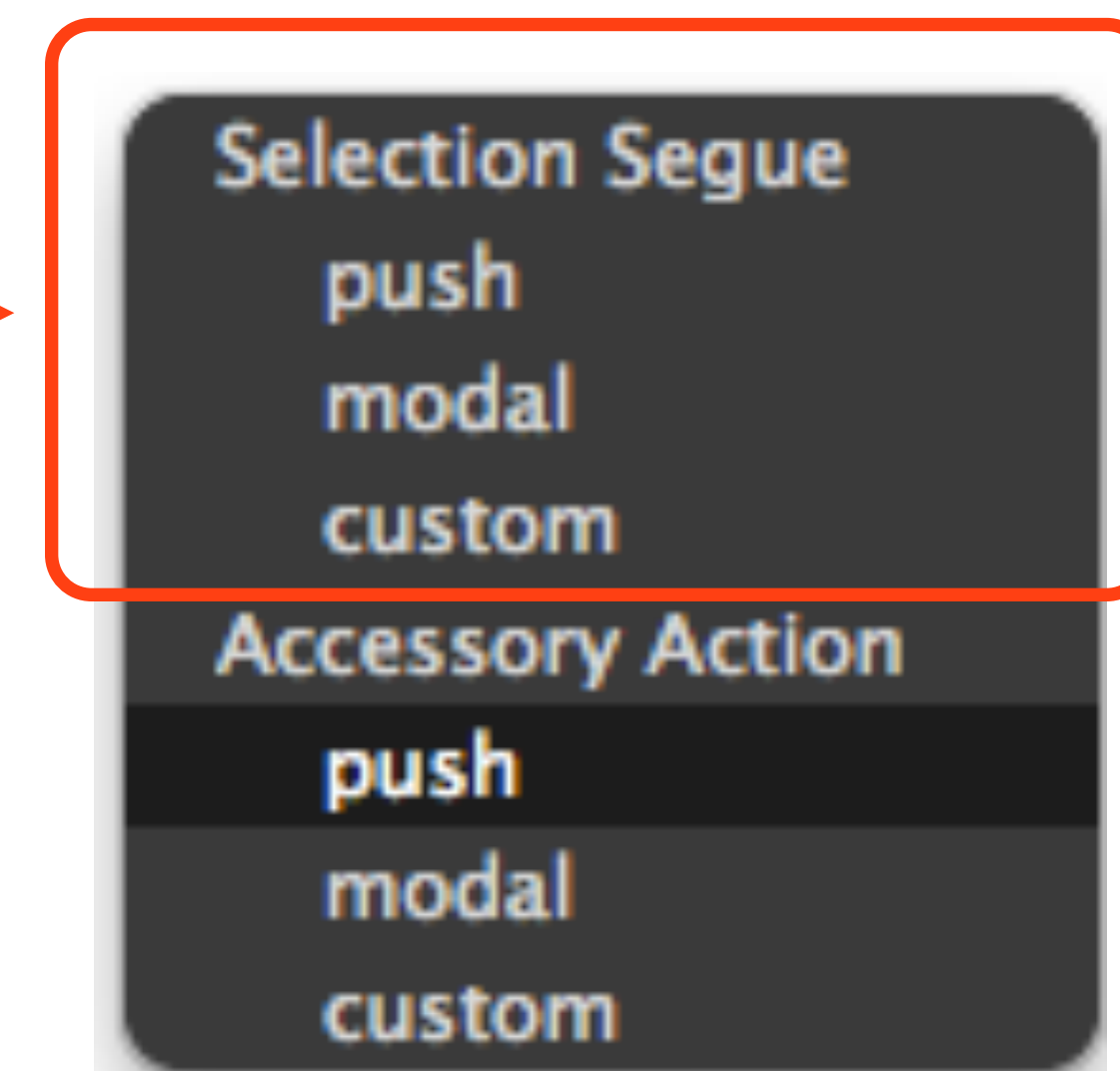
# Segue from a UITableViewCell



**Ctrl-drag from the cell prototype to the destination view controller**

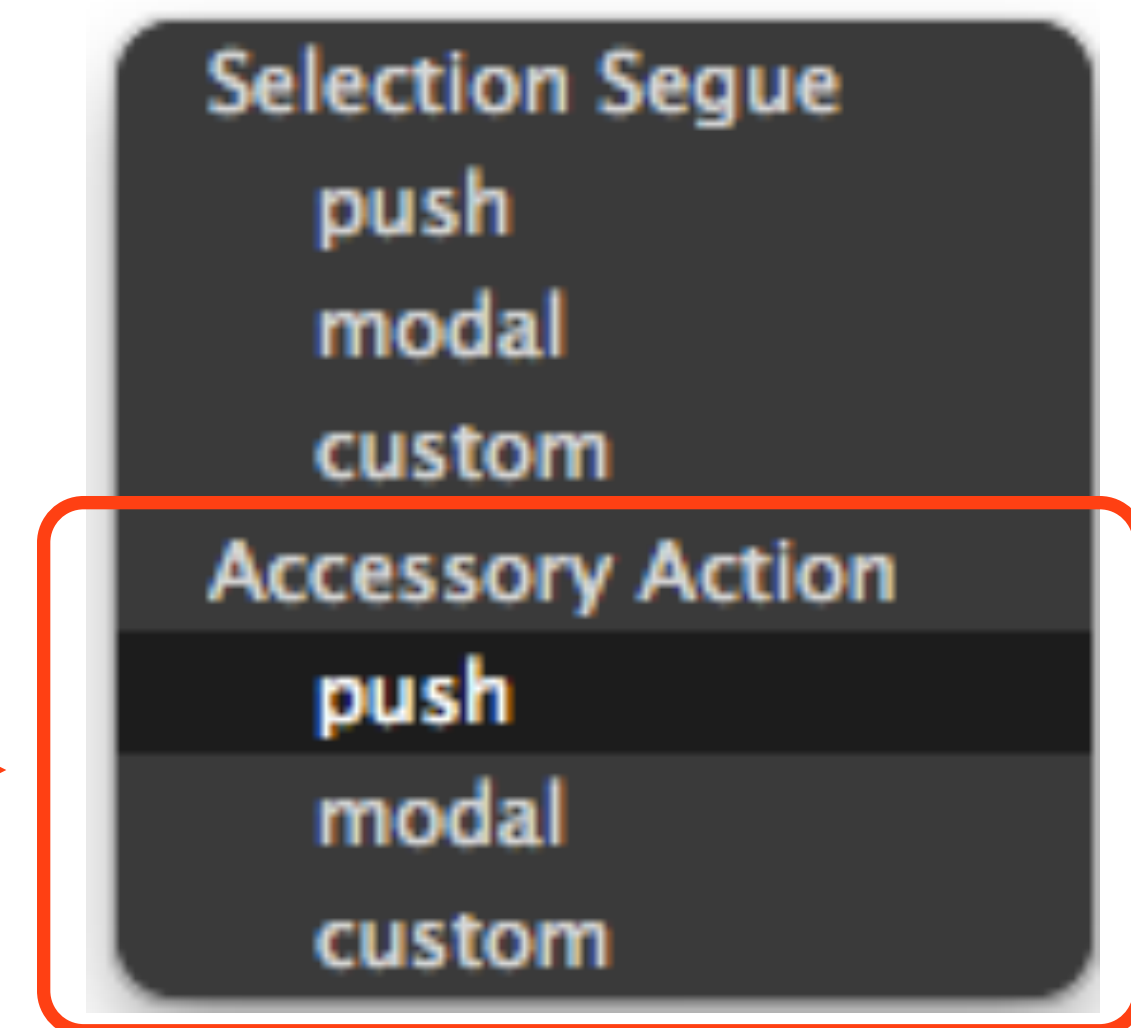
# Segue from a UITableViewCell

**A segue can be configured to start when the cell is selected...**



# Segue from a UITableViewCell

**...or when the cell's accessory is tapped**



# Preparing for a segue from a UITableViewCell

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender{  
    // Get the new view controller using [segue destinationViewController].  
    // Pass the selected object to the new view controller.  
  
    NSIndexPath *indexPath = [self.tableView indexPathForCell:sender];  
  
    if([segue.destinationViewController isKindOfClass:[SomeViewController class]]){  
        SomeViewController *vc = (SomeViewController *)segue.destinationViewController;  
        vc.title = [NSString stringWithFormat:@"Detail %d", indexPath.row];  
    }  
}
```

# Preparing for a segue from a UITableViewCell

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender{  
    // Get the new view controller using [segue destinationViewController].  
    // Pass the selected object to the new view controller.  
    NSIndexPath *indexPath = [self.tableView indexPathForCell:sender];  
    if([segue.destinationViewController isKindOfClass:[SomeViewController class]]){  
        SomeViewController *vc = (SomeViewController *)segue.destinationViewController;  
        vc.title = [NSString stringWithFormat:@"Detail %d", indexPath.row];  
    }  
}
```

The sender is the table view cell that was selected

# Preparing for a segue from a UITableViewCell

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender{  
    // Get the new view controller using [segue destinationViewController].  
    // Pass the selected object to the new view controller.  
    NSIndexPath *indexPath = [self.tableView indexPathForCell:sender];  
    if([segue.destinationViewController isKindOfClass:[SomeViewController class]]){  
        SomeViewController *vc = (SomeViewController *)segue.destinationViewController;  
        vc.title = [NSString stringWithFormat:@"Detail %d", indexPath.row];  
    }  
}
```

The index path of the cell can be retrieved with the `UITableView` method `indexPathForCell:`

# Preparing for a segue from a UITableViewCell

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender{  
    // Get the new view controller using [segue destinationViewController].  
    // Pass the selected object to the new view controller.  
  
    NSIndexPath *indexPath = [self.tableView indexPathForCell:sender];  
  
    if([segue.destinationViewController isKindOfClass:[SomeViewController class]]){  
        SomeViewController *vc = (SomeViewController *)segue.destinationViewController;  
        vc.title = [NSString stringWithFormat:@"Detail %d", indexPath.row];  
    }  
}
```

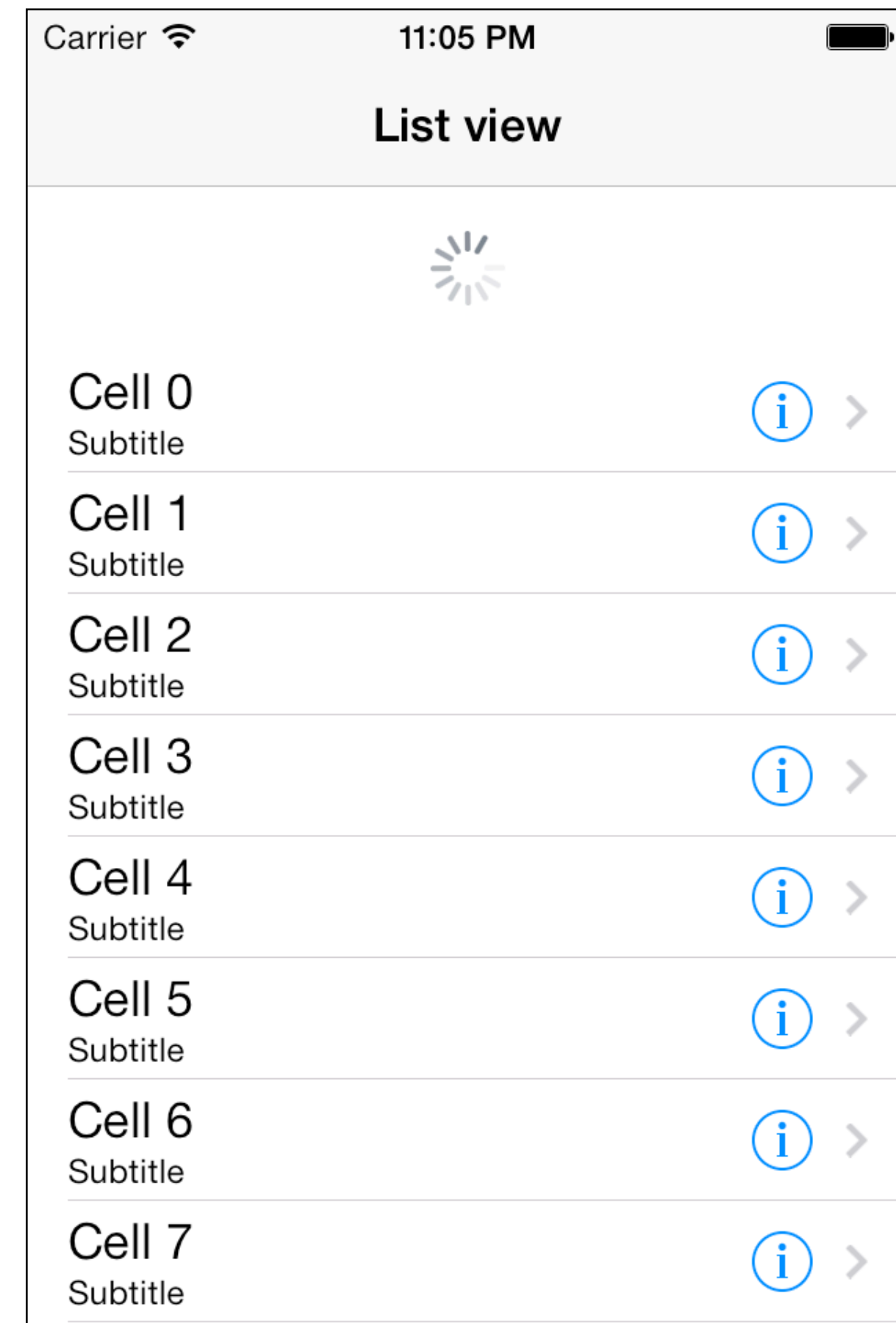


Pass data to the destination view controller

# Pull to refresh

- All `UITableViewController` instances come with a built-in activity indicator that can be used to indicate that table view data are being reloaded
- If an action has been set for the refresh control, it will be triggered when the user pulls the table view down (usually referred to as “pull-to-refresh”)

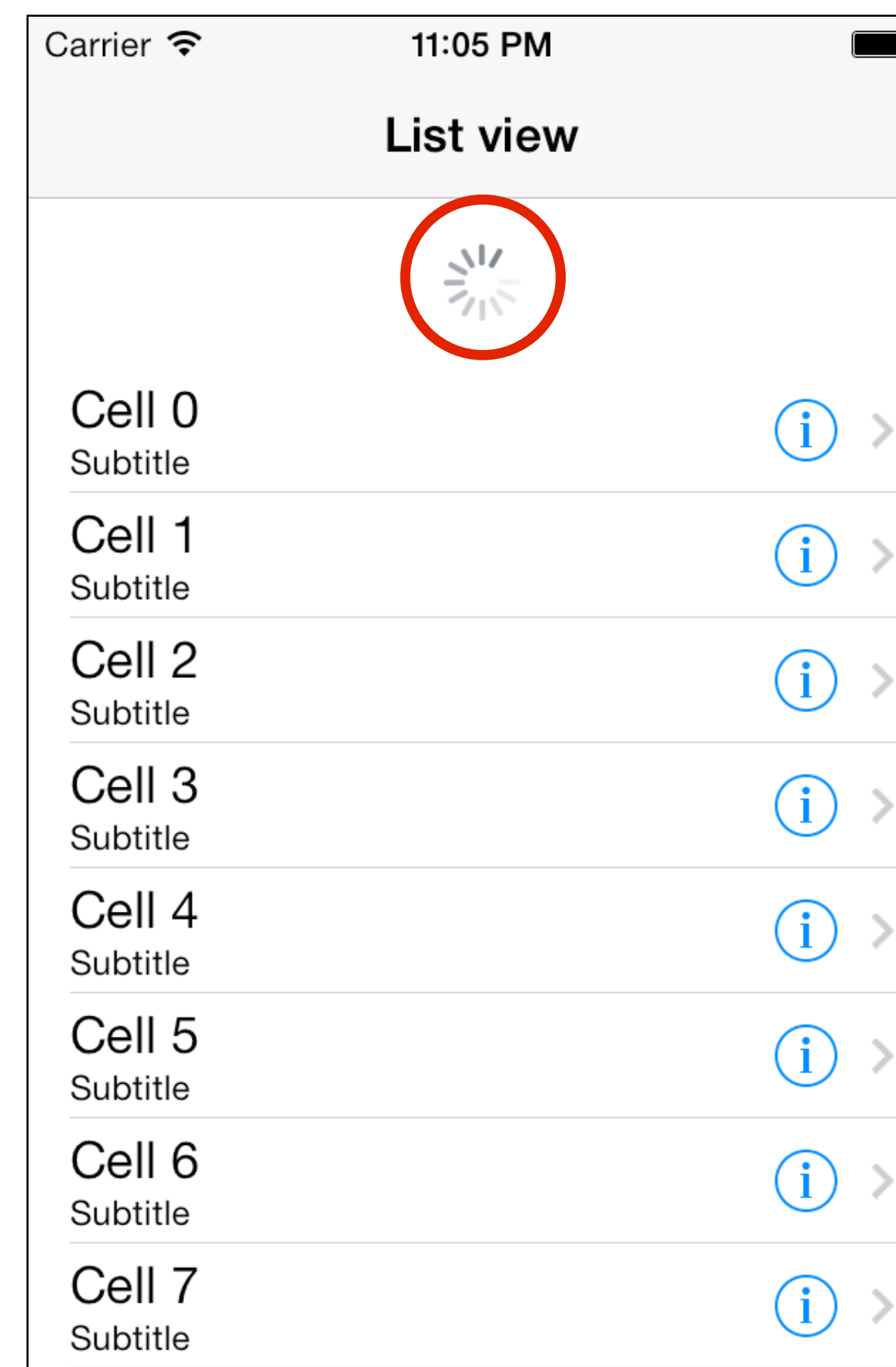
```
@property UIRefreshControl *refreshControl;
```



# Pull to refresh

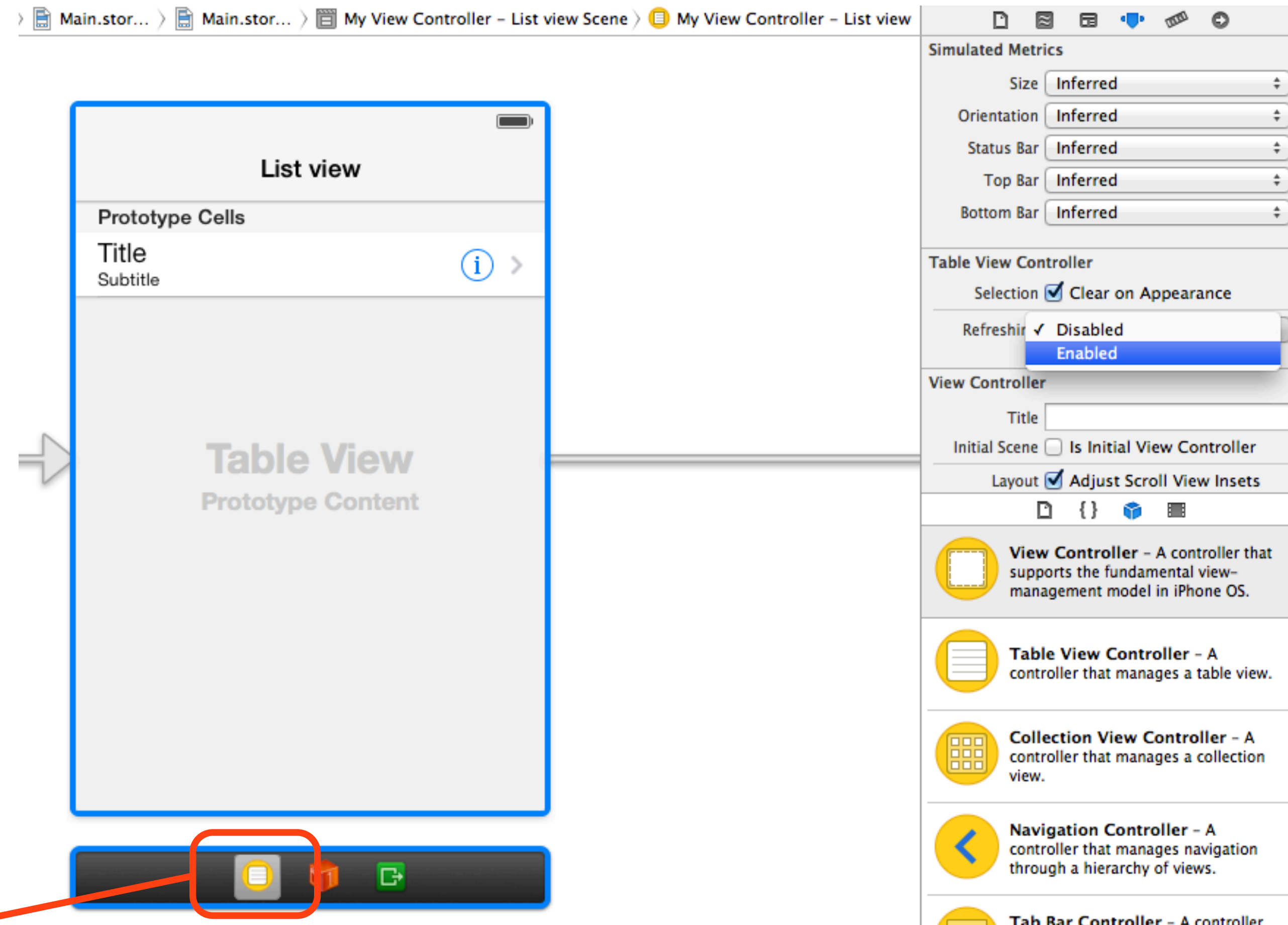
- All `UITableViewController` instances come with a built-in activity indicator that can be used to indicate that table view data are being reloaded
- If an action has been set for the refresh control, it will be triggered when the user pulls the table view down (usually referred to as “pull-to-refresh”)

```
@property UIRefreshControl *refreshControl;
```



# Pull to refresh

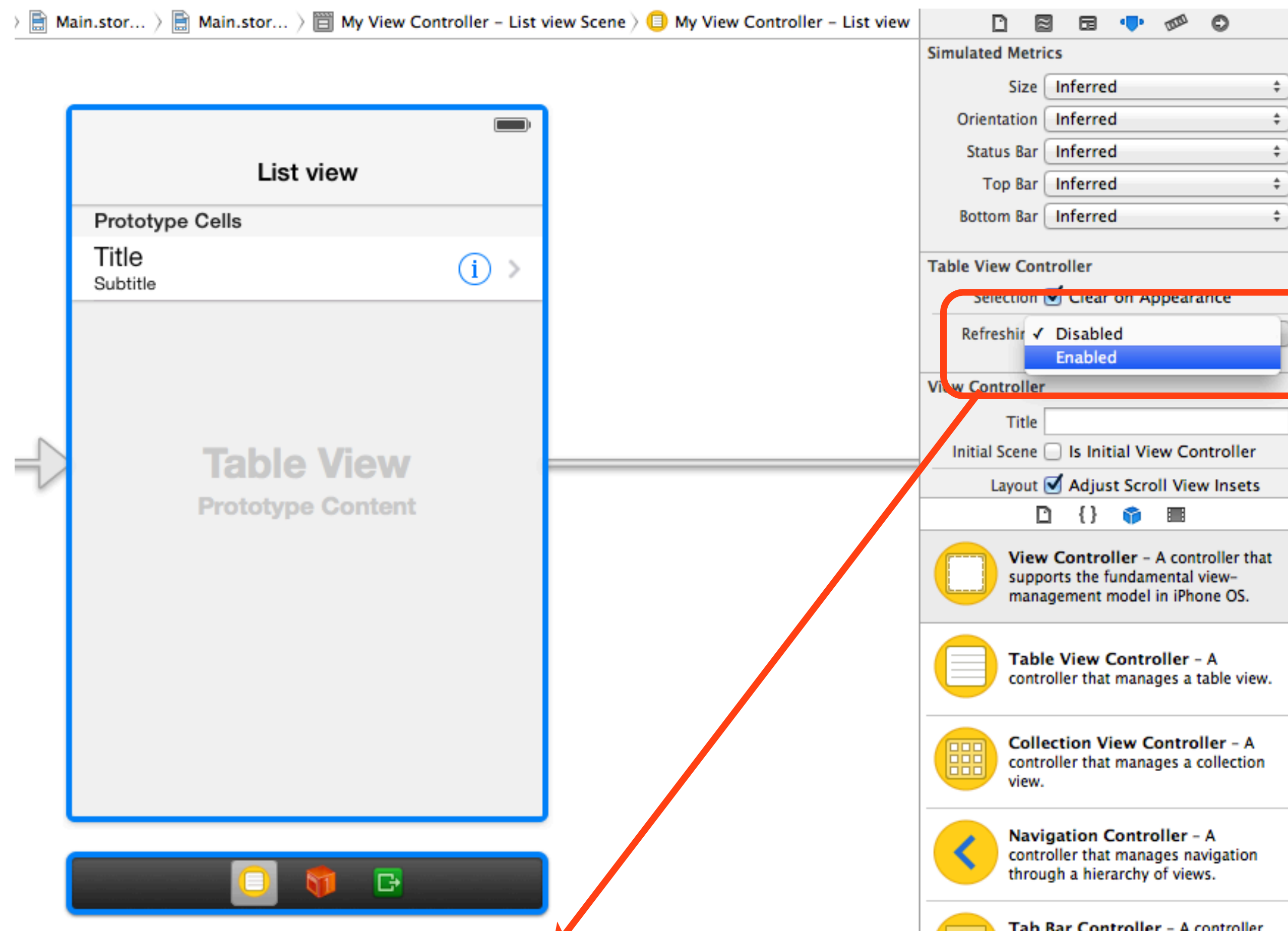
- The activity view is called refresh control and can be enabled in the Attribute inspector of the `UITableViewController`



Select the `UIViewController`

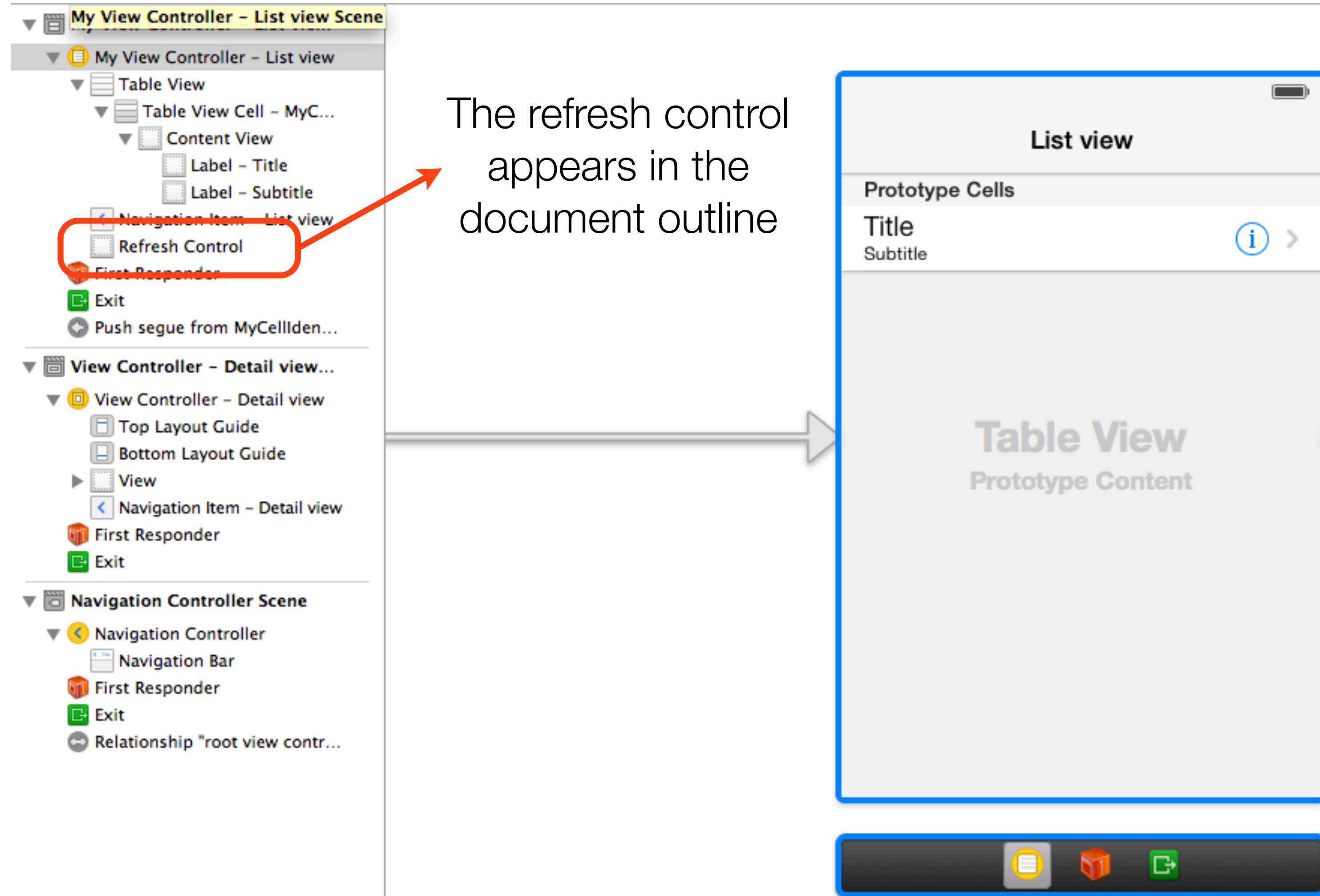
# Pull to refresh

- The activity view is called refresh control and can be enabled in the Attribute inspector of the `UITableViewController`



Enable the Refreshing option

# Pull to refresh



The refresh control appears in the document outline

# Pull to refresh

```
// MyViewController.m
// TableViews
// Created by Simone Cirani on 03/12/13.
// Copyright (c) 2013 Università degli Studi di Parma. All rights reserved.

#import "MyViewController.h"

@interface MyViewController ()
@end

@implementation MyViewController

- (id)initWithStyle:(UITableViewStyle)style
{
    self = [super initWithStyle:style];
    if (self) {
        // Custom initialization
    }
    return self;
}

- (void)viewDidLoad
{
    [super viewDidLoad];

    // Uncomment the following line to preserve selection between
    // presentations.
    // self.clearsSelectionOnViewWillAppear = NO;

    // Uncomment the following line to display an Edit button in the navigation
    // bar for this view controller.
    // self.navigationItem.rightBarButtonItem = self.editButtonItem;
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}
```

**Ctrl-drag from the refresh control to insert a target-action**

# Pull to refresh

```
- (IBAction)refreshTableView:(UIRefreshControl *)sender {  
    [sender beginRefreshing];  
  
    // ... reload data  
  
    [self.tableView reloadData];  
  
    [sender endRefreshing];  
}
```

# Pull to refresh

```
- (IBAction)refreshTableView:(UIRefreshControl *)sender {  
    [sender beginRefreshing];  
    // ... reload data  
    [self.tableView reloadData];  
    [sender endRefreshing];  
}
```

Start the spinner

# Pull to refresh

```
- (IBAction)refreshTableView:(UIRefreshControl *)sender {  
    [sender beginRefreshing];  
    // ... reload data  
    [self.tableView reloadData];  
    [sender endRefreshing];  
}
```

Fetch fresh data in  
background queue

# Pull to refresh

```
- (IBAction)refreshTableView:(UIRefreshControl *)sender {  
    [sender beginRefreshing];  
  
    // ... reload data  
    [self.tableView reloadData];  
  
    [sender endRefreshing];  
}
```

Redraw the cells

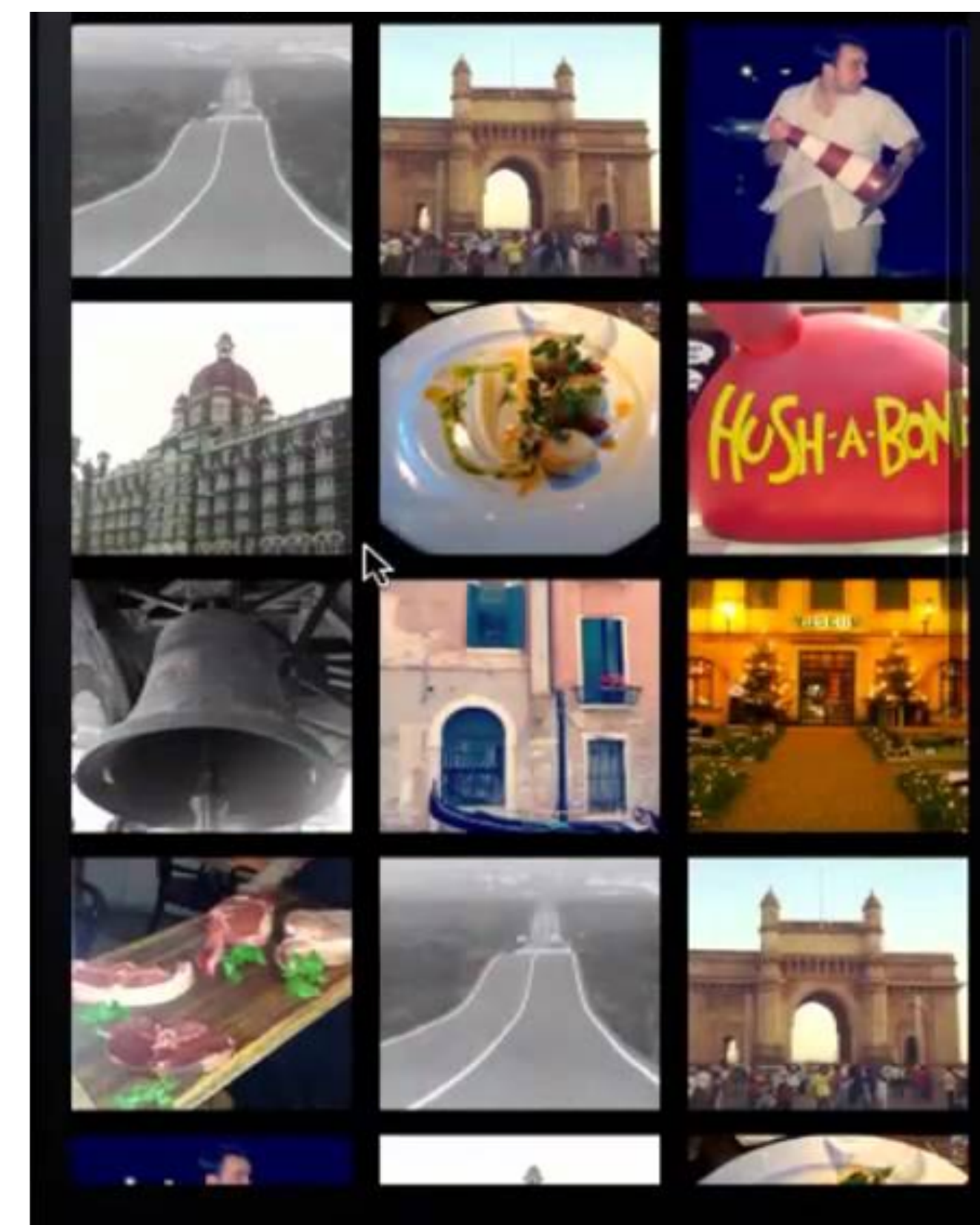
# Pull to refresh

```
- (IBAction)refreshTableView:(UIRefreshControl *)sender {  
    [sender beginRefreshing];  
  
    // ... reload data  
  
    [self.tableView reloadData];  
    [sender endRefreshing];  
}
```

→ Stop the spinner

# UICollectionView

- A collection view displays an ordered collection of data items using standard or custom layouts
  - ▶ Typically, collection views present items in a grid-like arrangement
  - ▶ More complex layouts can be presented, such as stacks, circular layouts, dynamically changing layouts
- A collection view should be used to display nonhierarchical, ordered data items
- The `UICollectionView` class implements the collection view
- The content of collection views is presented in cells (`UICollectionViewCell`)
- Just like a `UITableView`, a `UICollectionView` requires a data source (`UICollectionViewDataSource`) to provide content and a delegate (`UICollectionViewDelegate`) to provide behavior
- The placement of cells, supplementary views, and decoration views inside the collection view is determined by a subclass of `UICollectionViewLayout`



# UIWebView

- A web view is used to embed web content, loaded from a URL, into an application
- Web views support browsing and moving back and forward in the history
- The **UIWebView** class implements the web view
- A **UIWebView** object can be dragged into the view controller in storyboard or created programmatically and added to the view
- A **UIWebView** can then be linked as an outlet by ctrl-dragging from the UIWebView to the interface declaration of the view controller
- It is possible to configure the **UIWebView** to automatically scale web content to fit on the screen by setting the **scalesPagesToFit** property

# UIWebView

– To load content into the web view, two methods can be used:

– `loadRequest:`

```
NSURL *url = [NSURL URLWithString:@"http://mobdev.ce.unipr.it"];
NSURLRequest *request = [NSURLRequest requestWithURL:url];
[self.webView loadRequest:request];
```

– `loadHTMLString:baseUrl:`

```
NSString *html = @"<html><body>Hello, MobDev!</body></html>";
[self.webView loadHTMLString:html baseUrl:url];
```

# UIWebViewDelegate

- A web view can have a delegate that can be used to track the loading of web content
- The delegate object must conform to the `UIWebViewDelegate` protocol
- Set as web view delegate:

```
self.webView.delegate = self;
```

- A delegate's class must declare to conform to the `UIWebViewDelegate` protocol and implement the optional methods:

```
@interface WebViewViewController () <UIWebViewDelegate>
```

- If a delegate has been set for a web view, it must be set to nil before disposing the web view instance (e.g. in the dealloc method where the web view is released)

```
- (void)dealloc{  
    self.webView.delegate = nil;  
}
```

# UIWebViewDelegate methods

```
- (BOOL)webView:(UIWebView *)webView  
shouldStartLoadWithRequest:(NSURLRequest *)request  
navigationType:(UIWebViewNavigationType)navigationType
```

- This method is called before a web view begins loading a frame and can be use to “trap” the loading
- If the method returns **YES** the content is loaded, otherwise the loading will not occur
- Arguments passed in:
  - **webView**: the web view that is about to load content
  - **request**: the URL loading request, which can be used to get the URL that is about to load
  - **navigationType**: the type of action (a link was clicked, a form submitted, a page was reloaded, ...)

# UIWebViewDelegate methods

– (void)webViewDidStartLoad:(UIWebView \*)webView

- This method is called after a web view has started loading a frame
- Arguments passed in:
  - **webView**: the web view that is loading content

# UIWebViewDelegate methods

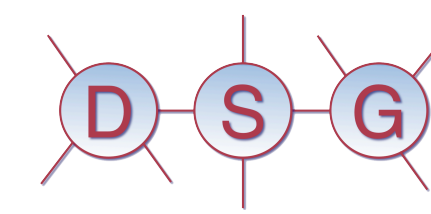
– (**void**)webViewDidFinishLoad:(UIWebView \*)webView

- This method is called after a web view has finished loading a frame
- Arguments passed in:
  - **webView**: the web view that is loaded content

# UIWebViewDelegate methods

```
- (void)webView:(UIWebView *)webView  
didFailLoadWithError:(NSError *)error
```

- This method is called if a web view fails to load
- Arguments passed in:
  - **webView**: the web view that is about to load content
  - **error**: the error that occurred during the loading



# Mobile Application Development

Lecture 18

Scroll View, Table View, Collection View, Web View