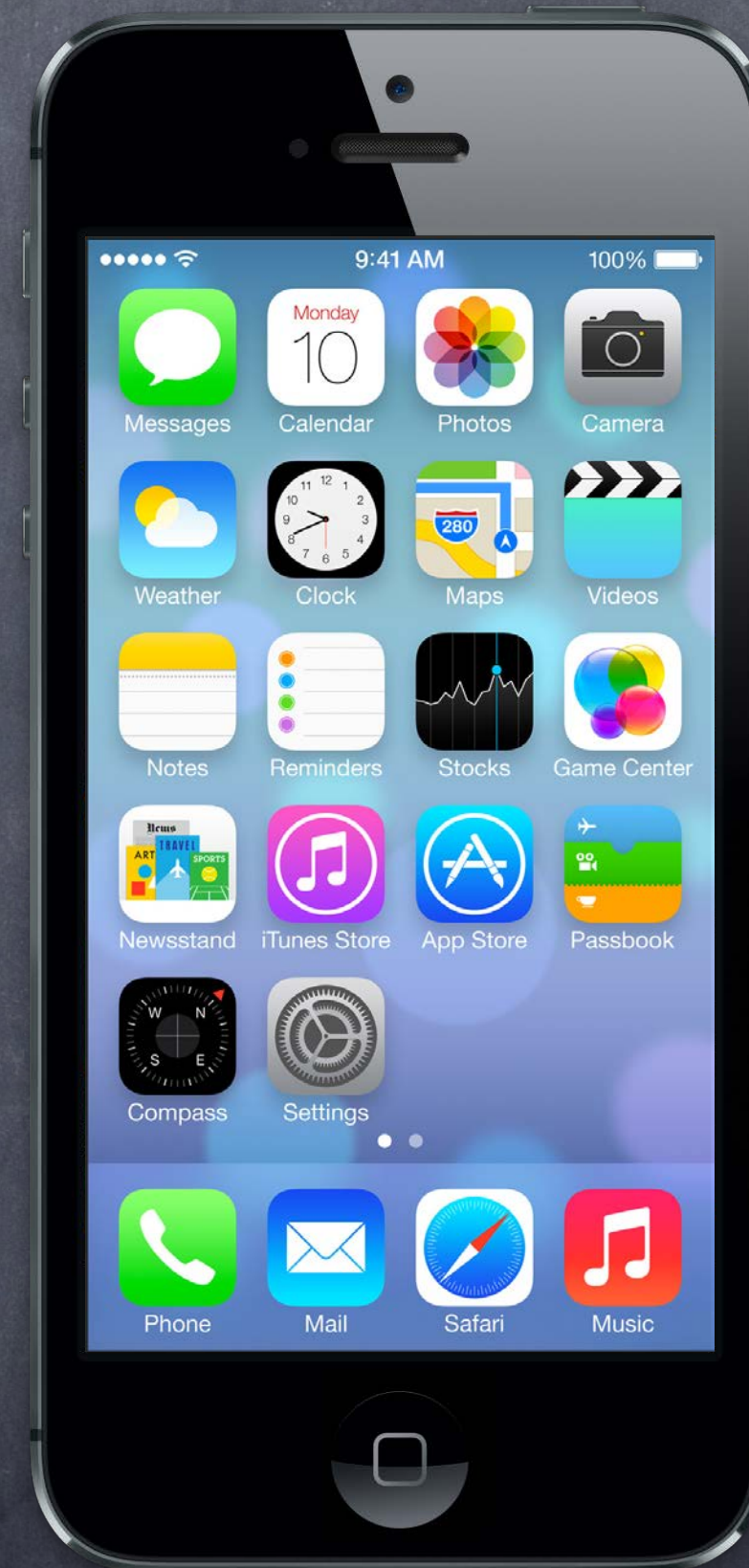


Stanford CS193p

Developing Applications for iOS
Fall 2013-14



Today

- UITableView
Data source-driven vertical list of views.
- iPad
Device-specific UI idioms.
- Demo
Shutterbug

UITableView

- Very important class for displaying data in a table

 - One-dimensional table.

 - It's a subclass of UIScrollView.

 - Table can be static or dynamic (i.e. a list of items).

 - Lots and lots of customization via a dataSource protocol and a delegate protocol.

 - Very efficient even with very large sets of data.

- Displaying multi-dimensional tables ...

 - Usually done via a UINavigationController with multiple MVC's where View is UITableView

- Kinds of UITableViews

 - Plain or Grouped

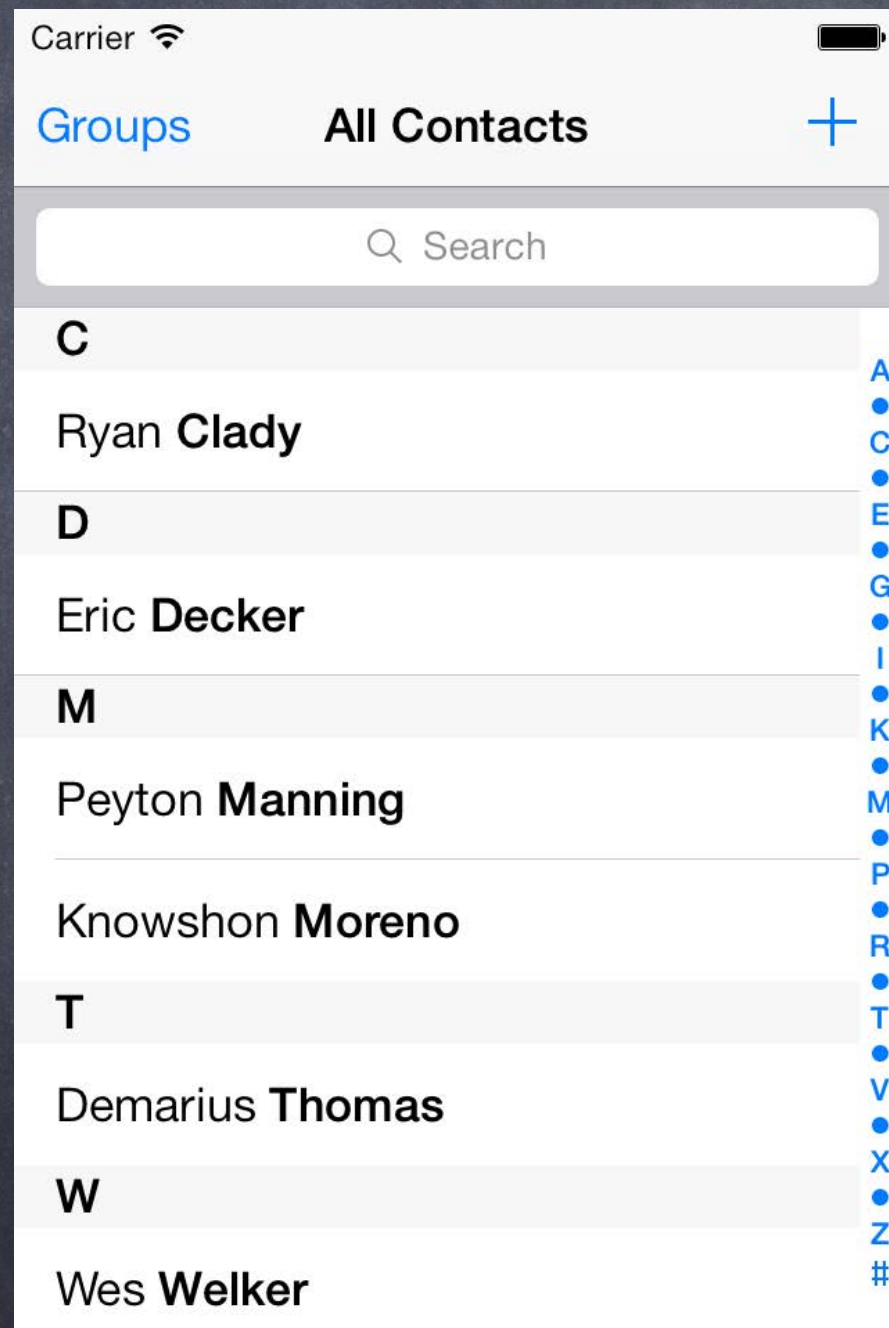
 - Static or Dynamic

 - Divided into sections or not

 - Different formats for each row in the table (including completely customized)

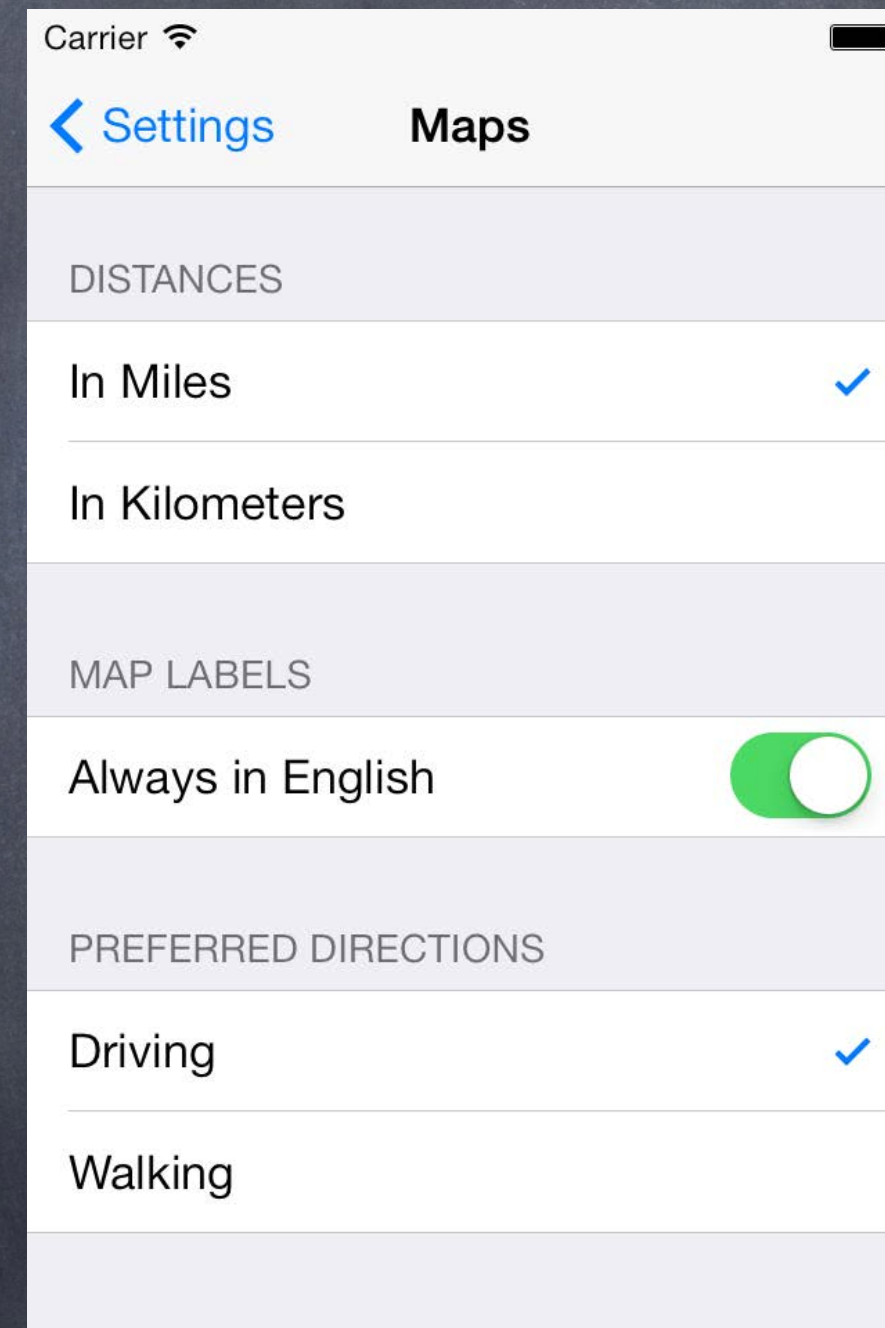
UITableView

UITableViewStylePlain



Dynamic (List)
& Plain
(ungrouped)

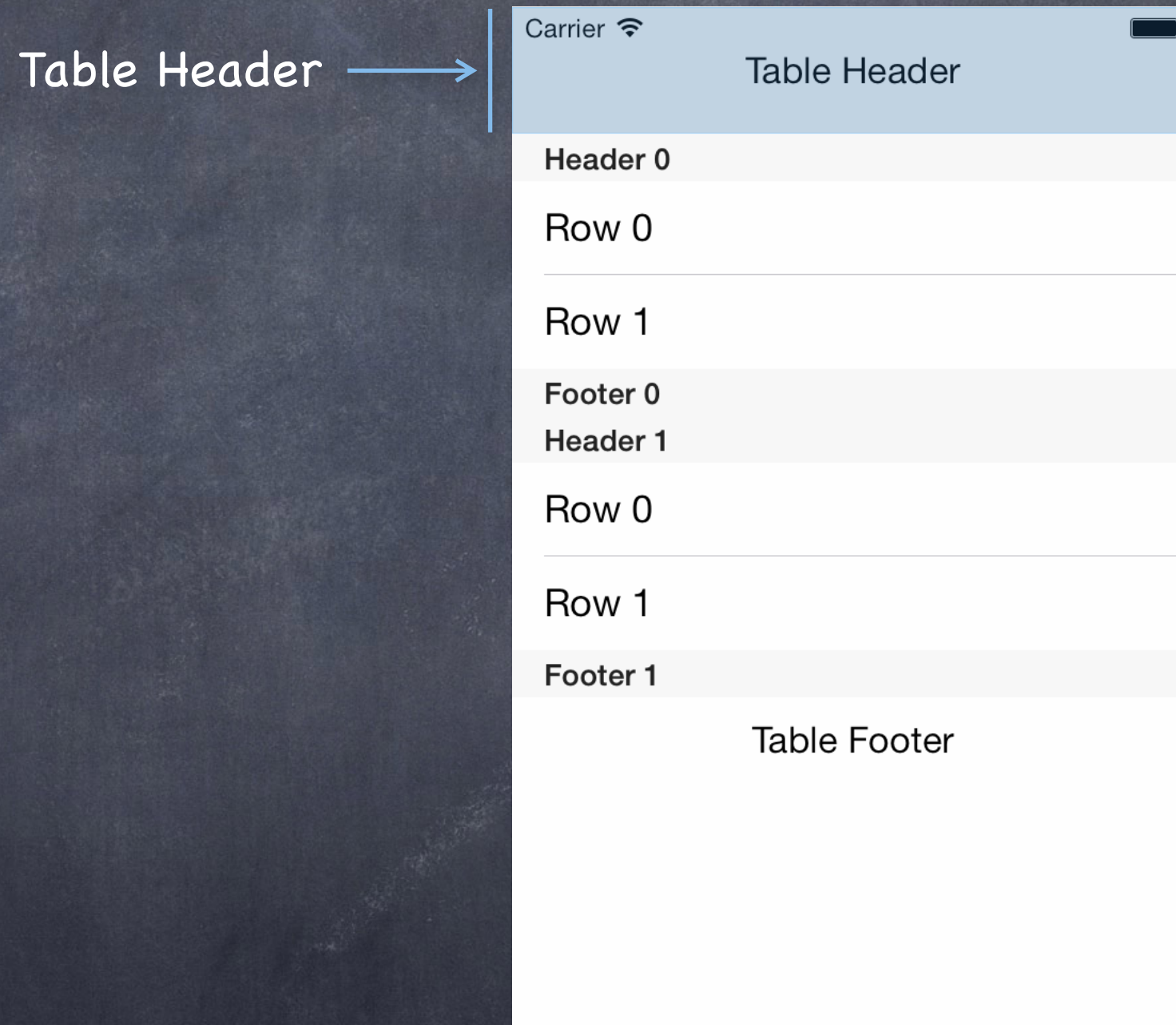
UITableViewStyleGrouped



Static
& Grouped

UITableView

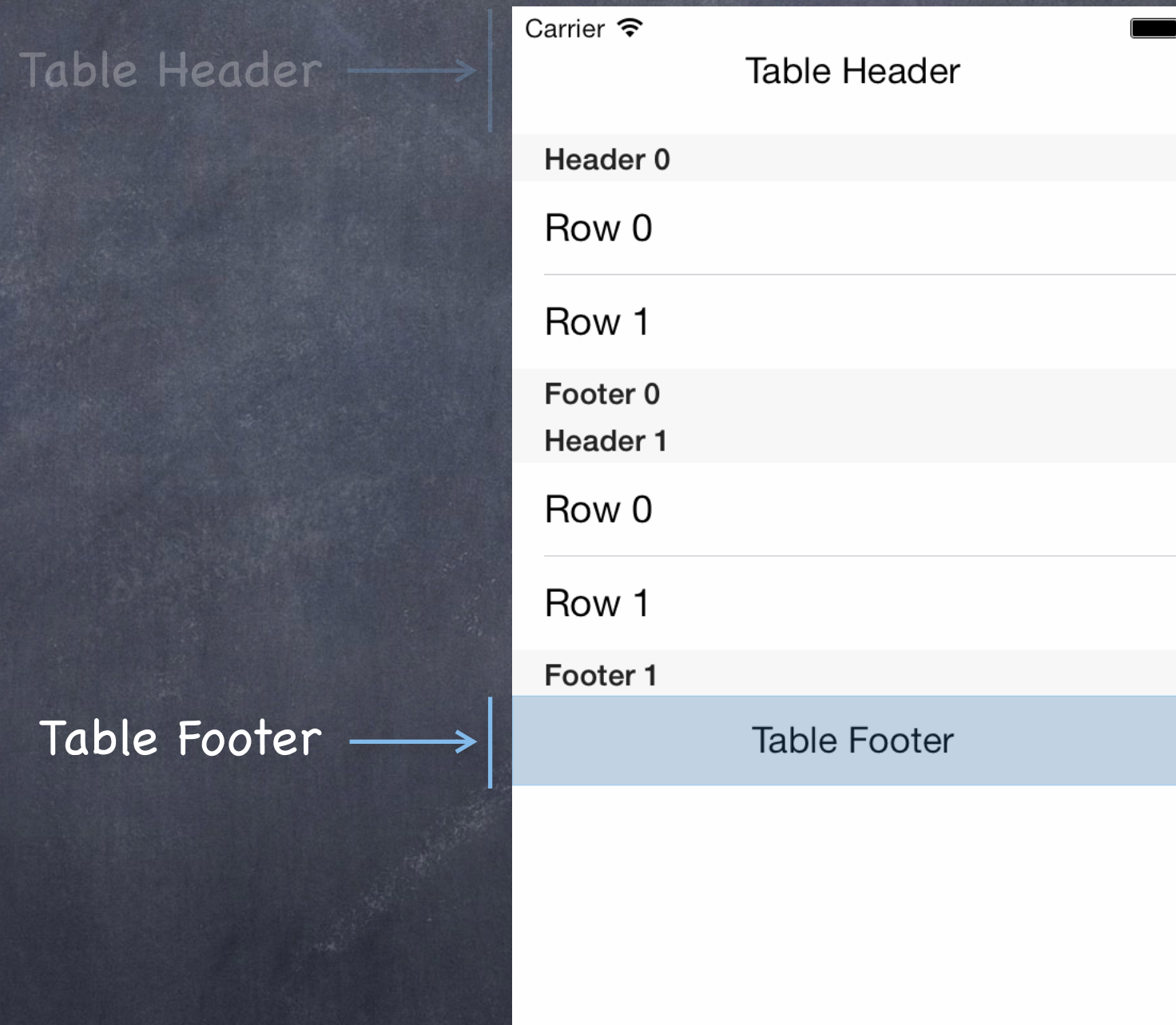
Plain Style



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

UITableView

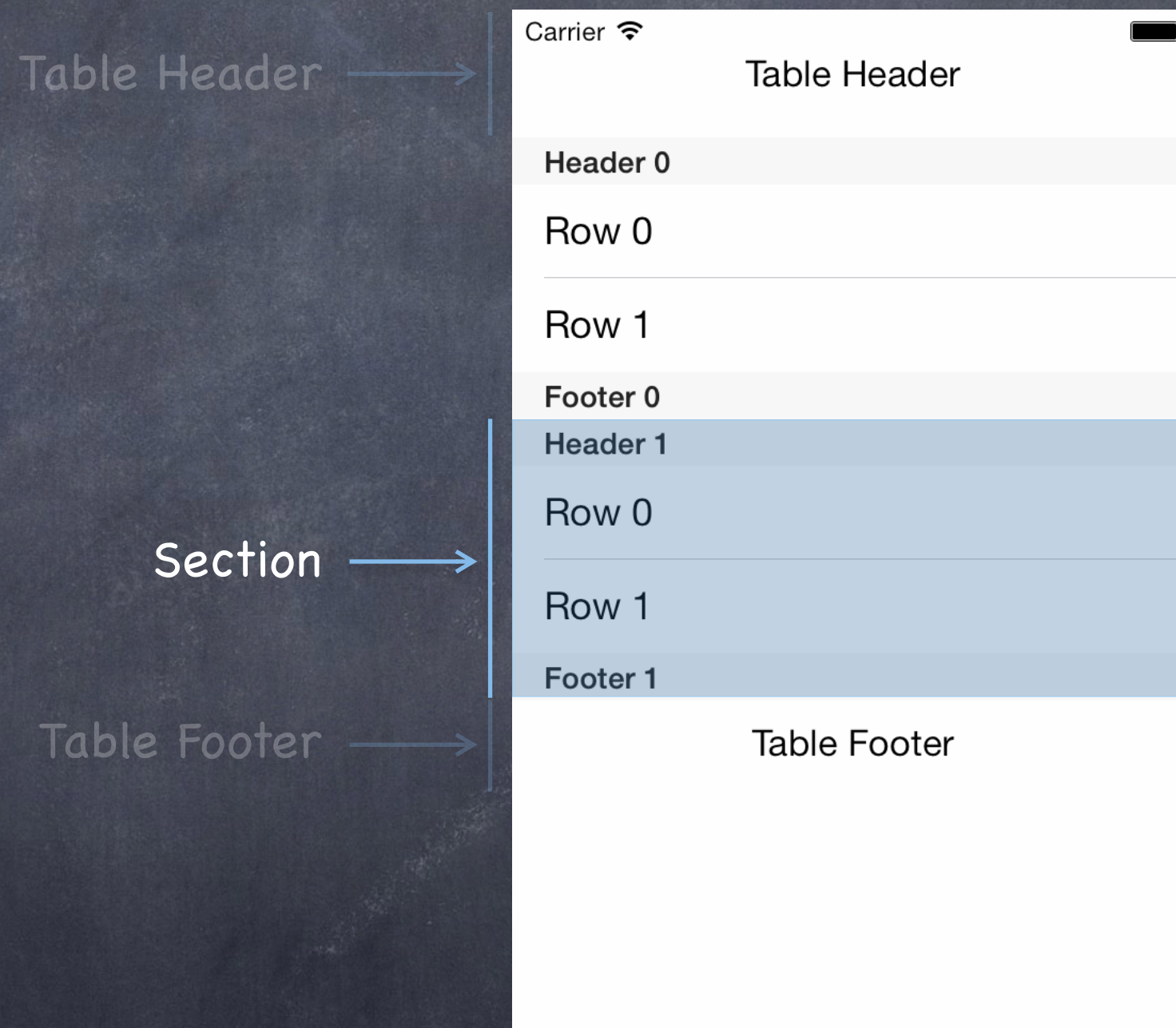
Plain Style



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

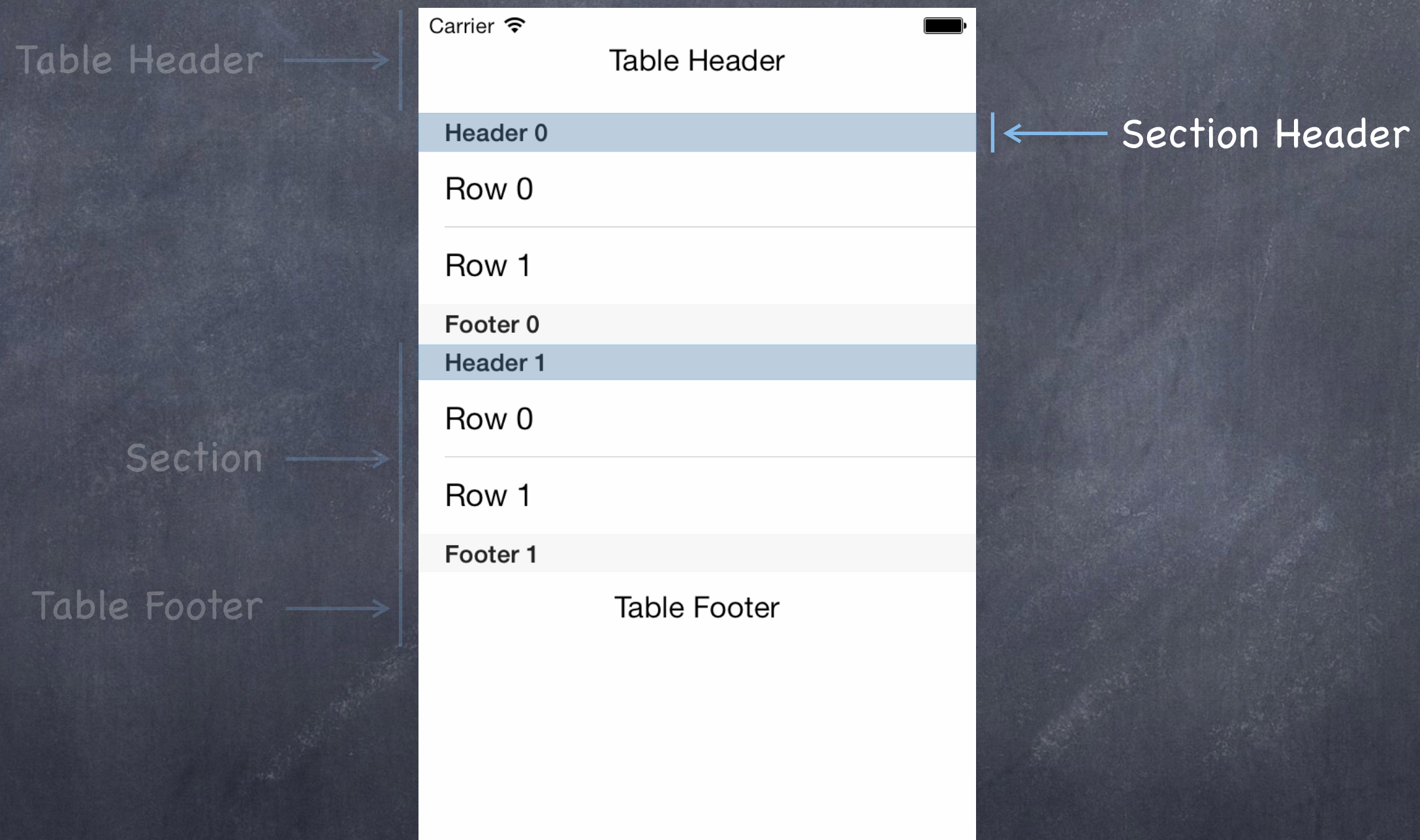
UITableView

Plain Style



UITableView

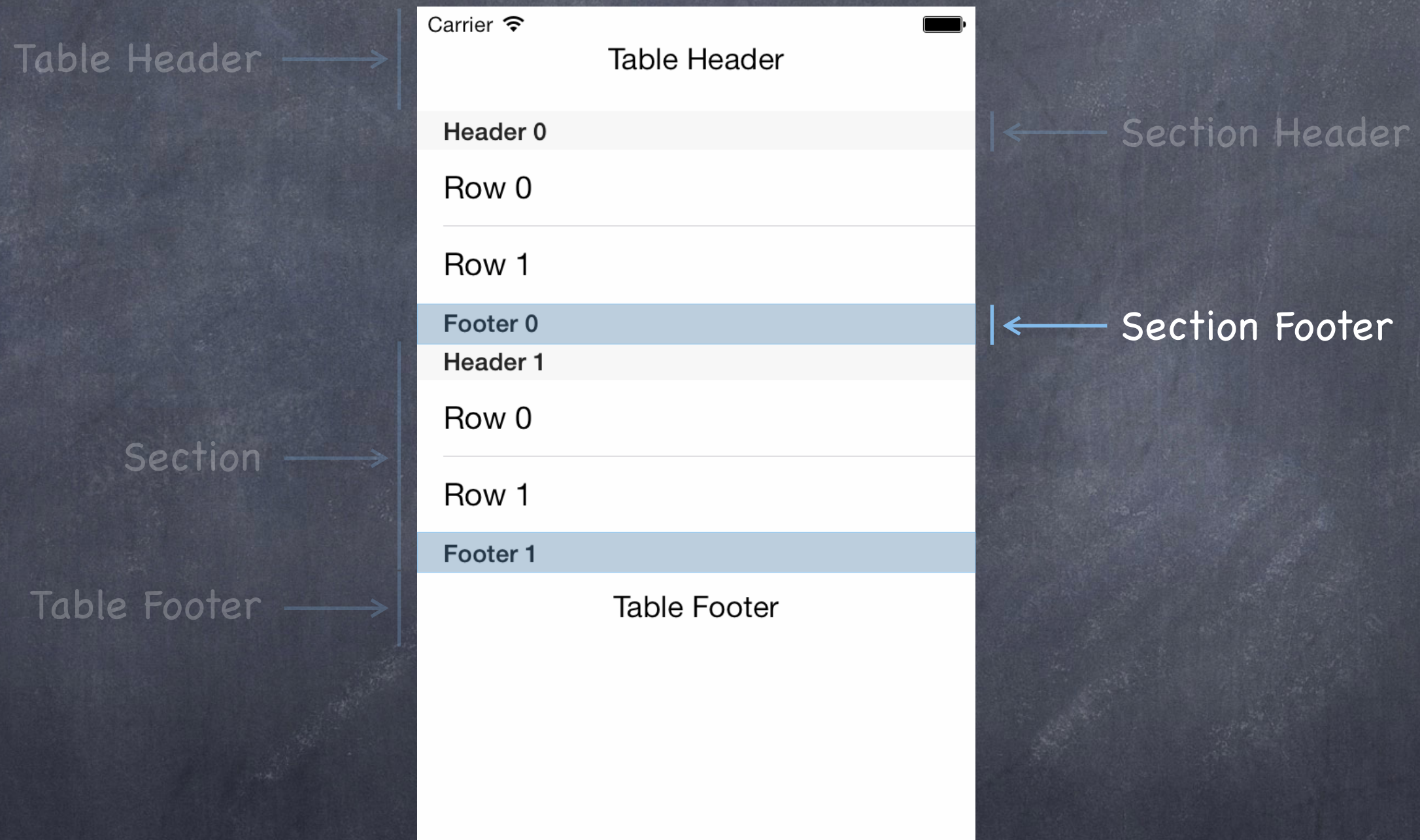
Plain Style



UITableViewDataSource's `tableView:titleForHeaderInSection:`

UITableView

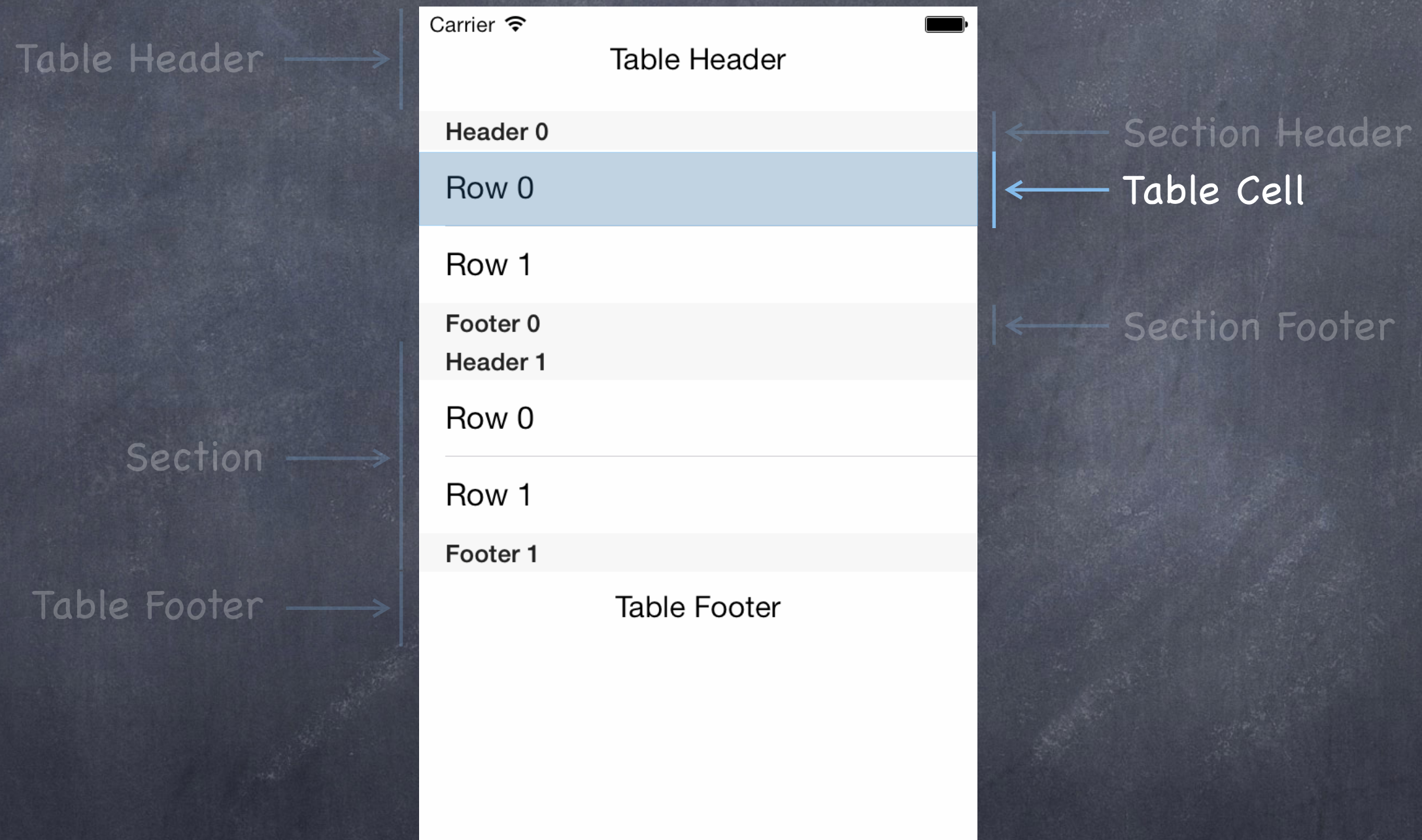
Plain Style



UITableViewDataSource's `tableView:titleForFooterInSection:`

UITableView

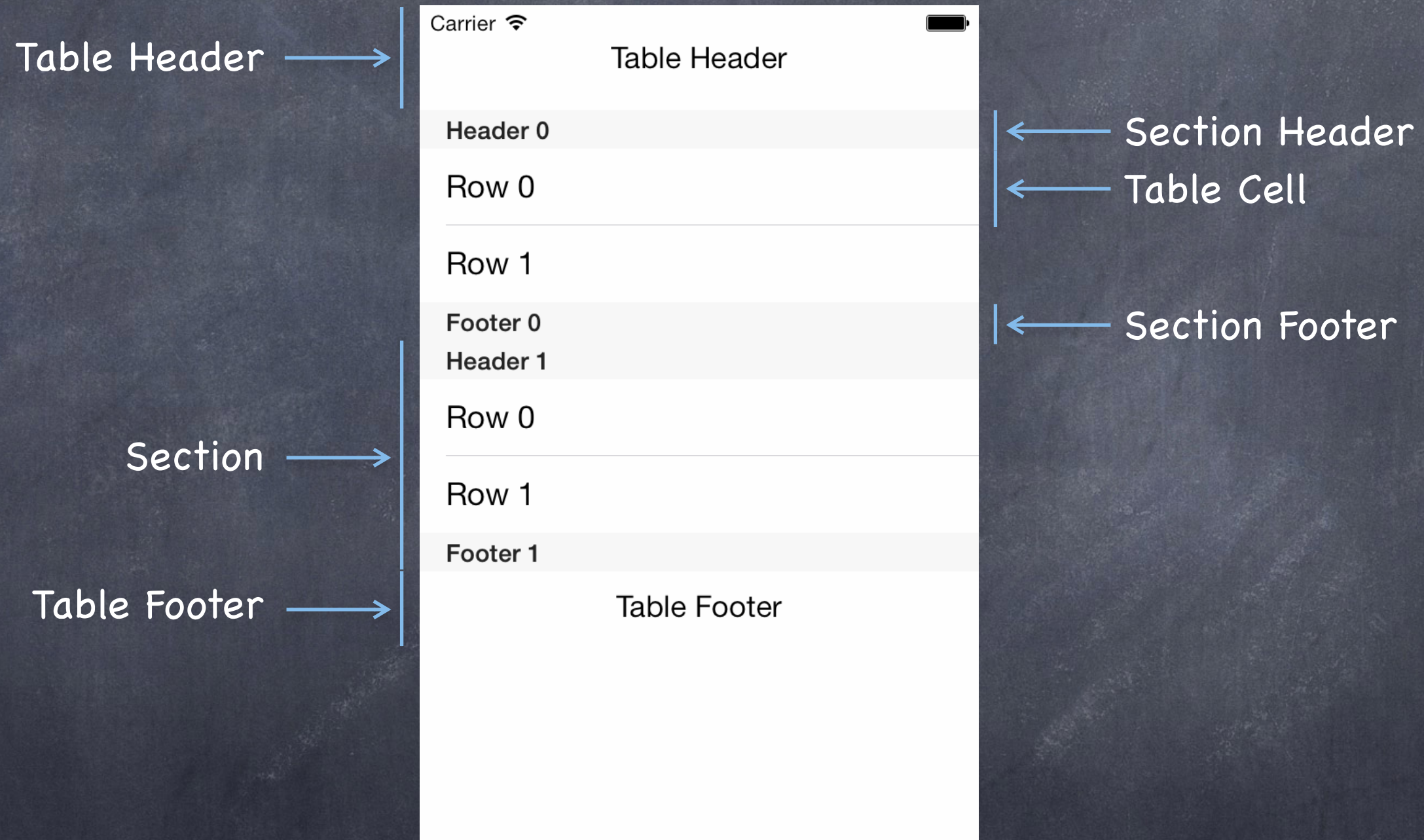
Plain Style



UITableViewDataSource's `tableView:cellForRowAtIndexPath:`

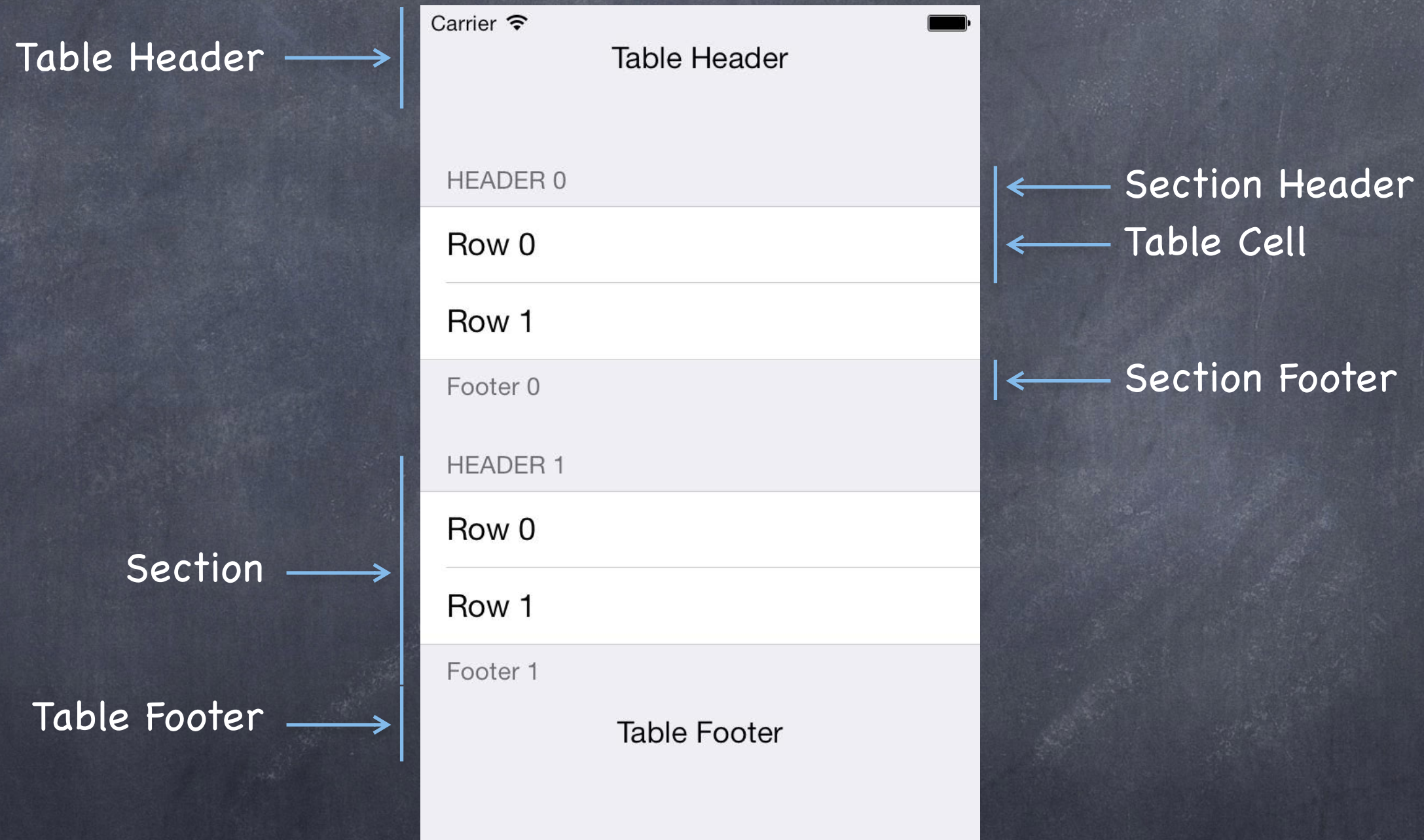
UITableView

Plain Style

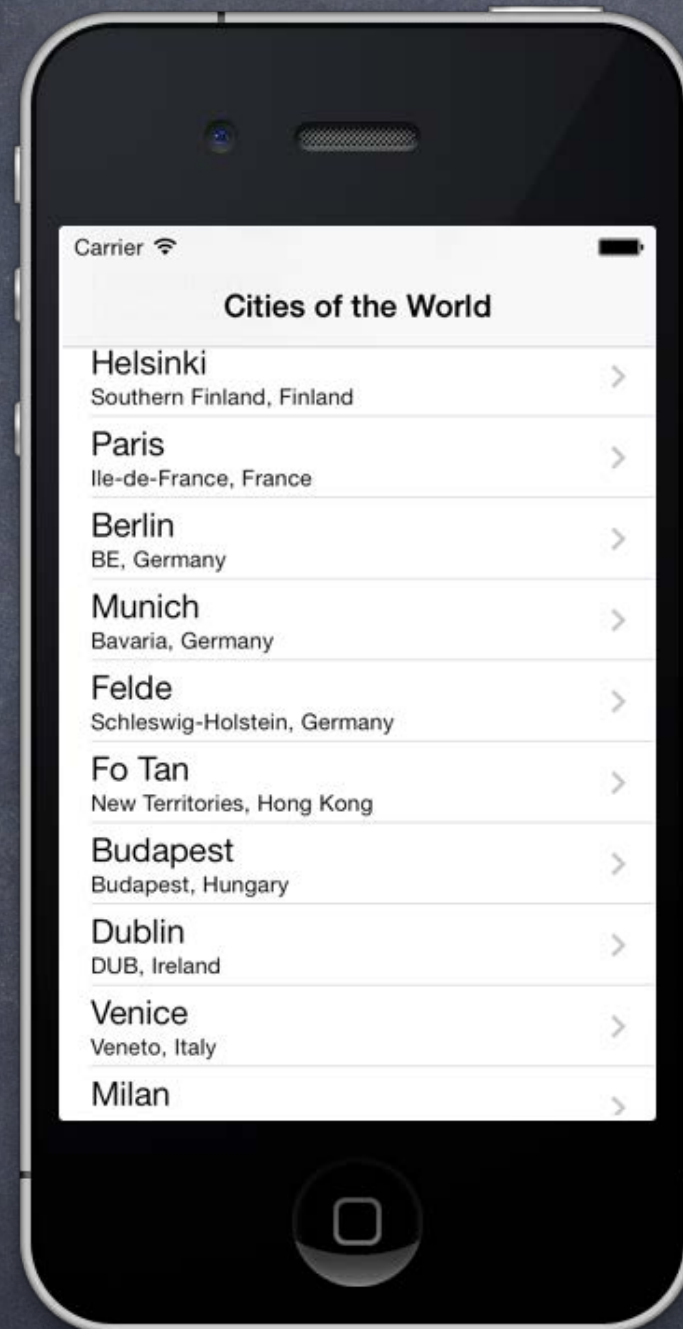


UITableView

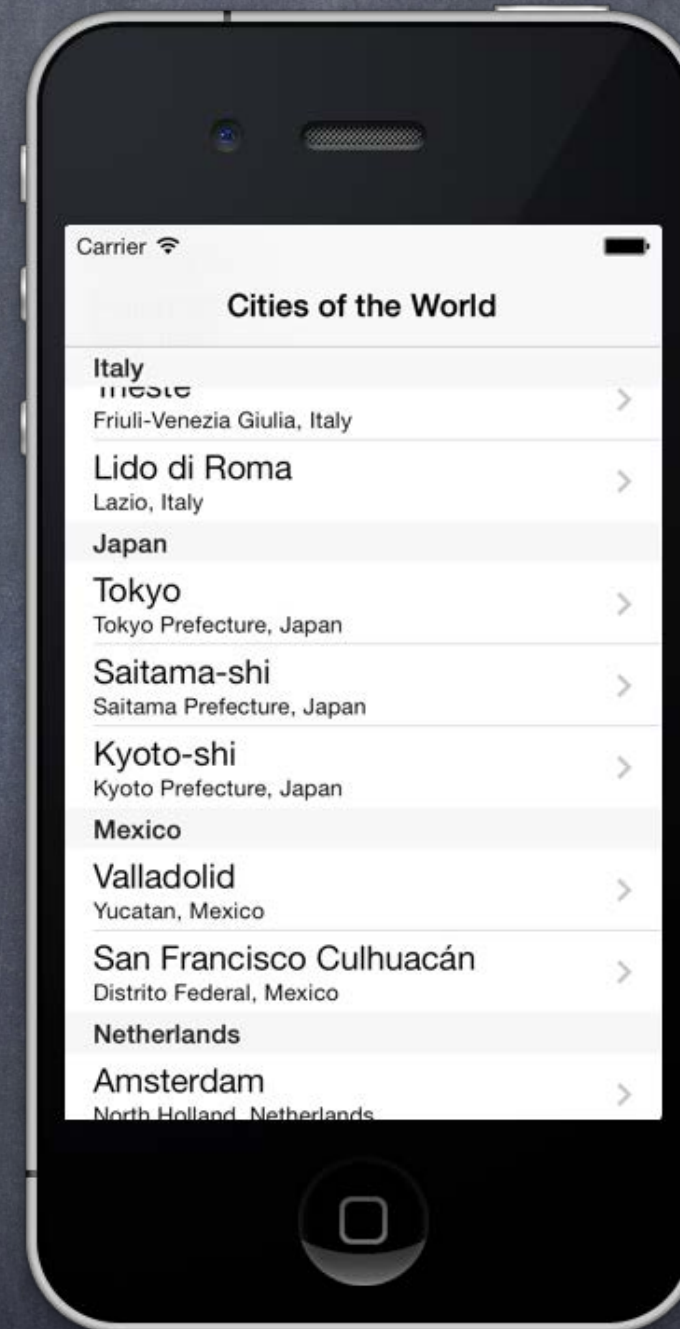
Grouped Style



Sections or Not

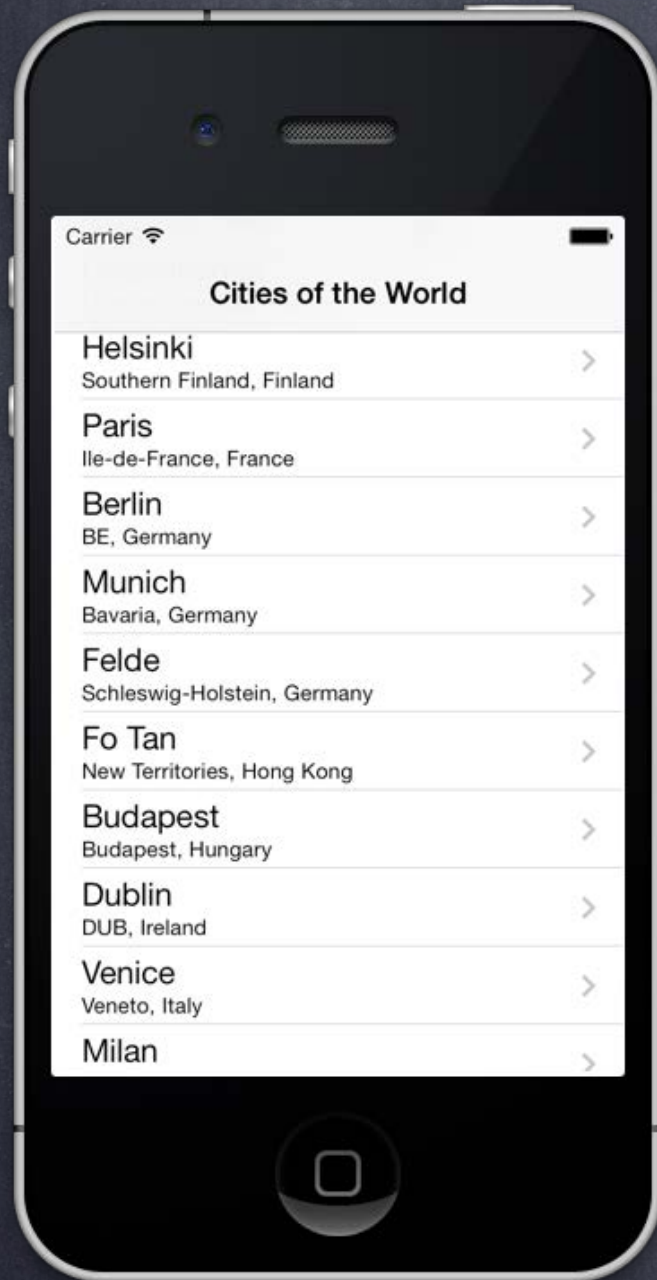


No Sections



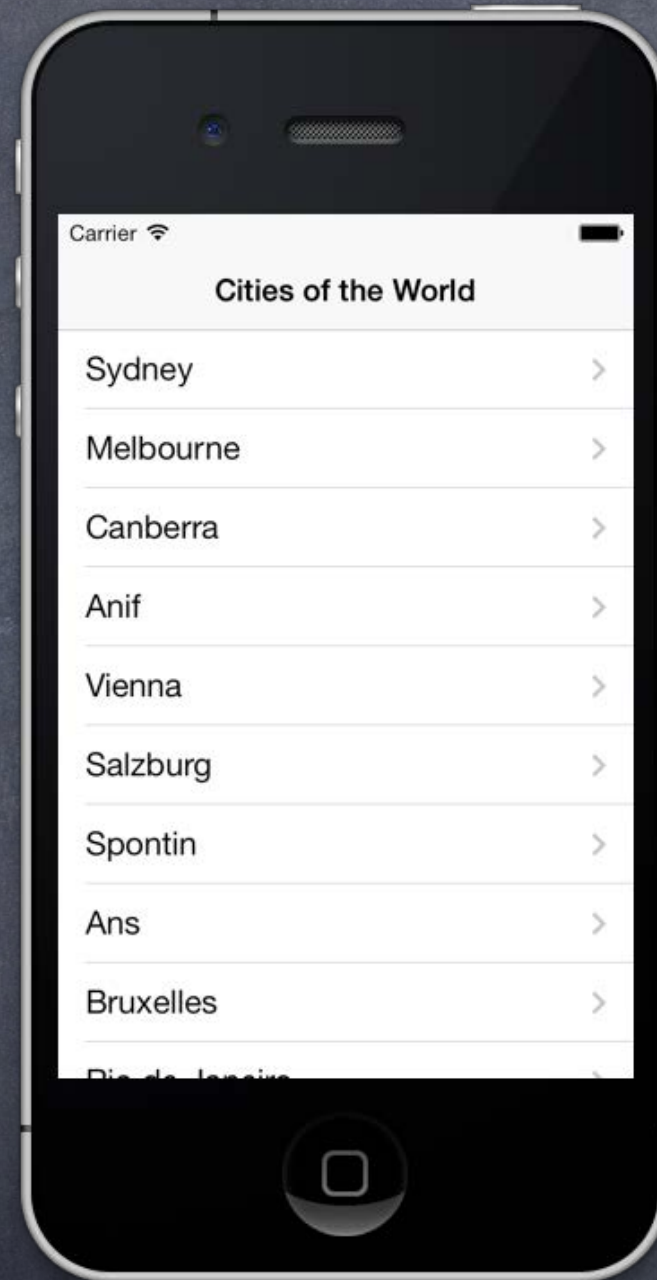
Sections

Cell Type



Subtitle

UITableViewCellStyleSubtitle



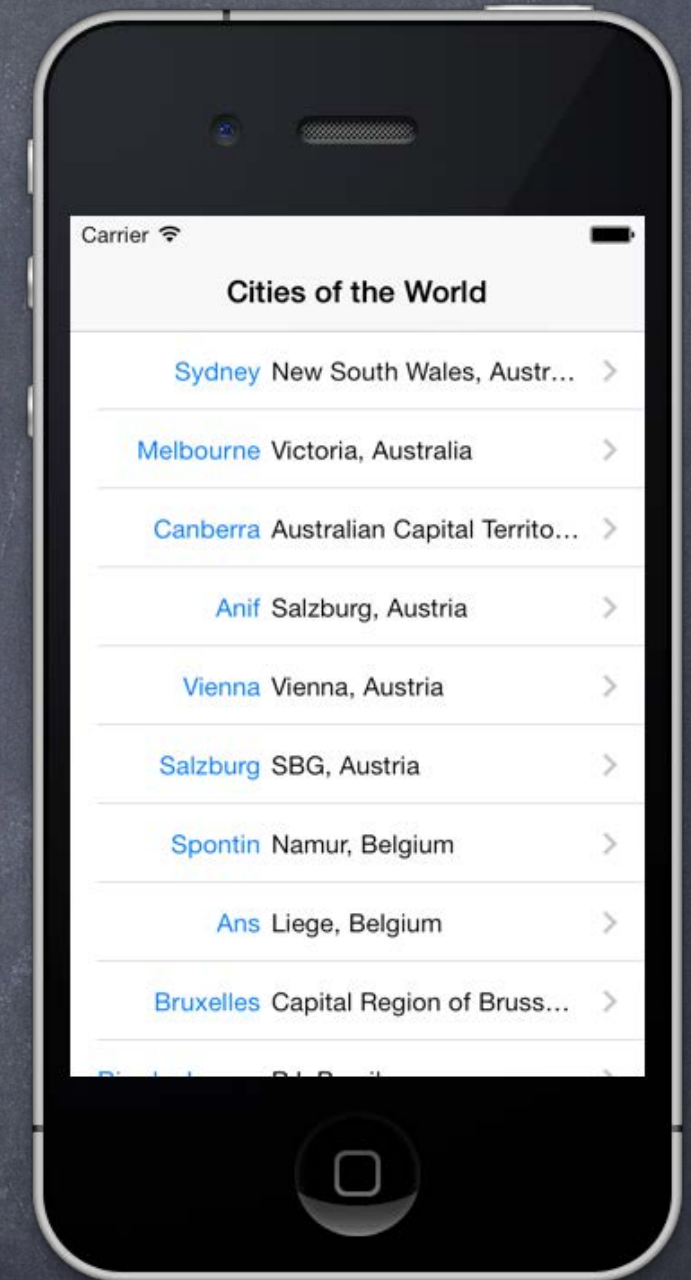
Basic

UITableViewCellStyleDefault



Right Detail

UITableViewCellStyleValue1

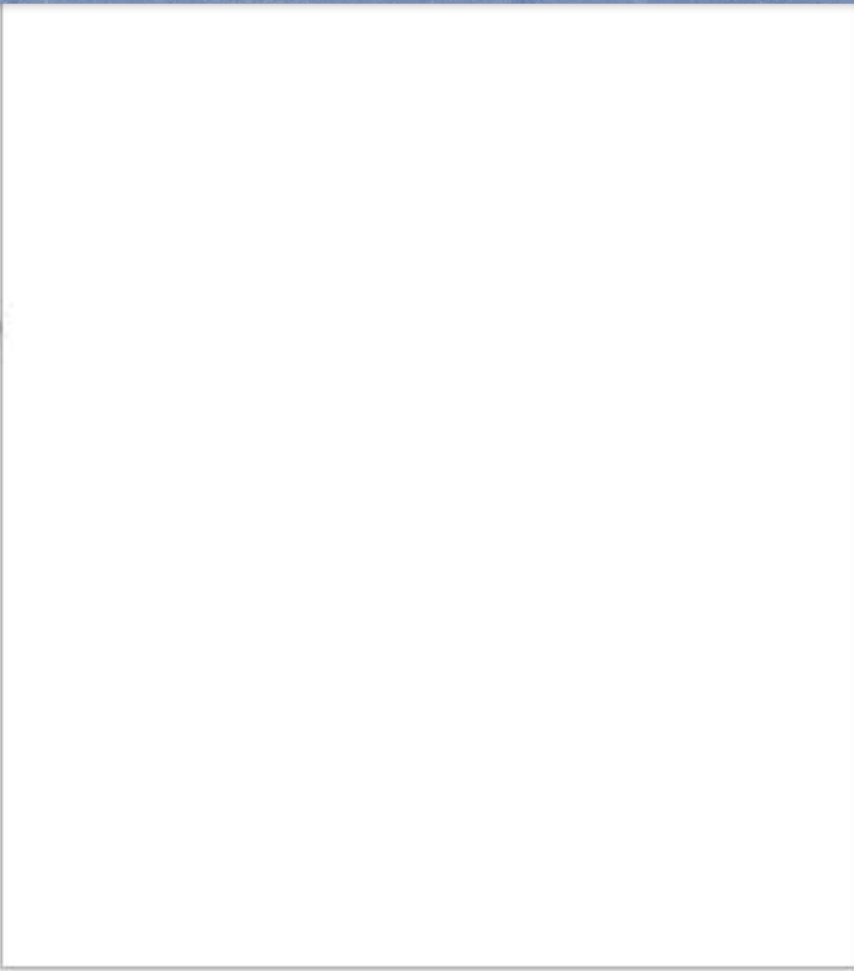


Left Detail

UITableViewCellStyleValue2

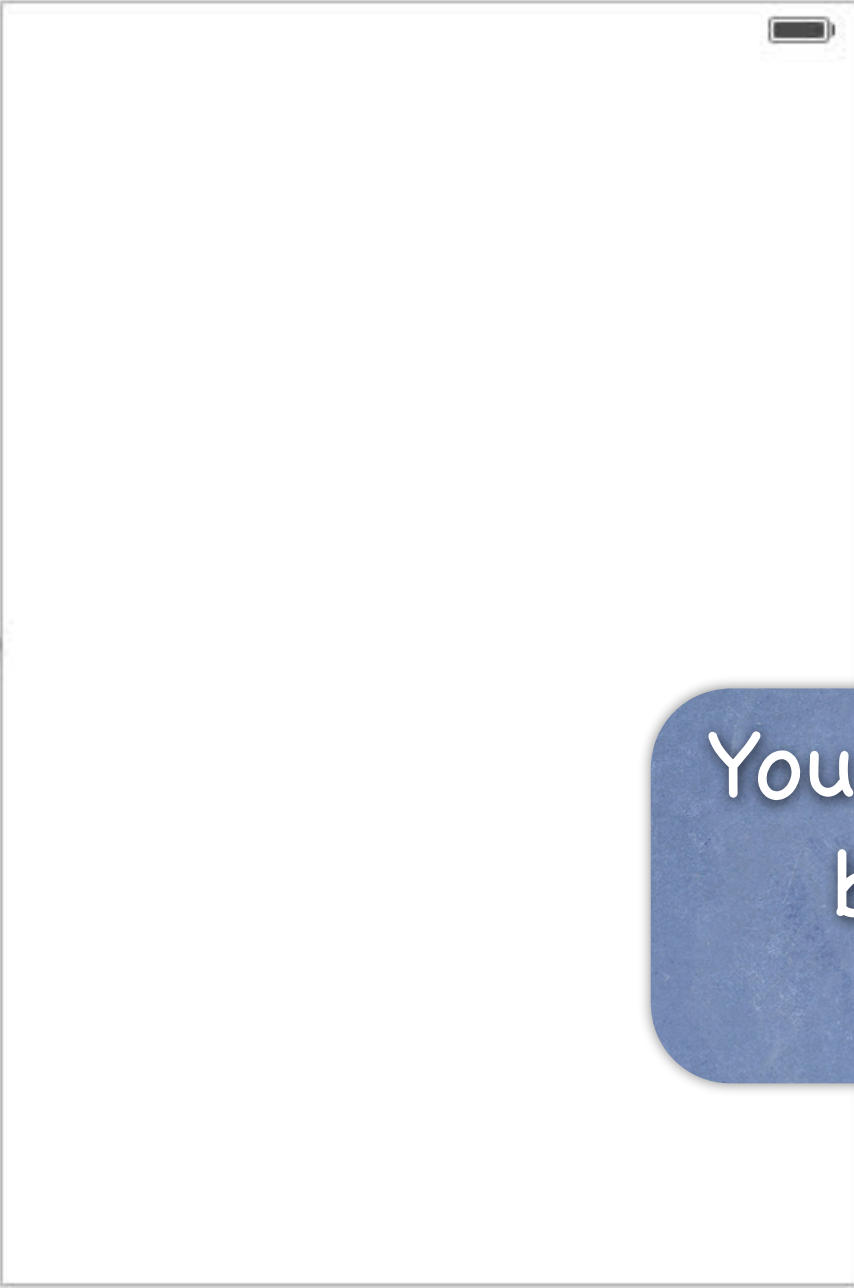
The class UITableViewController provides a convenient packaging of a UITableView in an MVC.

It's mostly useful when the UITableView is going to fill all of self.view (in fact self.view in a UITableViewController is the UITableView).



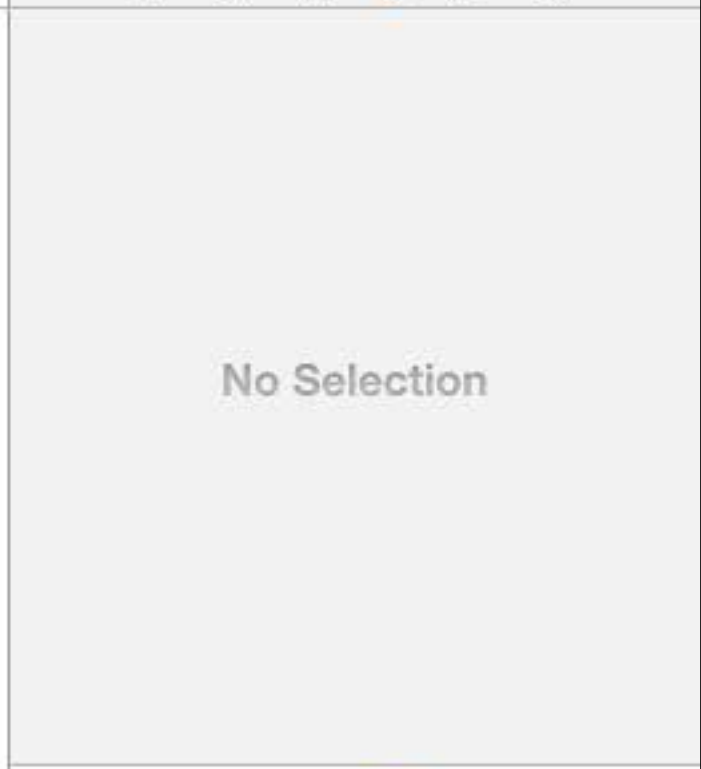
View Controller

- supports the fundamental view-management model in iPhone OS.
- Table View Controller** - A controller that manages a table view.
- CollectionView Controller** - A controller that manages a collection view.
- Navigation Controller** - A controller that manages navigation through a hierarchy of views.
- Tab Bar Controller** - A controller that manages a set of view controllers that represent tab bar items.
- Page View Controller** - Presents a sequence of view controllers as pages.

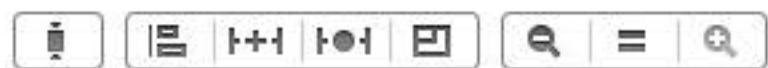


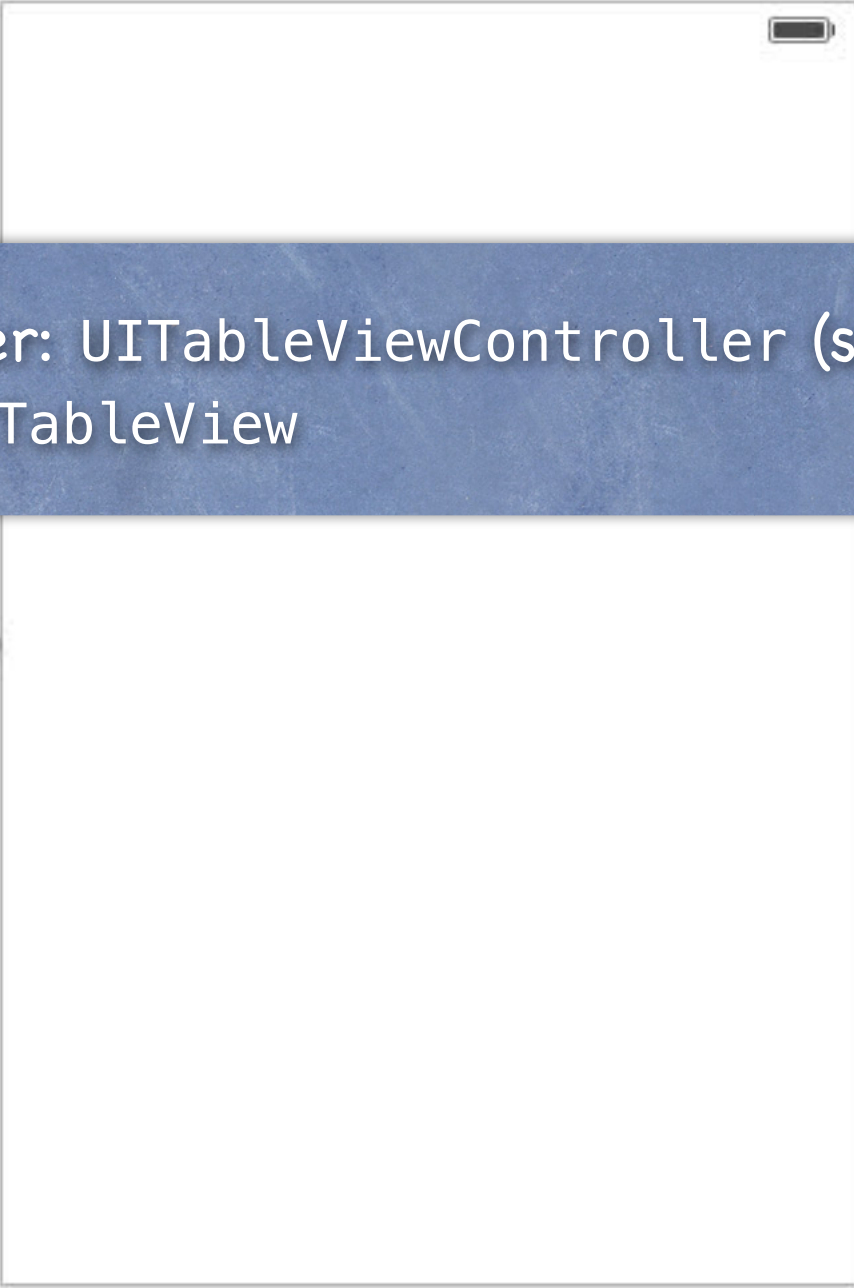
You can add an MVC like this by dragging it into your storyboard from here.

View Controller



- supports the fundamental view-management model in iPhone OS.
- Table View Controller** - A controller that manages a table view.
- Collection View Controller** - A controller that manages a collection view.
- Navigation Controller** - A controller that manages navigation through a hierarchy of views.
- Tab Bar Controller** - A controller that manages a set of view controllers that represent tab bar items.
- Page View Controller** - Presents a sequence of view controllers as pages.





View Controller

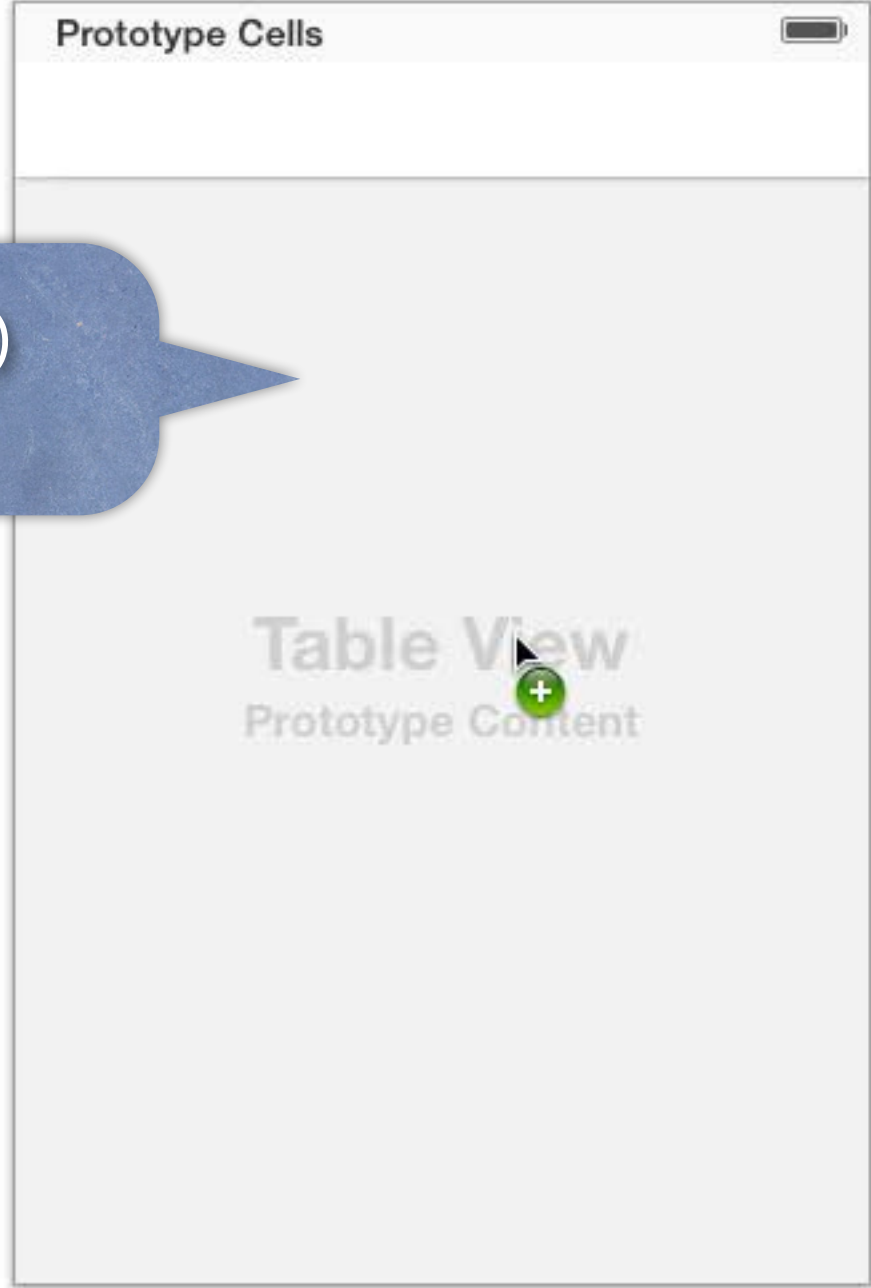
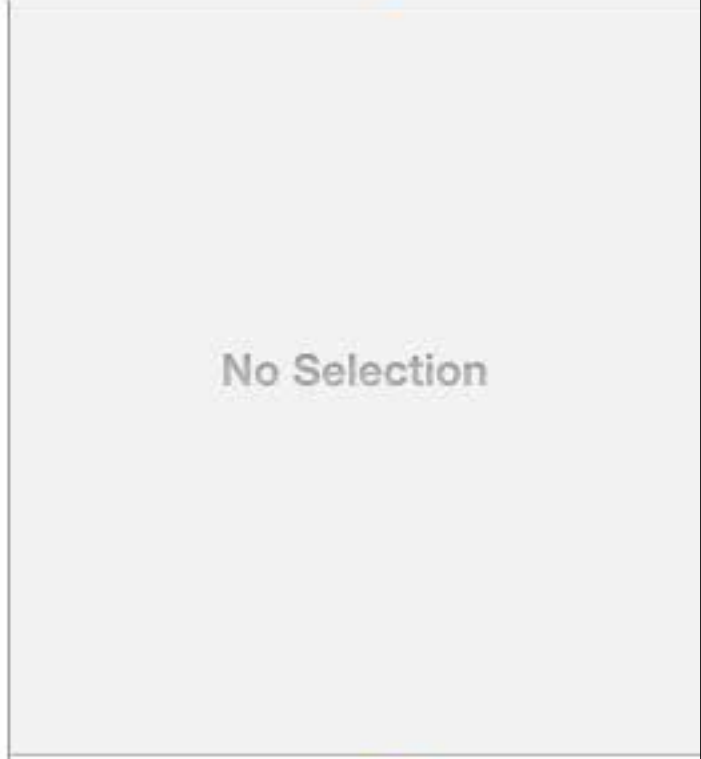
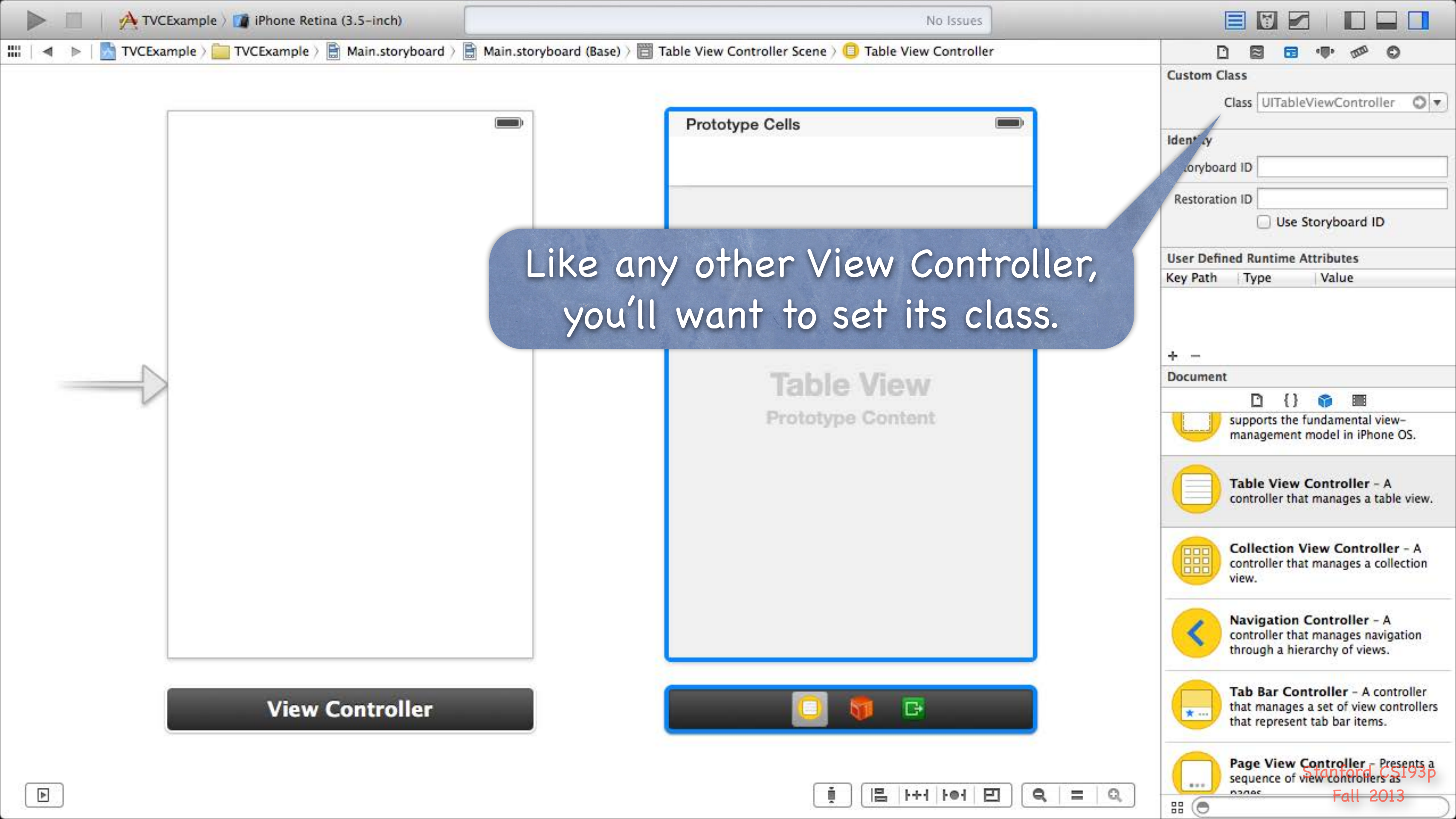


Table View Controller

Controller: UITableViewController (subclass of)
View: UITableView



- supports the fundamental view-management model in iPhone OS.
- Table View Controller** - A controller that manages a table view.
- Collection View Controller** - A controller that manages a collection view.
- Navigation Controller** - A controller that manages navigation through a hierarchy of views.
- Tab Bar Controller** - A controller that manages a set of view controllers that represent tab bar items.
- Page View Controller** - Presents a sequence of view controllers as pages.



Like any other View Controller, you'll want to set its class.

Custom Class
Class: UITableViewController

Identity
Storyboard ID:
Restoration ID:
 Use Storyboard ID

User Defined Runtime Attributes
Key Path | Type | Value

- + -
- Document
 - supports the fundamental view-management model in iPhone OS.
 - Table View Controller** - A controller that manages a table view.
 - Collection View Controller** - A controller that manages a collection view.
 - Navigation Controller** - A controller that manages navigation through a hierarchy of views.
 - Tab Bar Controller** - A controller that manages a set of view controllers that represent tab bar items.
 - Page View Controller** - Presents a sequence of view controllers as pages.

Choose options for your new file:

Class

Subclass of

Targeted for iPad

With XIB for user interface

Make sure you set the superclass to UITableViewController ...

Custom Class
Class

Identity
Storyboard ID
Restoration ID
 Use Storyboard ID

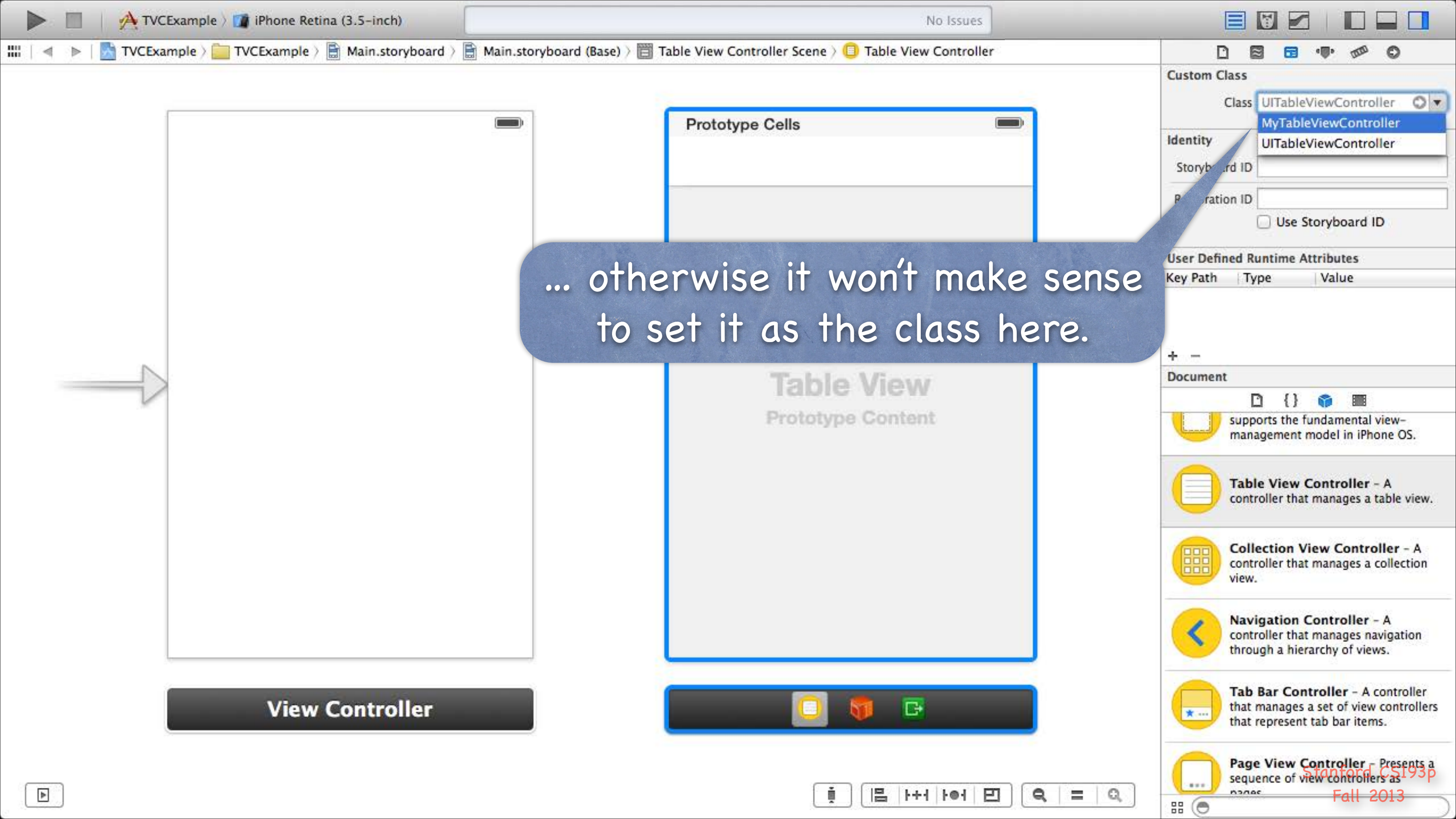
User Defined Runtime Attributes

Key Path	Type	Value
+ -		

Document

View Controller

- Collection View Controller** - A controller that manages a collection view.
- Navigation Controller** - A controller that manages navigation through a hierarchy of views.
- Tab Bar Controller** - A controller that manages a set of view controllers that represent tab bar items.
- Page View Controller** - Presents a sequence of view controllers as pages.



... otherwise it won't make sense to set it as the class here.

View Controller

Custom Class

Class: **MyTableViewController**

Identity: UITableViewController

Storyboard ID: []

Registration ID: []

Use Storyboard ID

User Defined Runtime Attributes

Key Path	Type	Value
----------	------	-------

Document

- supports the fundamental view-management model in iPhone OS.
- Table View Controller** - A controller that manages a table view.
- CollectionView Controller** - A controller that manages a collection view.
- Navigation Controller** - A controller that manages navigation through a hierarchy of views.
- Tab Bar Controller** - A controller that manages a set of view controllers that represent tab bar items.
- Page View Controller** - Presents a sequence of view controllers as pages.

Your UITableViewController subclass will also serve as the UITableView's dataSource and delegate (more on this in a moment).

You can see that if you right-click the Controller here.

dataSource and delegate @property

If you use UITableView without UITableViewController, you'll have to wire these up yourself.

View Controller

Table View
Prototype Content

My Table View Controller

Outlets
searchDisplayController
view * Table View
Presenting Segues
relationship
push
modal
custom
embed
Referencing Outlets
dataSource * Table View
delegate * Table View
New Referencing Outlet

No Selection

supports the fundamental view-management model in iPhone OS.

Table View Controller - A controller that manages a table view.

Navigation Controller - A controller that manages navigation through a hierarchy of views.

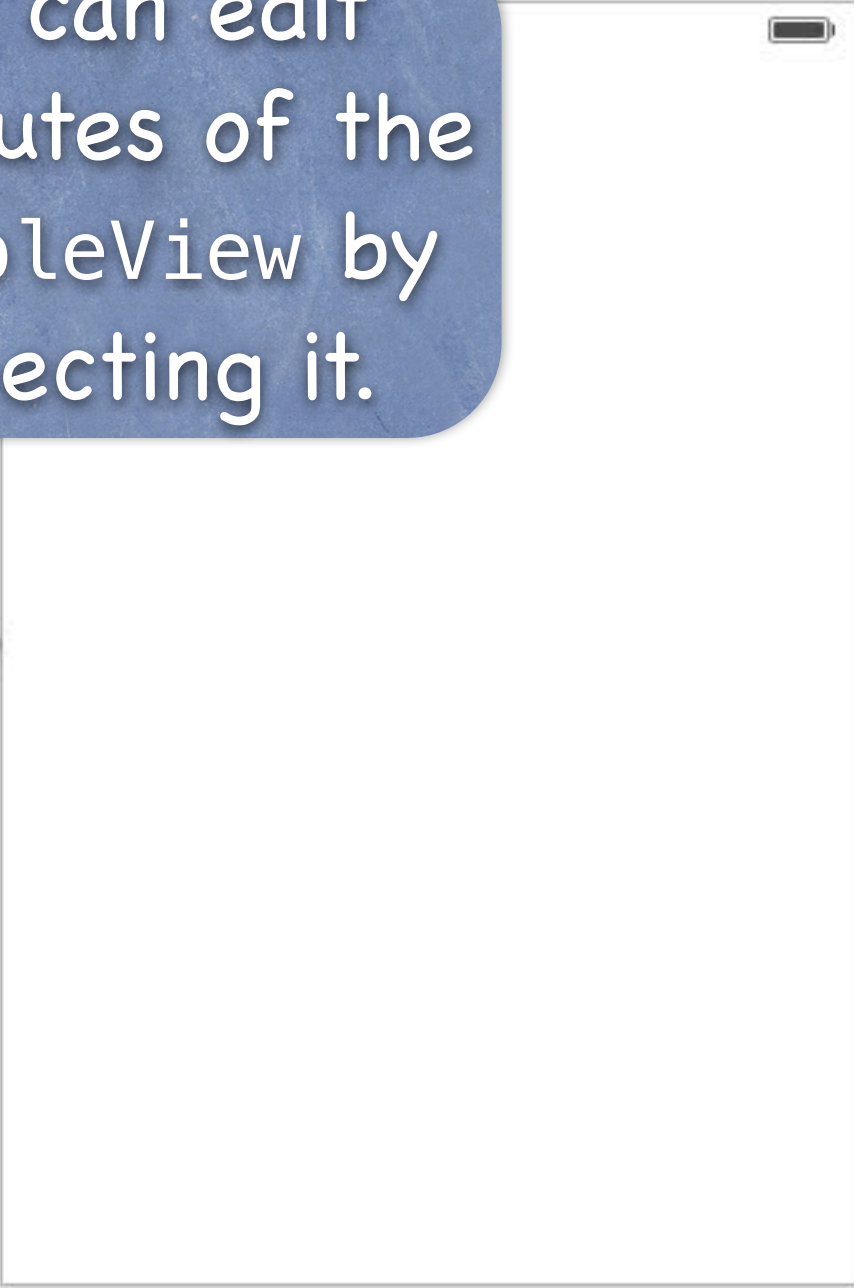
Tab Bar Controller - A controller that manages a set of view controllers that represent tab bar items.

View Controller - Presents a view controller as the source of view controllers as

Fall 2013

Stanford CS193p

You can edit attributes of the UITableView by inspecting it.



View Controller



One important attribute is the Plain vs. Grouped style ...

Table View

Content: Dynamic Prototypes

Prototype Cells: 1

Style: **Grouped** (selected), Plain

Separator: Default

Separator Insets: Default

Selection: Single Selection

Editing: No Selection During Editing

Show Selection on Touch

Index Row Limit: 0

Text Color: Default

Background: Default

Scroll View

View

Mode: Scale To Fill

Tag: 0

Interaction: User Interaction Enabled, Multiple Touch

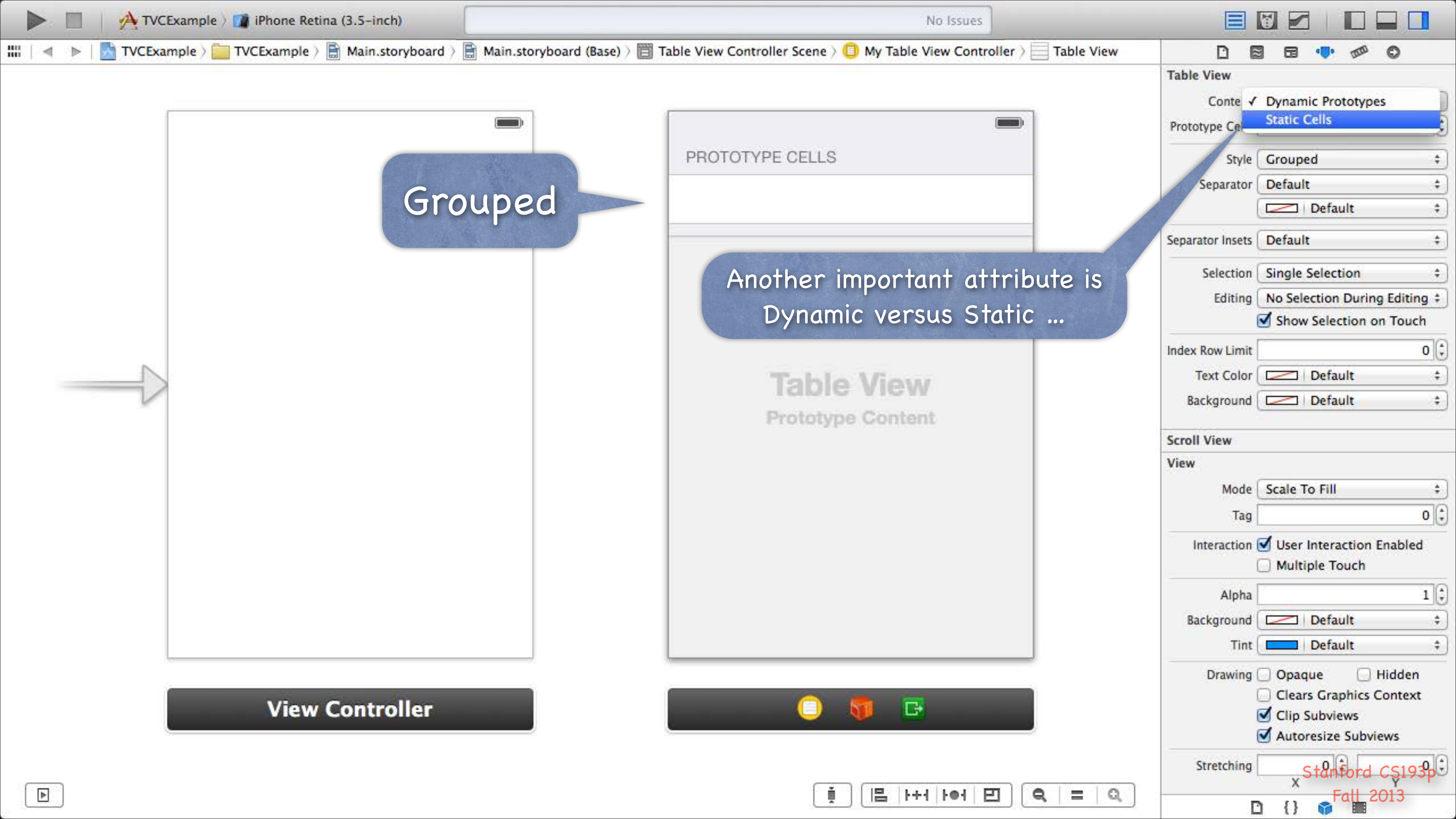
Alpha: 1

Background: Default

Tint: Default

Drawing: Opaque, Hidden, Clears Graphics Context, Clip Subviews, Autoresize Subviews

Stretching: X: 0, Y: 0



Grouped

Another important attribute is Dynamic versus Static ...

Dynamic Prototypes
Static Cells

Style: Grouped

Separator: Default

Separator Insets: Default

Selection: Single Selection

Editing: No Selection During Editing

Show Selection on Touch

Index Row Limit: 0

Text Color: Default

Background: Default

Scroll View

View

Mode: Scale To Fill

Tag: 0

Interaction: User Interaction Enabled

Multiple Touch

Alpha: 1

Background: Default

Tint: Default

Drawing: Opaque, Hidden

Clears Graphics Context

Clip Subviews

Autosize Subviews

Stretching: X, Y

Stanford CS193p
Fall 2013

View Controller

Navigation bar icons



View Controller

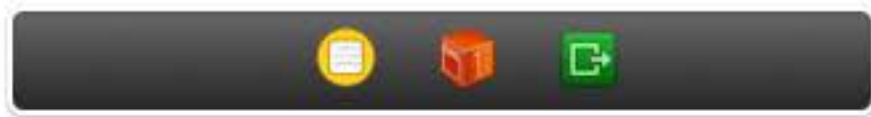
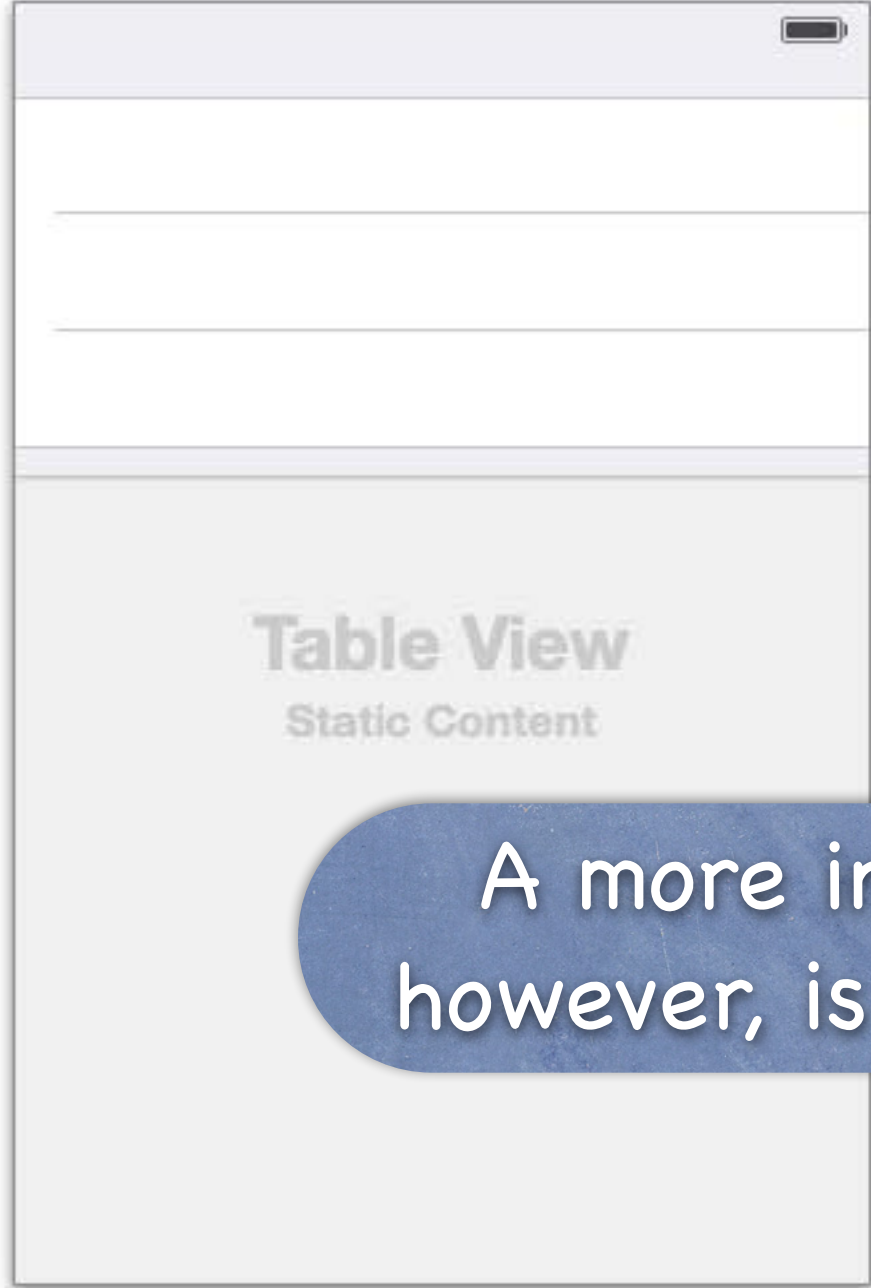


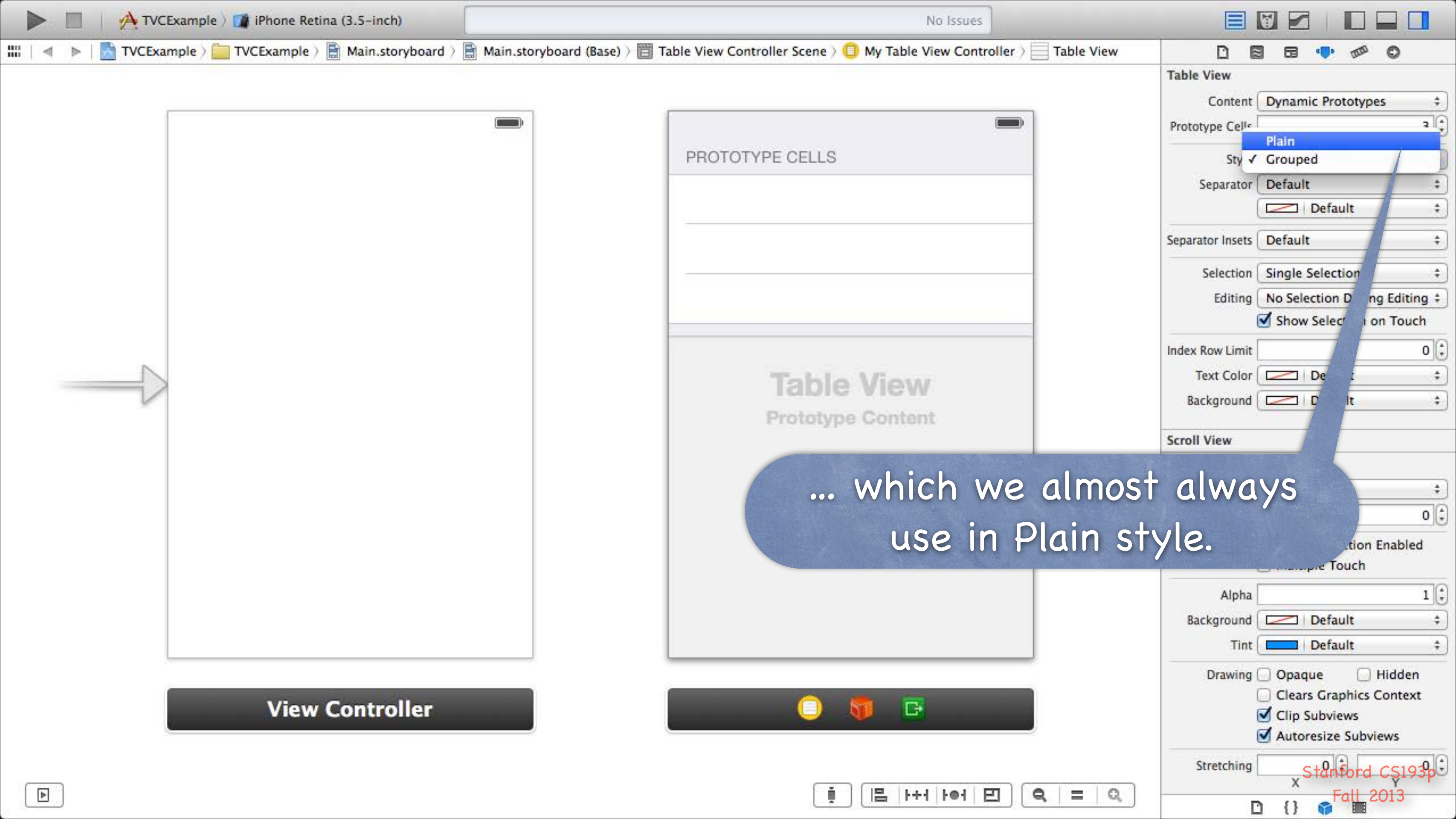
Table View

- Dynamic Prototypes
 - Static Cells
- Sections: 1
- Style: Grouped
- Separator: Default
- Separator Insets: Default
- Selection: Single Selection
- Editing: No Selection During Editing
- Show Selection on Touch
- Index Row Limit: 0
- Text Color: Default
- Background: Default

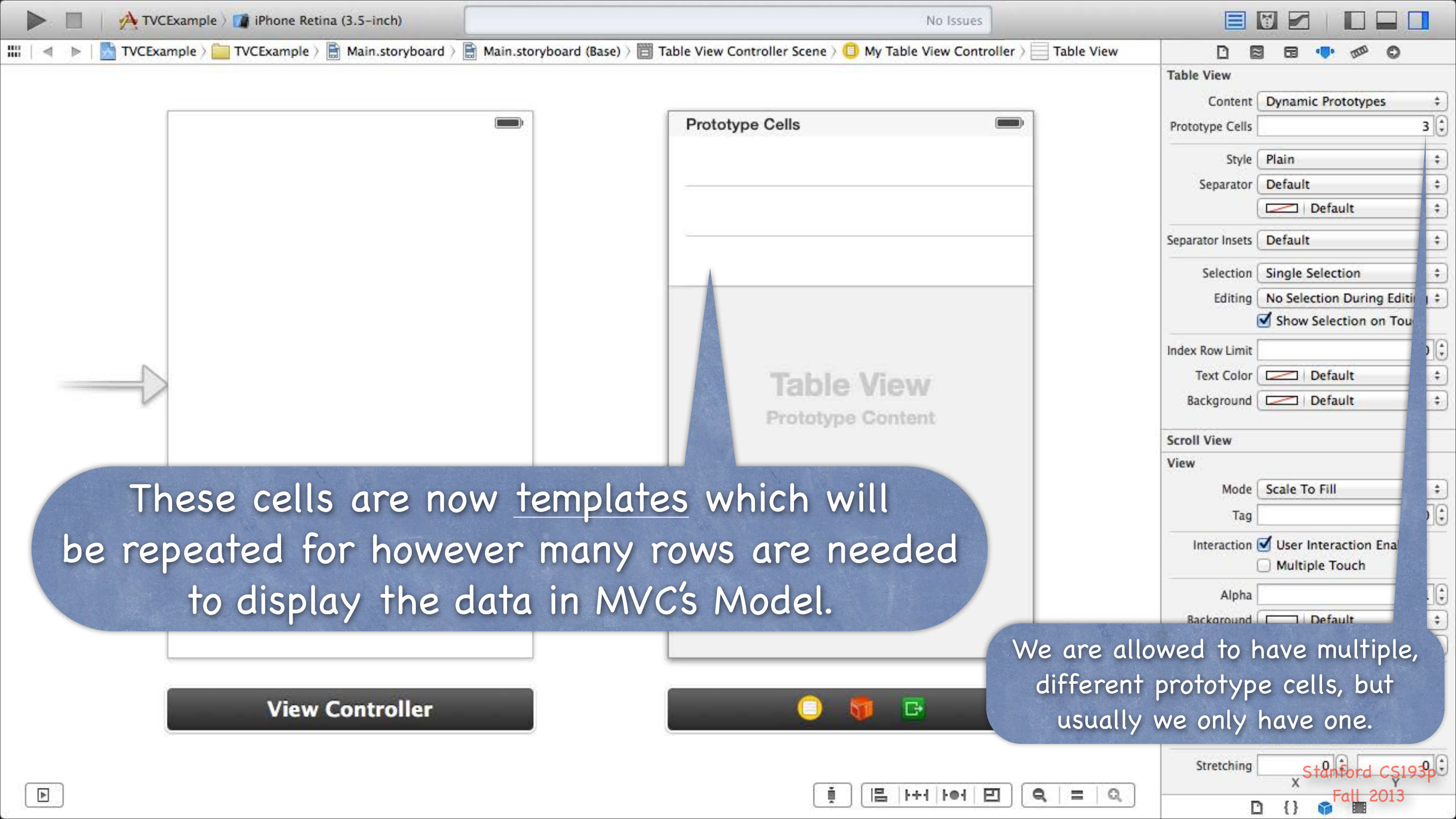
A more interesting table, however, is a Dynamic one ...

Scroll View

- Alpha: 1
- Background: Default
- Tint: Default
- Drawing
 - Opaque
 - Hidden
 - Clears Graphics Context
 - Clip Subviews
 - Autorelease Subviews
- Stretching: X: 0, Y: 0



... which we almost always use in Plain style.



These cells are now templates which will be repeated for however many rows are needed to display the data in MVC's Model.

We are allowed to have multiple, different prototype cells, but usually we only have one.

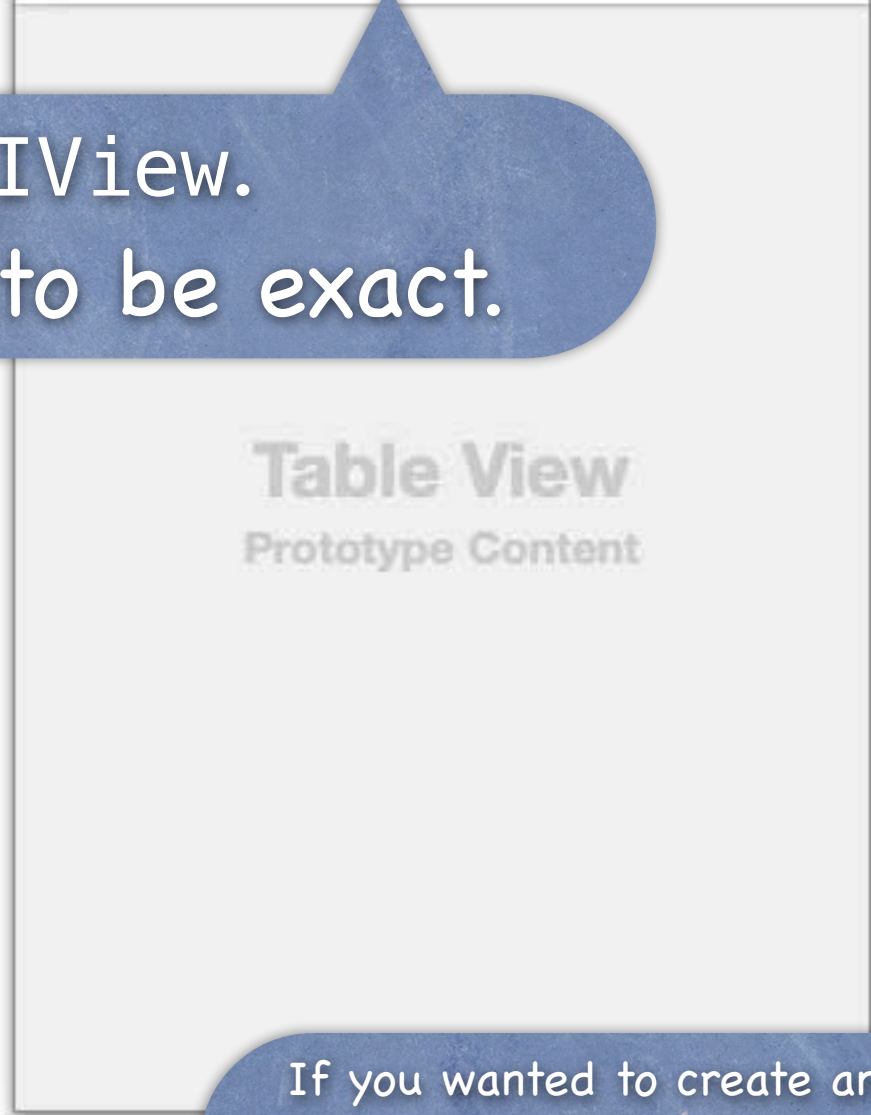


Table View

Content: Dynamic Prototypes

Prototype Cells: 1

Style: Plain

Separator: Default

Separator Insets: Default

Selection: Single Selection

Editing: No Selection During Editing

Show Selection on Touch

Index Row Limit: 0

Text Color: Default

Background: Default

Scroll View

View

Mode: Scale To Fill

Tag: 0

Interaction: User Interaction Enabled

Multiple Touch

Alpha: 1

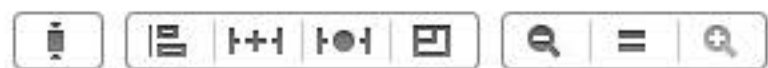
Background: Default

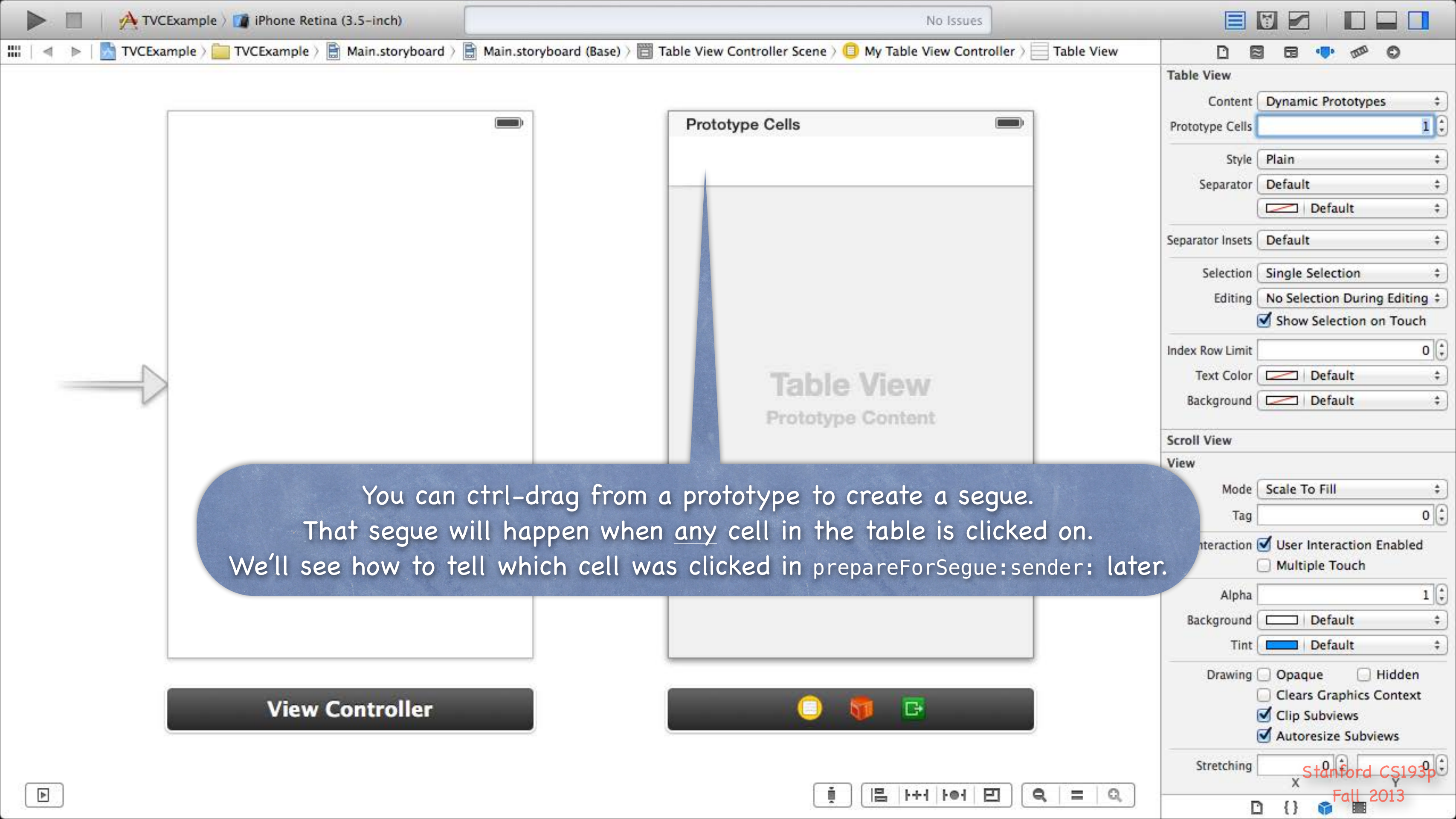
Each of these rows is a UIView.
A subclass of UITableViewCell to be exact.



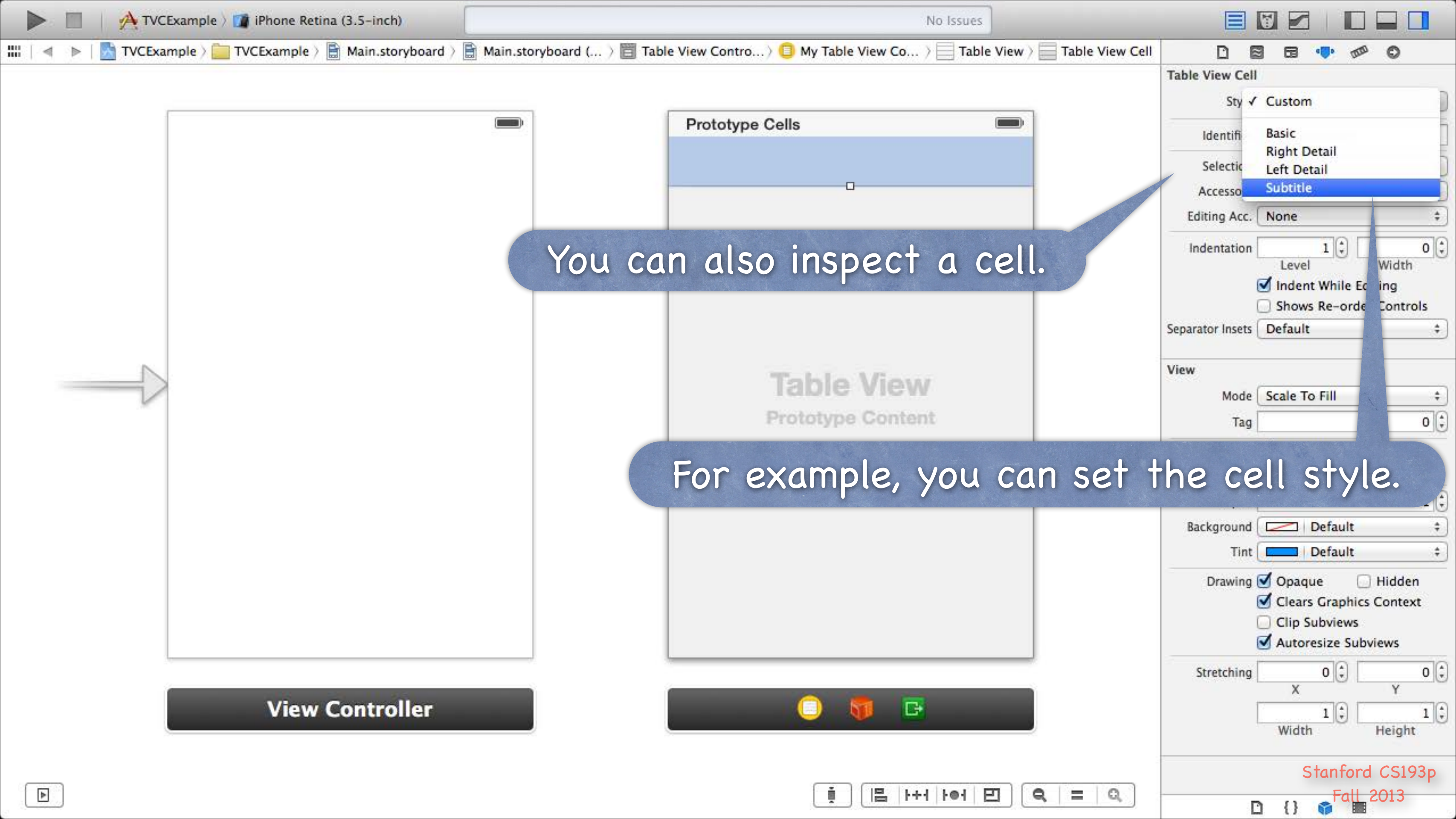
View Controller

If you wanted to create an outlet to something you drag into one of these prototypes, you'd have to subclass UITableViewCell, set its class in the Identity Inspector, and wire up to that. That's a little bit advanced for us right now!



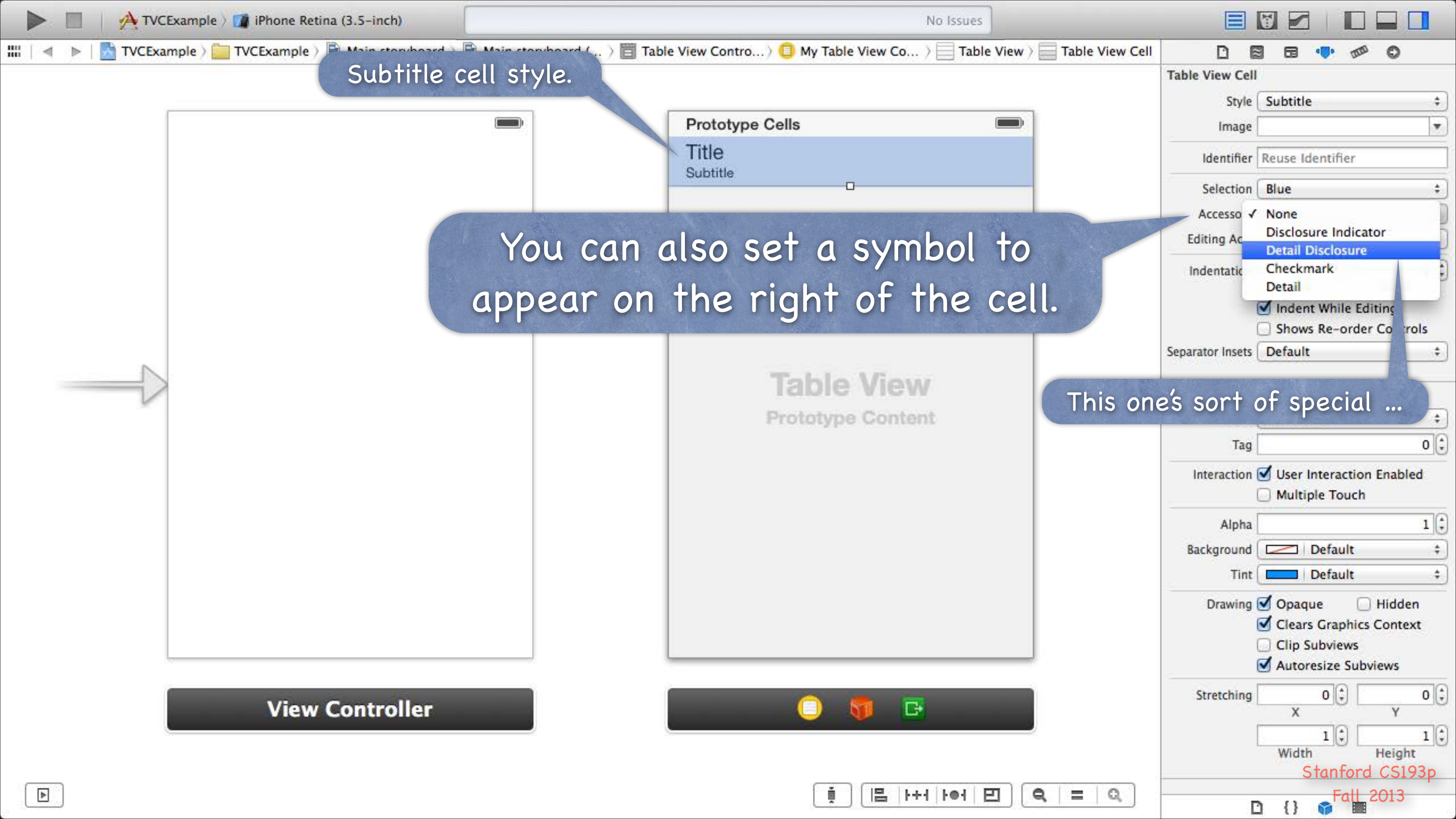


You can ctrl-drag from a prototype to create a segue. That segue will happen when any cell in the table is clicked on. We'll see how to tell which cell was clicked in `prepareForSegue:sender:` later.



You can also inspect a cell.

For example, you can set the cell style.



Subtitle cell style.

You can also set a symbol to appear on the right of the cell.

This one's sort of special ...

... we'll talk about the code behind this later.

The most important attribute of a UITableViewCell prototype, however, is this Reuse Identifier.

Table View Cell

Style **Subtitle**

Image

Identifier **Reuse Identifier**

Selection **Blue**

Accessory **Detail Disclosure**

Editing Acc. **None**

Indentation **1** Level **0** Width

Indent While Editing

Shows Re-order Controls

Separator Insets **Default**

View

Mode **Scale To Fill**

Tag **0**

Interaction User Interaction Enabled

Multiple Touch

Alpha **1**

Background **Default**

Tint **Default**

Drawing Opaque Hidden

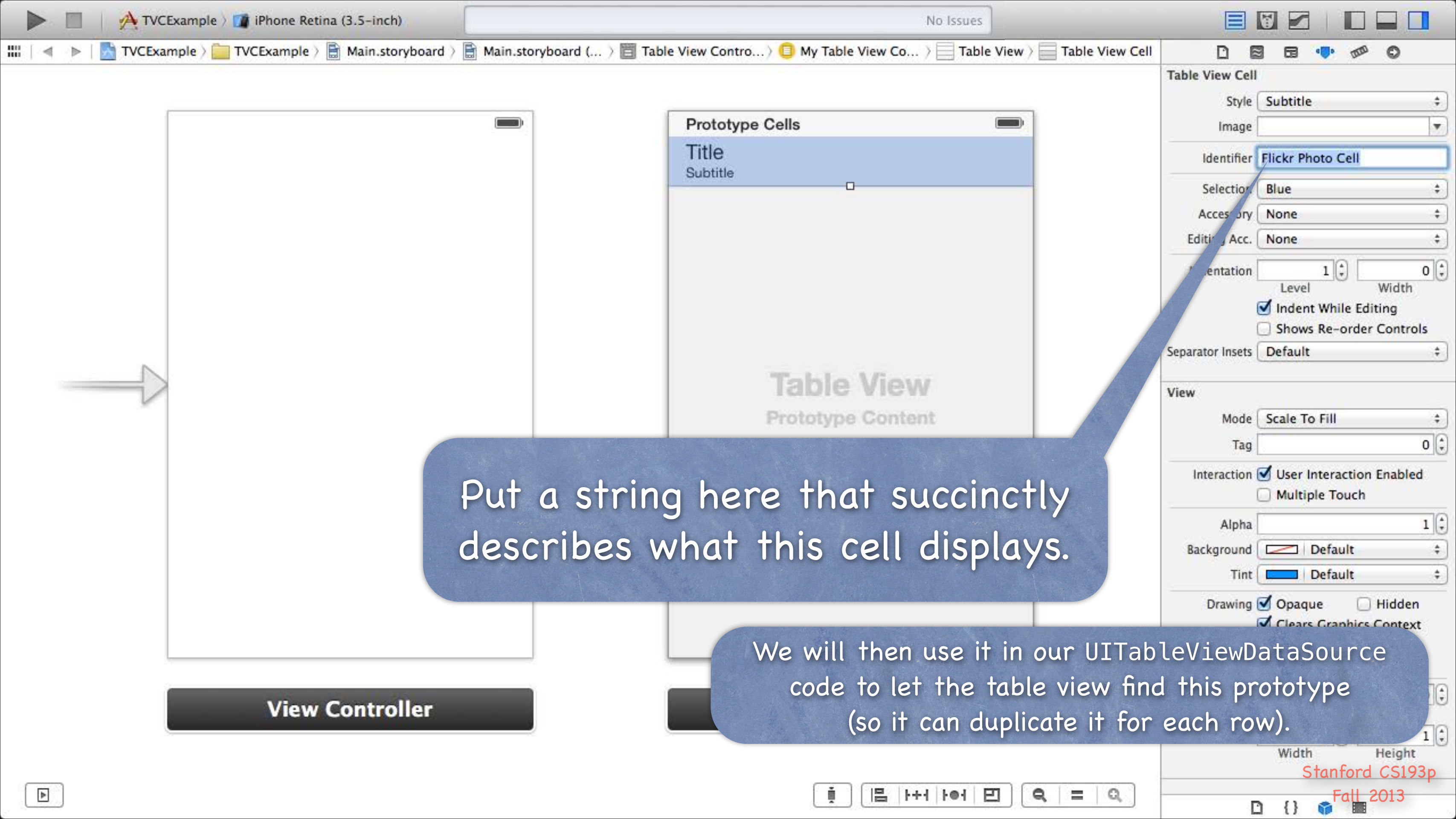
Clears Graphics Context

Clip Subviews

Autoresize Subviews

Stretching **0** X **0** Y

1 Width **1** Height



Put a string here that succinctly describes what this cell displays.

We will then use it in our UITableViewDataSource code to let the table view find this prototype (so it can duplicate it for each row).

View Controller

Table View
Prototype Content

Table View Cell

Style: Subtitle

Image: [empty]

Identifier: Flickr Photo Cell

Selection: Blue

Accessory: None

Editing Acc.: None

Indentation: Level 1, Width 0

Indent While Editing

Shows Re-order Controls

Separator Insets: Default

View

Mode: Scale To Fill

Tag: 0

Interaction: User Interaction Enabled, Multiple Touch

Alpha: 1

Background: Default

Tint: Default

Drawing: Opaque, Hidden, Clears Graphics Context

UITableView Protocols

- How do we connect to all this stuff in our code?

Via the `UITableView`'s `dataSource` and `delegate`.

The delegate is used to control how the table is displayed.

The dataSource provides the data what is displayed inside the cells.

- `UITableViewController`

Automatically sets itself as its `UITableView`'s delegate & dataSource.

Also has a property pointing to its `UITableView`:

```
@property (nonatomic, strong) UITableView *tableView;
```

(this property is actually `== self.view` in `UITableViewController`!)

UITableViewDataSource

- Important `dataSource` methods

We have to implement these 3 to be a “dynamic” (arbitrary number of rows) table ...

How many `sections` in the table?

How many `rows` in each section?

Give me a `UITableViewCell` to use to draw each cell at a given row in a given section.

Let's cover the last one first (since the first two are very straightforward) ...

UITableViewDataSource

- How do we control what is drawn in each cell in a dynamic table?

Each row is drawn by its own instance of `UITableViewCell` (a `UIView` subclass).

Here is the `UITableViewDataSource` method to get that cell for a given row in a section ...

```
- (UITableViewCell *)tableView:(UITableView *)sender  
    cellForRowAtIndexPath:(NSIndexPath *)indexPath  
{  
  
}  
}
```

In a static table, you do not need to implement this method (though you can if you want to ignore what's in the storyboard).

UITableViewDataSource

- How do we control what is drawn in each cell in a dynamic table?

Each row is drawn by its own instance of `UITableViewCell` (a `UIView` subclass).

Here is the `UITableViewDataSource` method to get that cell for a given row in a section ...

```
- (UITableViewCell *)tableView:(UITableView *)sender  
    cellForRowAtIndexPath:(NSIndexPath *)indexPath  
{  
  
}  
}
```

`NSIndexPath` is just an object with two important properties for use with `UITableView`: `row` and `section`.

UITableViewDataSource

- How do we control what is drawn in each cell in a dynamic table?

Each row is drawn by its own instance of `UITableViewCell` (a `UIView` subclass).

Here is the `UITableViewDataSource` method to get that cell for a given row in a section ...

```
- (UITableViewCell *)tableView:(UITableView *)sender
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // get a cell to use (instance of UITableViewCell)
    // set @propertys on the cell to prepare it to display
}
```

UITableViewDataSource

• How do we control what is drawn in each cell in a dynamic table?

Each row is drawn by its own instance of `UITableViewCell` (a `UIView` subclass).

Here is the `UITableViewDataSource` method to get that cell for a given row in a section ...

```
- (UITableViewCell *)tableView:(UITableView *)sender
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell;
    cell = [self.tableView dequeueReusableCellWithIdentifier:@"Flickr Photo Cell"
                                                forIndexPath:indexPath];

    // set @propertys on the cell to prepare it to display
}
```

This MUST match what is in your storyboard if you want to use the prototype you defined there!

UITableViewDataSource

How do we control what is drawn in each cell in a dynamic table?

Each row is drawn by its own instance of `UITableViewCell` (a `UIView` subclass).

Here is the `UITableViewDataSource` method to get that cell for a given row in a section ...

```
- (UITableViewCell *)tableView:(UITableView *)sender
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell;
    cell = [self.tableView dequeueReusableCellWithIdentifier:@"Flickr Photo Cell"
                                                forIndexPath:indexPath];
    // set @propertys on the cell to prepare it to display
}
```

The cells in the table are actually reused.

When one goes off-screen, it gets put into a "reuse pool."

The next time a cell is needed, one is grabbed from the reuse pool if available.

If none is available, one will be put into the reuse pool if there's a prototype in the storyboard.

Otherwise this dequeue method will return `nil`.

UITableViewDataSource

• How do we control what is drawn in each cell in a dynamic table?

Each row is drawn by its own instance of `UITableViewCell` (a `UIView` subclass).

Here is the `UITableViewDataSource` method to get that cell for a given row in a section ...

```
- (UITableViewCell *)tableView:(UITableView *)sender
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell;
    cell = [self.tableView dequeueReusableCellWithIdentifier:@"Flickr Photo Cell"
                                                forIndexPath:indexPath];
    cell.textLabel.text = [self getTitleForRow:indexPath.row inSection:indexPath.section];
    return cell;
}
```

There are obviously other things you can do in the cell besides setting its text (detail text, image, checkmark, etc.).

UITableViewDataSource

- How do we control what is drawn in each cell in a dynamic table?

Each row is drawn by its own instance of `UITableViewCell` (a `UIView` subclass).

Here is the `UITableViewDataSource` method to get that cell for a given row in a section ...

```
- (UITableViewCell *)tableView:(UITableView *)sender
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell;
    cell = [self.tableView dequeueReusableCellWithIdentifier:@"Flickr Photo Cell"
                                                forIndexPath:indexPath];
    cell.textLabel.text = [self getTitleForRow:indexPath.row inSection:indexPath.section];
    return cell;
}
```

See how we are using `indexPath.section` and `indexPath.row` to get Model information to set up this cell.

UITableViewDataSource

• How does a dynamic table know how many rows there are?

And how many sections, too, of course?

Via these two UITableViewDataSource methods ...

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

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

• Number of sections is 1 by default

In other words, if you don't implement `numberOfSectionsInTableView:`, it will be 1.

• No default for `tableView:numberOfRowsInSection:`

This is a required method in this protocol (as is `tableView:cellForRowAtIndexPath:`).

• What about a static table?

Do not implement these dataSource methods for a static table.

UITableViewController will take care of that for you.

UITableViewDataSource

- There are a number of other methods in this protocol
 - But we're not going to cover them today.
 - They are mostly about getting the headers and footers for sections.
 - And about keeping the Model in sync with table edits (moving/deleting/inserting rows).

UITableViewDelegate

- All of the above was the UITableView's dataSource
But `UITableView` has another protocol-driven delegate called its `delegate`.
- The delegate controls how the UITableView is displayed
Not what it displays (that's the dataSource's job).
- Common for dataSource and delegate to be the same object
Usually the Controller of the MVC in which the UITableView is part of the View.
This is the way UITableViewController sets it up for you.
- The delegate also lets you observe what the table view is doing
The classic "will/did" sorts of things.
An important one is "user did select a row."
Usually we don't need this because we simply segue when a row is touched.
But there are some occasions where it will be useful ...

UITableView "Target/Action"

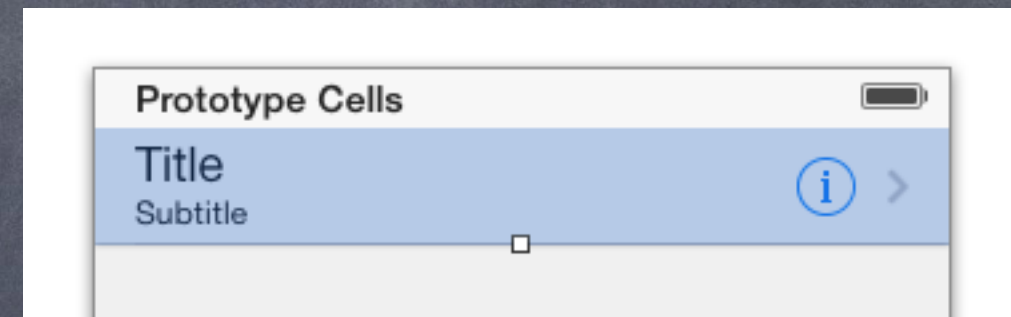
- UITableViewDelegate method sent when row is selected

This is sort of like "table view target/action" (only needed if you're not segueing, of course).
On the iPad, where the table might be on screen with what it updates, you might need this.

```
- (void)tableView:(UITableView *)sender didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    // go do something based on information about my Model
    // corresponding to indexPath.row in indexPath.section
}
```

UITableView Detail Disclosure

- Remember the little circled i?
Clicking on this will not segue.

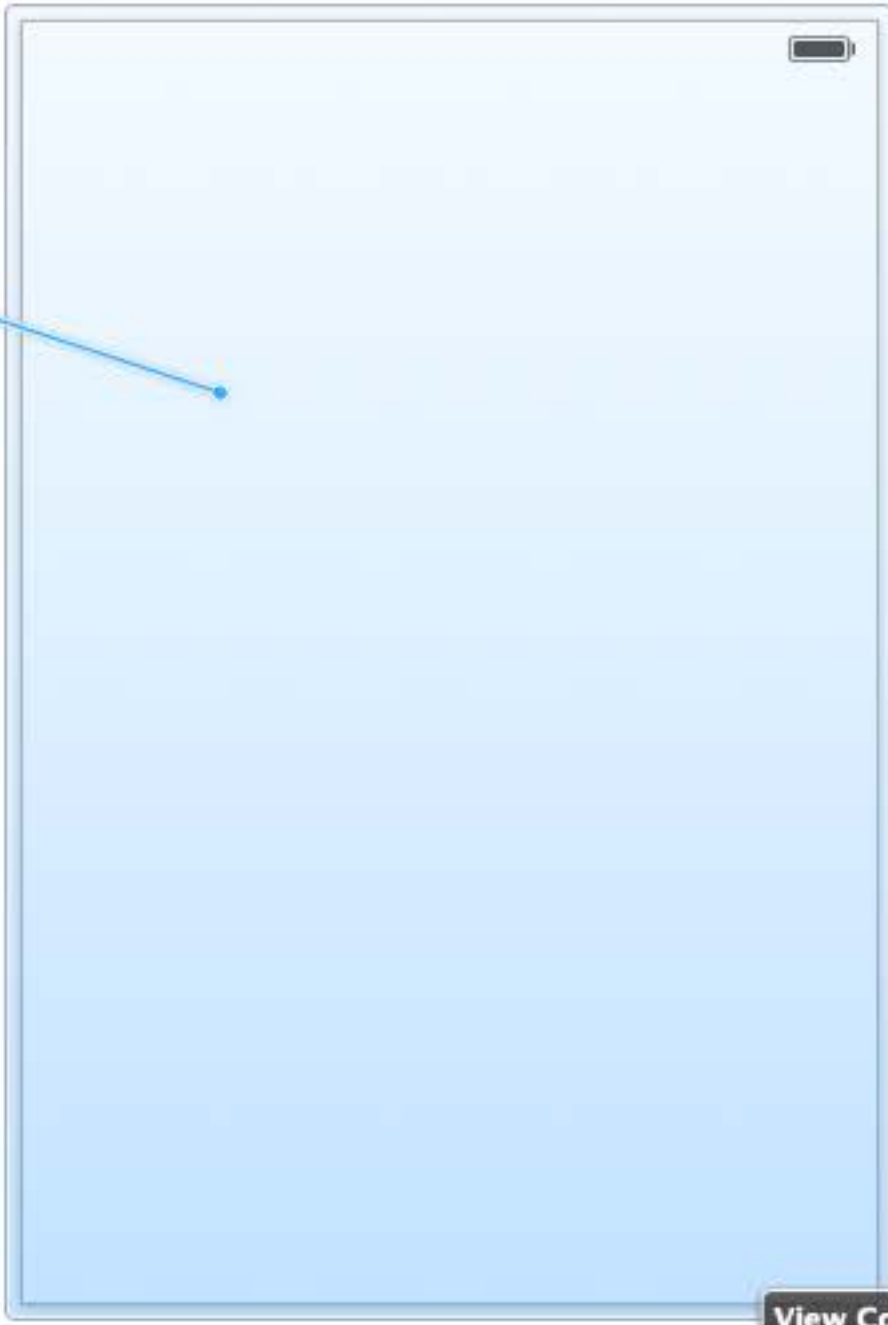
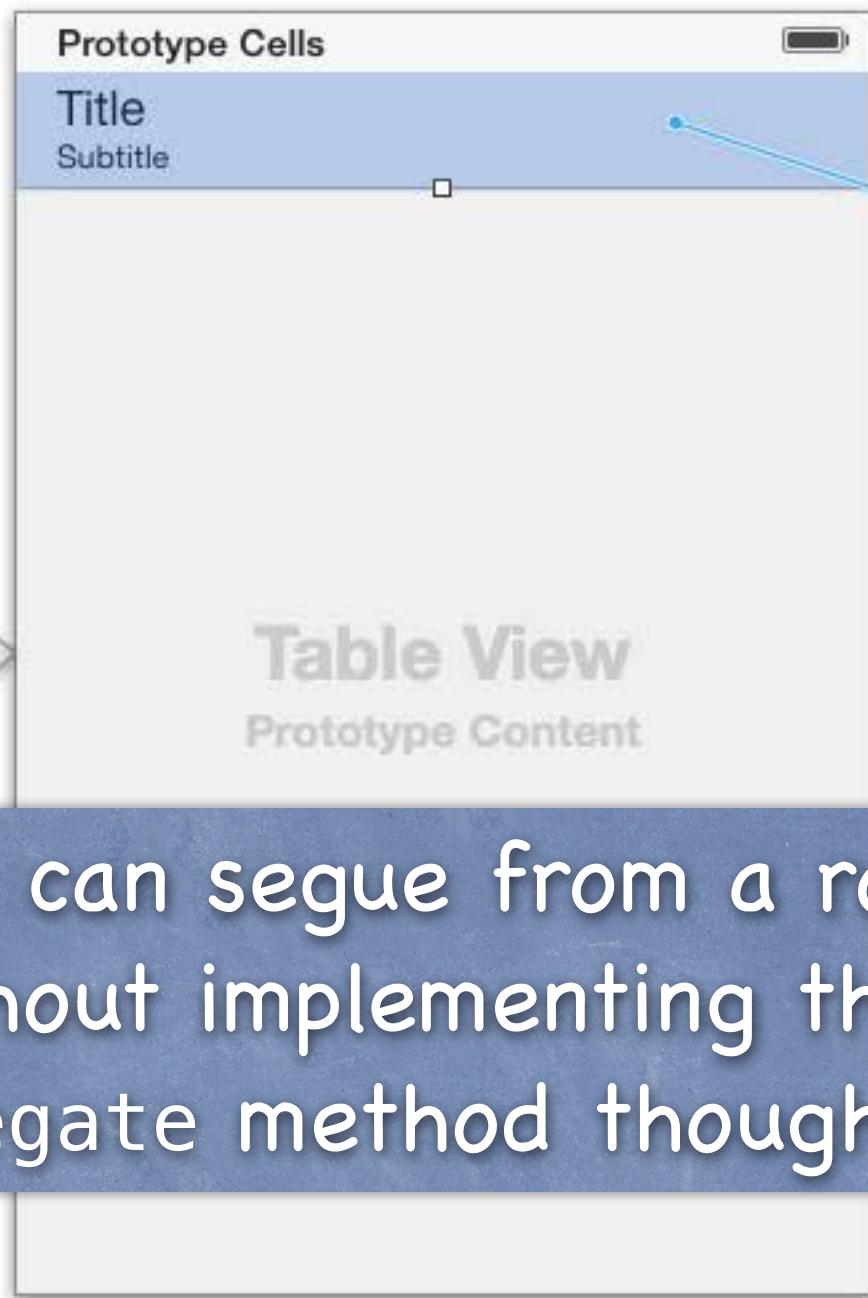


Instead it will invoke this method in the UITableViewDelegate protocol ...

```
- (void)tableView:(UITableView *)sender  
    accessoryButtonTappedForRowWithIndexPath:(NSIndexPath *)indexPath  
{  
    // Do something related to the row at indexPath,  
    // but not the primary action associated with touching the row  
}
```

UITableViewDelegate

- Lots and lots of other **delegate** methods
 - will/did** methods for both selecting and deselecting rows.
 - Providing UIView objects to draw section headers and footers.
 - Handling editing rows (moving them around with touch gestures).
 - willBegin/didEnd** notifications for editing (i.e. removing/moving) rows.
 - Copying/pasting rows.



You can segue from a row without implementing that delegate method though ...

Table View Cell

Style: **Subtitle**

Image: [Empty]

Identifier: **Reuse Identifier**

Selection: **Blue**

Accessory: **None**

Editing Acc.: **None**

Indentation: Level: **1** Width: **0**

Indent While Editing

Shows Re-order Controls

Separator Insets: **Default**

View

Mode: **Scale To Fill**

Tag: **0**

Interaction: User Interaction Enabled

Multiple Touch

Alpha: **1**

Background: **Default**

Tint: **Default**

Drawing: Opaque Hidden

Clears Graphics Context

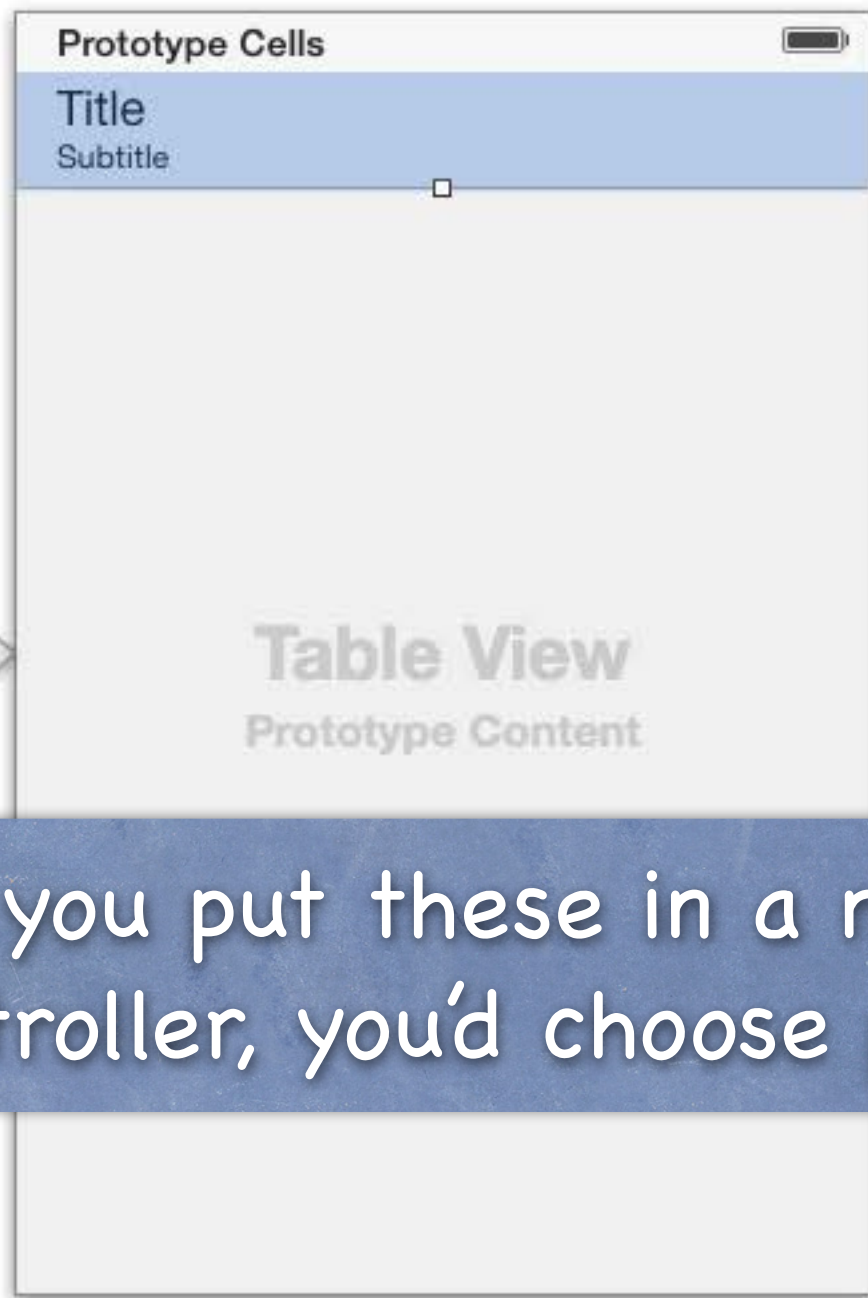
Clip Subviews

Autoresize Subviews

Stretching: X: **0** Y: **0**

Width: **1** Height: **1**

View Controller



If you put these in a navigation controller, you'd choose push here.

Table View Cell

Style: **Subtitle**

Image: [Empty]

Identifier: Reuse Identifier

Selection: **Blue**

Accessory: **None**

Editing Acc.: **None**

Indentation: Level: 1, Width: 0

Indent While Editing

Shows Re-order Controls

Separator Insets: **Default**

View

Mode: **Scale To Fill**

Tag: 0

Interaction: User Interaction Enabled, Multiple Touch

Alpha: 1

Background: **Default**

Tint: **Default**

Drawing: Opaque, Hidden, Clears Graphics Context, Clip Subviews, Autoresize Subviews

Stretching: X: 0, Y: 0, Width: 1, Height: 1

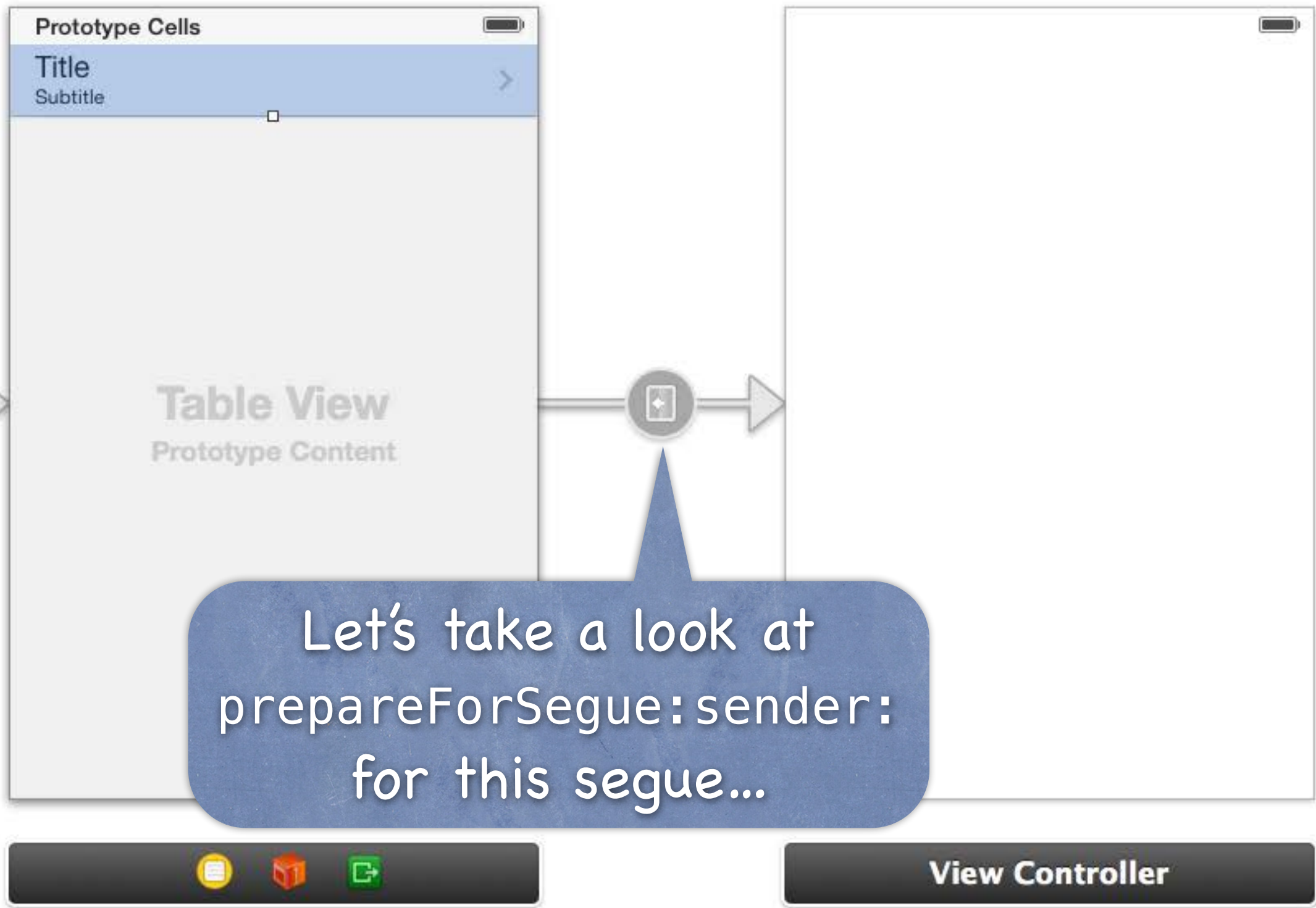


Table View Cell

Style: **Subtitle**

Image: [Empty]

Identifier: **Reuse Identifier**

Selection: **Blue**

Accessory: **Disclosure Indicator**

Editing Acc.: **None**

Indentation: Level: **1** Width: **0**

Indent While Editing

Shows Re-order Controls

Separator Insets: **Default**

View

Mode: **Scale To Fill**

Tag: **0**

Interaction: User Interaction Enabled

Multiple Touch

Alpha: **1**

Background: **Default**

Tint: **Default**

Drawing: Opaque Hidden

Clears Graphics Context

Clip Subviews

Autoresize Subviews

Stretching: X: **0** Y: **0**

Width: **1** Height: **1**

UITableView Segue

- The sender of `prepareForSegue:sender:` is the `UITableViewCell`. Use the important method `indexPathForCell:` to find out the `indexPath` of the row that's segueing.

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    NSIndexPath *indexPath = [self.tableView indexPathForCell:sender];
    // prepare segue.destinationController to display based on information
    // about my Model corresponding to indexPath.row in indexPath.section
}
```

UITableView Spinner

- UITableViewController has an “activity indicator” built in

You get it via this property in UITableViewController ...

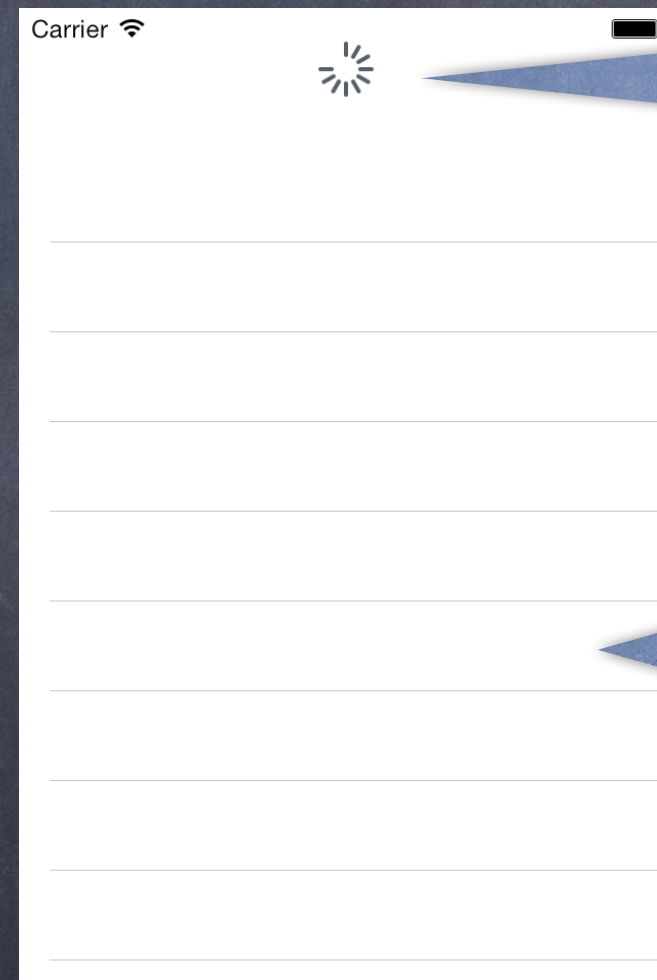
```
@property (strong) UIRefreshControl *refreshControl;
```

Start it with ...

```
- (void)beginRefreshing;
```

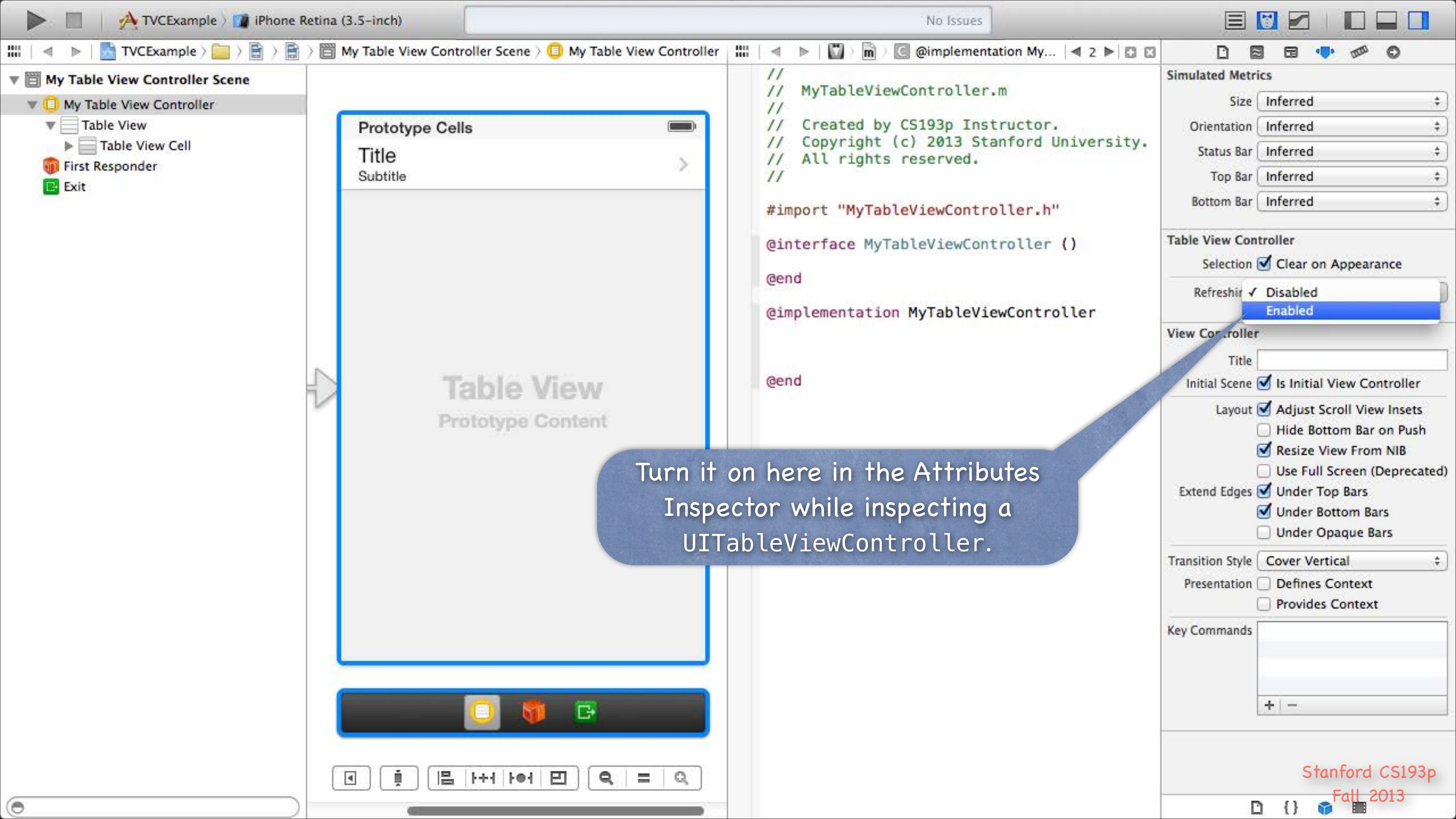
Stop it with ...

```
- (void)endRefreshing;
```



It appears here at the top of the table view.

Also, users can “pull down” on the table view and the refresh control will send its action to its target.



```
//  
// MyTableViewController.m  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "MyTableViewController.h"  
  
@interface MyTableViewController ()  
  
@end  
  
@implementation MyTableViewController  
  
  
@end
```

Turn it on here in the Attributes Inspector while inspecting a UITableViewController.

Simulated Metrics

- Size: Inferred
- Orientation: Inferred
- Status Bar: Inferred
- Top Bar: Inferred
- Bottom Bar: Inferred

Table View Controller

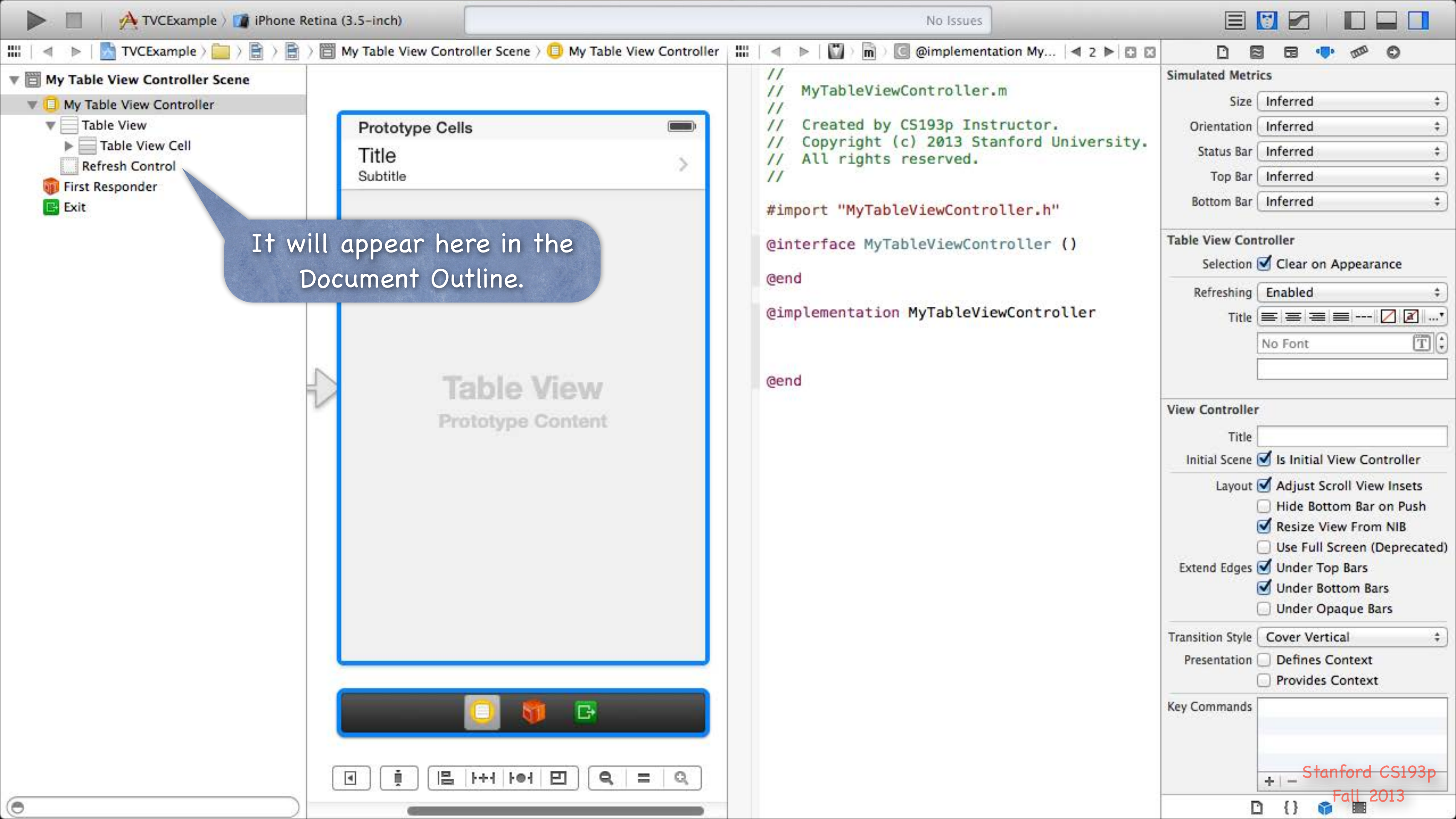
- Selection: Clear on Appearance
- Refresh Rate: Disabled, **Enabled**

View Controller

- Title: []
- Initial Scene: Is Initial View Controller
- Layout: Adjust Scroll View Insets, Hide Bottom Bar on Push, Resize View From NIB, Use Full Screen (Deprecated)
- Extend Edges: Under Top Bars, Under Bottom Bars, Under Opaque Bars
- Transition Style: Cover Vertical
- Presentation: Defines Context, Provides Context

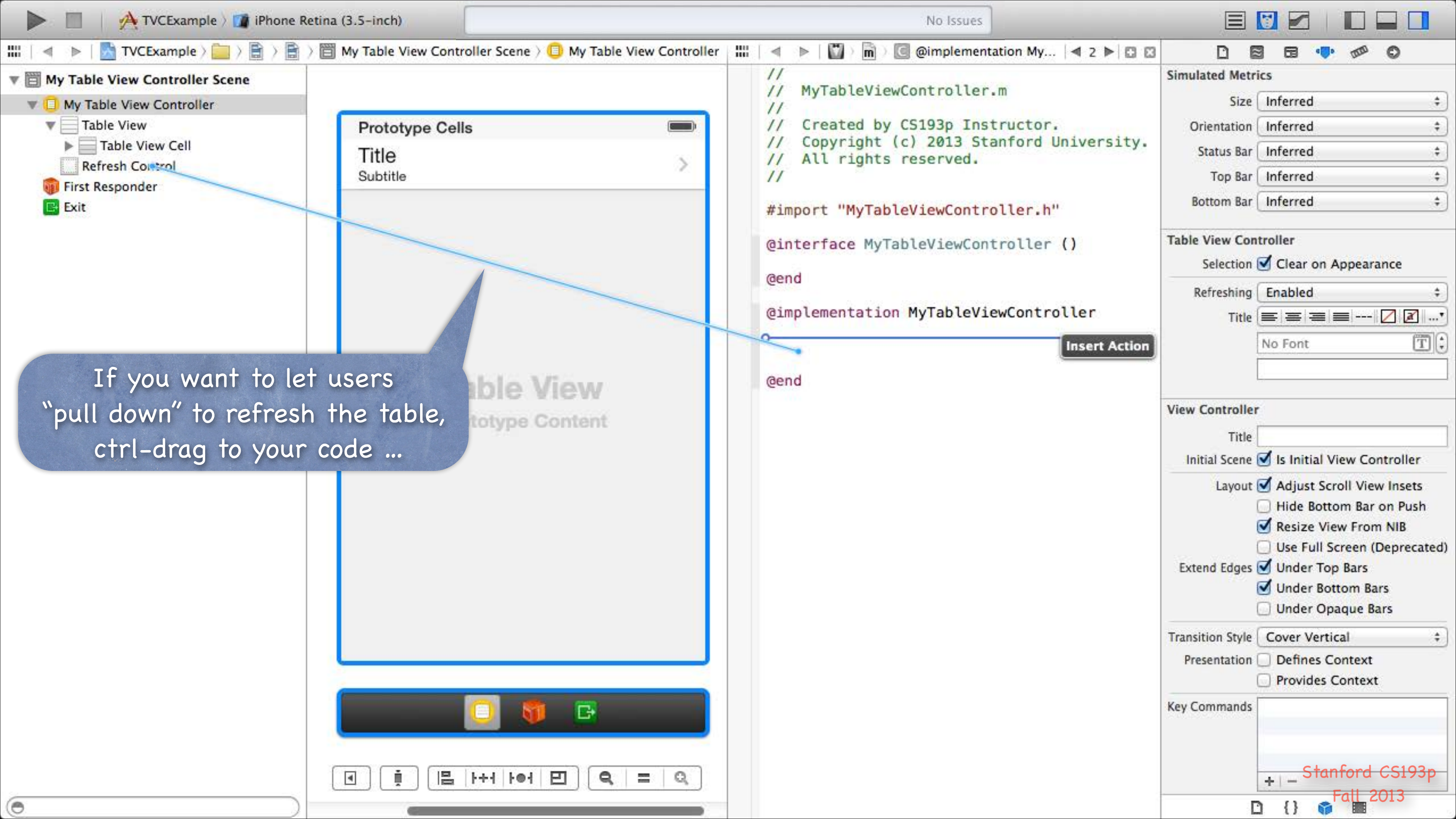
Key Commands

+ -



It will appear here in the Document Outline.

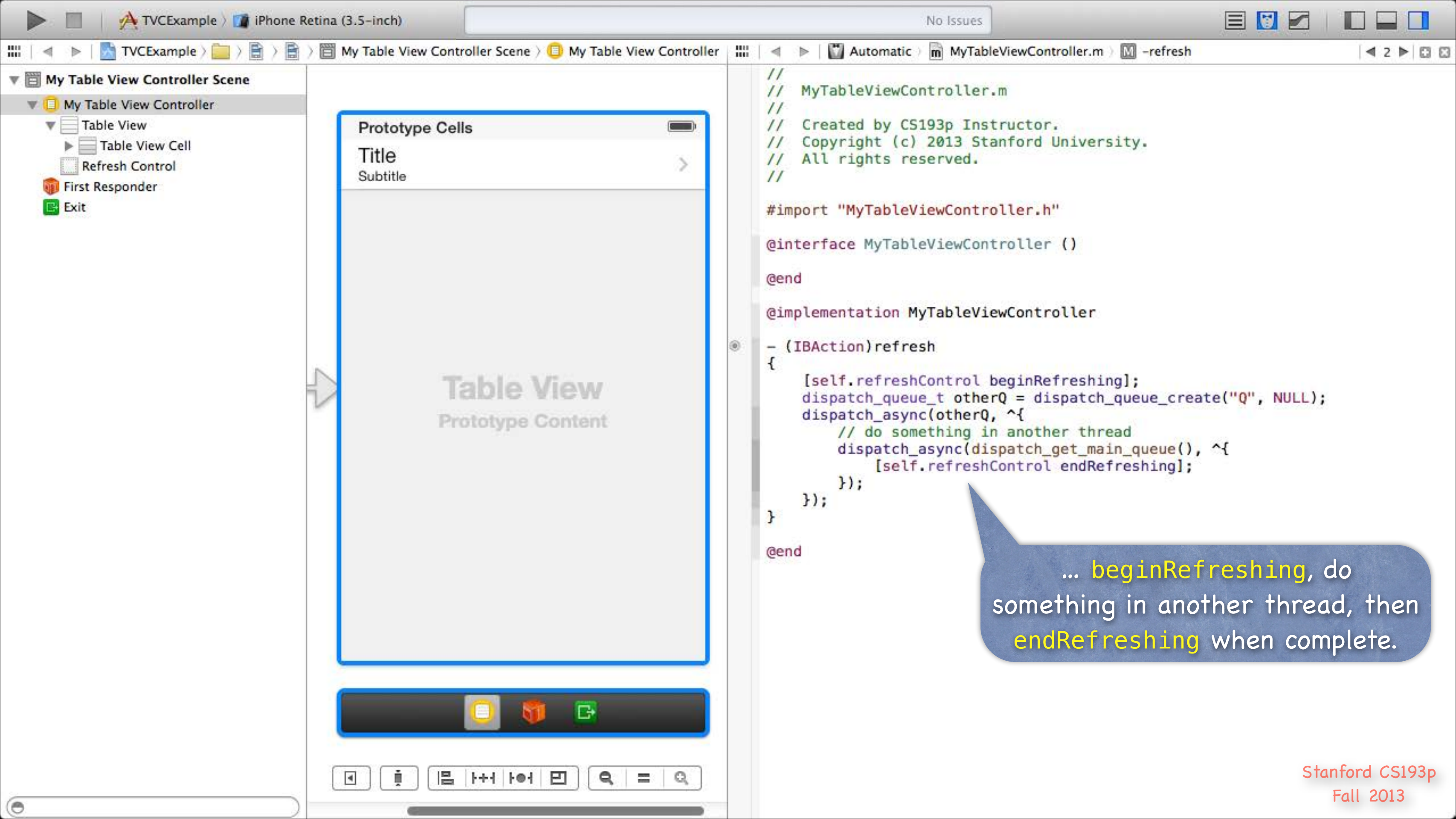
```
//  
// MyTableViewController.m  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "MyTableViewController.h"  
  
@interface MyTableViewController ()  
  
@end  
  
@implementation MyTableViewController  
  
  
@end
```



If you want to let users "pull down" to refresh the table, ctrl-drag to your code ...

```
//  
// MyTableViewController.m  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "MyTableViewController.h"  
  
@interface MyTableViewController ()  
  
@end  
  
@implementation MyTableViewController  
  
@end
```

Insert Action



Prototype Cells

Title

Subtitle

Table View

Prototype Content

```
//  
// MyTableViewController.m  
//  
// Created by CS193p Instructor.  
// Copyright (c) 2013 Stanford University.  
// All rights reserved.  
//  
  
#import "MyTableViewController.h"  
  
@interface MyTableViewController ()  
  
@end  
  
@implementation MyTableViewController  
  
- (IBAction)refresh  
{  
    [self.refreshControl beginRefreshing];  
    dispatch_queue_t otherQ = dispatch_queue_create("Q", NULL);  
    dispatch_async(otherQ, ^{  
        // do something in another thread  
        dispatch_async(dispatch_get_main_queue(), ^{  
            [self.refreshControl endRefreshing];  
        });  
    });  
}  
  
@end
```

... **beginRefreshing**, do something in another thread, then **endRefreshing** when complete.

UITableView

• What if your Model changes?

- `(void) reloadData;`

Causes the table view to call `numberOfSectionsInTableView:` and `numberOfRowsInSection:` all over again and then `cellForRowAtIndexPath:` on each visible cell.

Relatively heavyweight, but if your entire data structure changes, that's what you need.

If only part of your Model changes, there are lighter-weight reloaders, for example ...

- `(void) reloadRowsAtIndexPaths:(NSArray *)indexPaths
withRowAnimation:(UITableViewRowAnimation)animationStyle;`

• There are dozens of other methods in UITableView

Setting headers and footers for the entire table.

Controlling the look (separator style and color, default row height, etc.).

Getting cell information (cell for index path, index path for cell, visible cells, etc.).

Scrolling to a row.

Selection management (allows multiple selection, getting the selected row, etc.).

Moving, inserting and deleting rows, etc.

Universal Applications

- A “Universal” Application will run on both iPhone and iPad

It might look different on each.

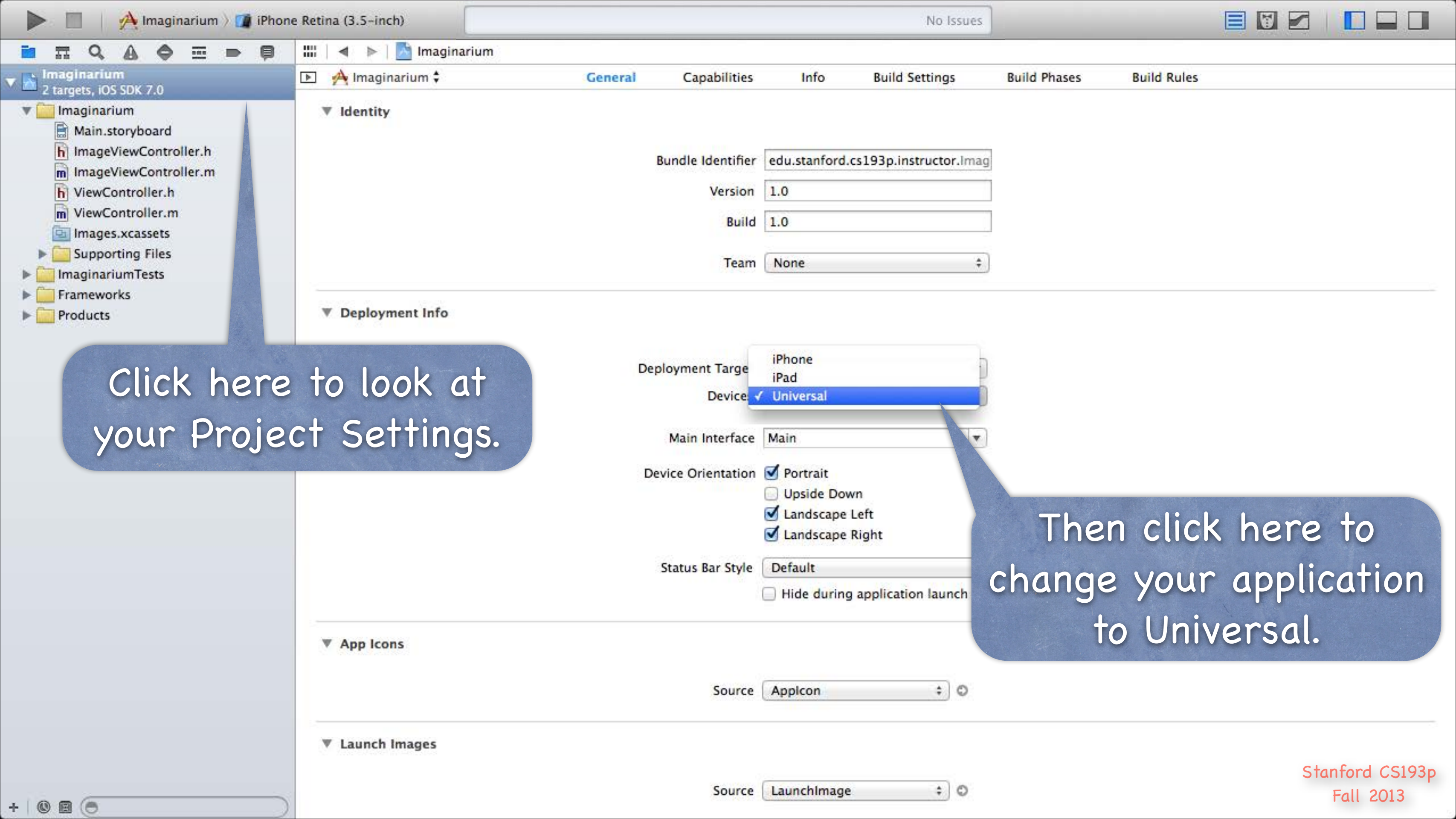
But it's a single binary image (i.e. it's one app, not two).

Two different storyboards.

- How to create one

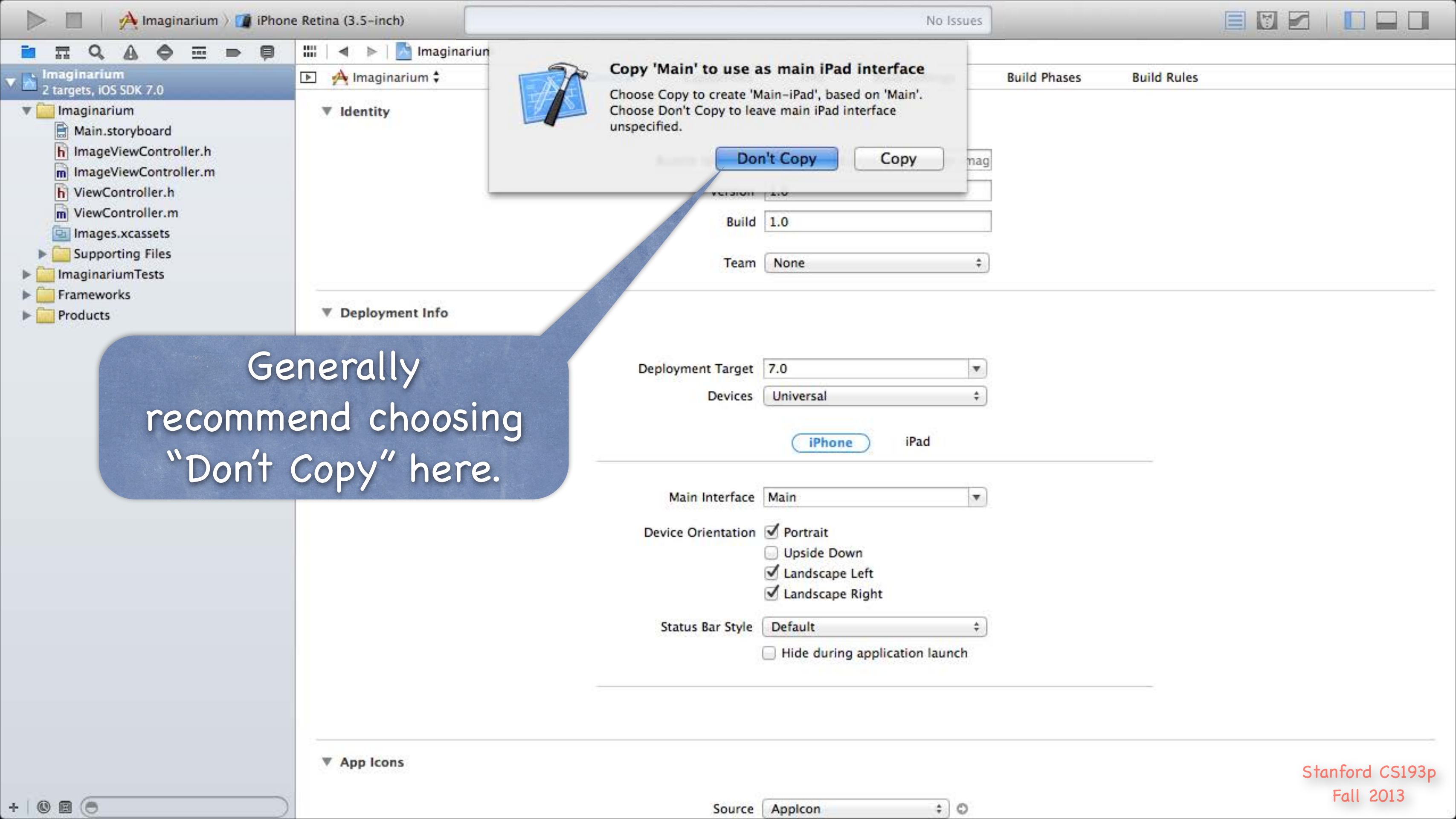
When you create the project, pick Universal instead of iPhone or iPad.

If you have an existing iPhone- or iPad-only project, you must edit your Project Settings ...



Click here to look at your Project Settings.

Then click here to change your application to Universal.



Generally recommend choosing "Don't Copy" here.

Copy 'Main' to use as main iPad interface

Choose Copy to create 'Main-iPad', based on 'Main'. Choose Don't Copy to leave main iPad interface unspecified.

Don't Copy Copy

Deployment Target: 7.0

Devices: Universal

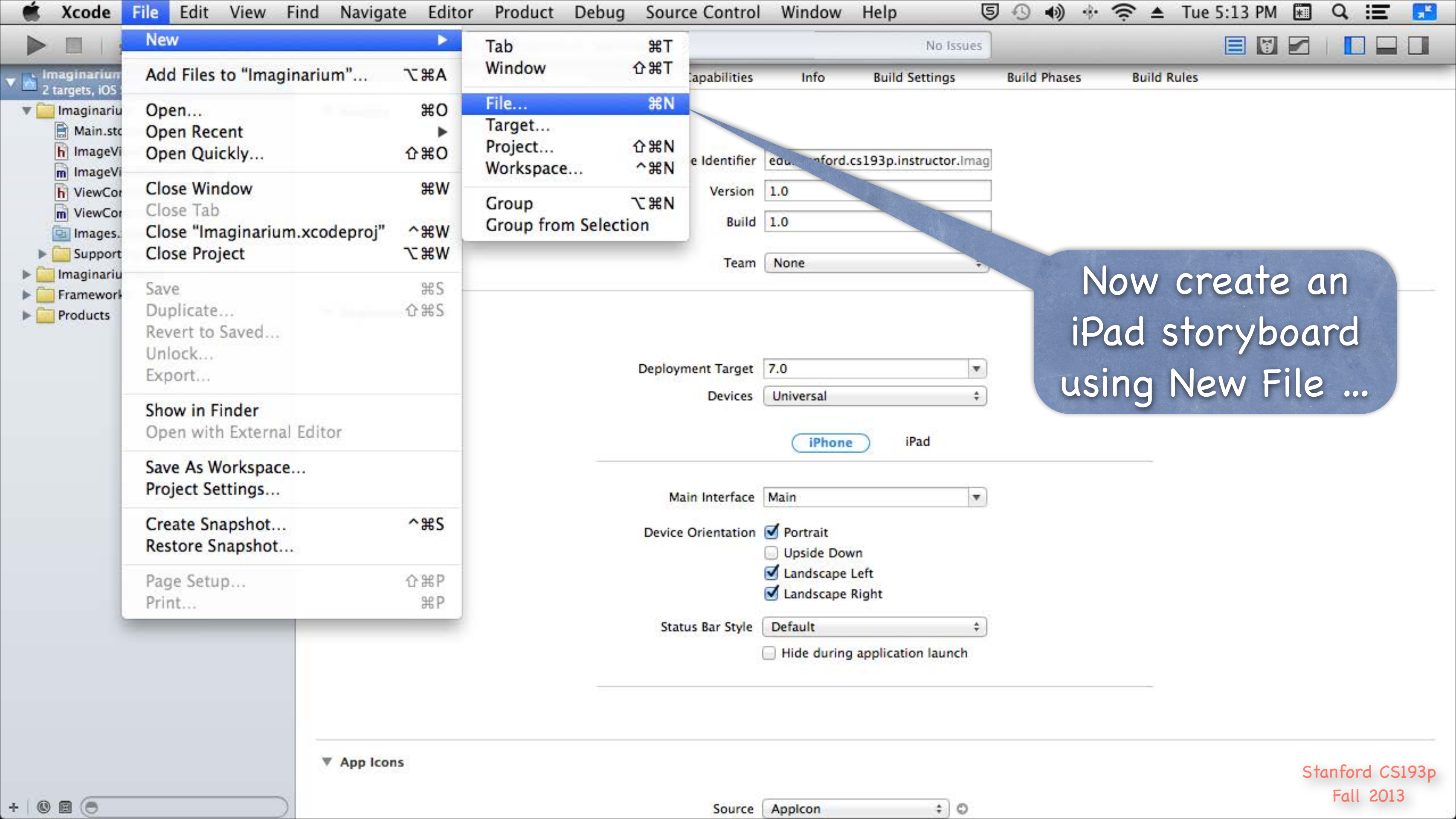
iPhone iPad

Main Interface: Main

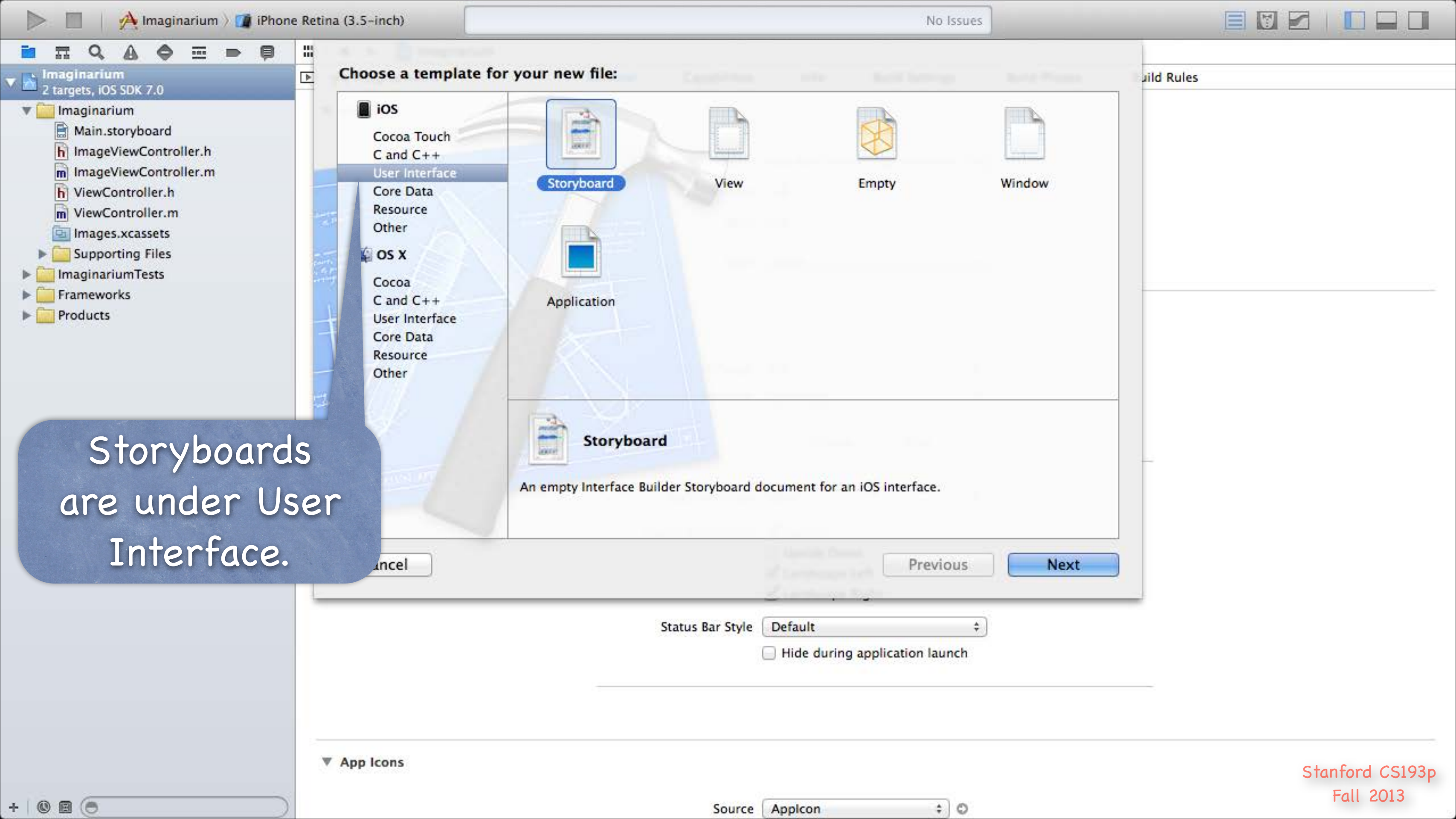
Device Orientation: Portrait, Upside Down, Landscape Left, Landscape Right

Status Bar Style: Default

Hide during application launch



Now create an iPad storyboard using New File ...



Storyboards are under User Interface.

Choose a template for your new file:

- iOS
 - Cocoa Touch
 - C and C++
 - User Interface
 - Core Data
 - Resource
 - Other
 - OS X
 - Cocoa
 - C and C++
 - User Interface
 - Core Data
 - Resource
 - Other

Storyboard

View

Empty

Window

Application

Storyboard

An empty Interface Builder Storyboard document for an iOS interface.

Cancel Previous Next

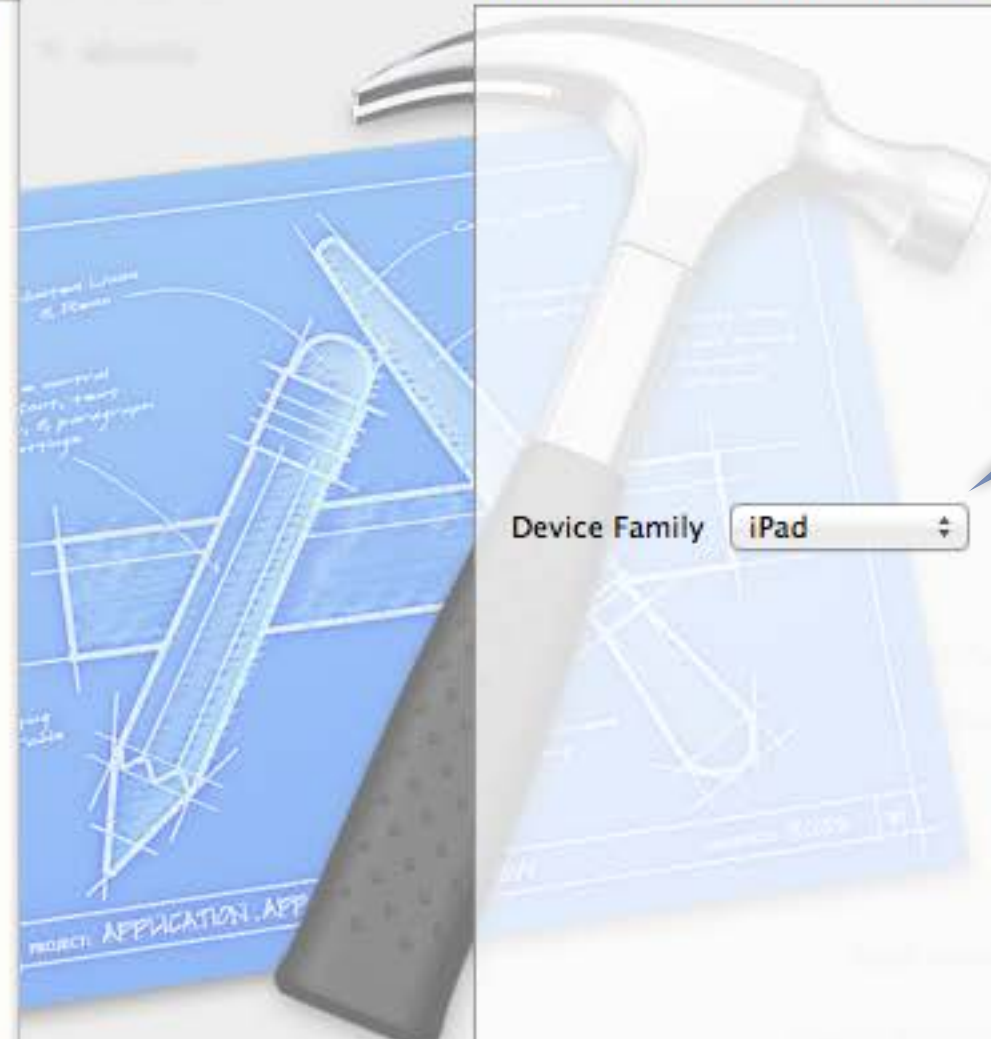
Status Bar Style Default
 Hide during application launch

App Icons

Source Applcon

- Imaginarium
2 targets, iOS SDK 7.0
 - Imaginarium
 - Main.storyboard
 - ImageViewController.h
 - ImageViewController.m
 - ViewController.h
 - ViewController.m
 - Images.xcassets
 - Supporting Files
 - ImaginariumTests
 - Frameworks
 - Products

Choose options for your new file:



Device Family iPad

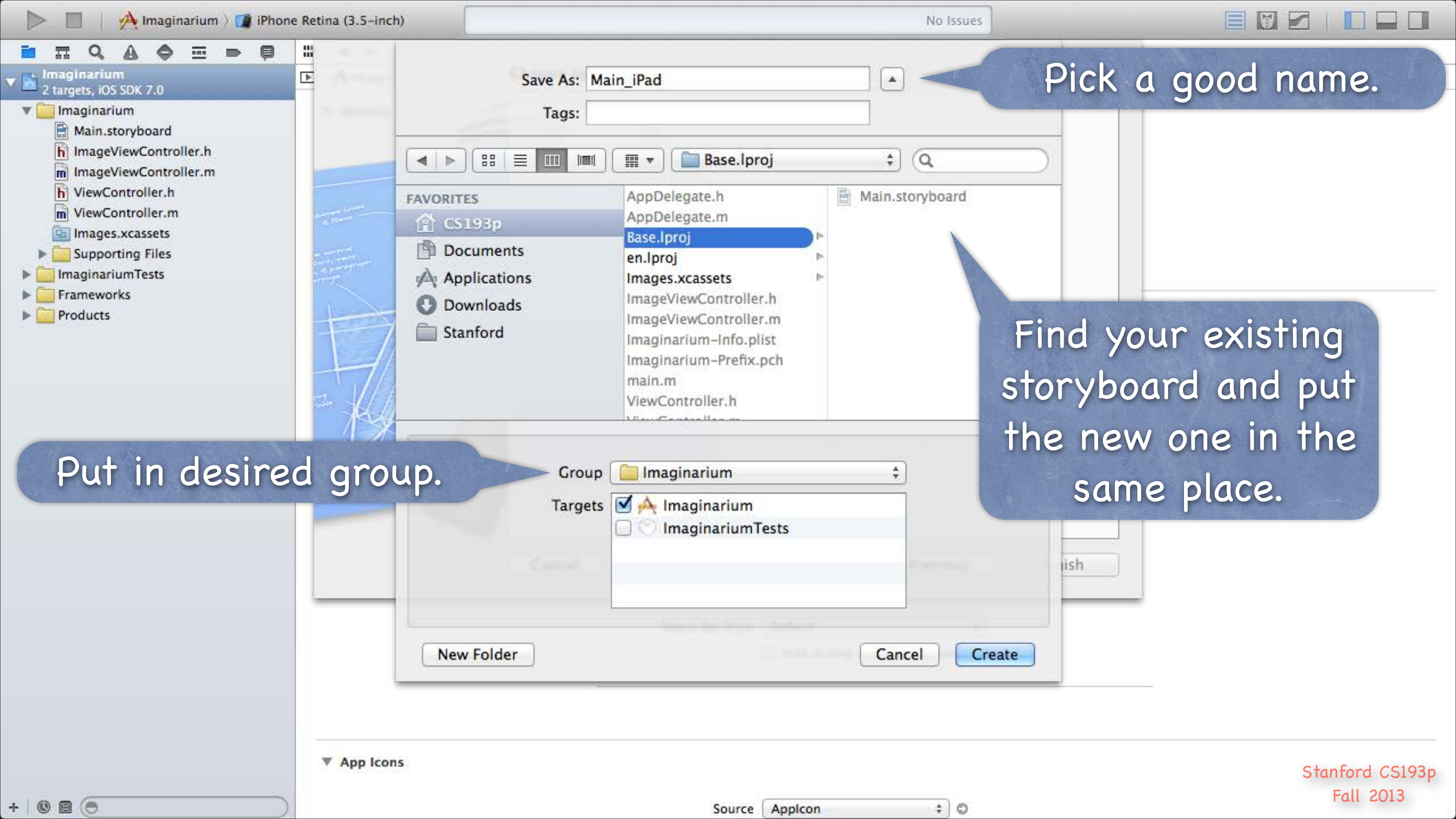
Cancel Previous Next

Pick appropriate device.

Status Bar Style Default
 Hide during application launch

App Icons

Source Applcon



Save As: Main_iPad

Pick a good name.

Base.lproj

Find your existing storyboard and put the new one in the same place.

Put in desired group.

Group: Imaginarium

Targets: Imaginarium, ImaginariumTests

New Folder

Cancel

Create

App Icons

Source: Applcon

- Imaginarium
 - 2 targets, iOS SDK 7.0
 - Imaginarium
 - Main_iPad.storyboard
 - Main.storyboard
 - ImageViewController.h
 - ImageViewController.m
 - ViewController.h
 - ViewController.m
 - Images.xcassets
 - Supporting Files
 - ImaginariumTests
 - Frameworks
 - Products

Here's your iPad storyboard.

Imaginarium **General** Capabilities Info Build Settings Build Phases Build Rules

Bundle Identifier: edu.stanford.cs193p.instructor.Imag
 Version: 1.0
 Build: 1.0
 Team: None

Deployment Info

Deployment Target: 7.0
 Devices: Universal

iPhone iPad

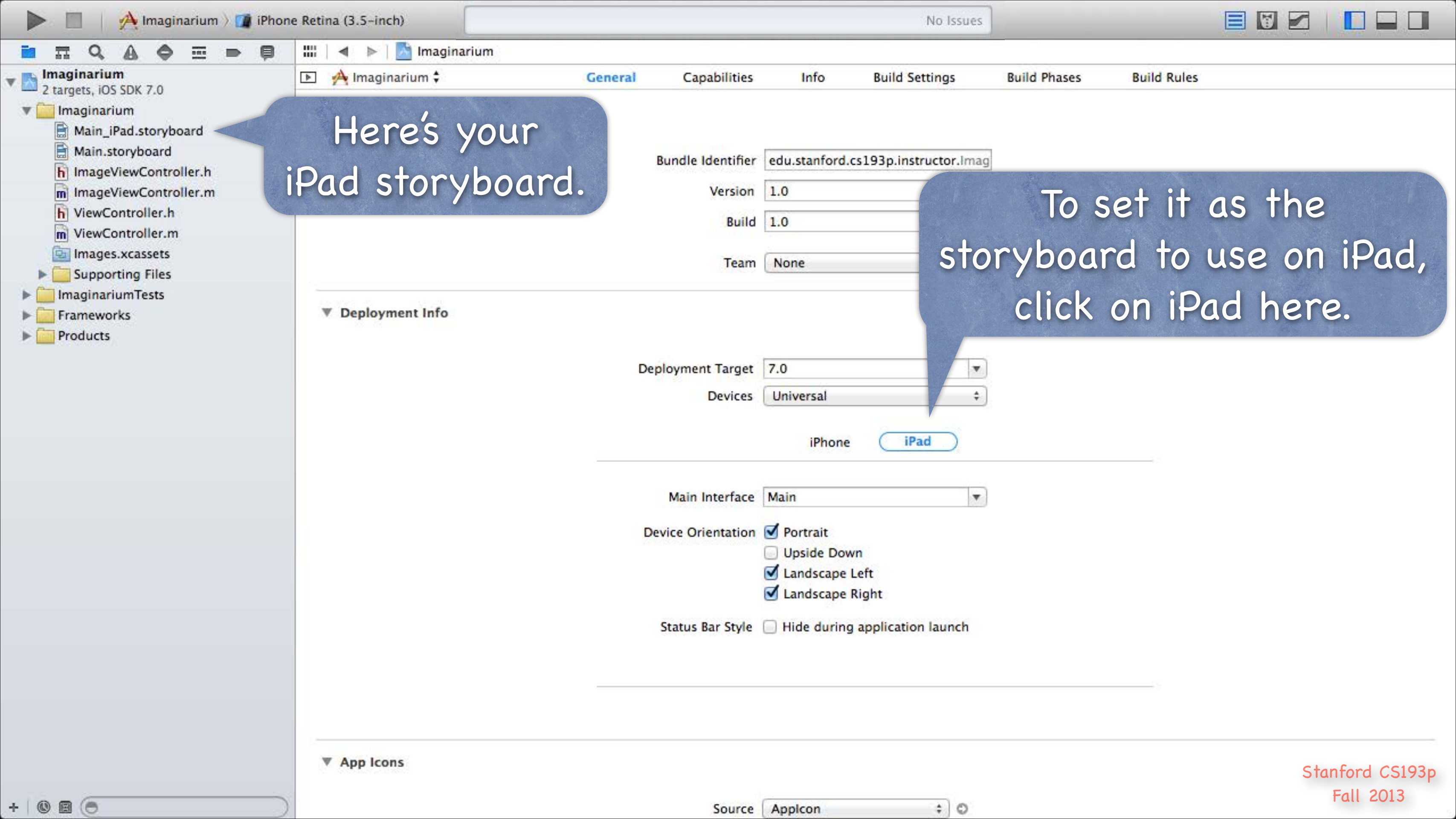
Main Interface: Main

- Device Orientation
- Portrait
 - Upside Down
 - Landscape Left
 - Landscape Right

Status Bar Style: Default
 Hide during application launch

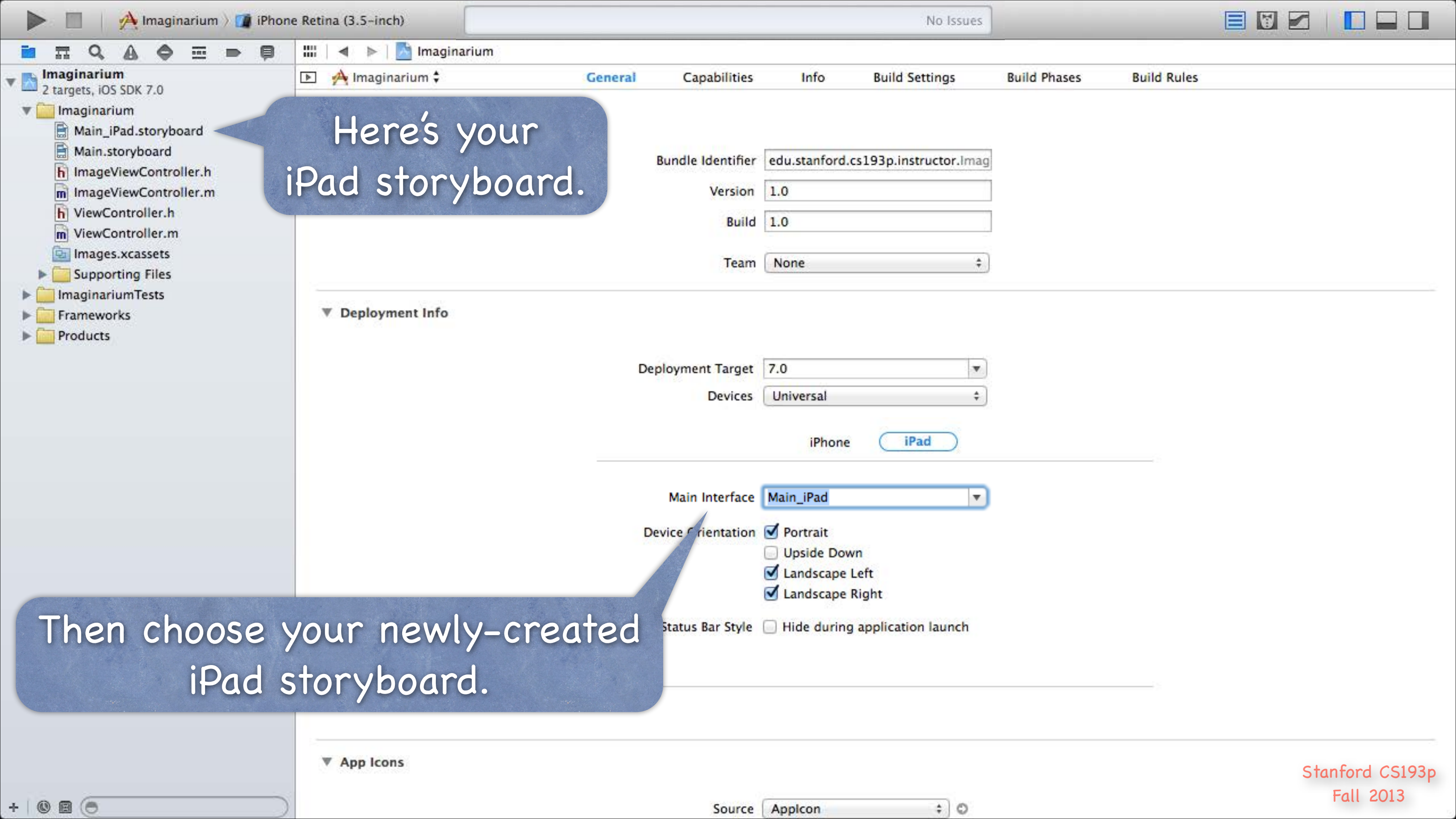
App Icons

Source: Applcon



Here's your iPad storyboard.

To set it as the storyboard to use on iPad, click on iPad here.



Here's your iPad storyboard.

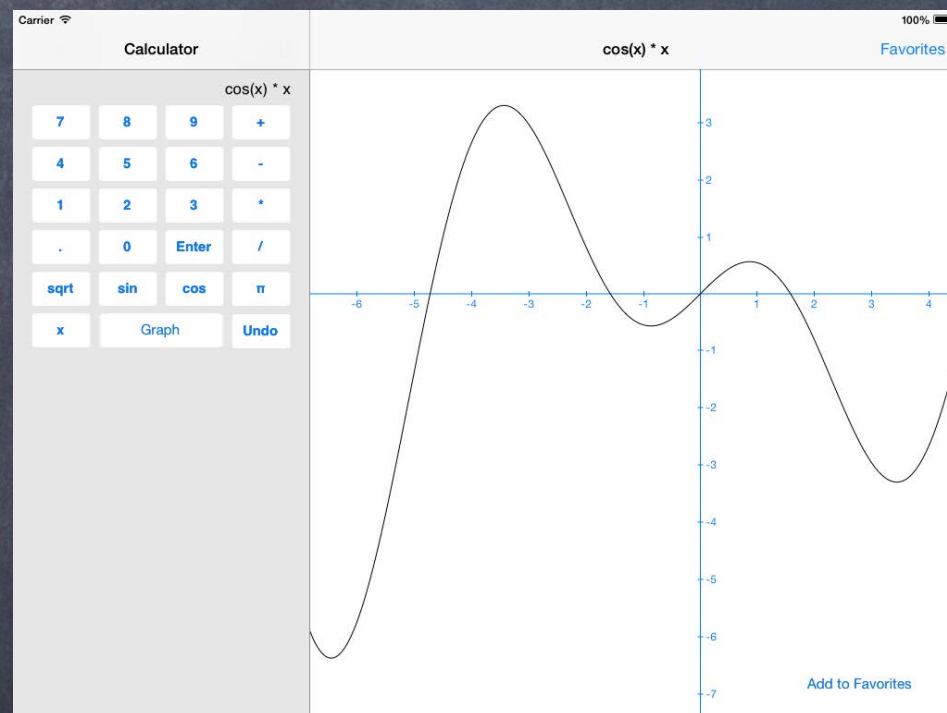
Then choose your newly-created iPad storyboard.

Universal Applications

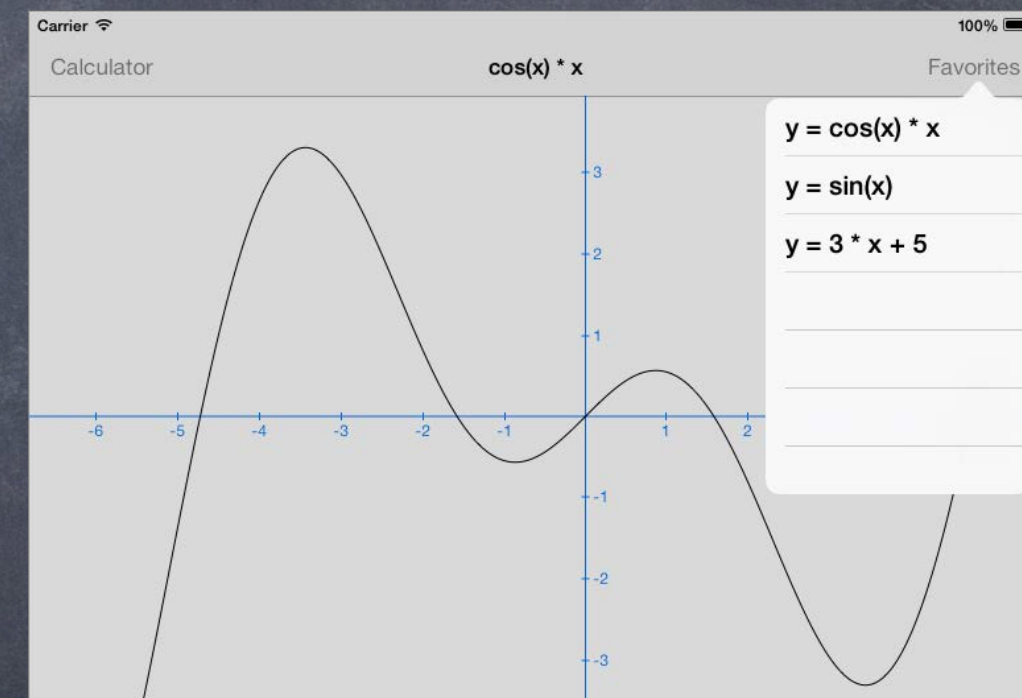
👁 iPad user-interface idioms

The iPad has more screen real estate, so it can present MVCs in a couple of other ways.

Split View



Popover



Universal Applications

- How do I figure out "am I on an iPad?"

```
BOOL iPad = ([[UIDevice currentDevice] userInterfaceIdiom] == UIUserInterfaceIdiomPad);
```

Use this sparingly!

Checking other things (like whether you are in a split view or popover) might be better design.

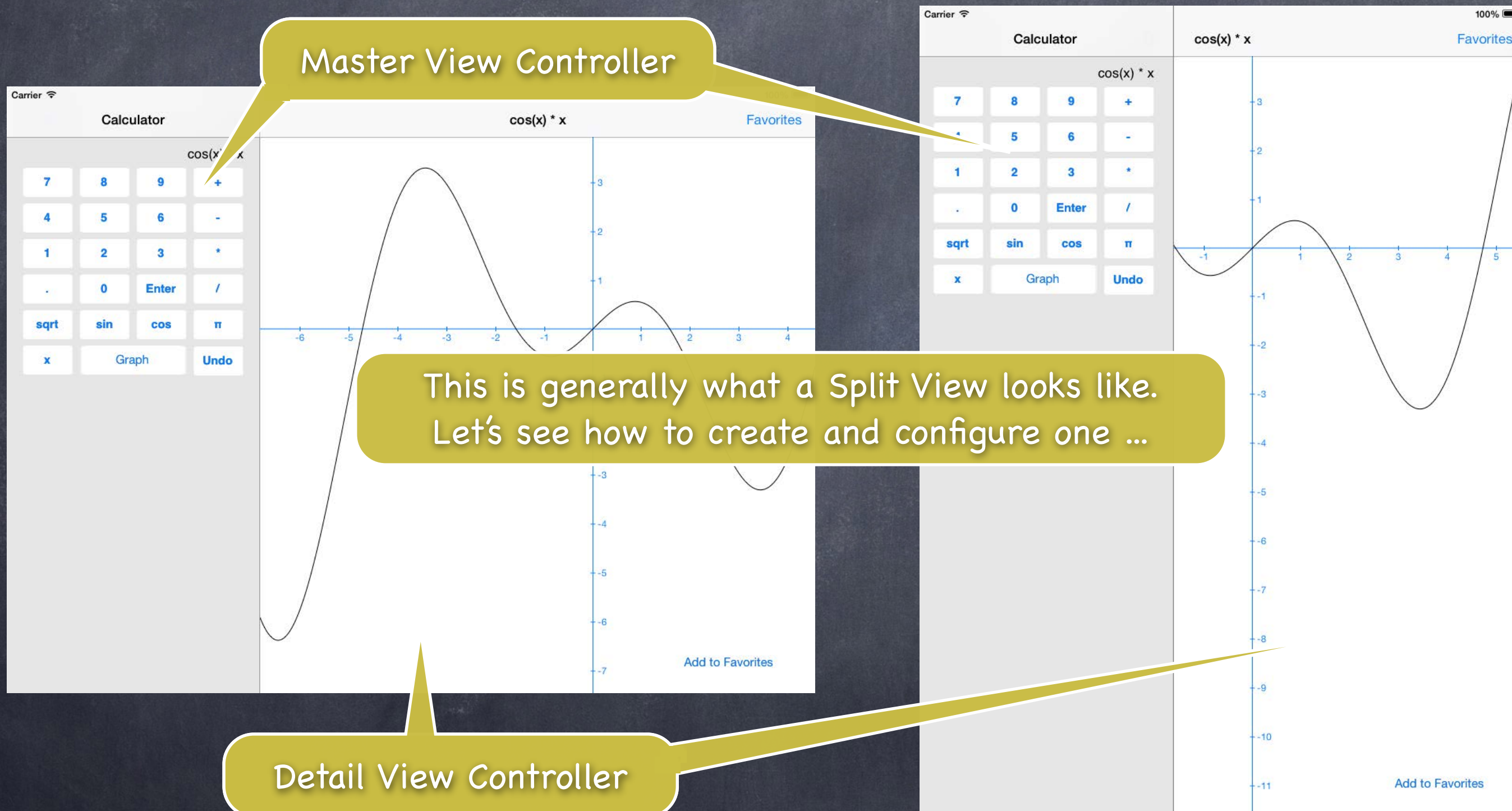
Or maybe check to see if your MVC or another MVC are "on screen" now

(because with more screen real estate, iPad can often have multiple MVCs showing at once).

Remember this? `if (self.view.window == nil) { /* I am not on screen right now */ }`

UISplitViewController

Master View Controller



Detail View Controller

UISplitViewController

- Designed to be at the top-level of your storyboard

Don't try to put it inside a tab bar controller or navigation controller!

But you can put either of those inside either side of the split view.

- Simple to add to your storyboard

Just drag it out (and usually delete the "free" Master and Detail it gives you).

If you don't see a Split View in your Object Palette, then you're not editing an iPad storyboard.

Then ctrl-drag to each of the two sides (Master and Detail) of the split view.

UISplitViewController

• Accessing the Master and Detail MVCs from code

All UIViewController know the UISplitViewController they are contained in (if in one):

```
@property (strong) UISplitViewController *splitViewController;
```

e.g. if (self.splitViewController) { /* I am in a UISplitViewController */ }

The UISplitViewController has a property which is an array containing Master and Detail:

```
@property (copy) NSArray *viewControllers; // index 0 is Master, 1 is Detail
```

This property is not readonly, so you can change the Master & Detail of a Split View.

The array is immutable though, so you must set both Master & Detail together at once.

Usually you set this by ctrl-dragging in your storyboard though, not in code.

e.g. A Master VC wants to get ahold of the Detail VC of the Split View both are in ...

```
UIViewController *detailVC = self.splitViewController.viewControllers[1];
```

If the Master VC is not in a Split View, this would nicely return nil.

UISplitViewControllerDelegate

- UISplitViewController requires its delegate to be set
Or, at least, if you don't set it, then in portrait mode, the Master will be inaccessible.
`@property (assign) id <UISplitViewControllerDelegate> delegate;`
By the way, "assign" above is like "weak" except it doesn't set to nil when it leaves the heap!
Seems dangerous (and it can be), except that a Controller is almost always the delegate.
And a Controller is unlikely to leave the heap before elements of the View do.
- You must set this delegate very early!
Probably in `awakeFromNib`.
e.g., UISplitViewController starts sending its delegate methods way before `viewDidLoad`.
And then, unfortunately, when its delegate methods get sent to you, your outlets aren't set yet!
This can make being a UISplitViewController's delegate a real pain.
- What is the delegate's responsibility?
To control how the Master and Detail are presented when device rotation occurs ...

UISplitViewControllerDelegate

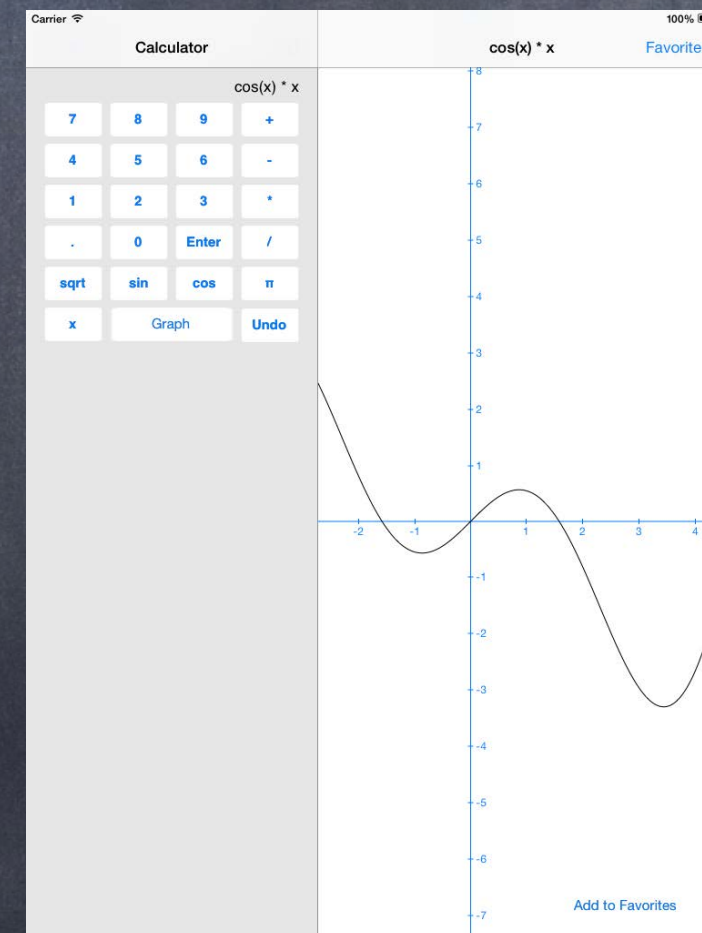
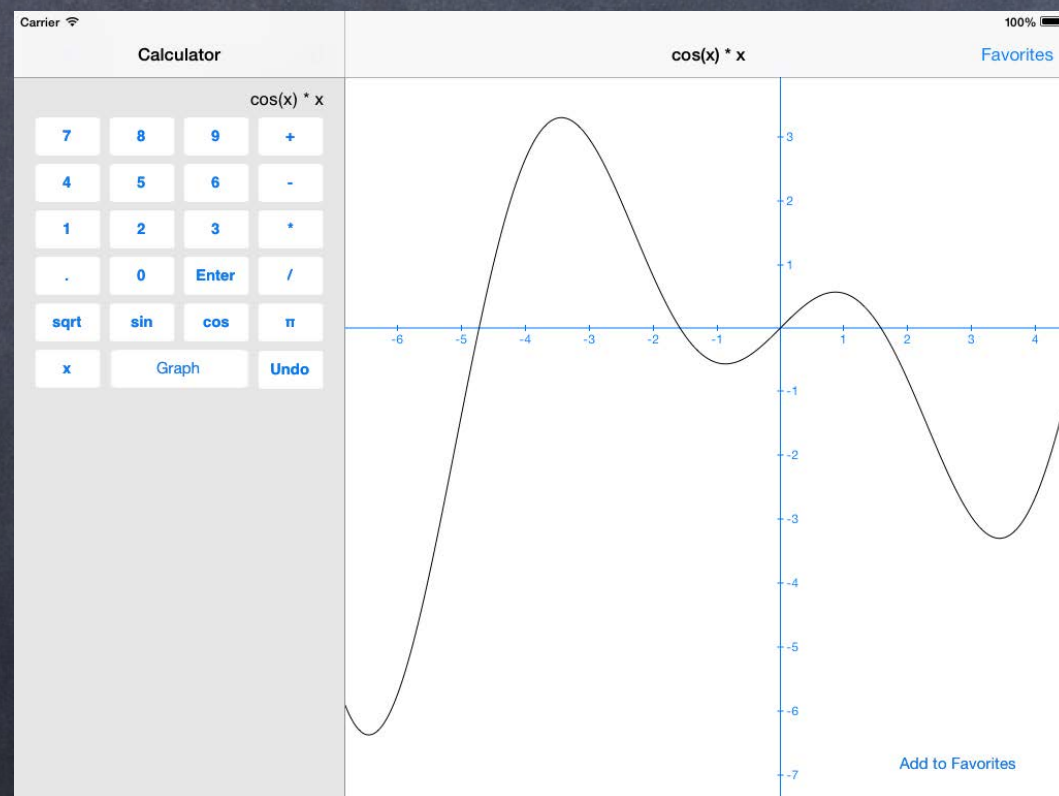
- Never hide the left side (Master) behind a bar button

```
- (BOOL)splitViewController:(UISplitViewController *)sender  
    shouldHideViewController:(UIViewController *)master  
    inOrientation:(UIInterfaceOrientation)orientation
```

```
{
```

```
    return NO; // never hide it
```

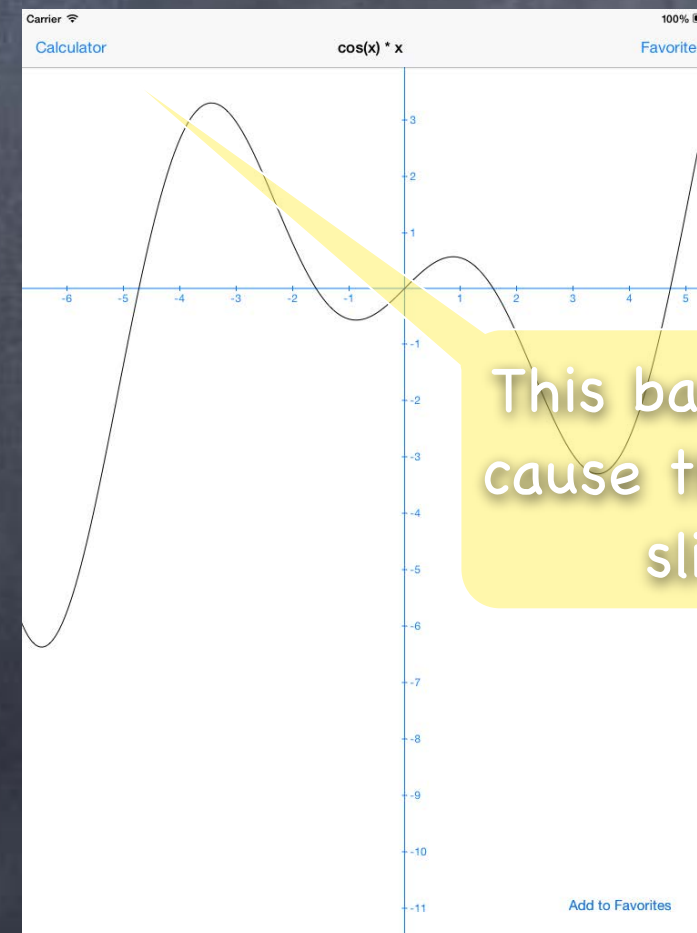
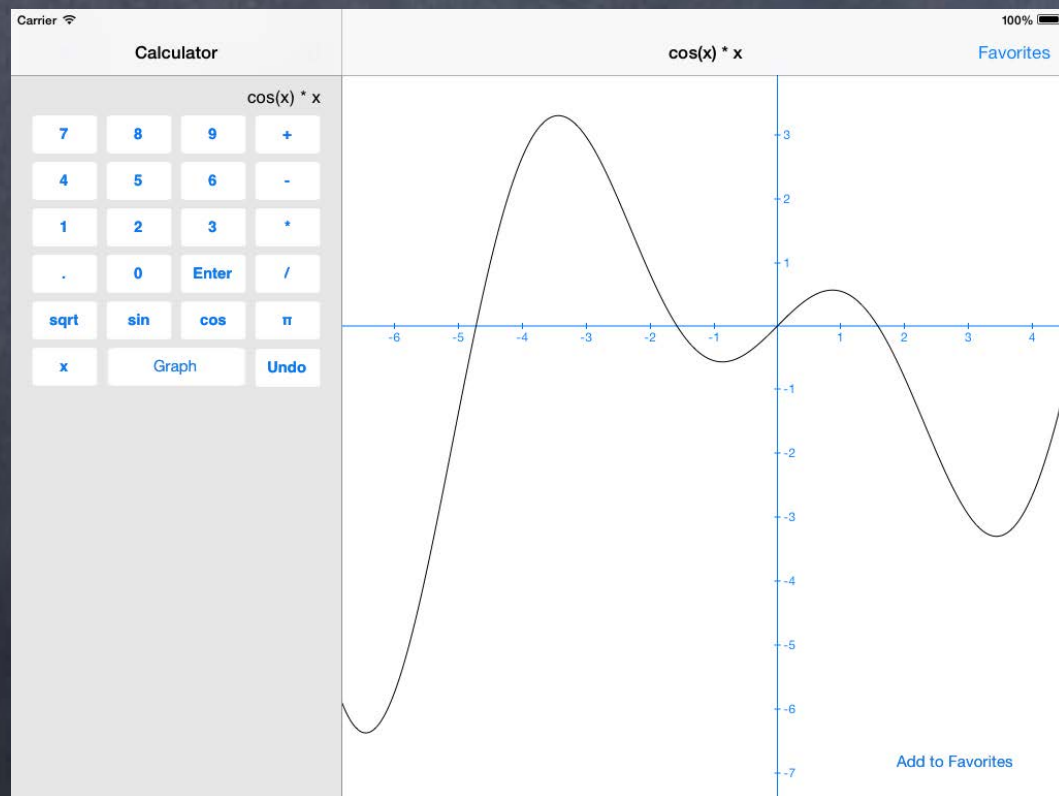
```
}
```



UISplitViewControllerDelegate

- Hide Master in portrait orientation only (the default)

```
- (BOOL)splitViewController:(UISplitViewController *)sender  
    shouldHideViewController:(UIViewController *)master  
    inOrientation:(UIInterfaceOrientation)orientation  
{  
    return UIInterfaceOrientationIsPortrait(orientation);  
}
```



UISplitViewControllerDelegate

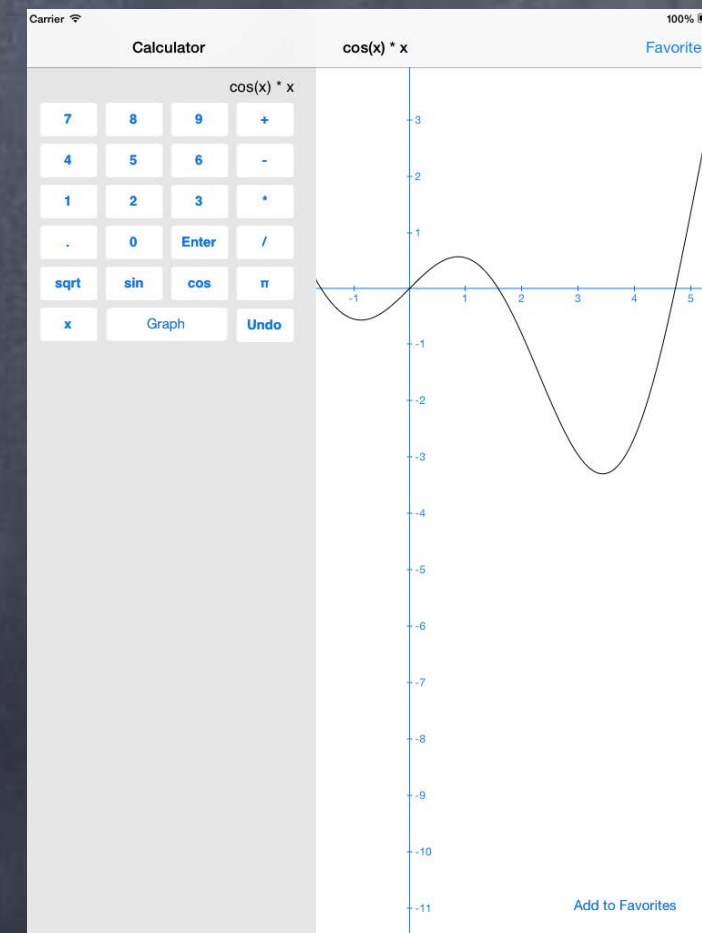
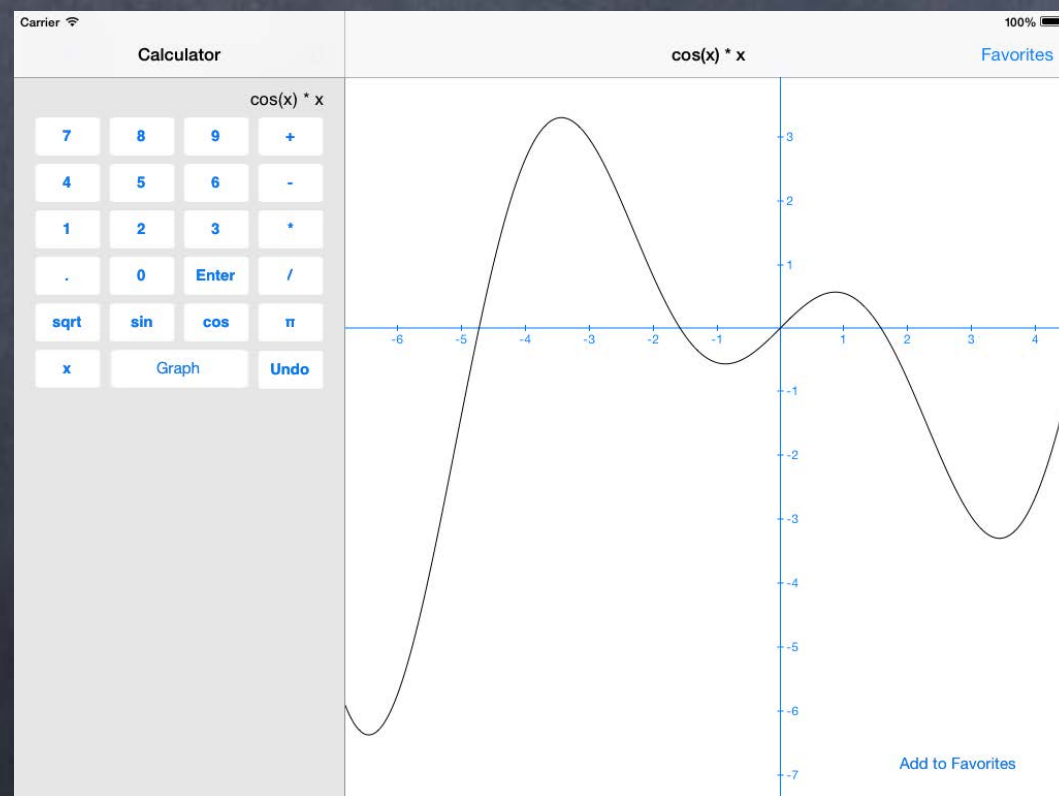
- Hide Master in portrait orientation only (the default)

```
- (BOOL)splitViewController:(UISplitViewController *)sender  
    shouldHideViewController:(UIViewController *)master  
    inOrientation:(UIInterfaceOrientation)orientation
```

```
{
```

```
    return UIInterfaceOrientationIsPortrait(orientation);
```

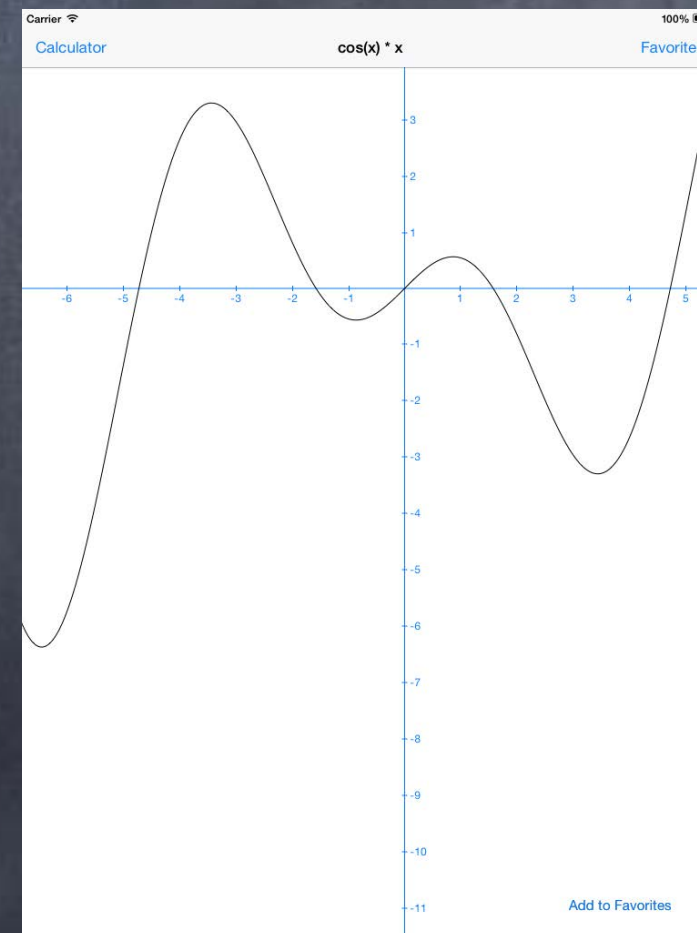
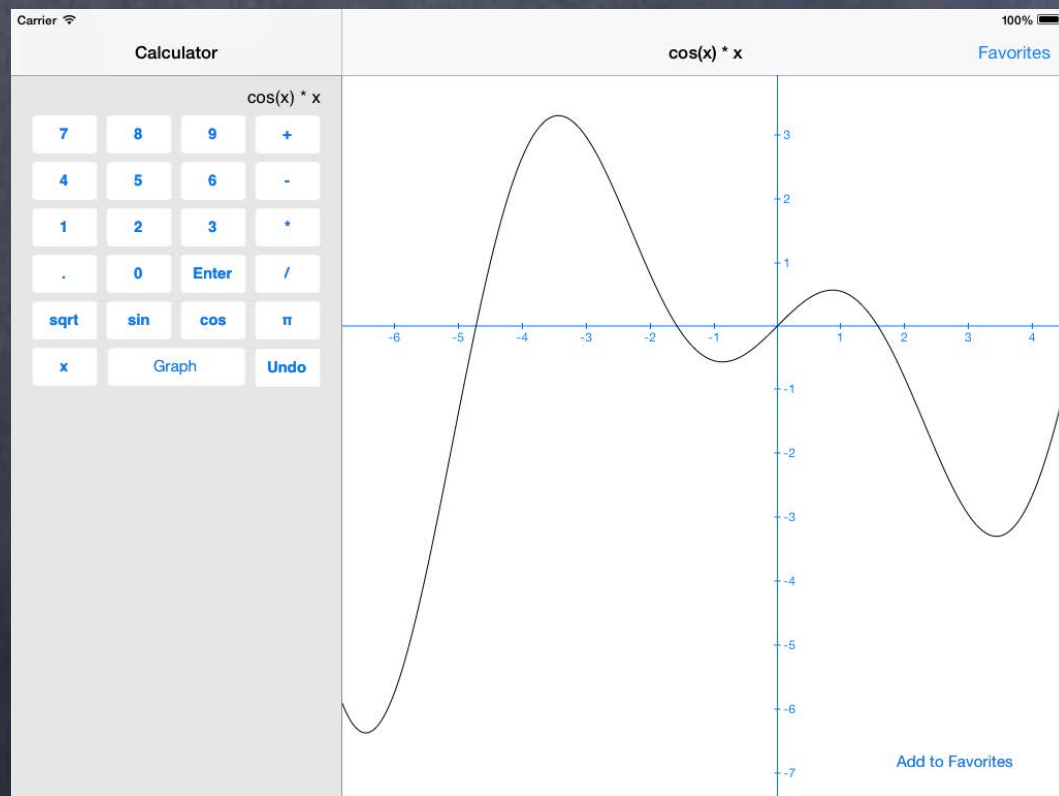
```
}
```



UISplitViewControllerDelegate

- Hide Master in portrait orientation only (the default)

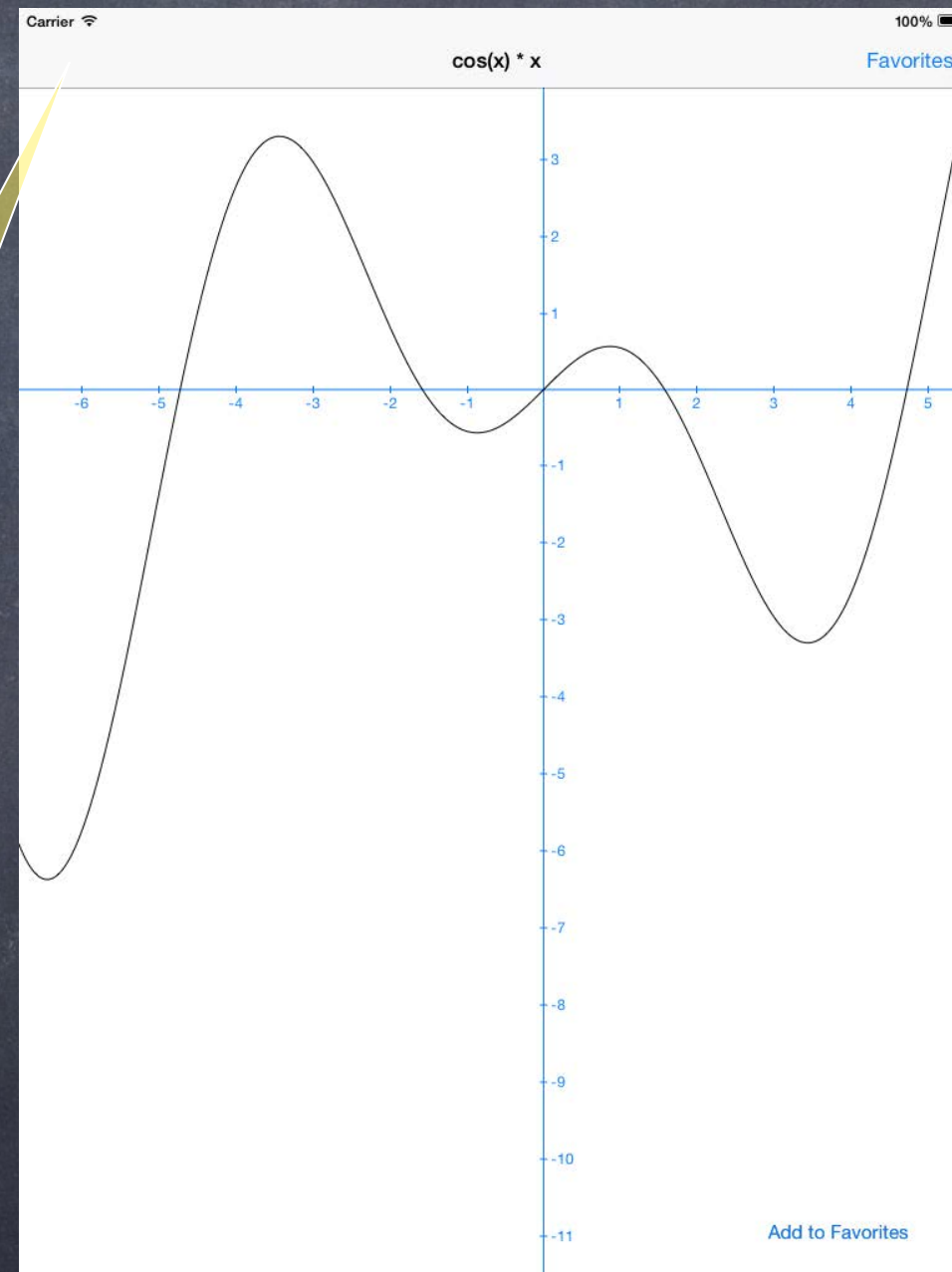
```
- (BOOL)splitViewController:(UISplitViewController *)sender  
    shouldHideViewController:(UIViewController *)master  
    inOrientation:(UIInterfaceOrientation)orientation  
{  
    return UIInterfaceOrientationIsPortrait(orientation);  
}
```



UISplitViewControllerDelegate

- If you forget to set the delegate, you'll get this ...

No button to slide the Master on screen!



UISplitViewControllerDelegate

- Split View helps you by providing that bar button

This gets called in your delegate when the master gets hidden ...

```
- (void)splitViewController:(UISplitViewController *)sender
    willHideViewController:(UIViewController *)master
    withBarButtonItem:(UIBarButtonItem *)barButtonItem
    forPopoverController:(UIPopoverController *)popover
{
    barButtonItem.title = master.title;
    // this next line would only work in the Detail
    // and only if it was in a UINavigationController
    self.navigationItem.leftBarButtonItem = barButtonItem;
}
```

See? You are being provided the bar button item. You just need to put it on screen somewhere.

UISplitViewControllerDelegate

- When it's time for the bar button to go away ...

This gets called in your delegate when the master reappears ...

```
- (void)splitViewController:(UISplitViewController *)sender
    willShowViewController:(UIViewController *)master
    invalidatingBarButtonItem:(UIBarButtonItem *)barButtonItem
{
    // this next line would only work in the Detail
    // and only if it was in a UINavigationController
    self.navigationItem.leftBarButtonItem = nil;
}
```

UISplitViewController

- Updating the Detail when the Master is touched?

There are 2 choices for how to do this: Target/Action or Replace Segue

- Target/Action

Example (code in the Master view controller) ...

```
- (IBAction)doit
{
    id detailViewController = self.splitViewController.viewControllers[1];
    [detailViewController setSomeProperty:...]; // might want some Introspection first
}
```

- Replace Segue (entirely replaces the Detail view controller)

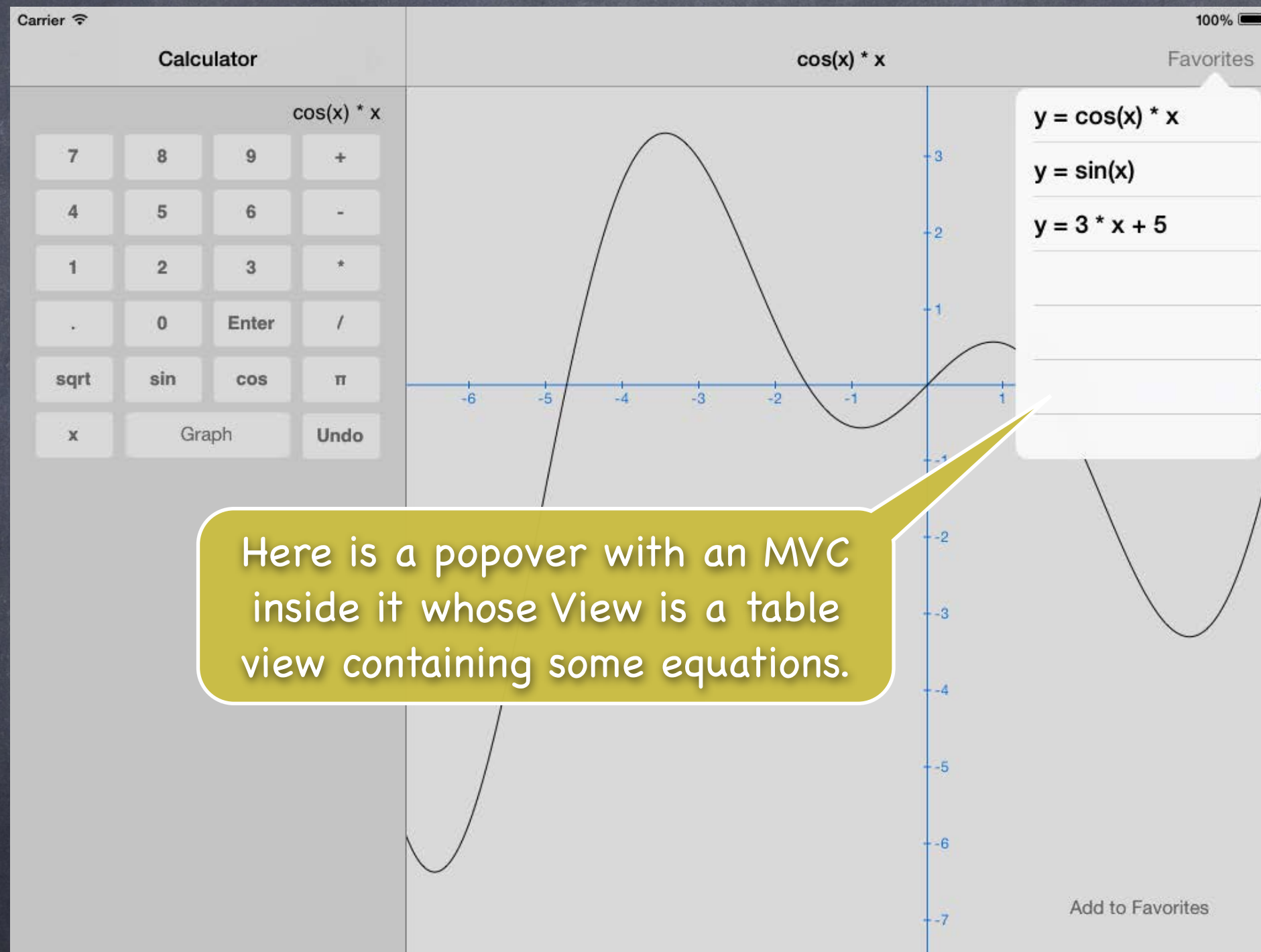
Remember, segues always instantiate a view controller (split view stops pointing to old one).

Can Replace either side, but much more common to replace the right side (since it's the "detail").

Be careful! You might lose the UIBarButtonItem used for revealing the hidden Master!

(you'd need to be sure to put it back into the newly instantiated view controller)

Popovers



Popovers

- `UIPopoverController` is not, itself, a `UIViewController`
Instead it has a `@property` that holds the `UIViewController` that is inside it ...
`@property (nonatomic, strong) UIViewController *contentViewController;`
This is usually wired up in a storyboard ...

Popovers

• Creating a Popover Segue in your Storyboard

Just drag from the UI element you want to cause the popover to the scene you want to pop up.

In your `prepareForSegue:sender:`, the argument will be `isKindOfClass:UIStoryboardPopoverSegue`. And `UIStoryboardPopoverSegue` has a `@property` you can use to get the `UIPopoverController`:

```
- (UIPopoverController *)popoverController;
```

Example:

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender  
{  
    if ([segue isKindOfClass:[UIStoryboardPopoverSegue class]]) {  
        UIPopoverController *popoverController =  
            ((UIStoryboardPopoverSegue *)segue).popoverController;  
        . . .  
    }  
}
```

Popover

• You can also present a popover from code

Popover has a little arrow that points to what (rectangle or button) brought it up.

You can specify which directions it is valid to point (and thus where the popover will pop up).

```
UIPopoverController *popover =
```

```
    [[UIPopoverController alloc] initWithContentViewController:myPoppedUpVC];
```

```
[popover presentPopoverFromRect:(CGRect)aRect // little arrow points to aRect in view's coords
```

```
    inView:(UIView *)view
```

```
    permittedArrowDirections:(UIPopoverArrowDirection)direction
```

```
    animated:(BOOL)flag];
```

... or (points to a bar button item) ...

```
[popover presentPopoverFromBarButtonItem:(UIBarButtonItem *)barButtonItem
```

```
    permittedArrowDirections:(UIPopoverArrowDirection)direction
```

```
    animated:(BOOL)flag;
```

* the "casts" on the arguments above are only for educational purposes, they are not required

Popover

- But don't forget to keep a strong pointer to the popover controller!

Example: a target/action method attached to a UIBarButtonItem that presents a popover ...

```
- (IBAction)presentPopover:(UIBarButtonItem *)item
{
    UIPopoverController *pop = [[UIPopoverController alloc] initWithViewController:vc];
    [pop presentPopoverFromBarButtonItem:item ...];
}
```

The above is bad because there is no strong pointer kept to the UIPopoverController!

Popover

- But don't forget to keep a strong pointer to the popover controller!

Example: a target/action method attached to a UIBarButtonItem that presents a popover ...

```
- (IBAction)presentPopover:(UIBarButtonItem *)item
```

```
{
```

```
    if (!self.popover) {
```

```
        self.popover = [[UIPopoverController alloc] initWithViewController:vc];
```

```
        [self.popover presentPopoverFromBarButtonItem:item ...];
```

```
    }
```

```
}
```

```
// then set self.popover to nil when the popover is dismissed at a later time
```

Speaking of which ... how do we dismiss a popover (or find out that the user has dismissed it)?

Popover

- The user dismisses a popover by touching outside of it
Unless the user touches in one of the views in this array property in UIPopoverController:
`@property (copy) NSArray *passthroughViews;`
- Dismissing a popover from code
UIPopoverController method:
– `(void)dismissPopoverAnimated:(BOOL)animated;`
- Finding out that the user dismissed the popover
UIPopoverController has a delegate too and it will be sent this message:
– `(void)popoverControllerDidDismissPopover:(UIPopoverController *)sender;`
This is only sent if the user dismisses the popover.

Demo

👁 Shutterbug

UITableView

Flickr

Universal Application

UISplitViewController

UIRefreshControl

GCD

No Popover, sorry, but you will not be asked to do that in your homework assignment.

Coming Up

- Homework

Due next Wednesday.

- Friday

Stanford Only Review Section

- Next Week

Core Data (Object-Oriented Database)

Maybe some Multitasking API