Stanford CS193p

Developing Applications for iOS

Fall 2013-14

Today

- UITableView

  Data source-driven vertical list of views.

- iPad

  Device-specific UI idioms.

- Demo

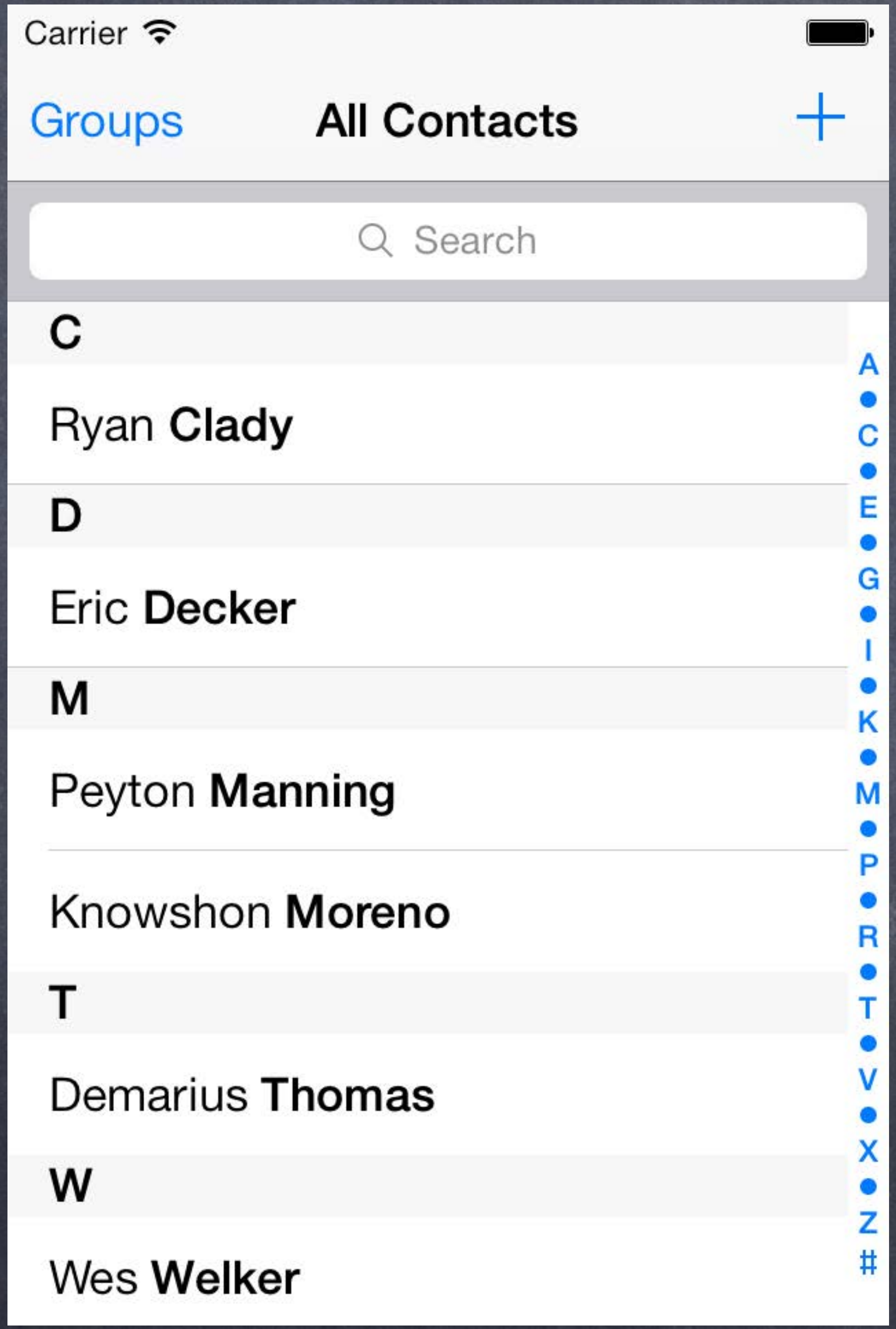
  Shutterbug

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


Carrier

Groups All Contacts +

Search

C

Ryan Clady

D

Eric Decker

M

Peyton Manning

Knowshon Moreno

T

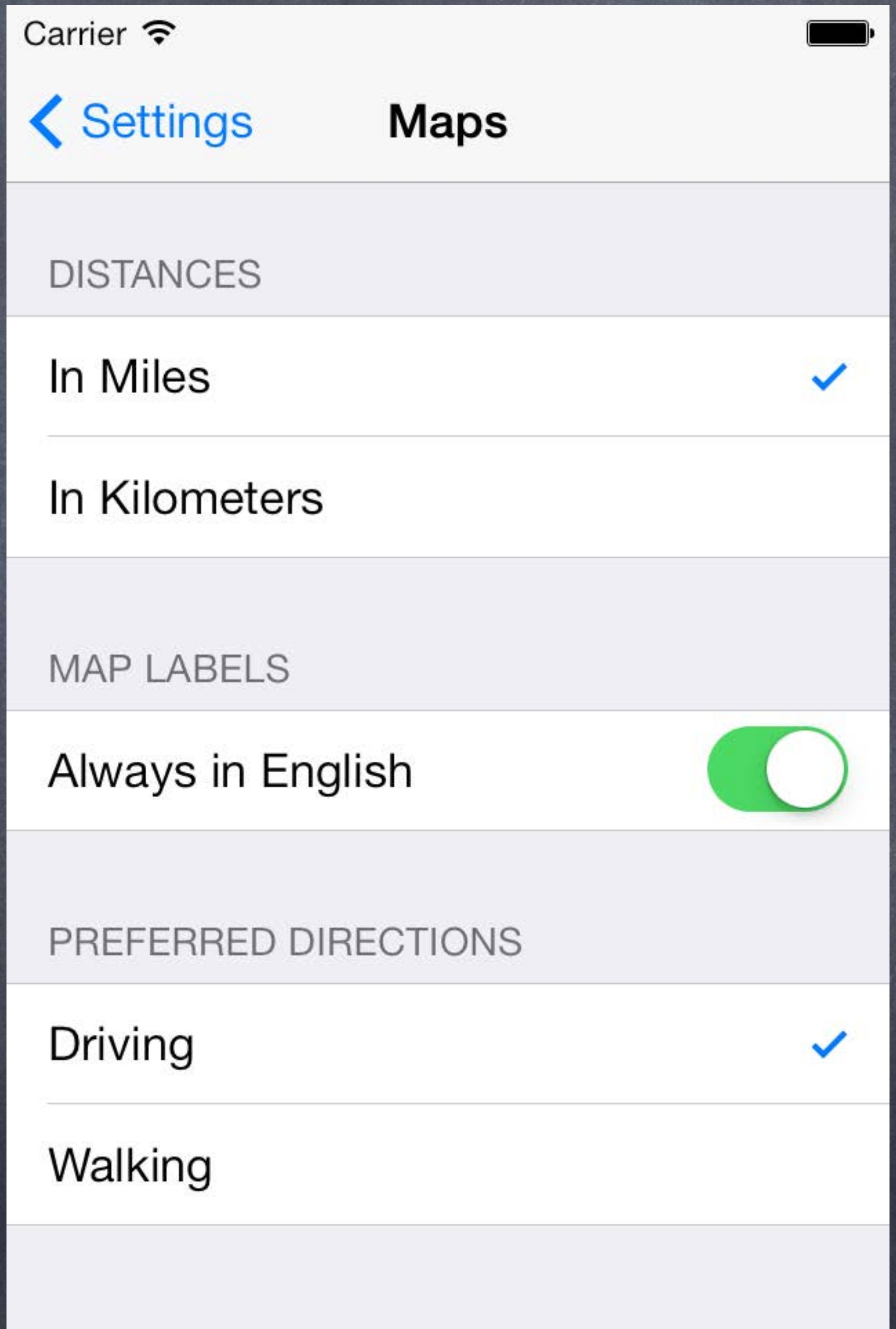
Demarius Thomas

W

Wes Welker

Dynamic (List) & Plain (ungrouped)

UITableViewStyleGrouped


Carrier

Settings Maps

DISTANCES

In Miles ✓

In Kilometers

MAP LABELS

Always in English

PREFERRED DIRECTIONS

Driving ✓

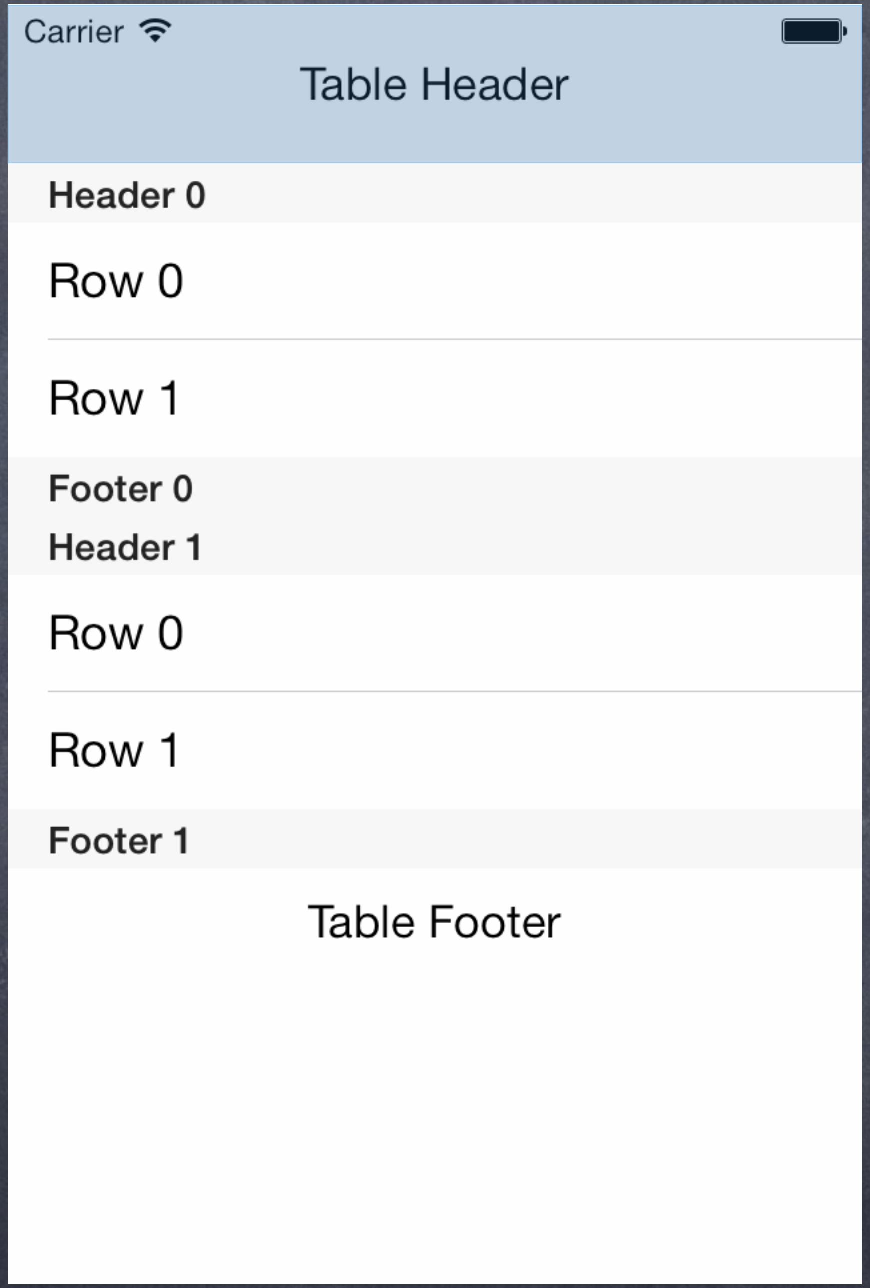
Walking

Static & Grouped

UITableView

Plain Style

Table Header



Carrier

Table Header

Header 0

Row 0

Row 1

Footer 0

Header 1

Row 0

Row 1

Footer 1

Table Footer

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

UITableView

Plain Style

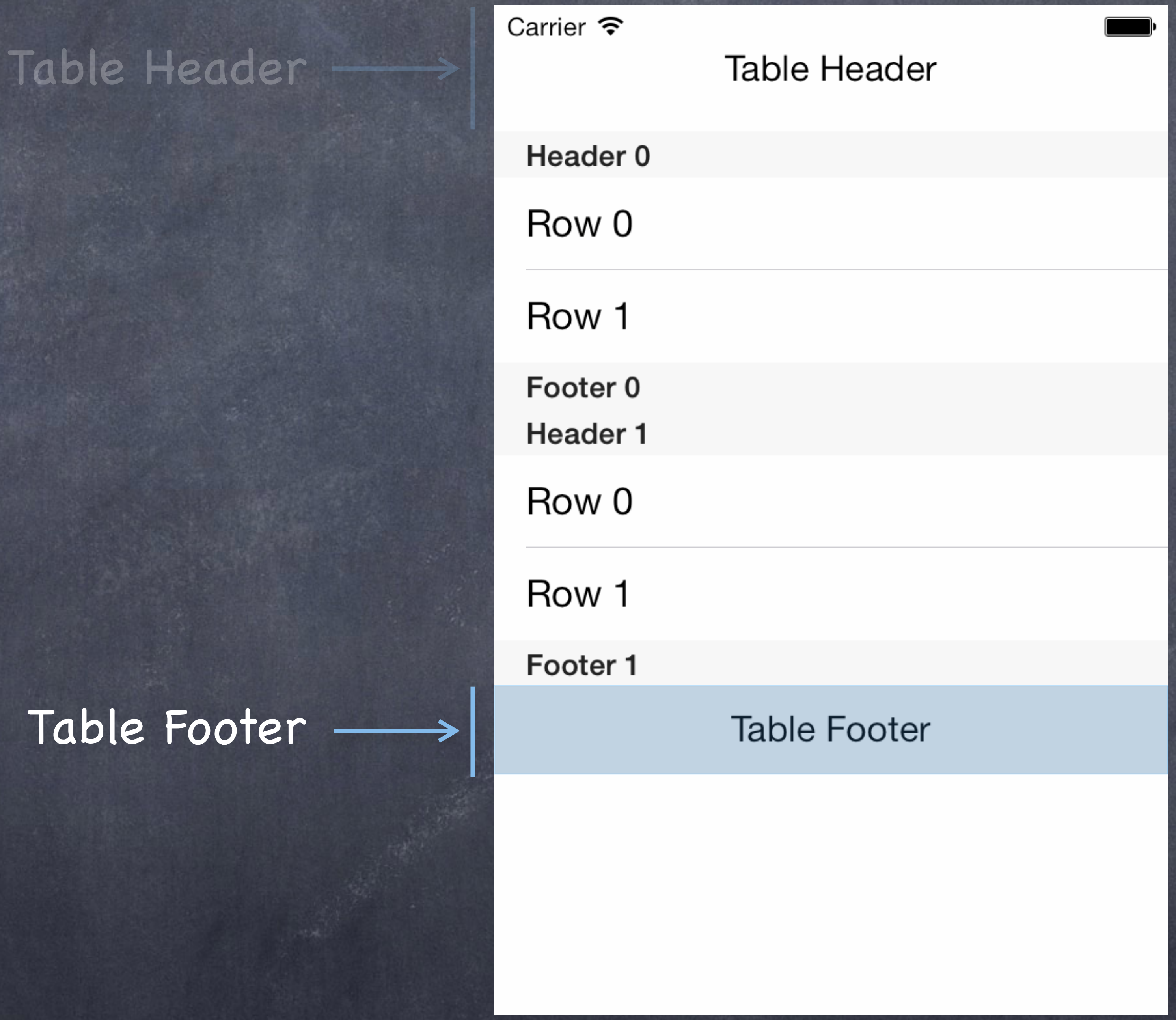


Table Header

Table Footer

Carrier

Table Header

Header 0

Row 0

Row 1

Footer 0

Header 1

Row 0

Row 1

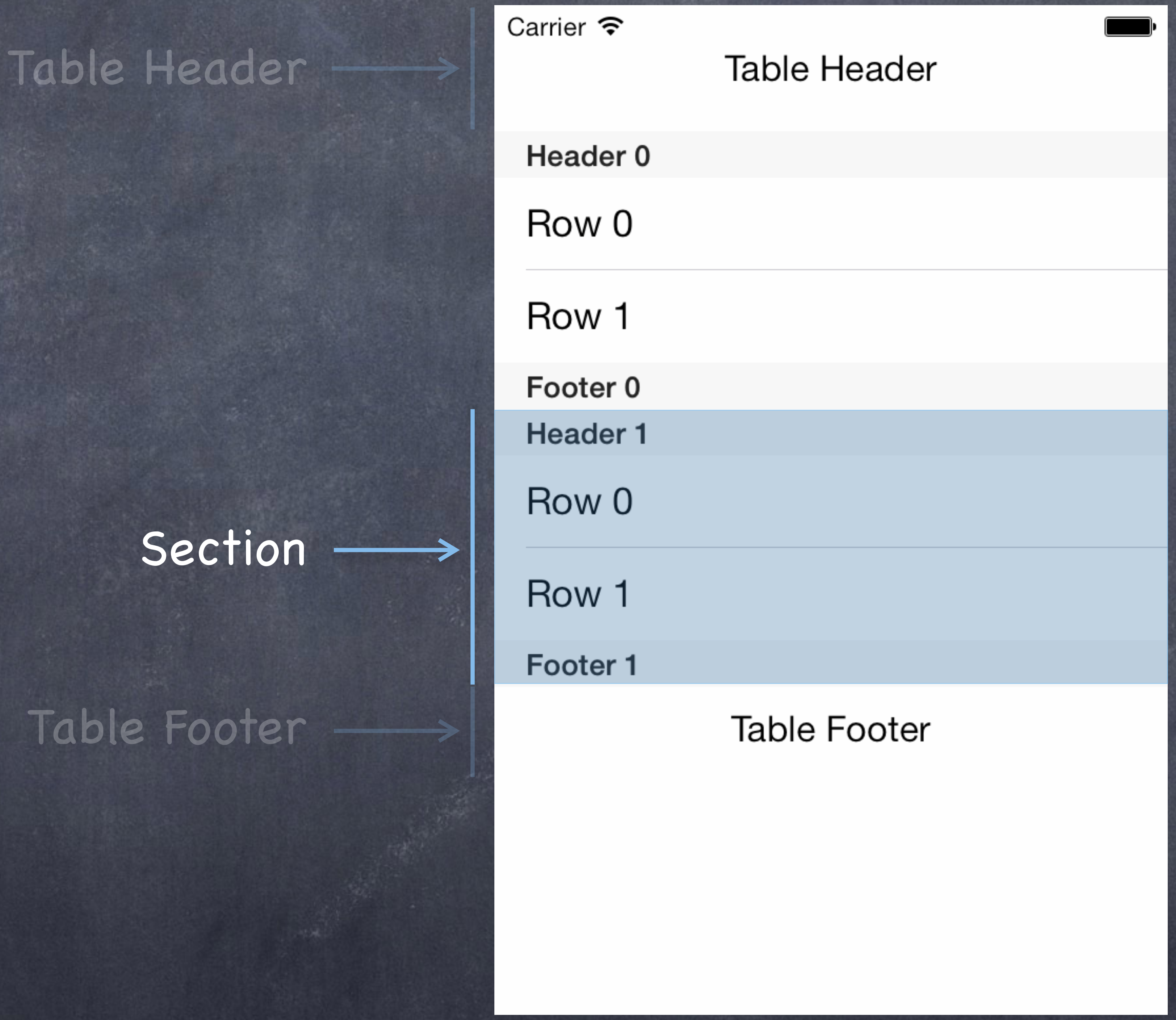
Footer 1

Table Footer

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

UITableView

Plain Style



UITableView

Plain Style

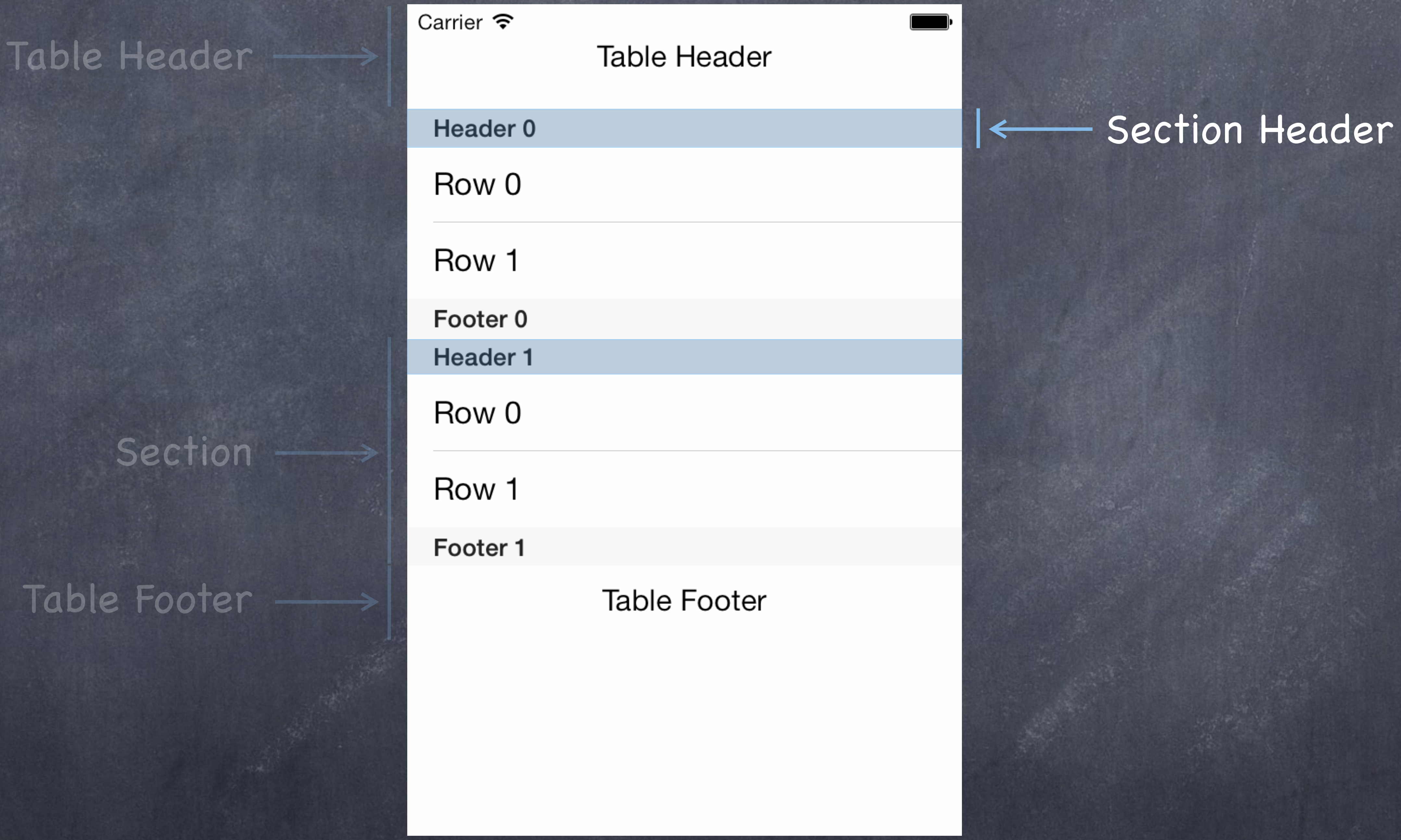


Table Header

Section

Table Footer

Section Header

Carrier

Table Header

Header 0

Row 0

Row 1

Footer 0

Header 1

Row 0

Row 1

Footer 1

Table Footer

UITableViewDataSource’s tableView:titleForHeaderInSection:

UITableView

Plain Style

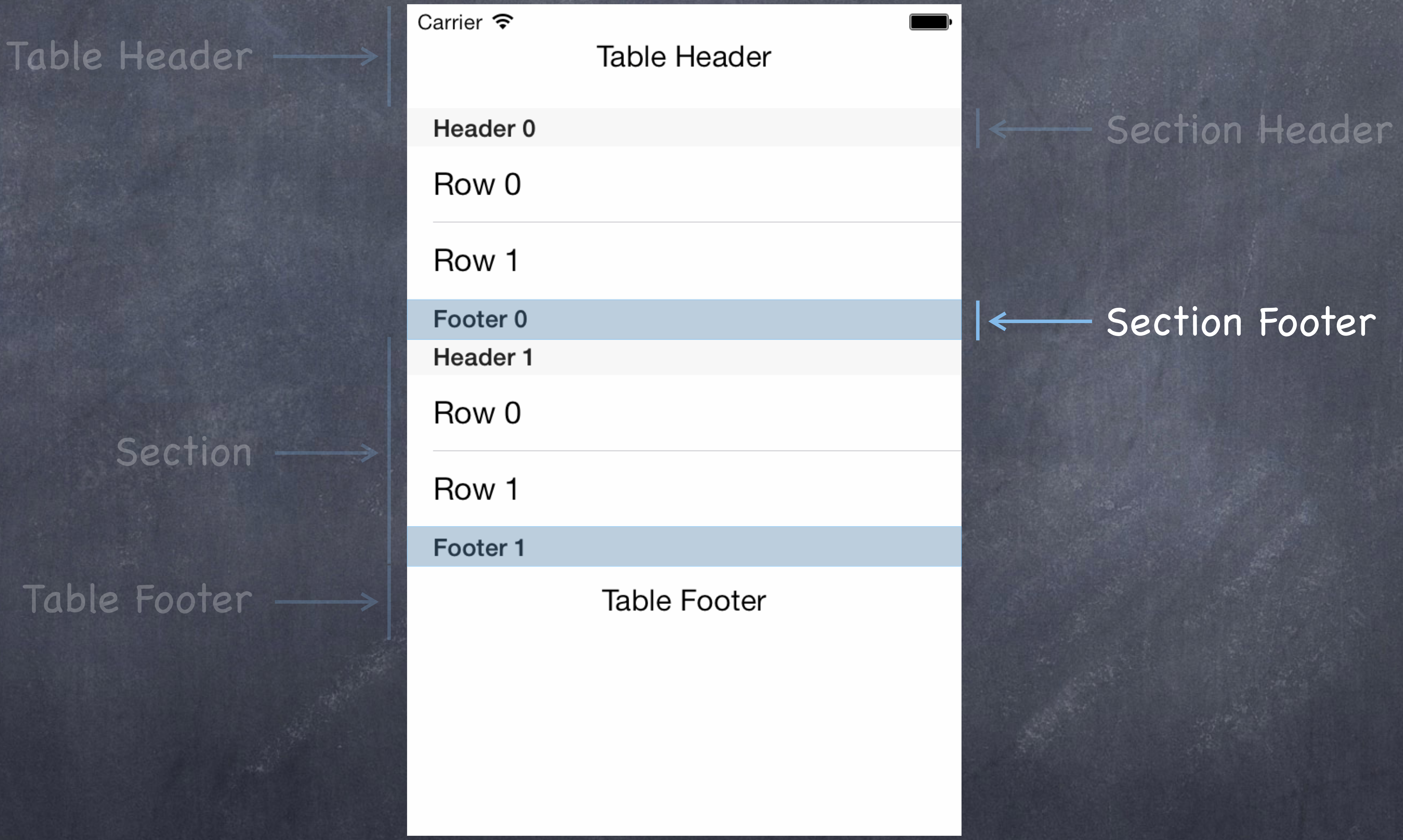


Table Header

Section Header

Section Footer

Section

Table Footer

Carrier

Table Header

Header 0

Row 0

Row 1

Footer 0

Header 1

Row 0

Row 1

Footer 1

Table Footer

`UITableViewDataSource’s tableView:titleForFooterInSection:`

UITableView

Plain Style

Table Header

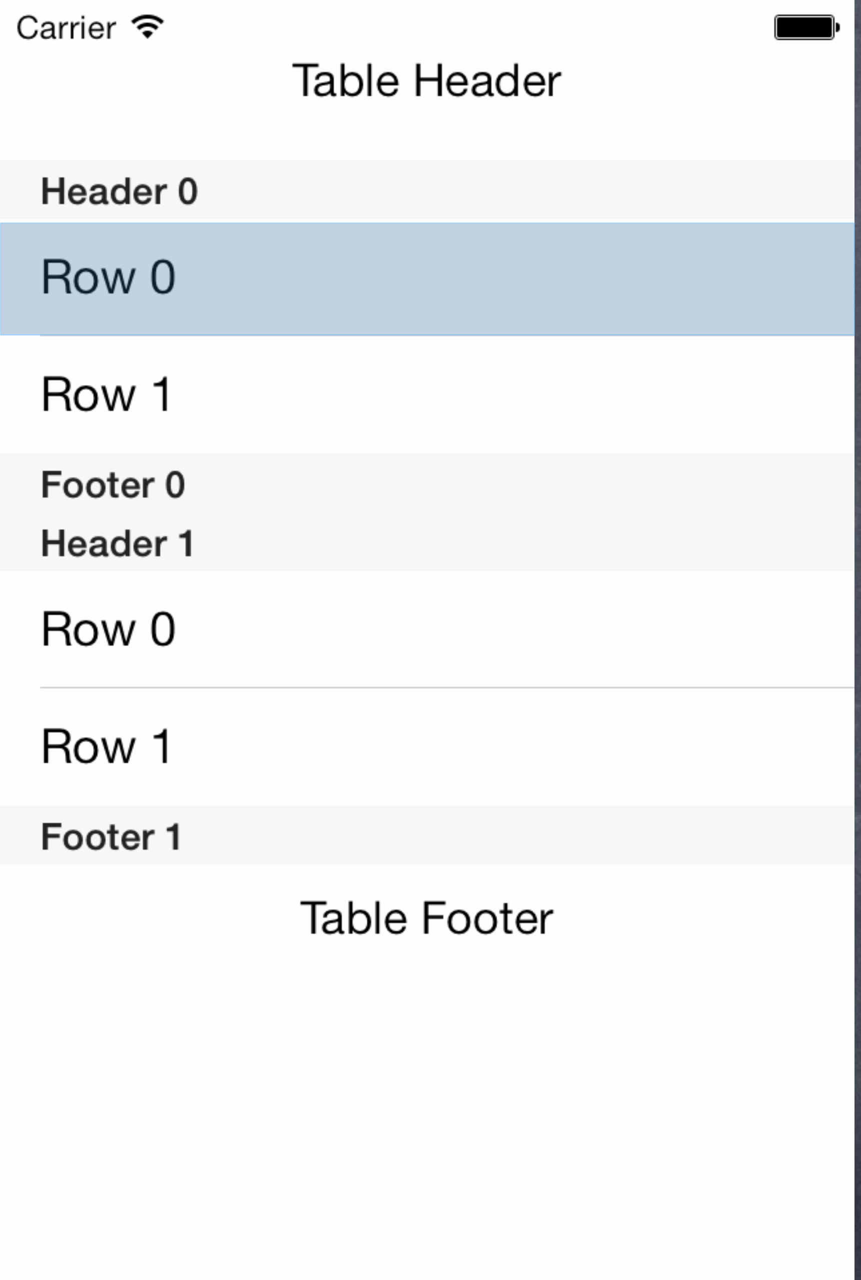
Section Header

Table Cell

Section Footer

Section

Table Footer



Carrier

Table Header

Header 0

Row 0

Row 1

Footer 0

Header 1

Row 0

Row 1

Footer 1

Table Footer

UITableViewDataSource’s tableView:cellForRowAtIndexPath:

UITableView

Plain Style

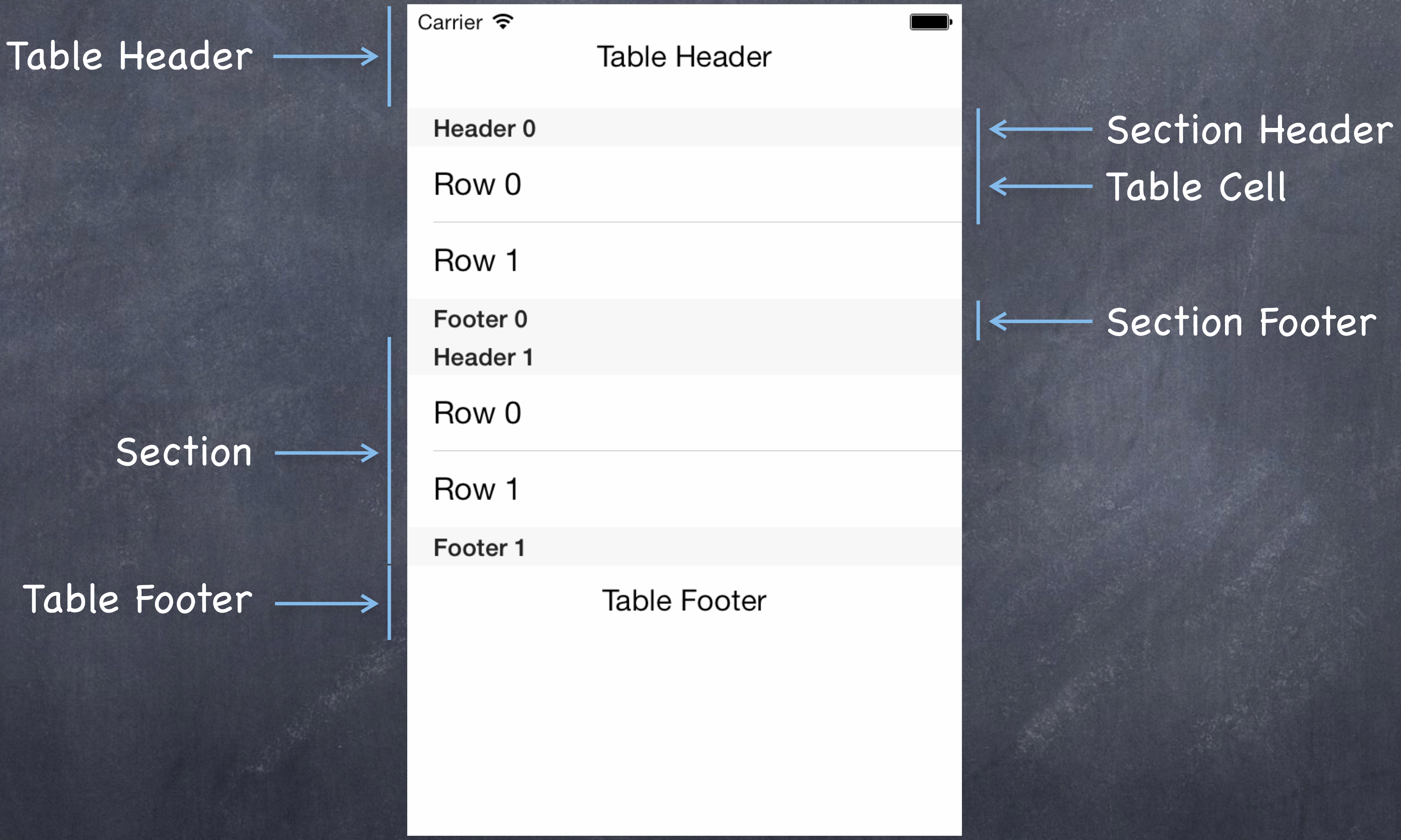


Table Header

Section

Table Footer

Carrier

Table Header

Header 0

Row 0

Row 1

Footer 0

Header 1

Row 0

Row 1

Footer 1

Table Footer

Section Header

Table Cell

Section Footer

UITableView

Grouped Style

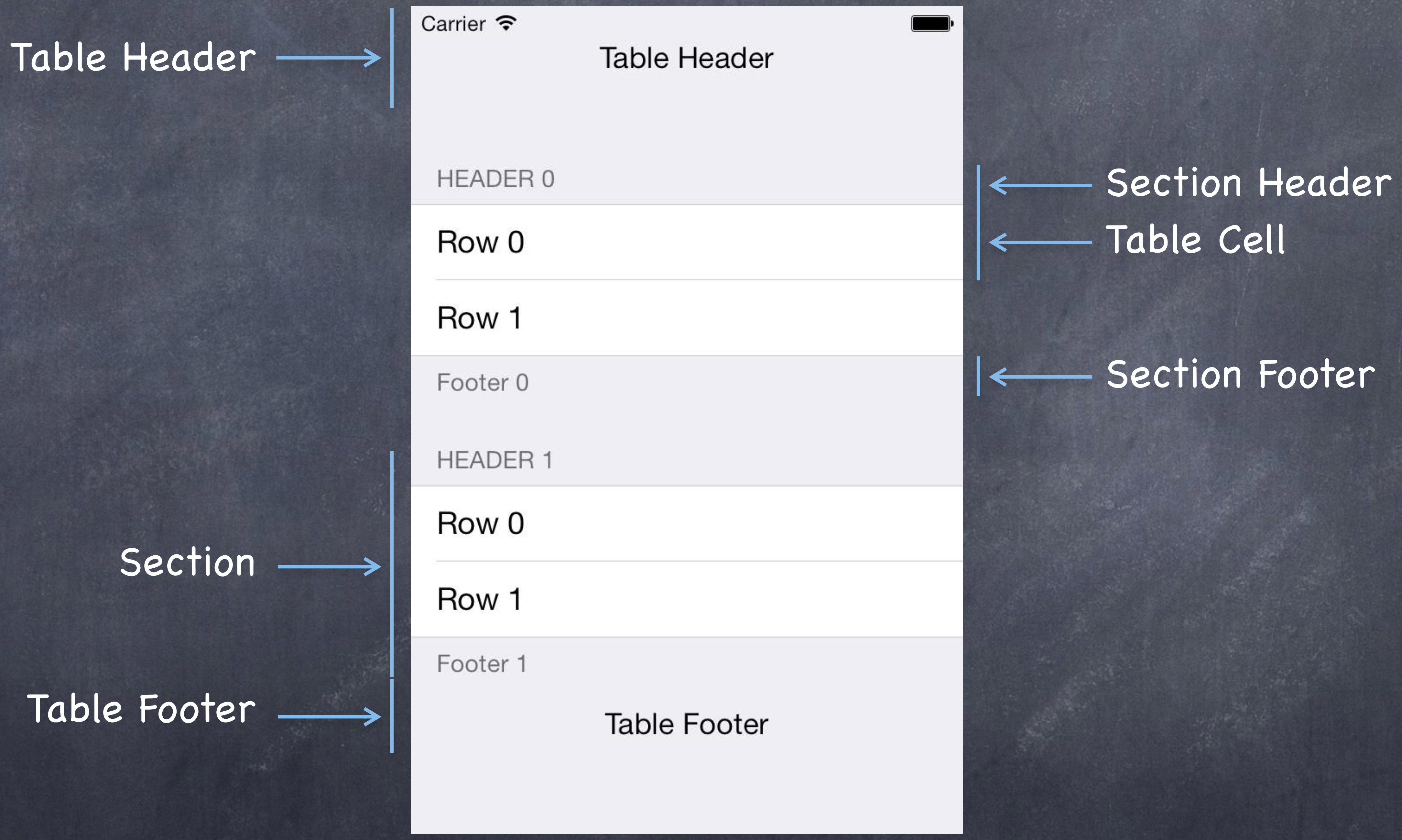


Table Header

Section

Table Footer

Section Header

Table Cell

Section Footer

Carrier

Table Header

HEADER 0

Row 0

Row 1

Footer 0

HEADER 1

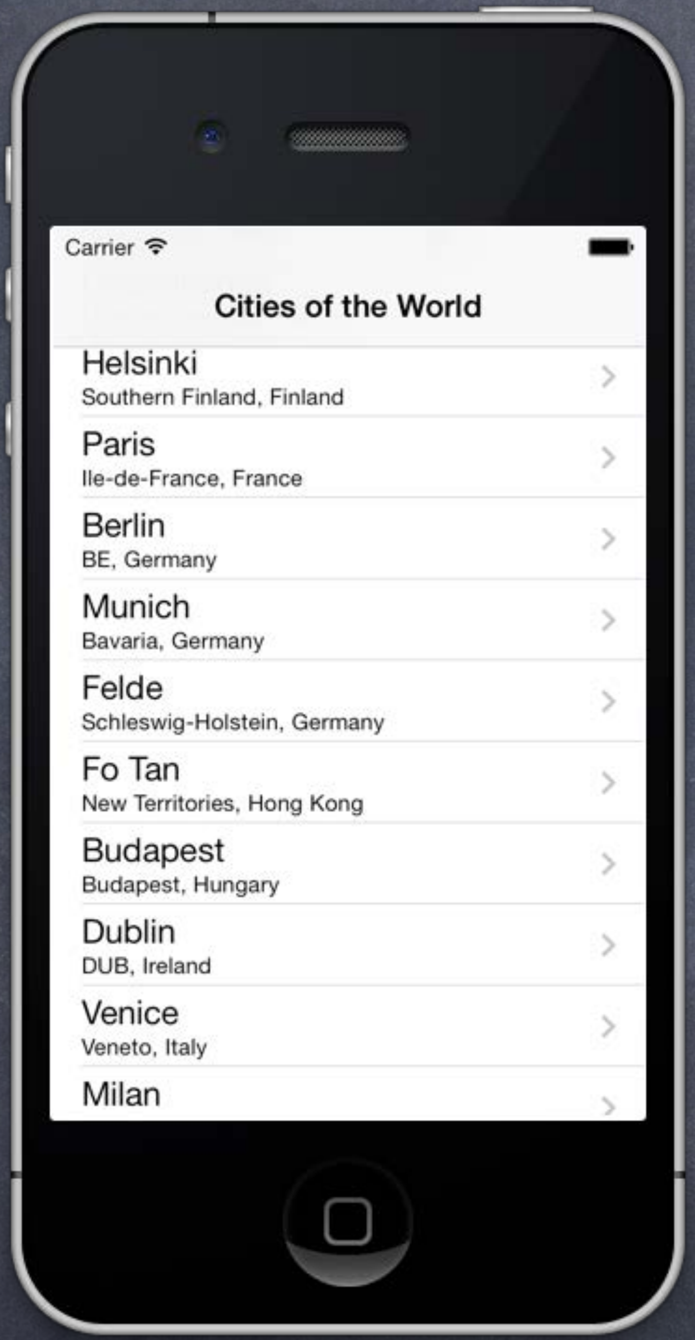
Row 0

Row 1

Footer 1

Table Footer

Sections or Not



Carrier

Cities of the World

Helsinki
Southern Finland, Finland

Paris
Ile-de-France, France

Berlin
BE, Germany

Munich
Bavaria, Germany

Felde
Schleswig-Holstein, Germany

Fo Tan
New Territories, Hong Kong

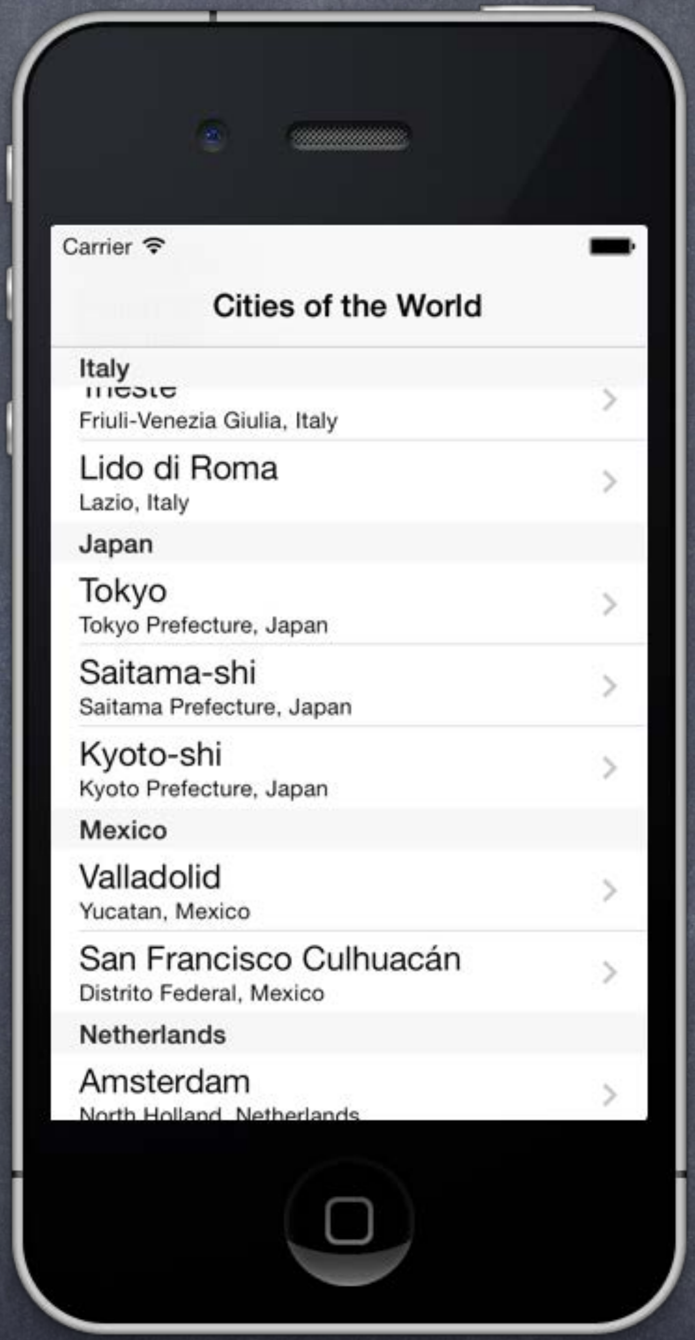
Budapest
Budapest, Hungary

Dublin
DUB, Ireland

Venice
Veneto, Italy

Milan

No Sections



Carrier

Cities of the World

Italy

Trieste
Friuli-Venezia Giulia, Italy

Lido di Roma
Lazio, Italy

Japan

Tokyo
Tokyo Prefecture, Japan

Saitama-shi
Saitama Prefecture, Japan

Kyoto-shi
Kyoto Prefecture, Japan

Mexico

Valladolid
Yucatan, Mexico

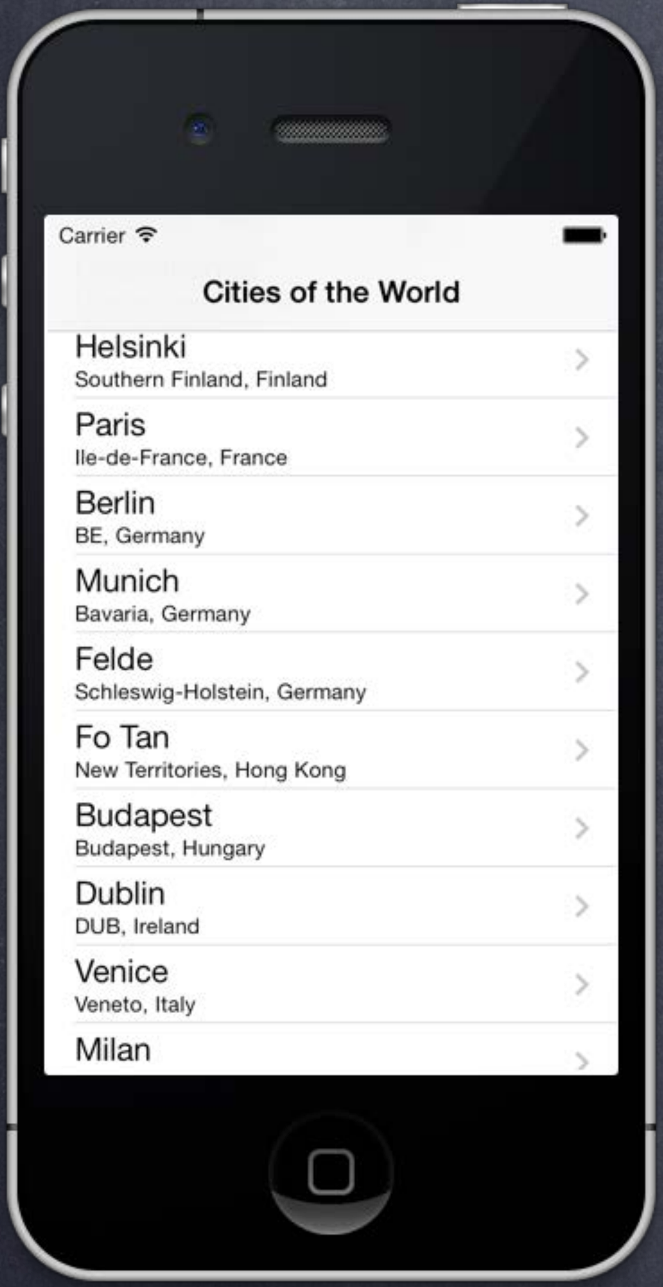
San Francisco Culhuacán
Distrito Federal, Mexico

Netherlands

Amsterdam
North Holland, Netherlands

Sections

Cell Type



Carrier

Cities of the World

Helsinki
Southern Finland, Finland

Paris
Ile-de-France, France

Berlin
BE, Germany

Munich
Bavaria, Germany

Felde
Schleswig-Holstein, Germany

Fo Tan
New Territories, Hong Kong

Budapest
Budapest, Hungary

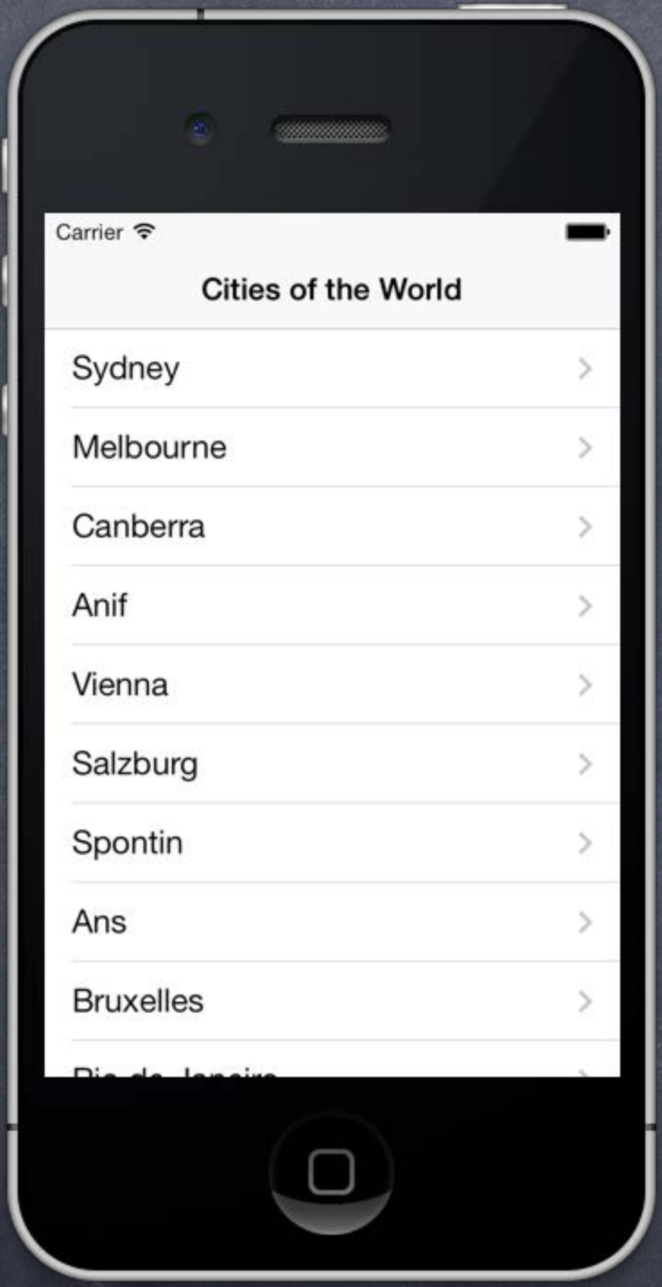
Dublin
DUB, Ireland

Venice
Veneto, Italy

Milan

Subtitle

`UITableViewCellStyleSubtitle`



Carrier

Cities of the World

Sydney

Melbourne

Canberra

Anif

Vienna

Salzburg

Spontin

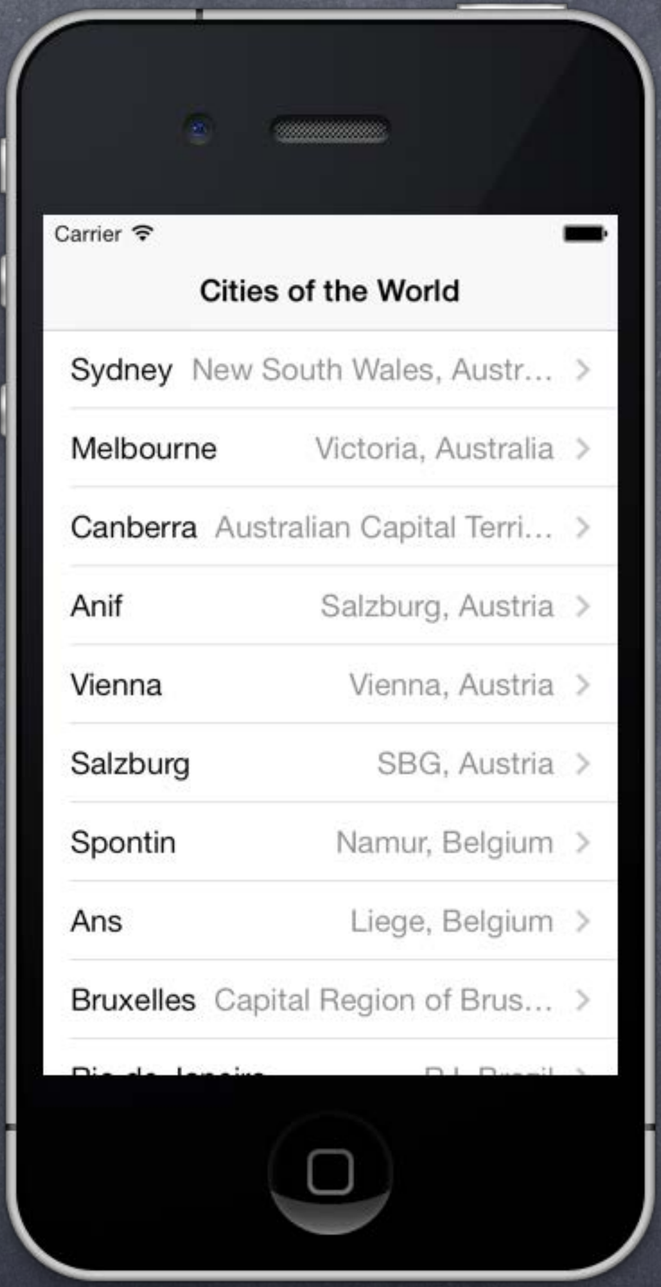
Ans

Bruxelles

Rio de Janeiro

Basic

`UITableViewCellStyleDefault`



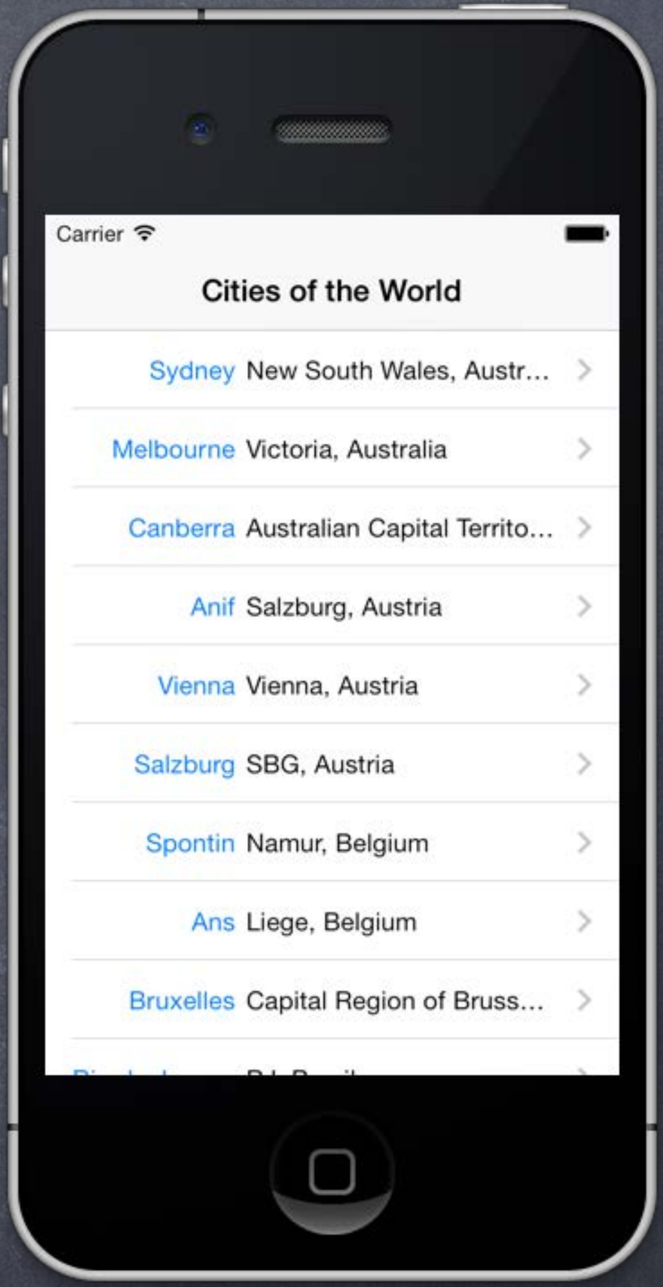
Carrier

Cities of the World

Sydney	New South Wales, Austr...
Melbourne	Victoria, Australia
Canberra	Australian Capital Terri...
Anif	Salzburg, Austria
Vienna	Vienna, Austria
Salzburg	SBG, Austria
Spontin	Namur, Belgium
Ans	Liege, Belgium
Bruxelles	Capital Region of Brus...
Rio de Janeiro	RJ, Brazil

Right Detail

`UITableViewCellStyleValue1`



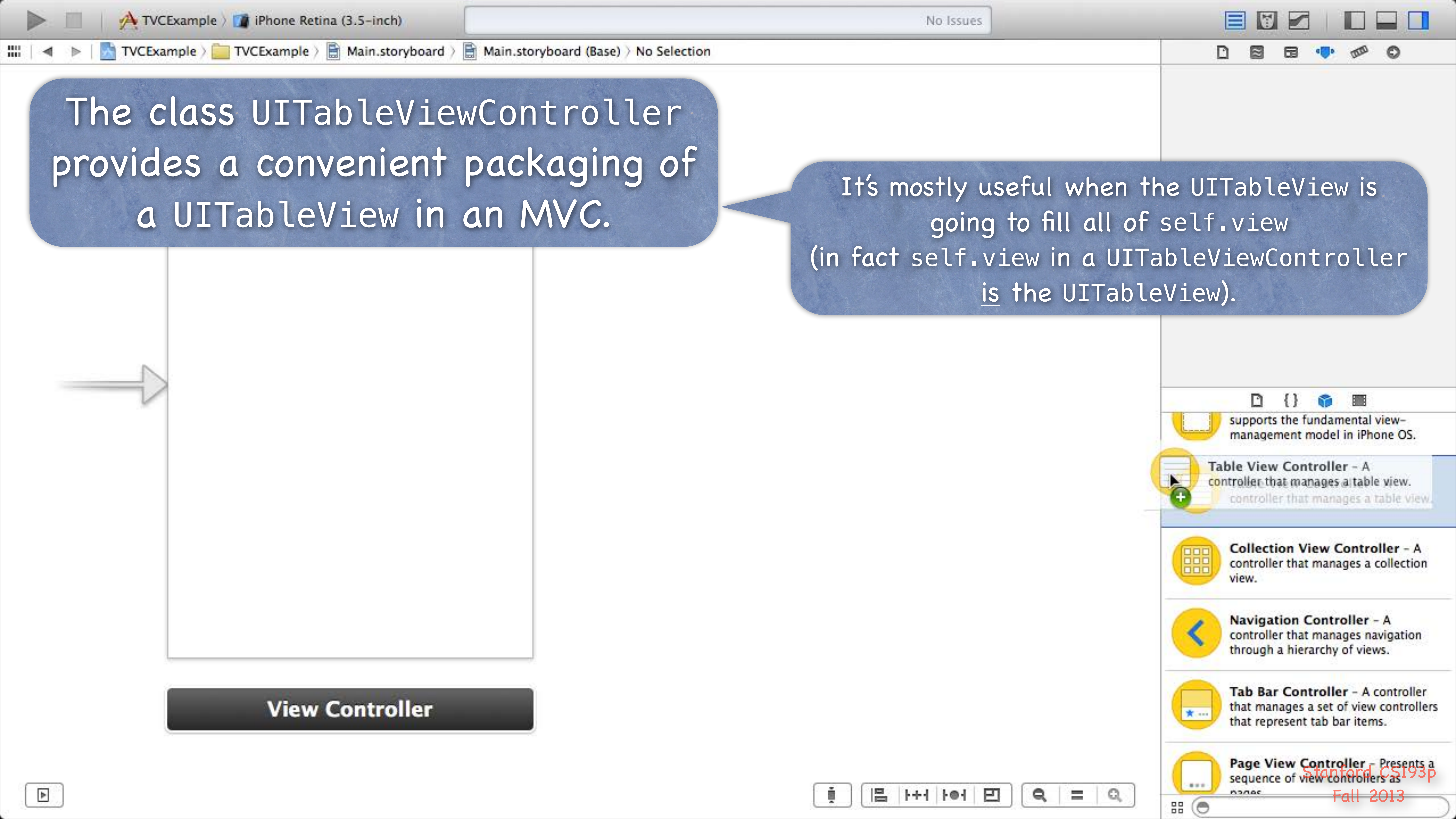
Carrier

Cities of the World

Sydney	New South Wales, Austr...
Melbourne	Victoria, Australia
Canberra	Australian Capital Territo...
Anif	Salzburg, Austria
Vienna	Vienna, Austria
Salzburg	SBG, Austria
Spontin	Namur, Belgium
Ans	Liege, Belgium
Bruxelles	Capital Region of Bruss...

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).

supports the fundamental view-management model in iPhone OS.

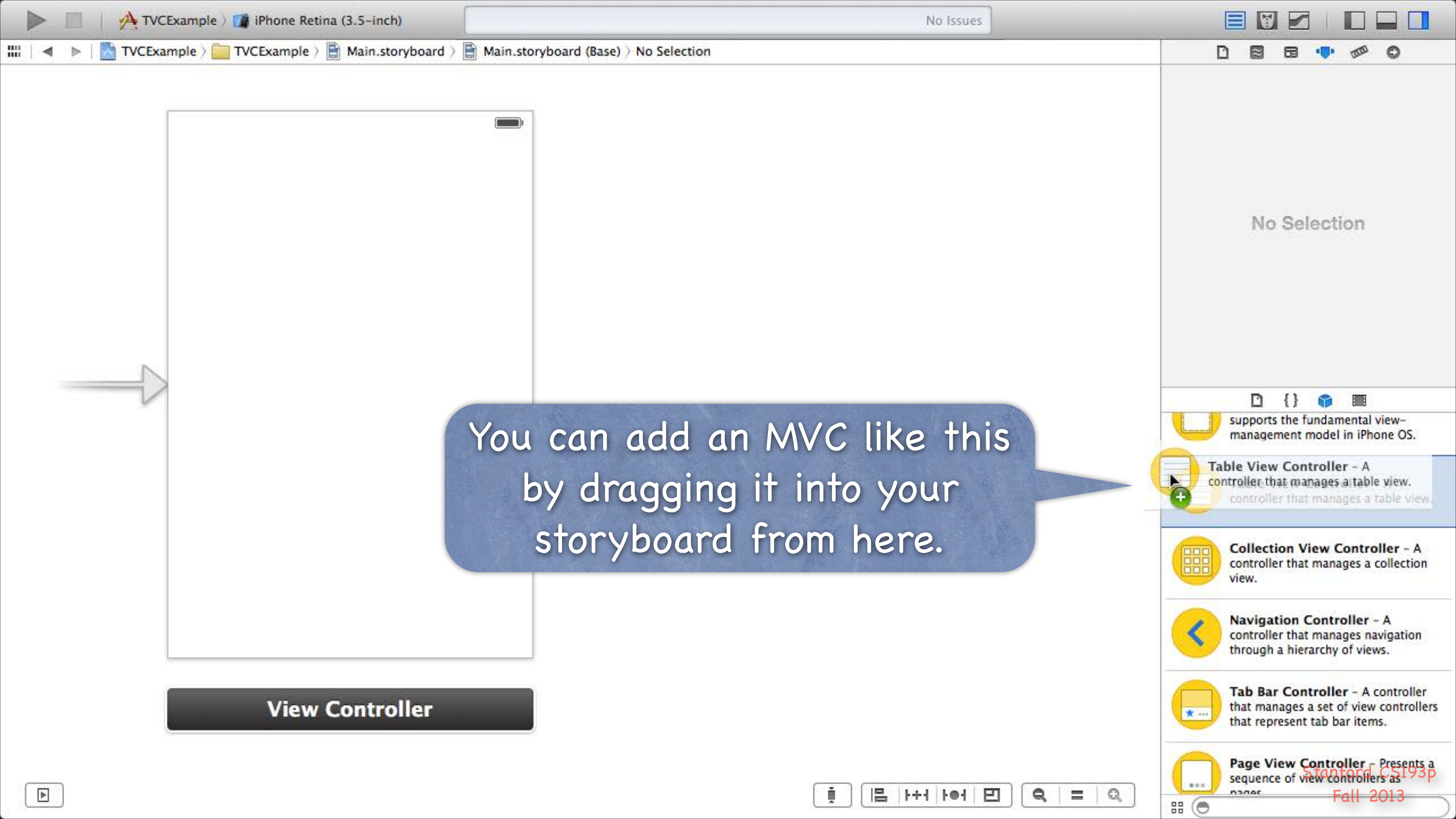
Table View Controller - A controller that manages a table view.

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.



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

supports the fundamental view-management model in iPhone OS.

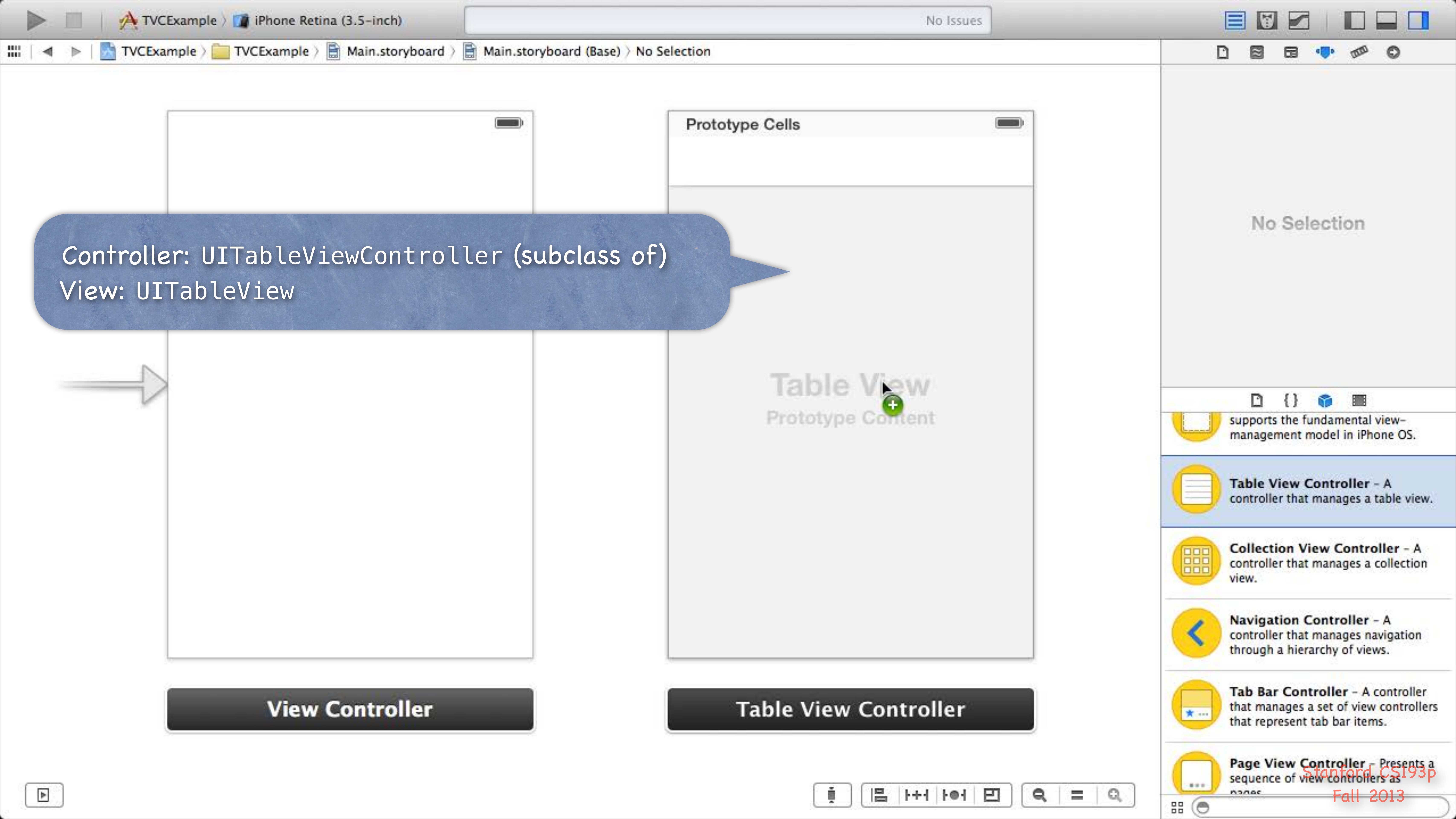
Table View Controller - A controller that manages a table view.

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.



Controller: UITableViewController (subclass of)
View: UITableView

supports the fundamental view-management model in iPhone OS.

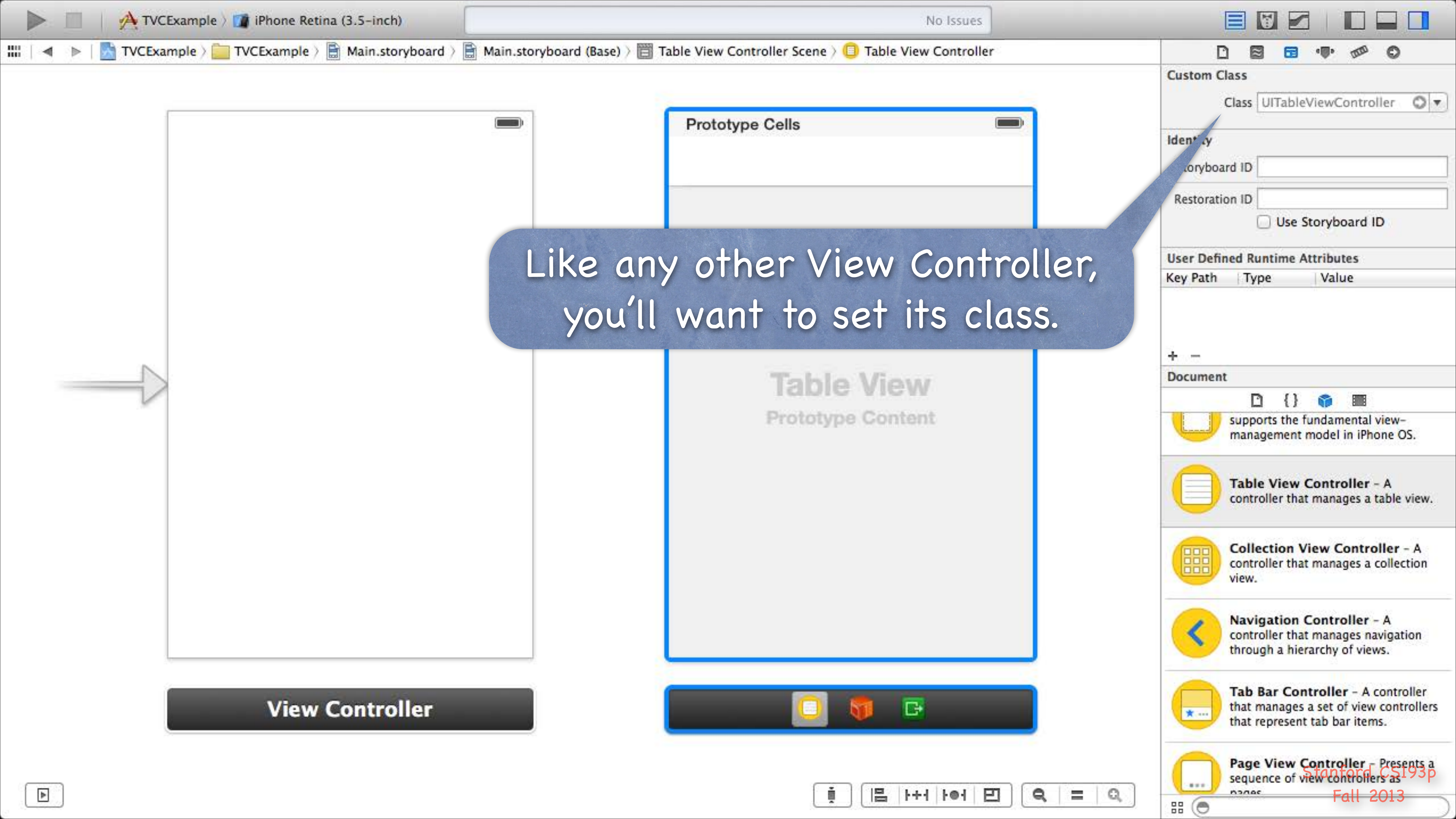
Table View Controller - A controller that manages a table view.

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.

supports the fundamental view-management model in iPhone OS.

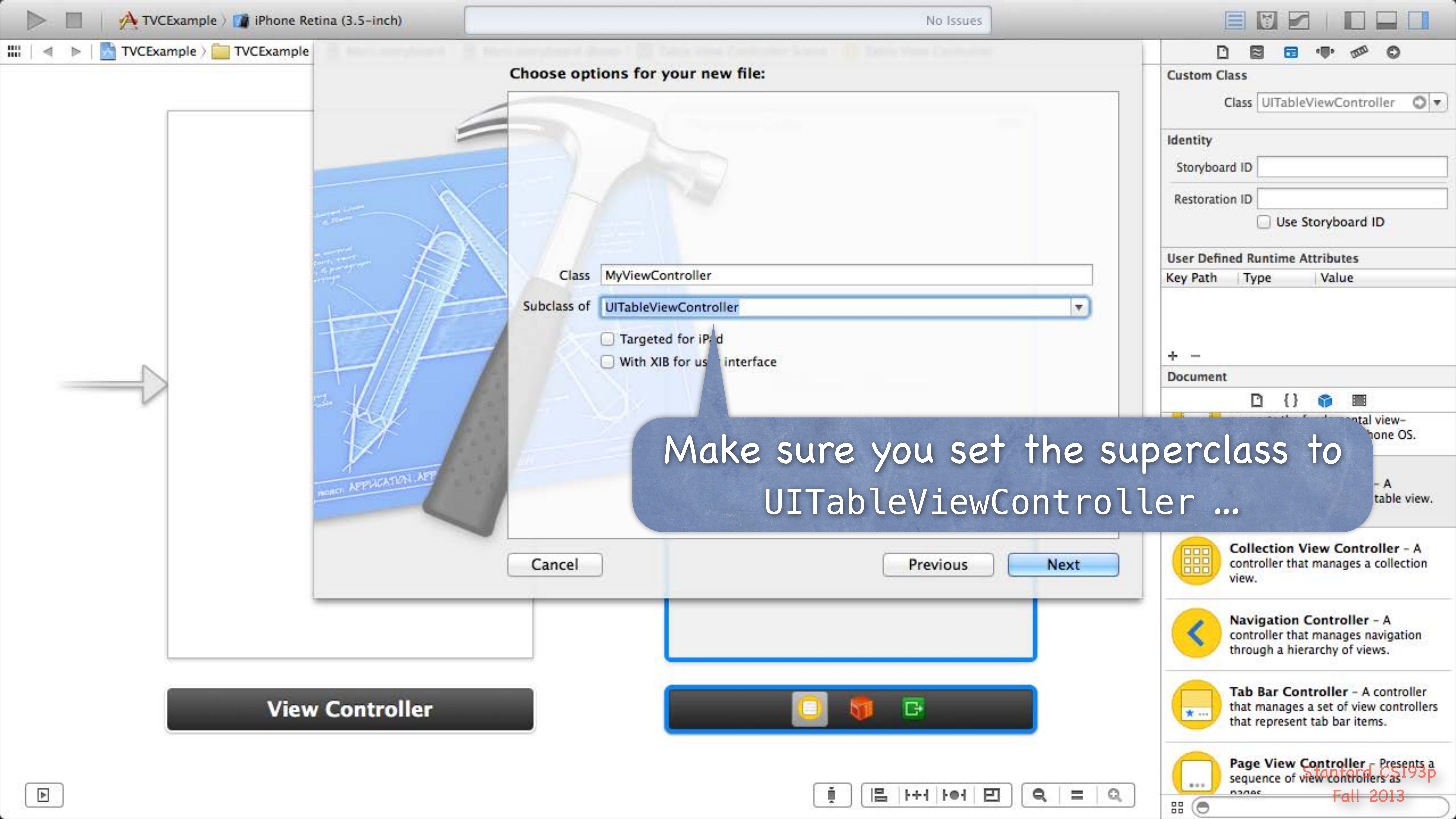
Table View Controller - A controller that manages a table view.

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: MyViewController

Subclass of: UITableViewController

□ Targeted for iPad

□ With XIB for user interface

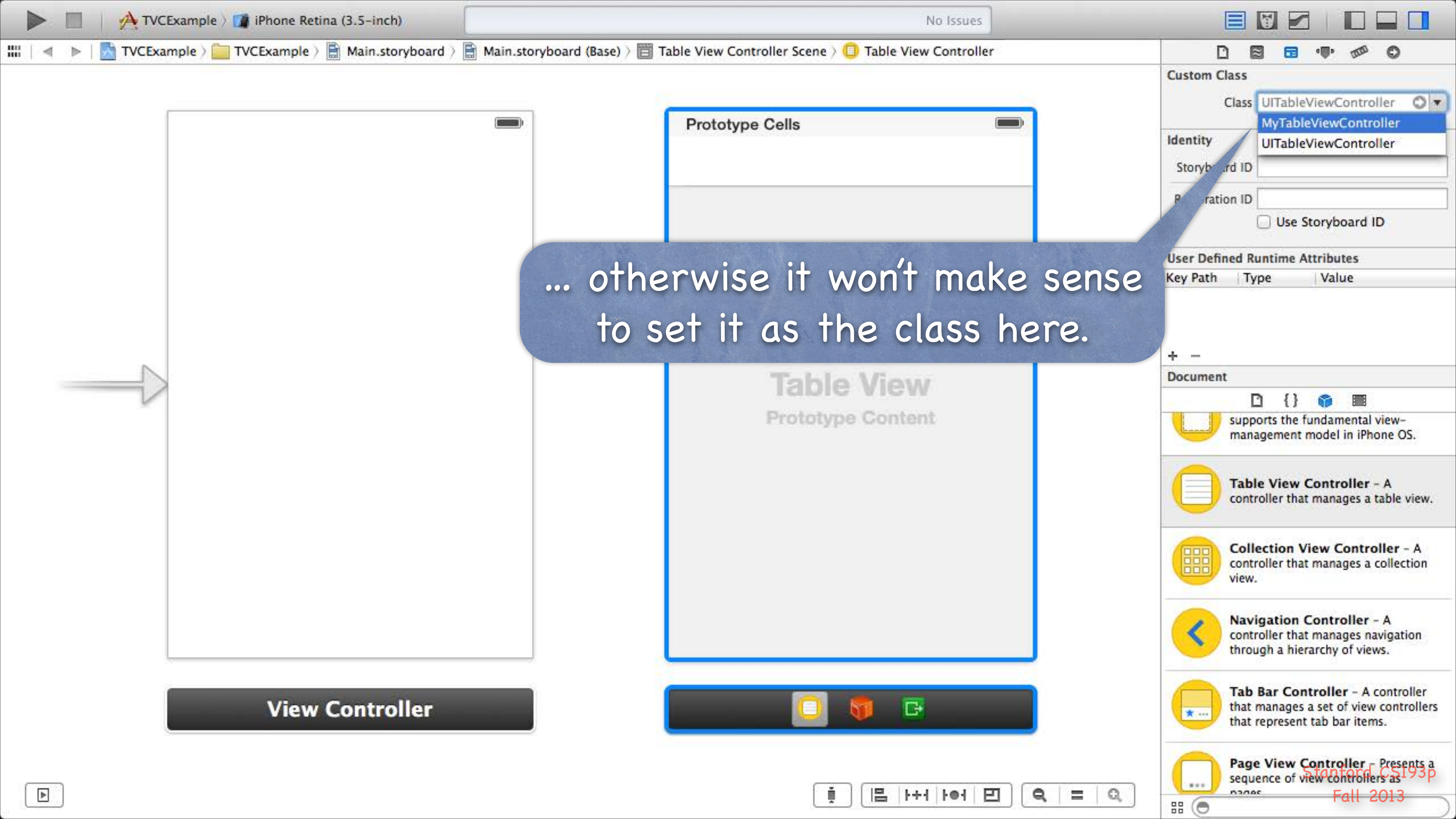
Make sure you set the superclass to UITableViewController …

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.



Class: UITableViewController

supports the fundamental view-management model in iPhone OS.

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

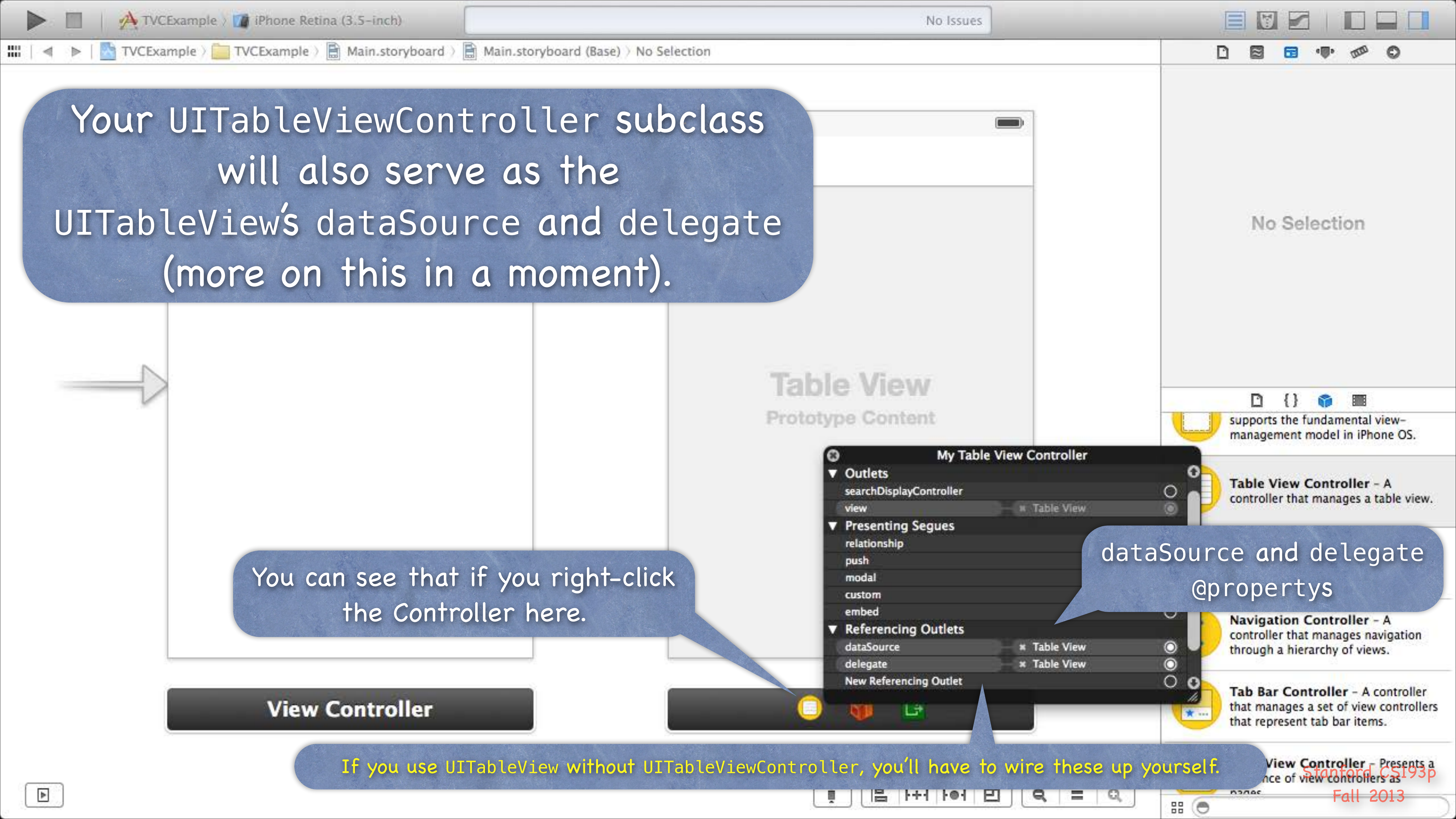
Collection View Controller - A controller that manages a collection view.

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.

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



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 @propertys

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

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 ○

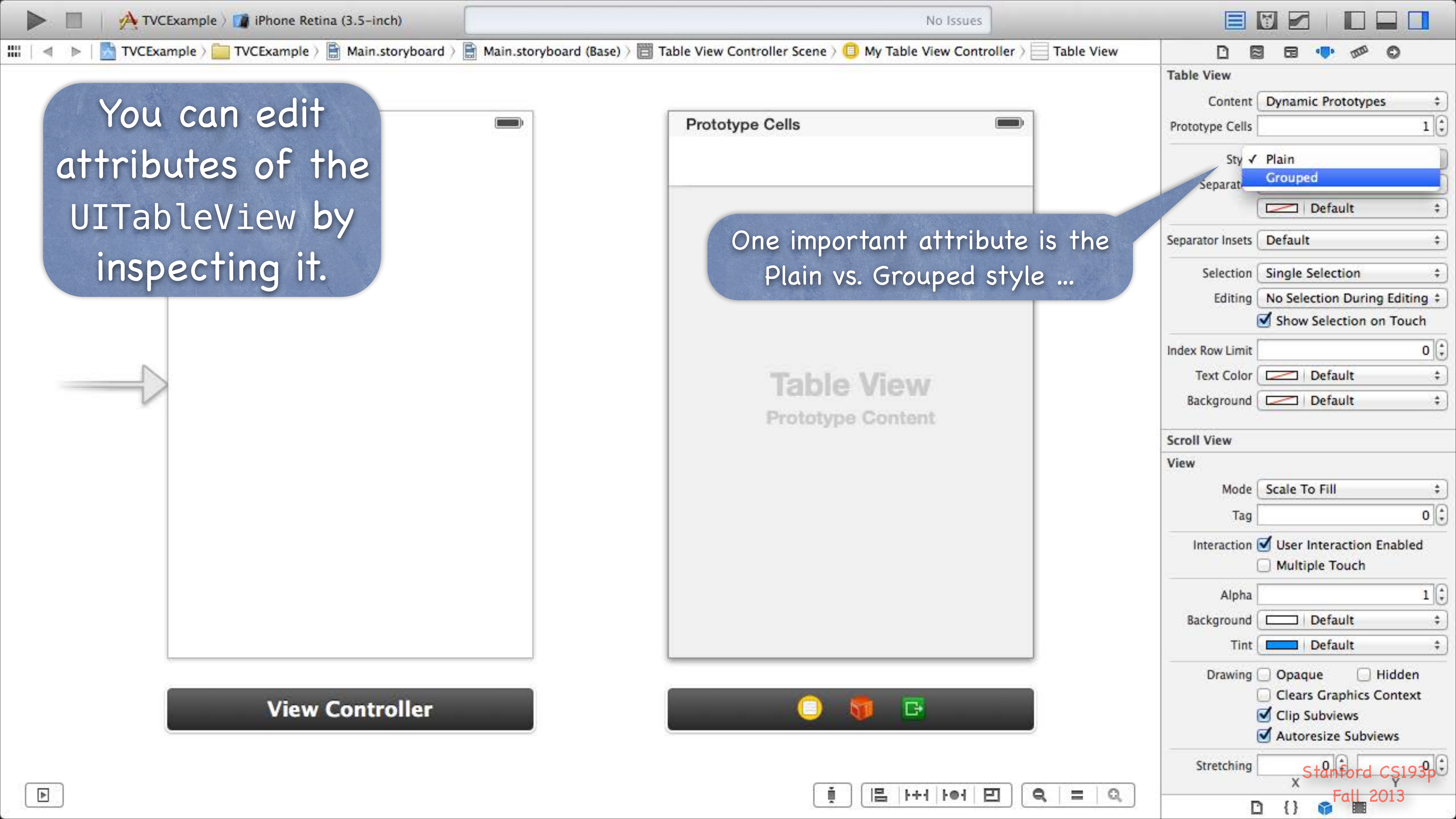
supports the fundamental view-management model in iPhone OS.

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 ...nce of view controllers as pages.



You can edit attributes of the UITableView by inspecting it.

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

Table View

Content: Dynamic Prototypes

Prototype Cells: 1

Sty ✓ Plain
Grouped

Separat

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

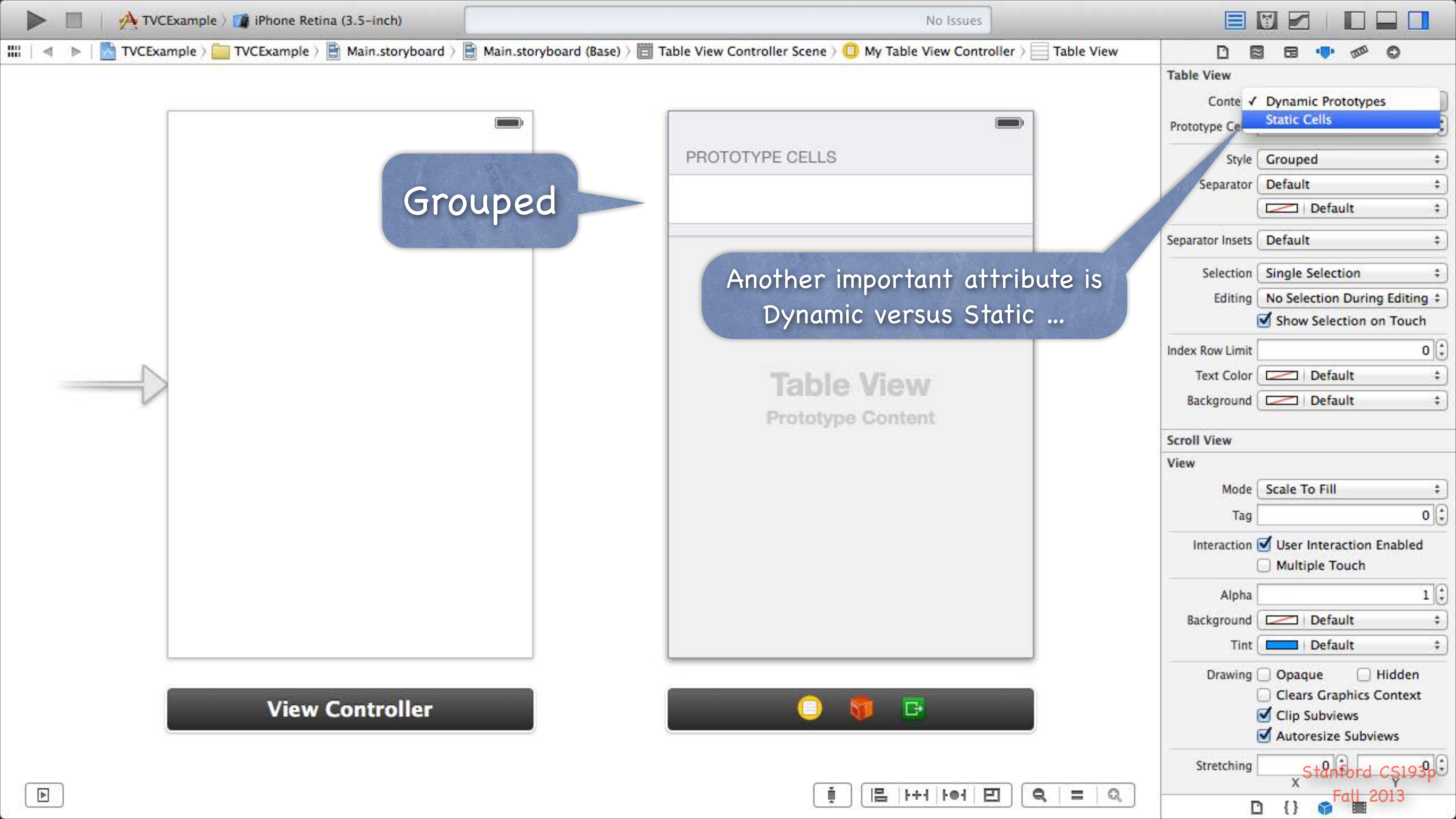
Alpha: 1

Background: Default

Tint: Default

Drawing: □ Opaque □ Hidden
□ Clears Graphics Context
☑ Clip Subviews
☑ Autoresize Subviews

Stretching: 0 0
X Y



Grouped

Another important attribute is Dynamic versus Static ...

Table View

Conte ✓ Dynamic Prototypes / Static Cells

Prototype Ce

Style: Grouped

Separator: Default

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

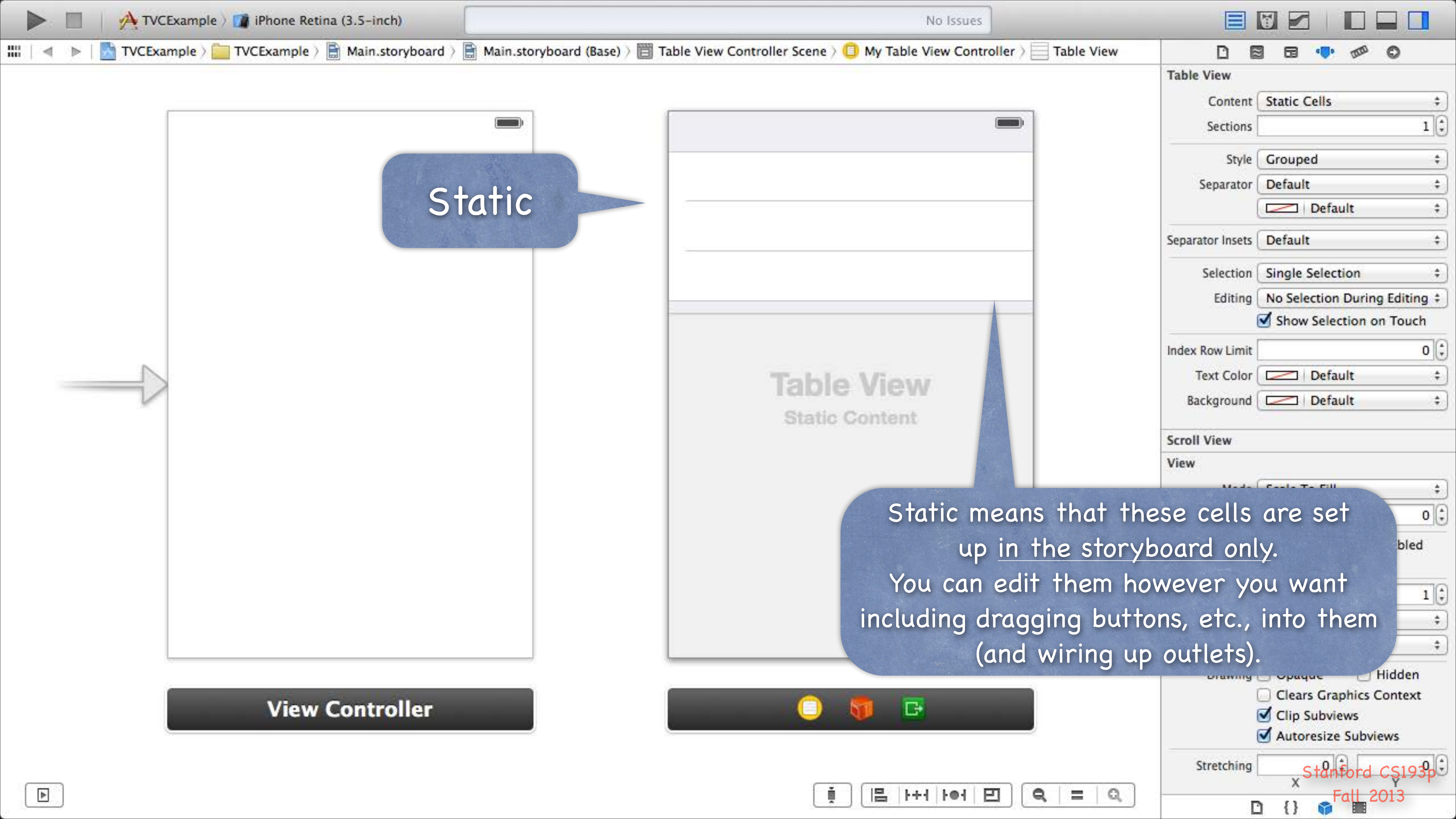
□ Clears Graphics Context

☑ Clip Subviews

☑ Autoresize Subviews

Stretching: 0 0

X Y



TVCExample › iPhone Retina (3.5-inch)
No Issues
TVCExample › TVCExample › Main.storyboard › Main.storyboard (Base) › Table View Controller Scene › My Table View Controller › Table View
Table View
Static Content
View Controller

Table View

- Content: Static Cells
- Sections: 1
- Style: Grouped
- Separator: Default
- 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

- □ Opaque □ Hidden
- □ Clears Graphics Context
- ☑ Clip Subviews
- ☑ Autoresize Subviews
- Stretching: 0 0

Static

Static means that these cells are set up in the storyboard only.
You can edit them however you want including dragging buttons, etc., into them (and wiring up outlets).

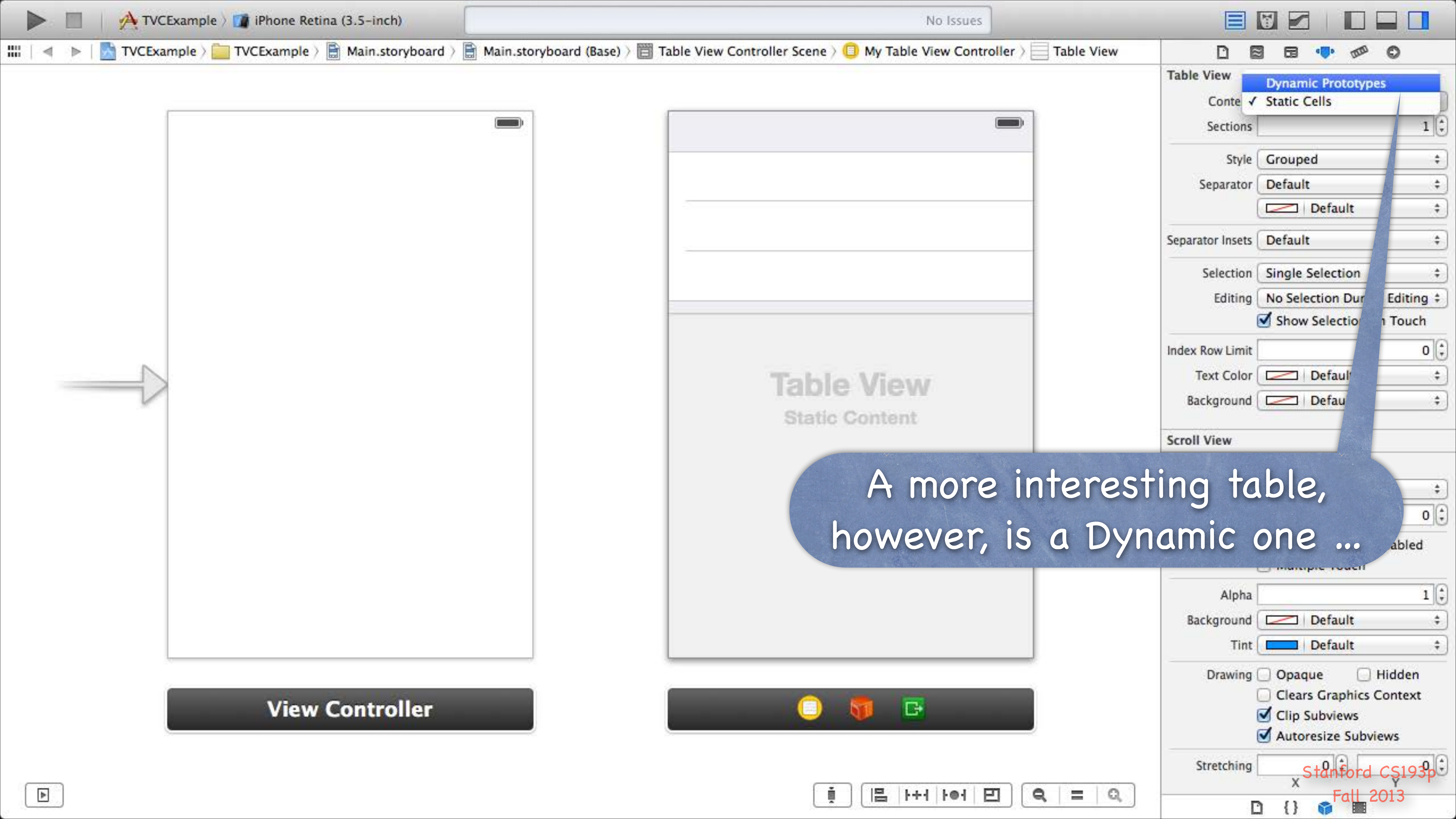

TVCExample 〉 iPhone Retina (3.5-inch)
No Issues
TVCExample 〉 TVCExample 〉 Main.storyboard 〉 Main.storyboard (Base) 〉 Table View Controller Scene 〉 My Table View Controller 〉 Table View
Table View
Static Content
View Controller

Table View

Content: Dynamic Prototypes / ✓ Static Cells

Sections: 1

Style: Grouped

Separator: Default / Default

Separator Insets: Default

Selection: Single Selection

Editing: No Selection Dur… Editing

☑ Show Selectio… Touch

Index Row Limit: 0

Text Color: Defaul…

Background: Defau…

Scroll View

…abled

Alpha: 1

Background: Default

Tint: Default

Drawing: □ Opaque □ Hidden

□ Clears Graphics Context

☑ Clip Subviews

☑ Autoresize Subviews

Stretching: 0 0 (X, Y)

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

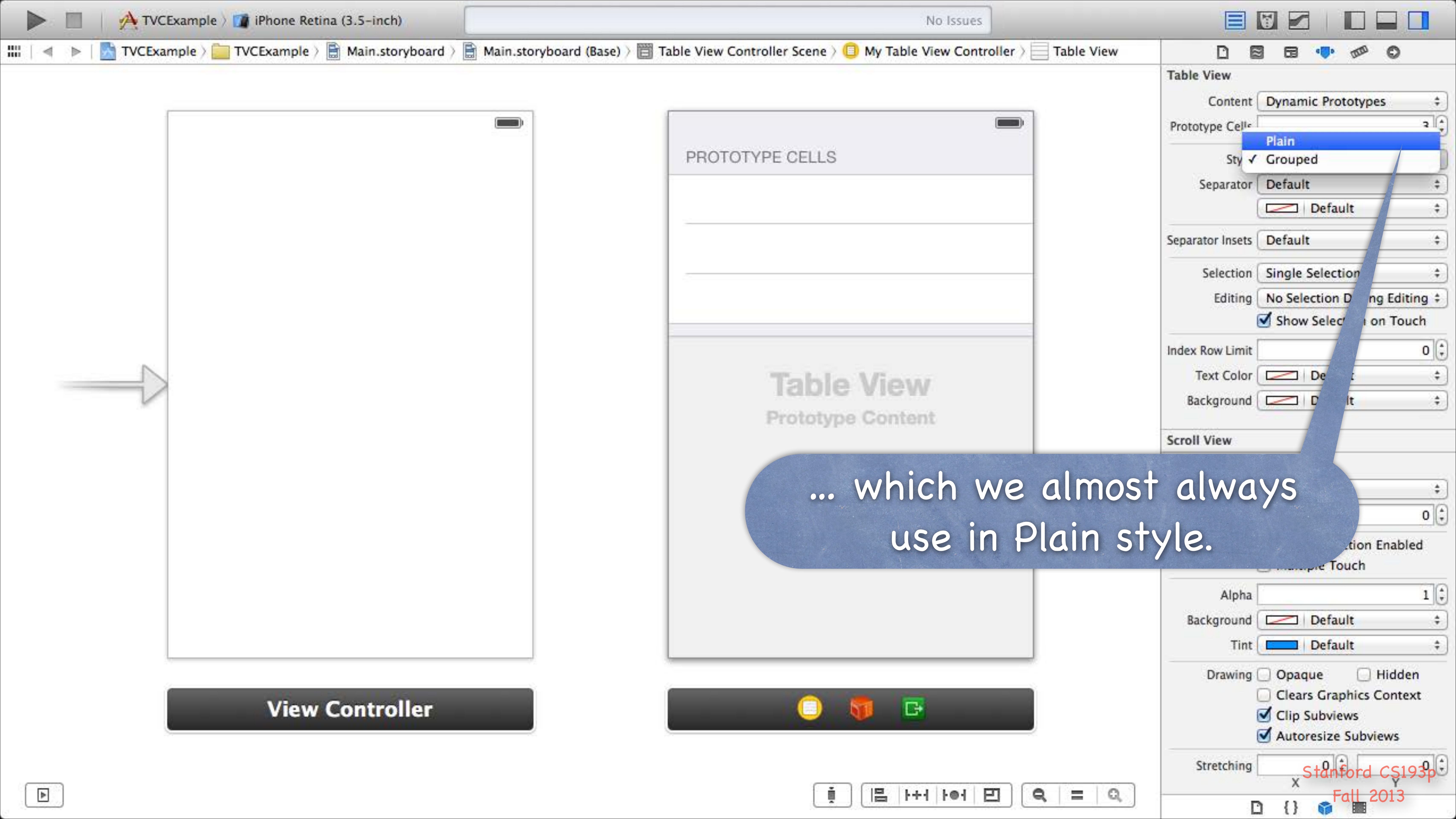


Table View

Content: Dynamic Prototypes

Prototype Cells: 3

Sty: Plain / ✓ Grouped

Separator: Default

Default

Separator Insets: Default

Selection: Single Selection

Editing: No Selection D ng Editing

☑ Show Selec on Touch

Index Row Limit: 0

Text Color: De t

Background: D lt

Scroll View

0

tion Enabled

Touch

Alpha: 1

Background: Default

Tint: Default

Drawing: □ Opaque □ Hidden

□ Clears Graphics Context

☑ Clip Subviews

☑ Autoresize Subviews

Stretching: 0 X 0 Y

… which we almost always use in Plain style.

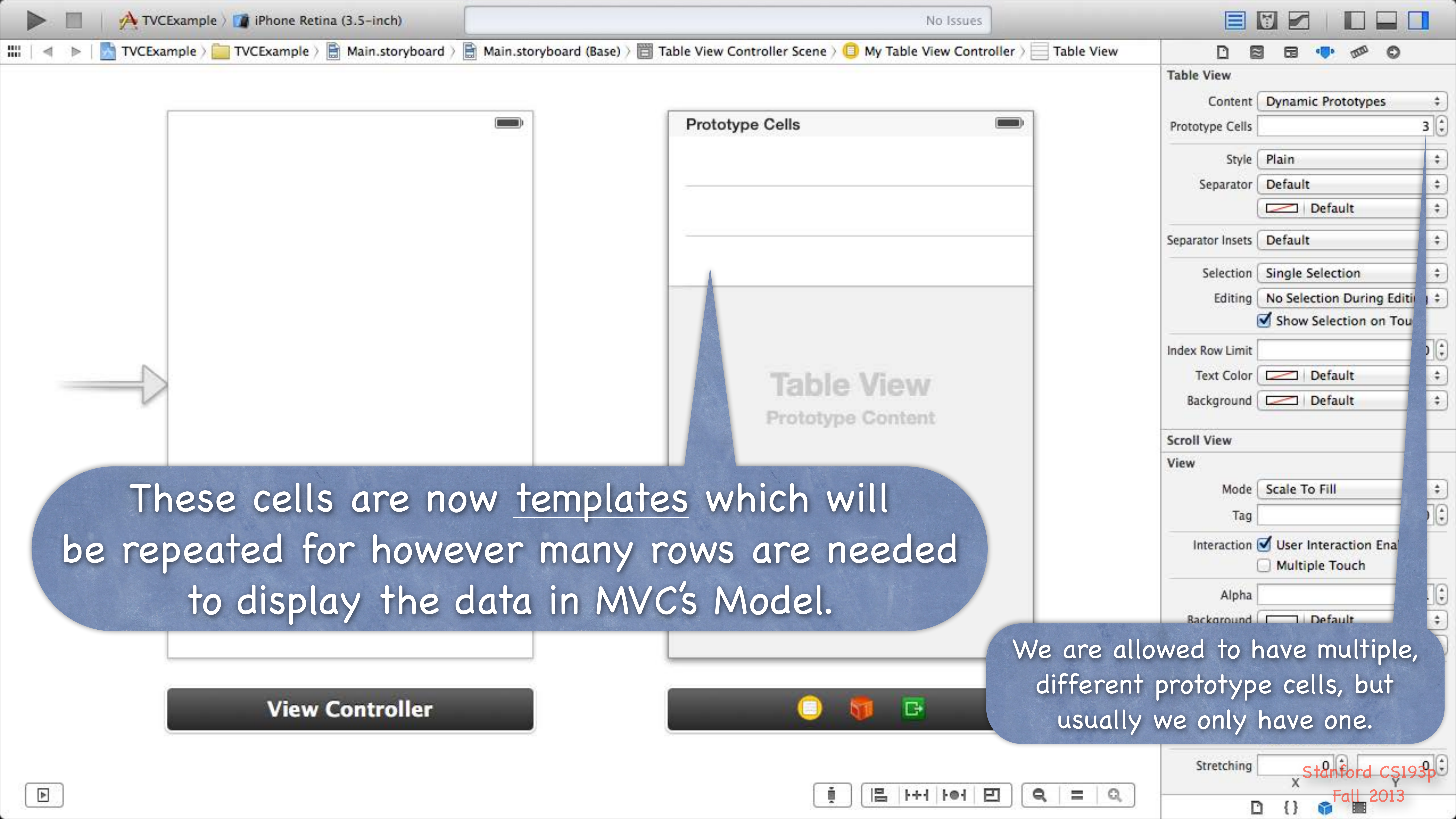

TVCExample
iPhone Retina (3.5-inch)
No Issues
TVCExample › TVCExample › Main.storyboard › Main.storyboard (Base) › Table View Controller Scene › My Table View Controller › Table View
Prototype Cells
Table View
Prototype Content
View Controller

Table View

Content: Dynamic Prototypes

Prototype Cells: 3

Style: Plain

Separator: Default

Default

Separator Insets: Default

Selection: Single Selection

Editing: No Selection During Editi

☑ Show Selection on Tou

Index Row Limit

Text Color: Default

Background: Default

Scroll View

View

Mode: Scale To Fill

Tag

Interaction: ☑ User Interaction Ena

□ Multiple Touch

Alpha

Background: Default

Stretching: 0 0

X Y

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.

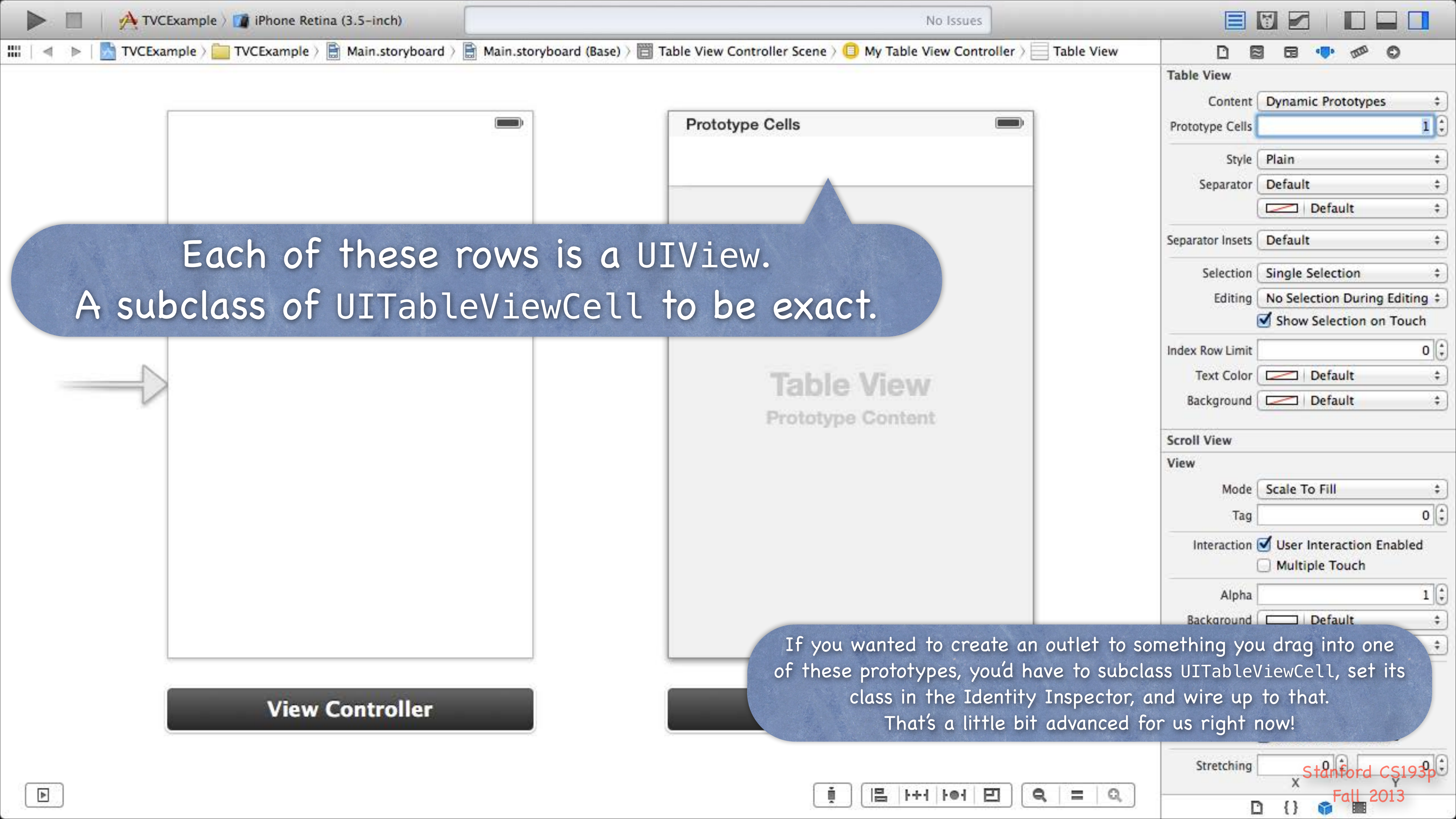


Table View

Content: Dynamic Prototypes

Prototype Cells: 1

Style: Plain

Separator: Default

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

Stretching: 0 0

X Y

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

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!

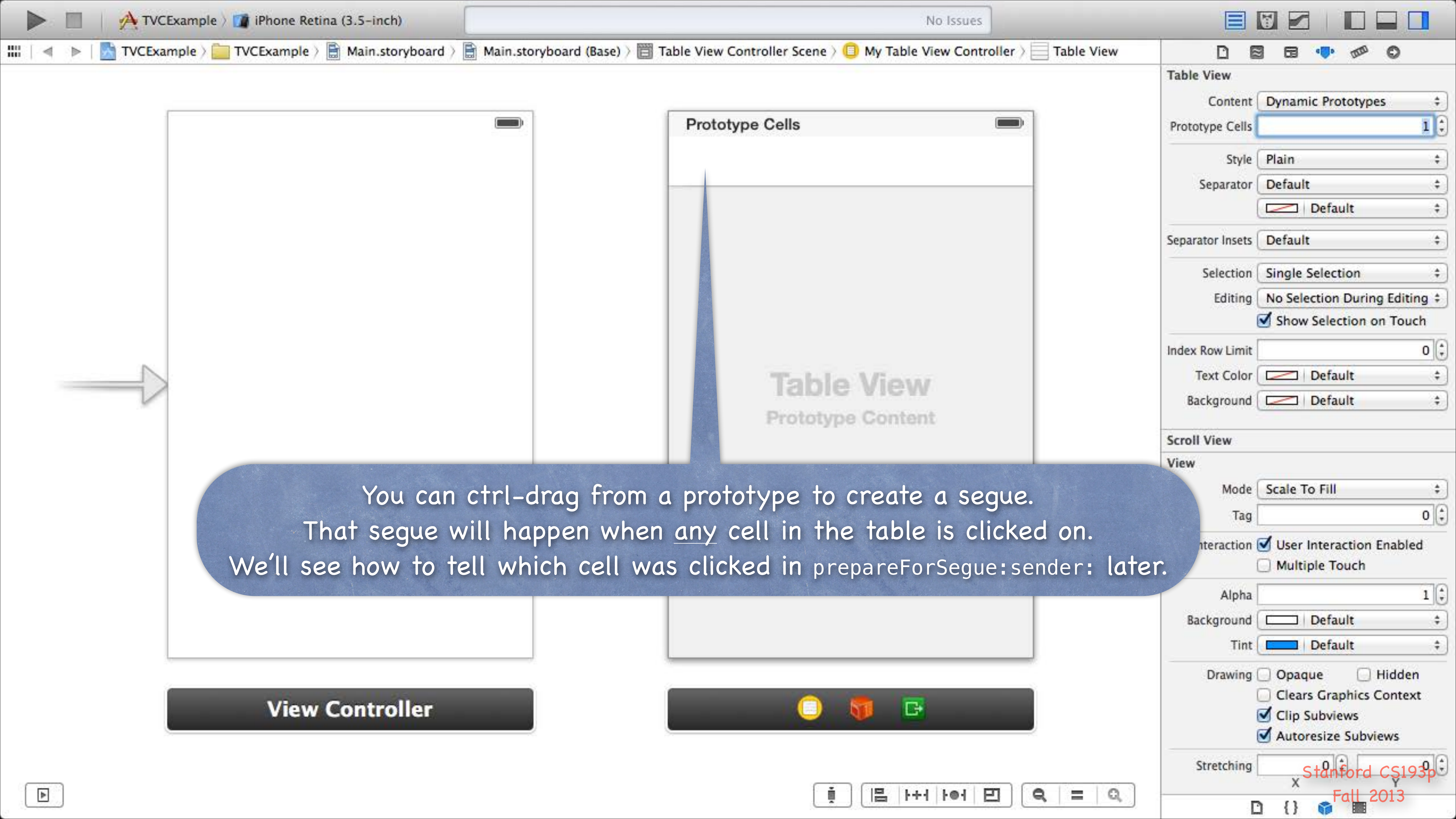

TVCExample
iPhone Retina (3.5-inch)
No Issues
TVCExample › TVCExample › Main.storyboard › Main.storyboard (Base) › Table View Controller Scene › My Table View Controller › Table View
Prototype Cells
Table View
Prototype Content
View Controller

Table View

Content: Dynamic Prototypes

Prototype Cells: 1

Style: Plain

Separator: Default

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: 0 0

X Y

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.

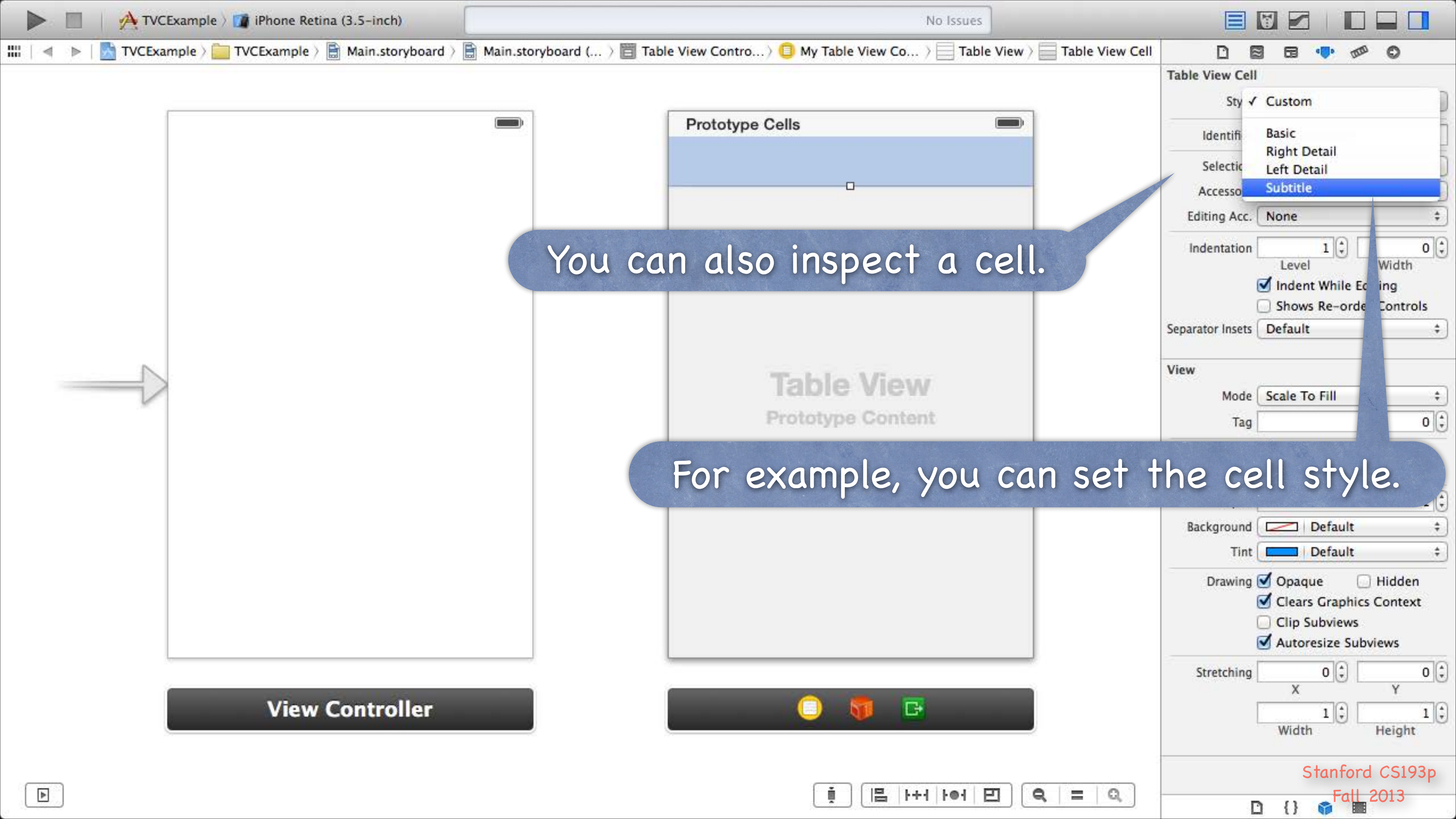


Table View Cell

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

Background Default

Tint Default

Drawing ☑ Opaque ☐ Hidden

☑ Clears Graphics Context

☐ Clip Subviews

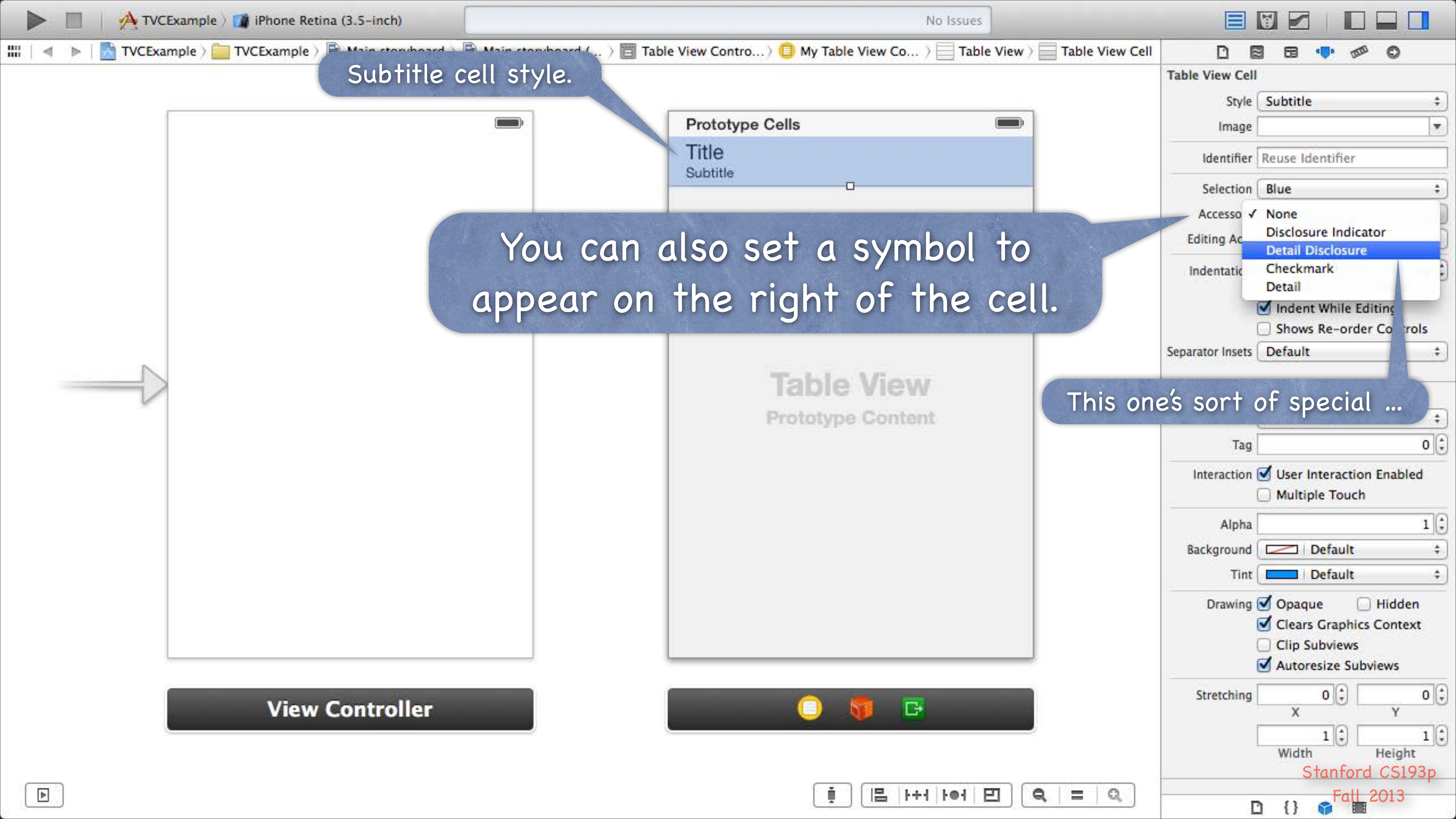
☑ Autoresize Subviews

Stretching 0 X 0 Y

1 Width 1 Height

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 …

Table View Cell

Style: Subtitle

Image:

Identifier: Reuse Identifier

Selection: Blue

Accesso…: ✓ None, Disclosure Indicator, Detail Disclosure, Checkmark, Detail

Editing Ac…

Indentatio…

☑ Indent While Editing

☐ Shows Re-order Co…rols

Separator Insets: Default

Tag: 0

Interaction: ☑ User Interaction Enabled

☐ Multiple Touch

Alpha: 1

Background: Default

Tint: Default

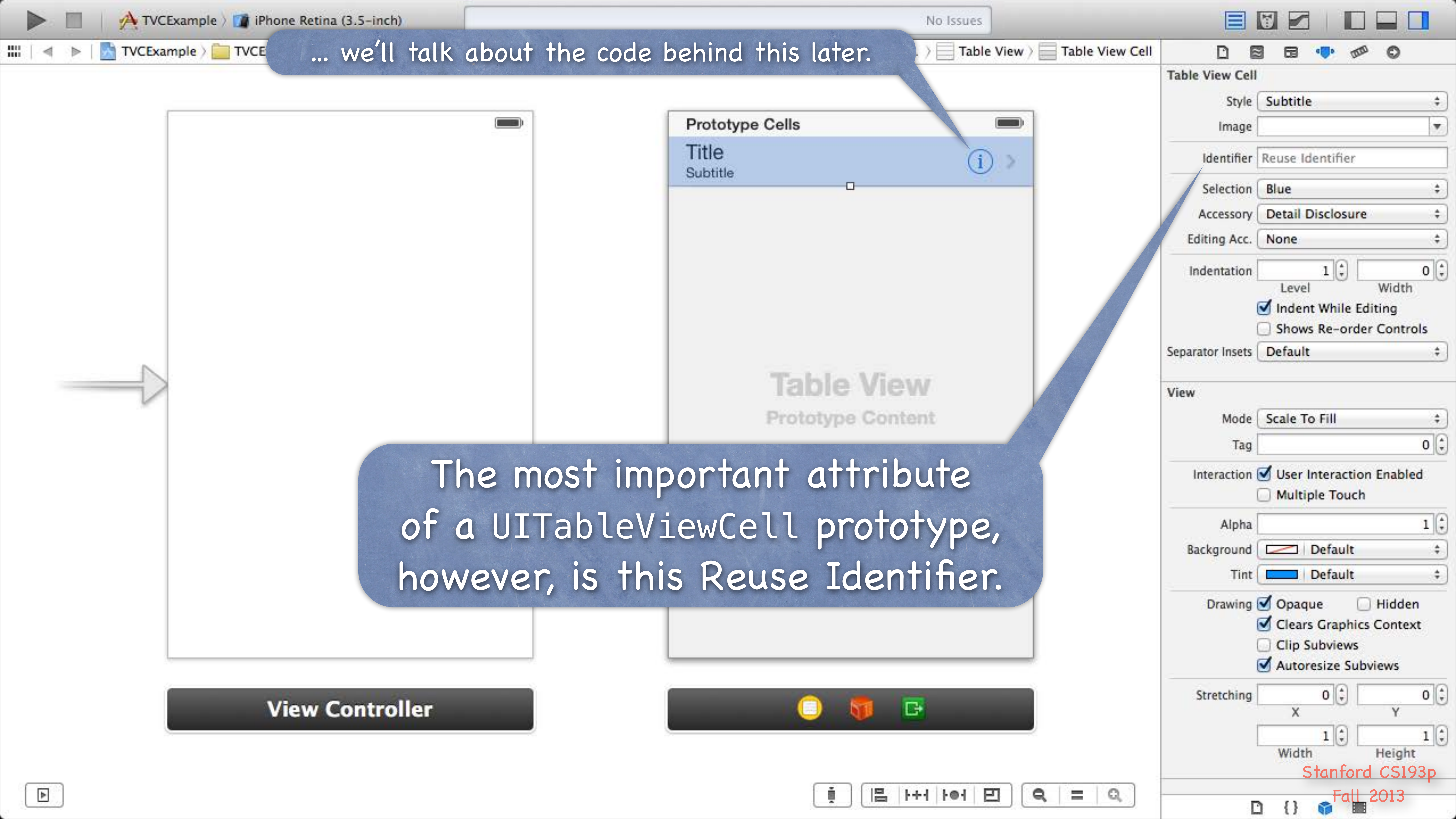
Drawing: ☑ Opaque ☐ Hidden

☑ Clears Graphics Context

☐ Clip Subviews

☑ Autoresize Subviews

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



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

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

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

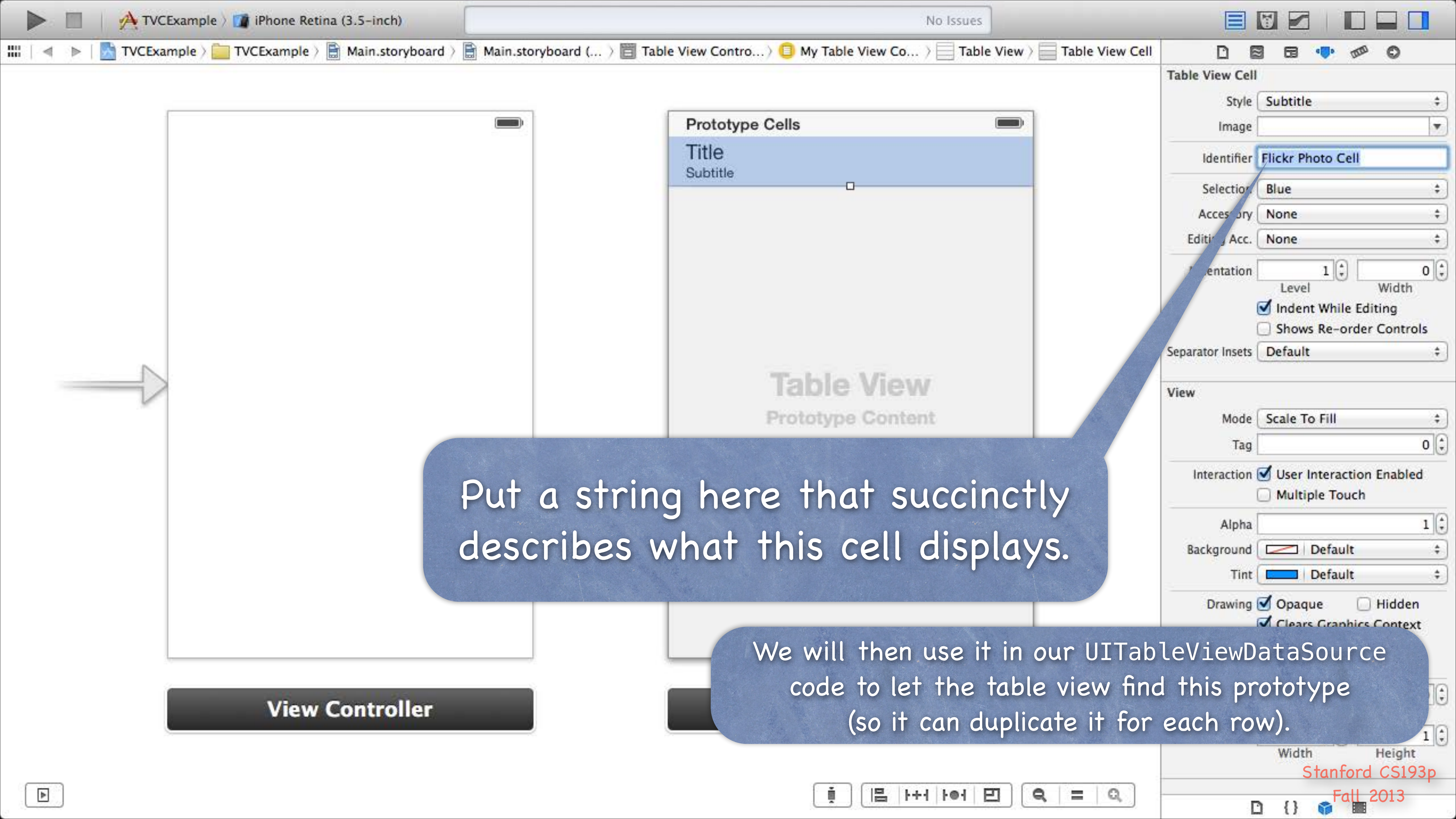

TVCExample
iPhone Retina (3.5-inch)
No Issues
TVCExample › TVCExample › Main.storyboard › Main.storyboard (… › Table View Contro… › My Table View Co… › Table View › Table View Cell
Prototype Cells
Title
Subtitle
Table View
Prototype Content
View Controller
Width
Height

Table View Cell

Style: Subtitle

Image:

Identifier: Flickr Photo Cell

Selection: Blue

Accessory: None

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

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).

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 getMyTitleForRow: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 getMyTitleForRow: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).

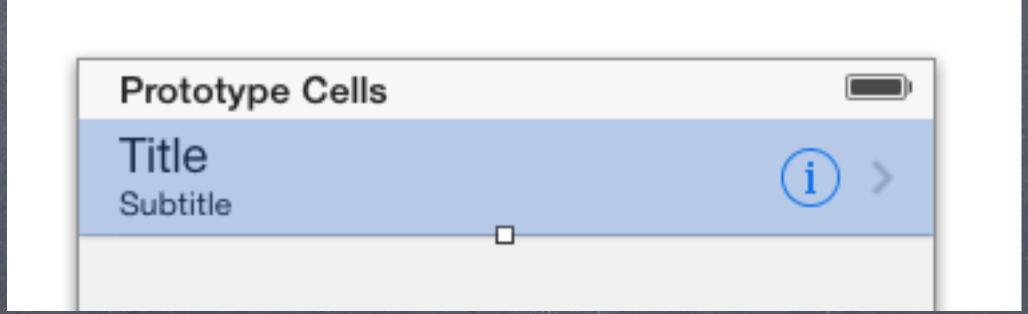
  On the iPad, where the table might be on screen with what it updates, you might need this.

```
- (void)tableView:(UITableView *)sender didSelectRowAtIndexPath:(NSIndexPath *)path
{
    // 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.

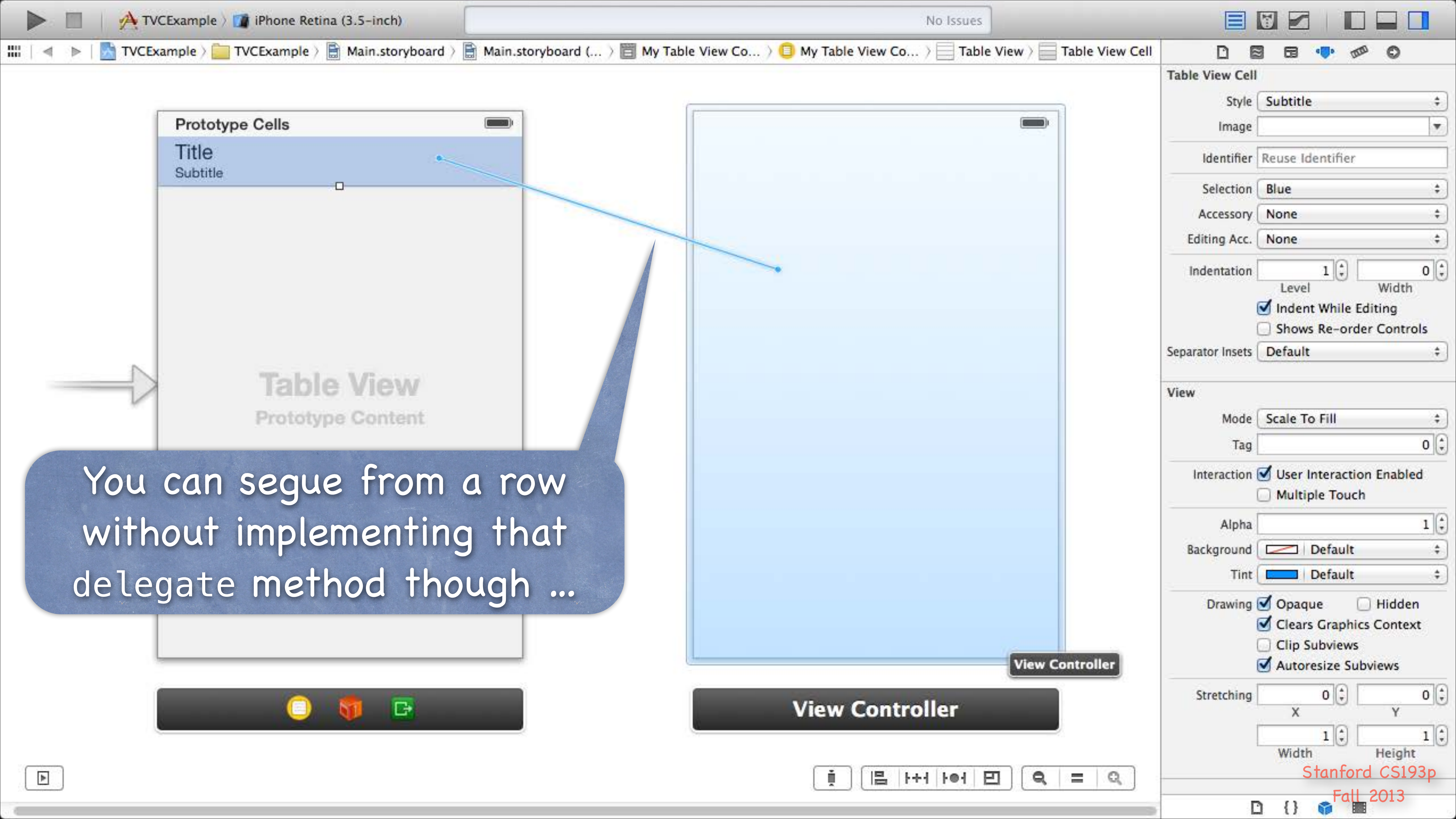


Table View Cell

Style: Subtitle

Image:

Identifier: Reuse Identifier

Selection: Blue

Accessory: None

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

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

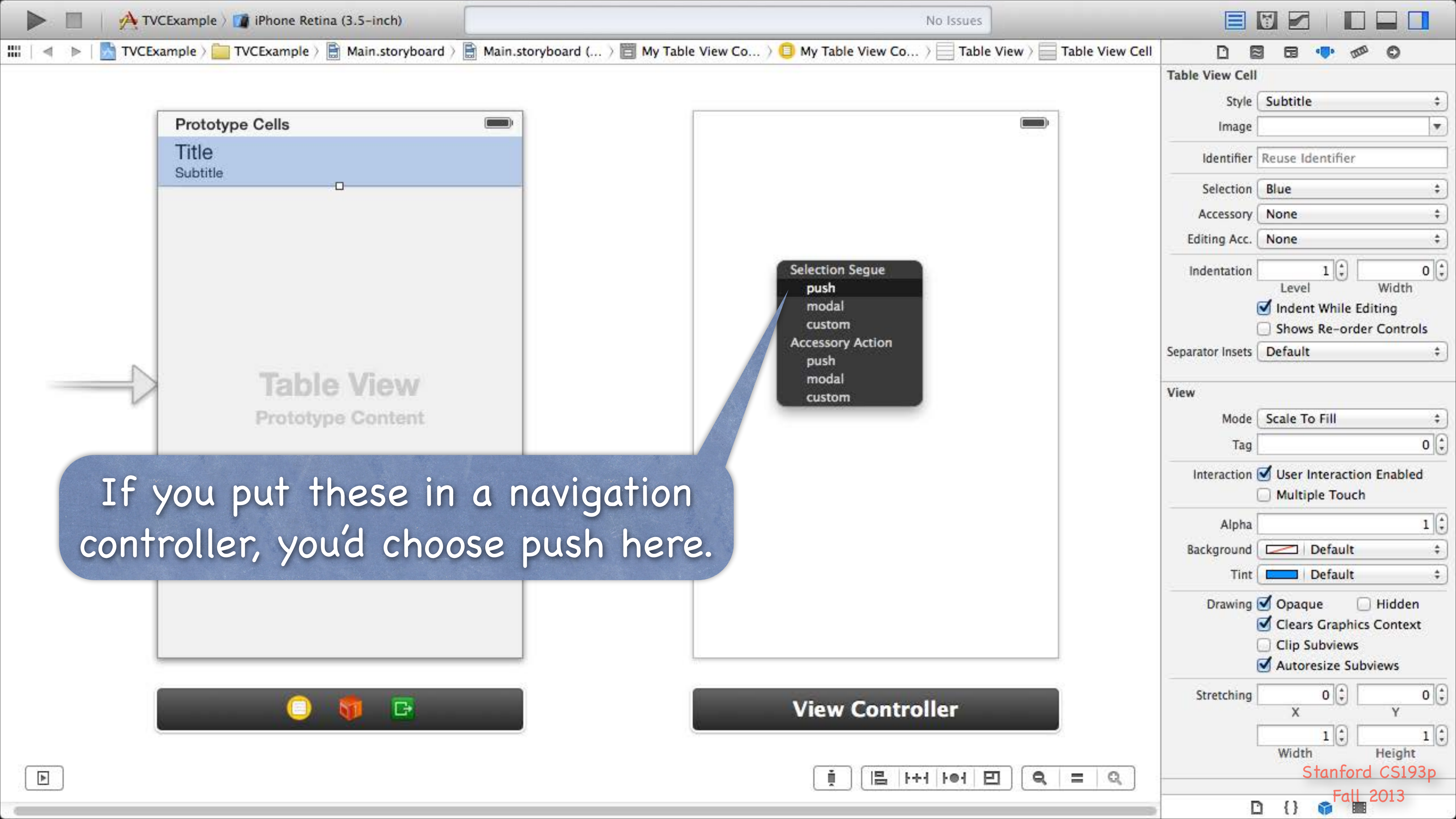


Table View Cell

Style: Subtitle

Image:

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

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

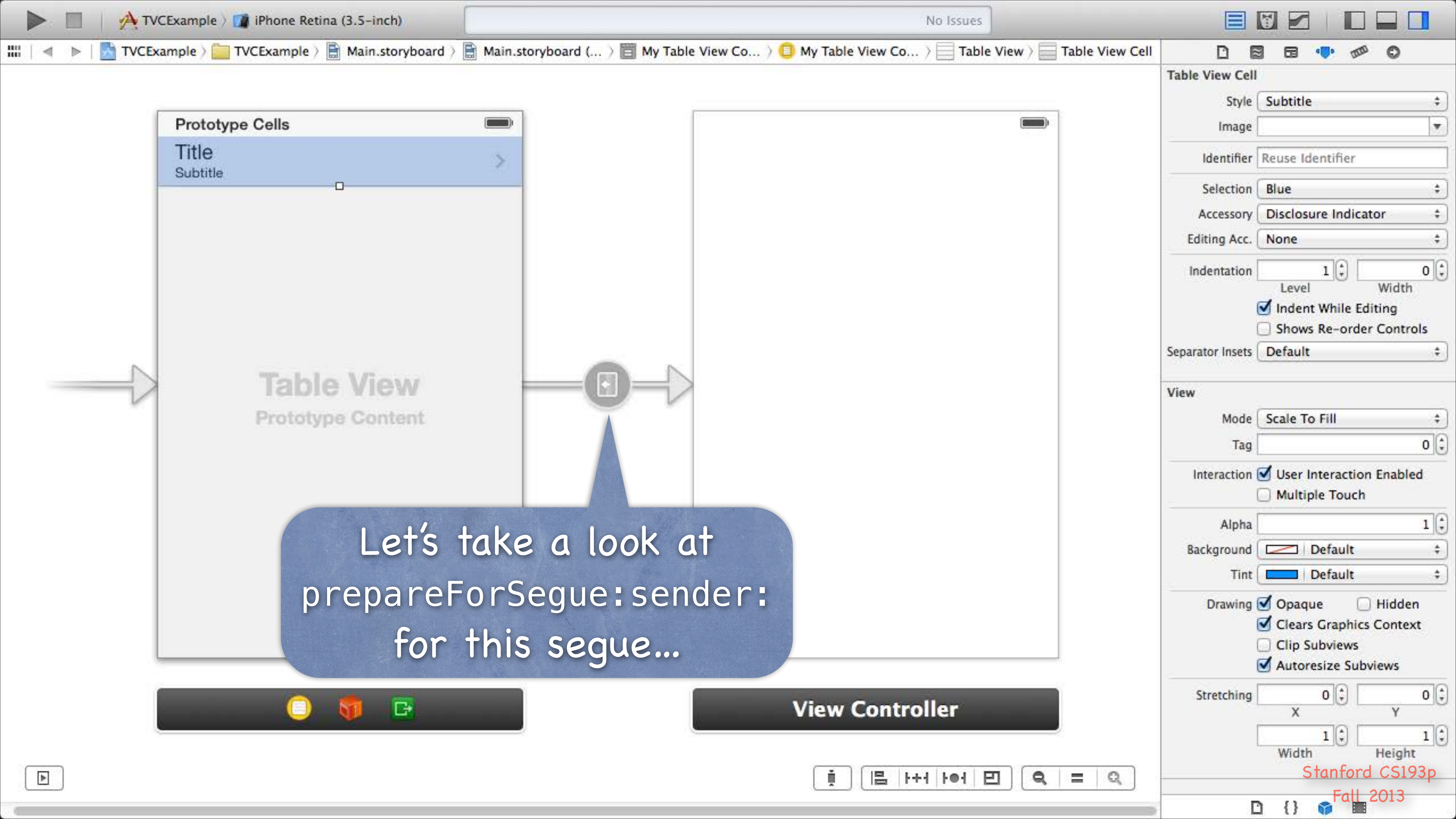

TVCExample
iPhone Retina (3.5-inch)
No Issues
TVCExample > TVCExample > Main.storyboard > Main.storyboard (... > My Table View Co... > My Table View Co... > Table View > Table View Cell
Prototype Cells
Title
Subtitle
Table View
Prototype Content
View Controller

Table View Cell

Style: Subtitle

Image:

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

Let’s take a look at prepareForSegue:sender: for this segue…

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 …

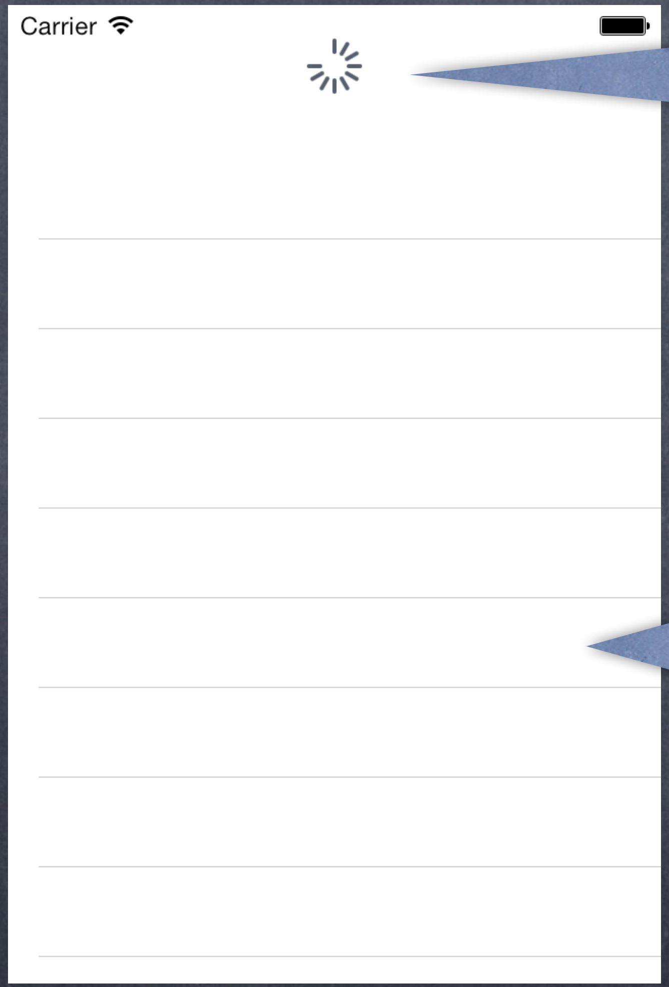
  ```
  @property (strong) UIRefreshControl *refreshControl;
  ```

  Start it with …

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

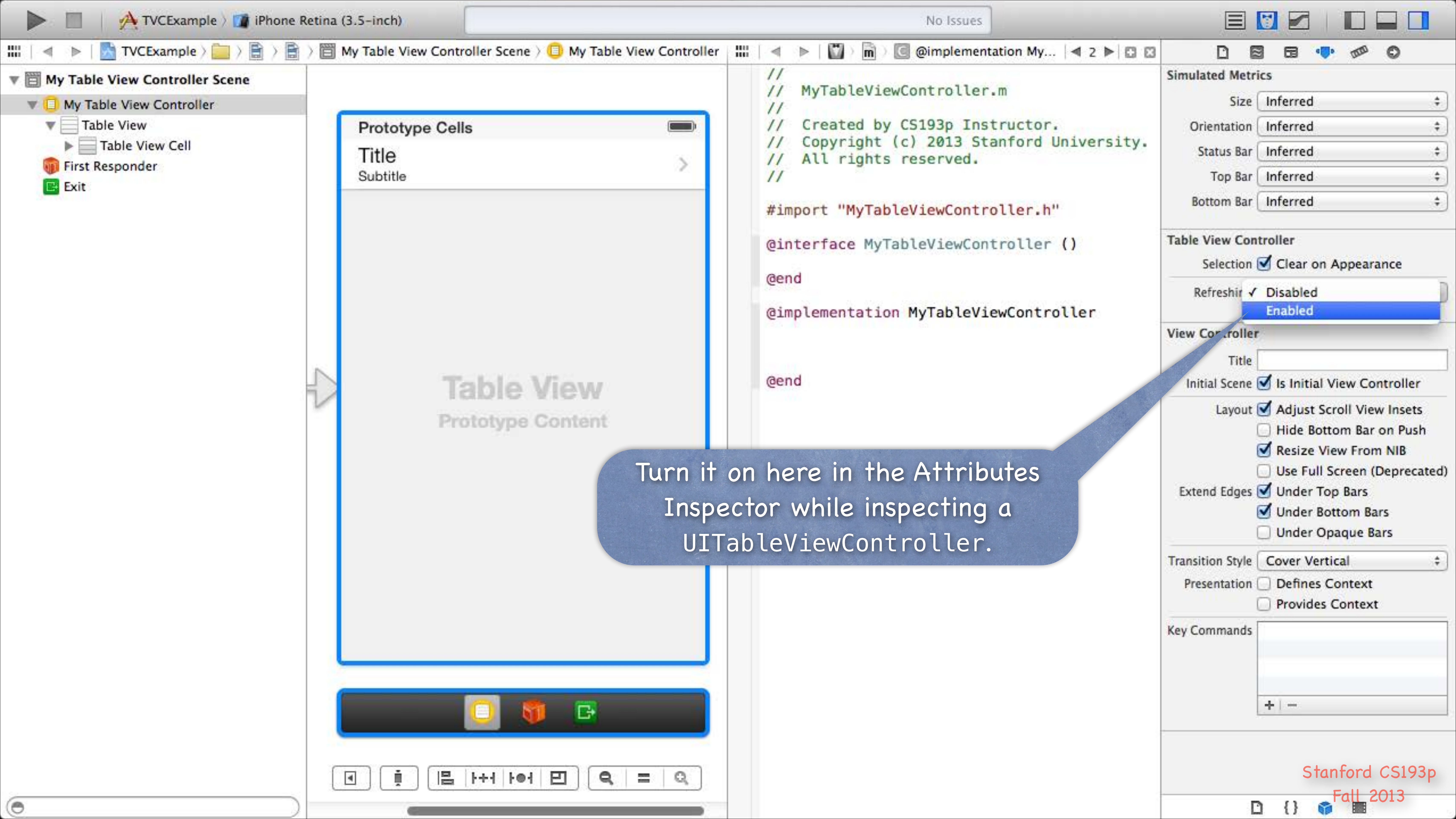
  Stop it with …

  ```
  - (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
```

Simulated Metrics

Size: Inferred

Orientation: Inferred

Status Bar: Inferred

Top Bar: Inferred

Bottom Bar: Inferred

Table View Controller

Selection ☑ Clear on Appearance

Refreshing ✓ 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

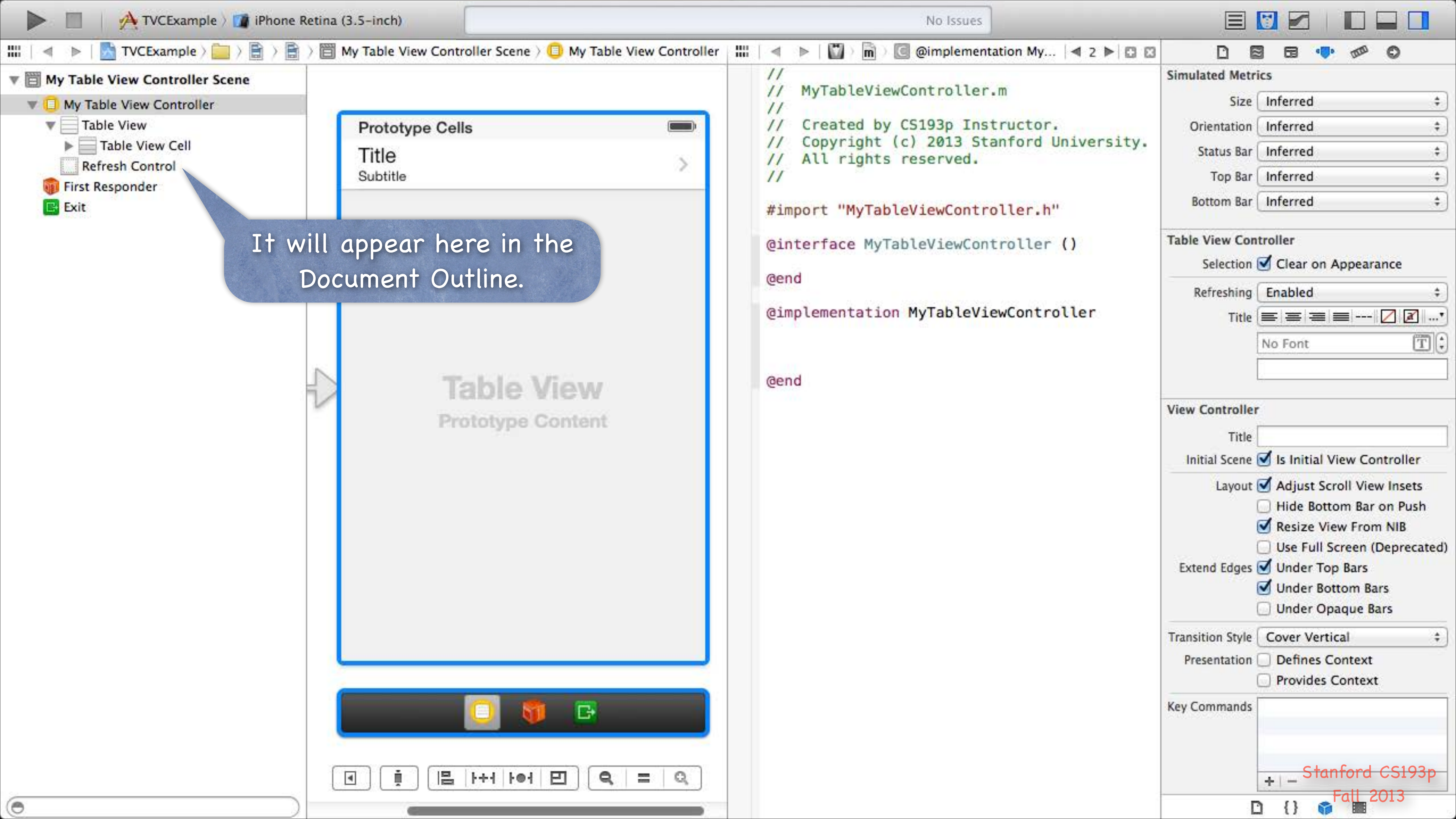
Presentation □ Defines Context

□ Provides Context

Key Commands

+ −

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



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
```

Simulated Metrics

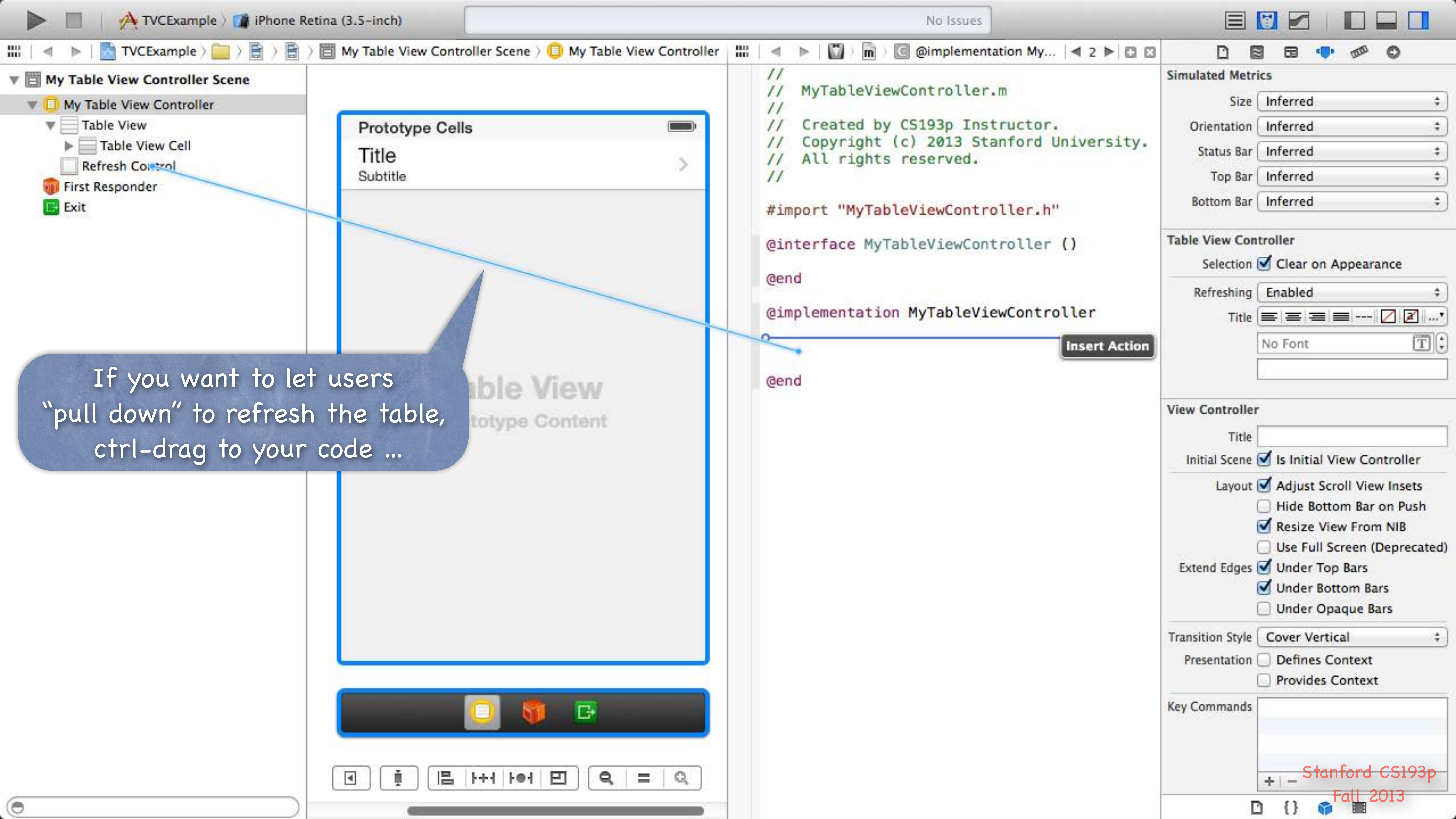
Size: Inferred
Orientation: Inferred
Status Bar: Inferred
Top Bar: Inferred
Bottom Bar: Inferred

Table View Controller

Selection ☑ Clear on Appearance
Refreshing: Enabled
Title: No Font

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



```
//
//  MyTableViewController.m
//
//  Created by CS193p Instructor.
//  Copyright (c) 2013 Stanford University.
//  All rights reserved.
//

#import "MyTableViewController.h"

@interface MyTableViewController ()

@end

@implementation MyTableViewController


@end
```

Simulated Metrics

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

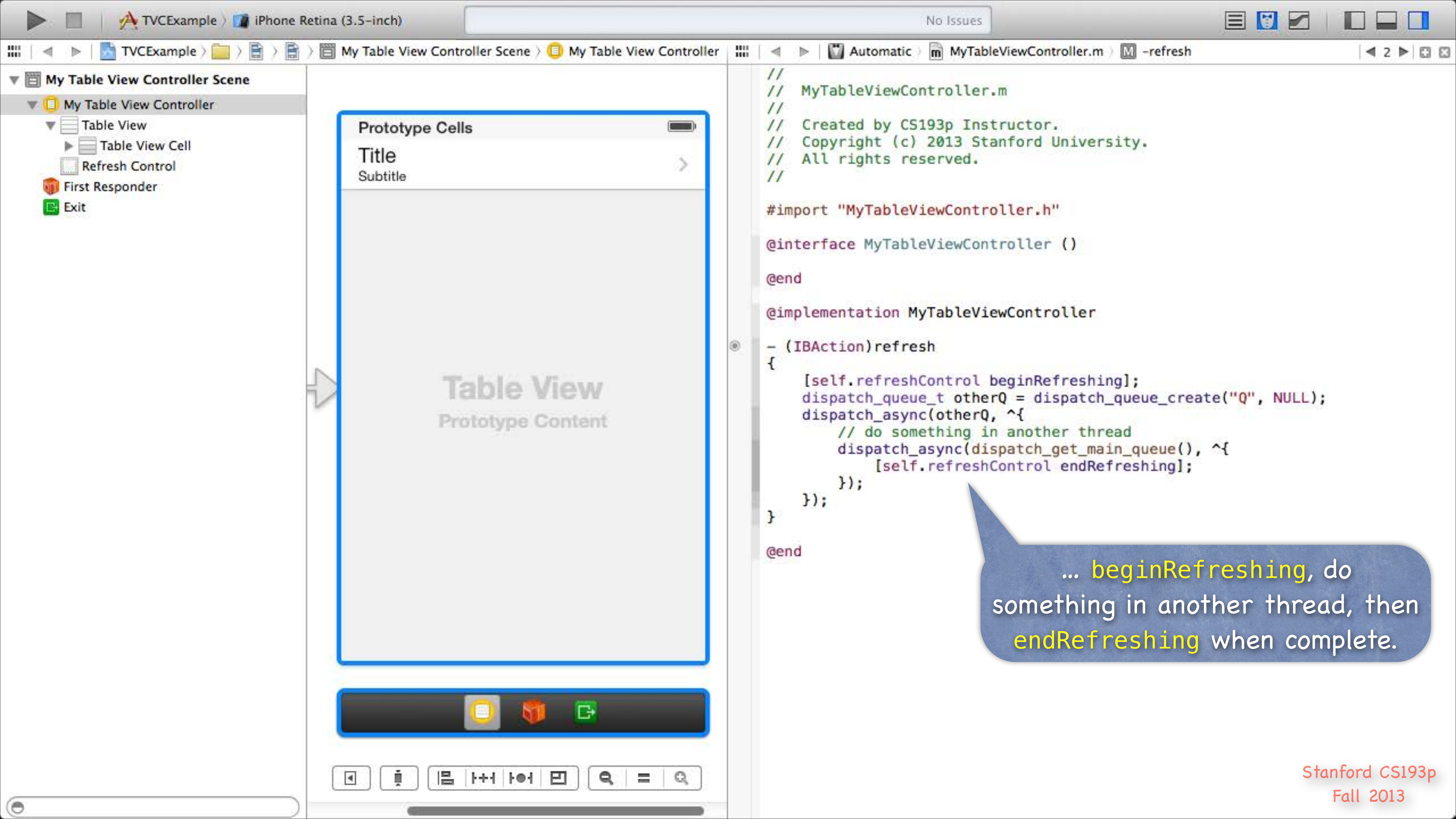
Table View Controller

Selection: ☑ Clear on Appearance
Refreshing: Enabled
Title
No Font

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

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

- (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.