Stanford CS193p

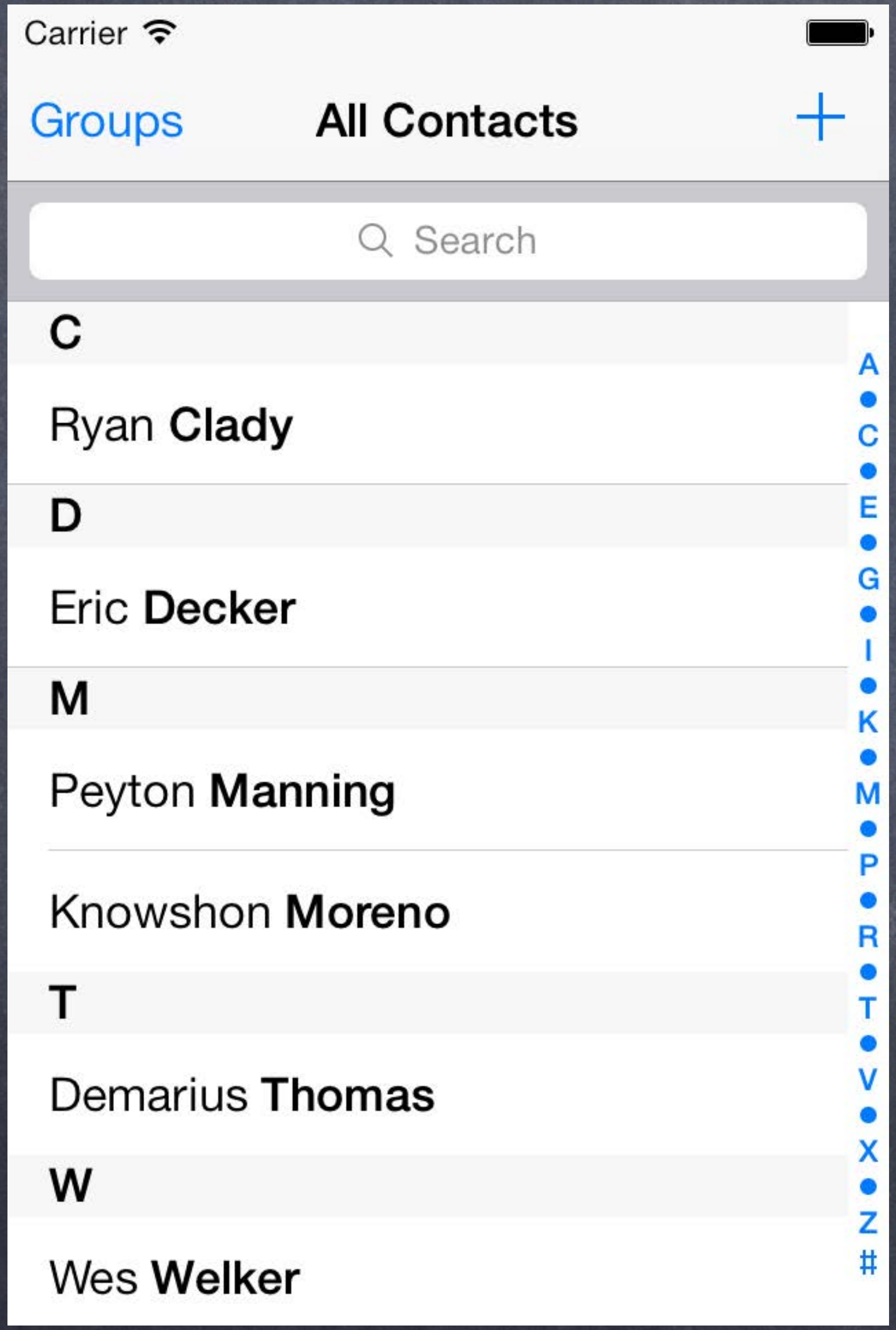
Developing Applications for iOS
Spring 2016

Today

- Table View
  - Way to display large data sets
  - Demo: Twitter Client

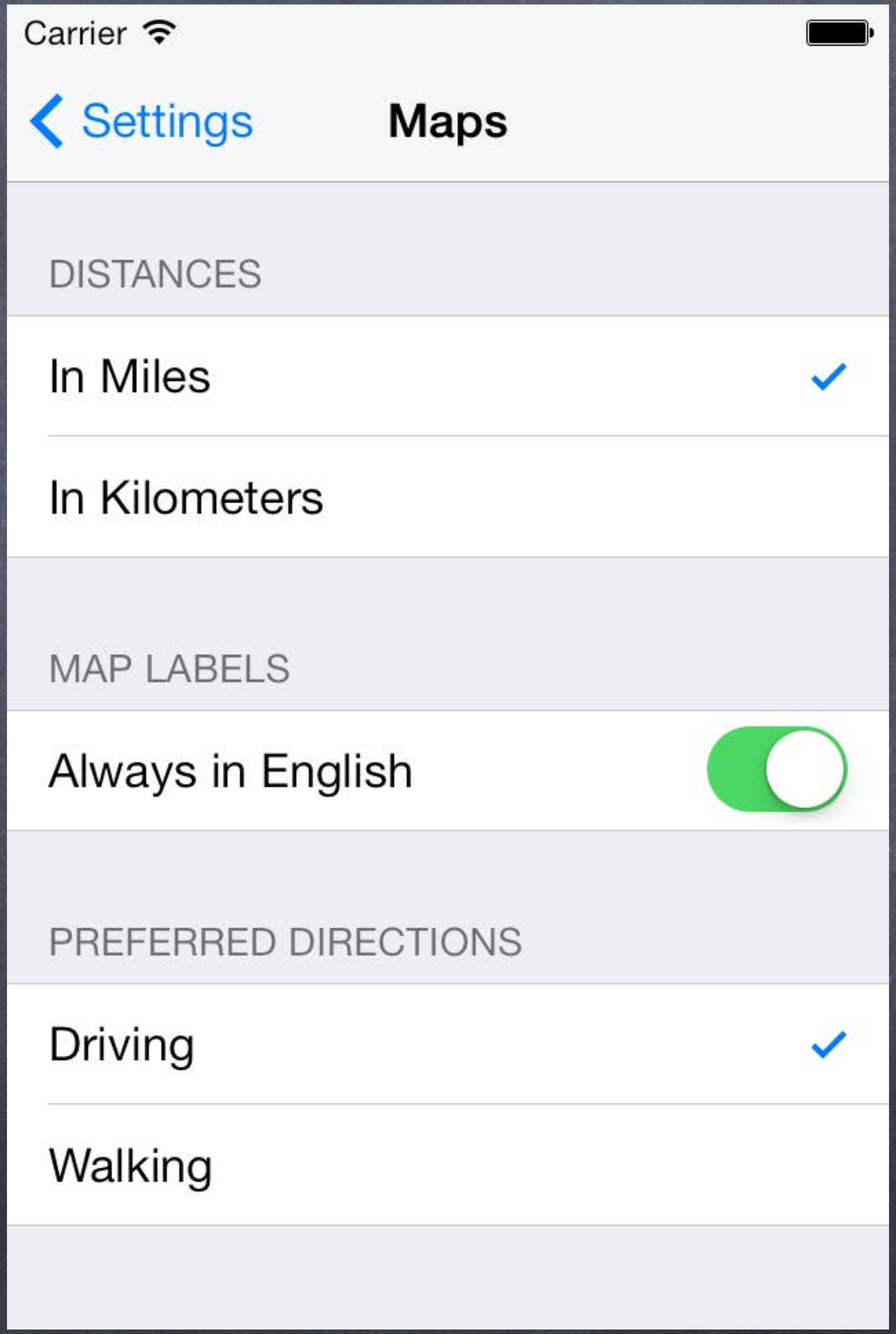
UITableView

`UITableViewStyle.Plain`



Dynamic (List)
& Plain
(ungrouped)

`.Grouped`



Static
& Grouped

UITableView

Plain Style

Table Header

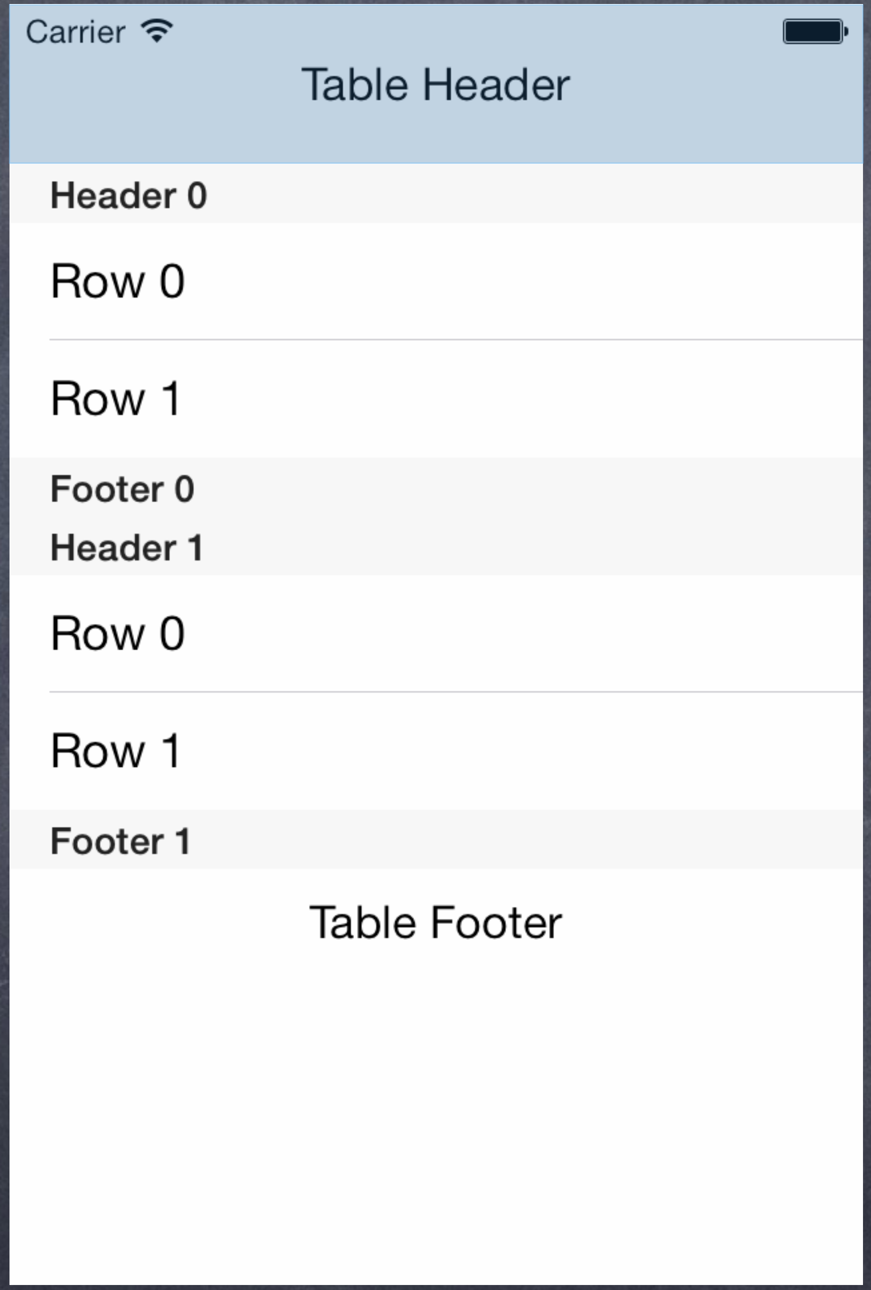

Carrier

Table Header

Header 0

Row 0

Row 1

Footer 0

Header 1

Row 0

Row 1

Footer 1

Table Footer

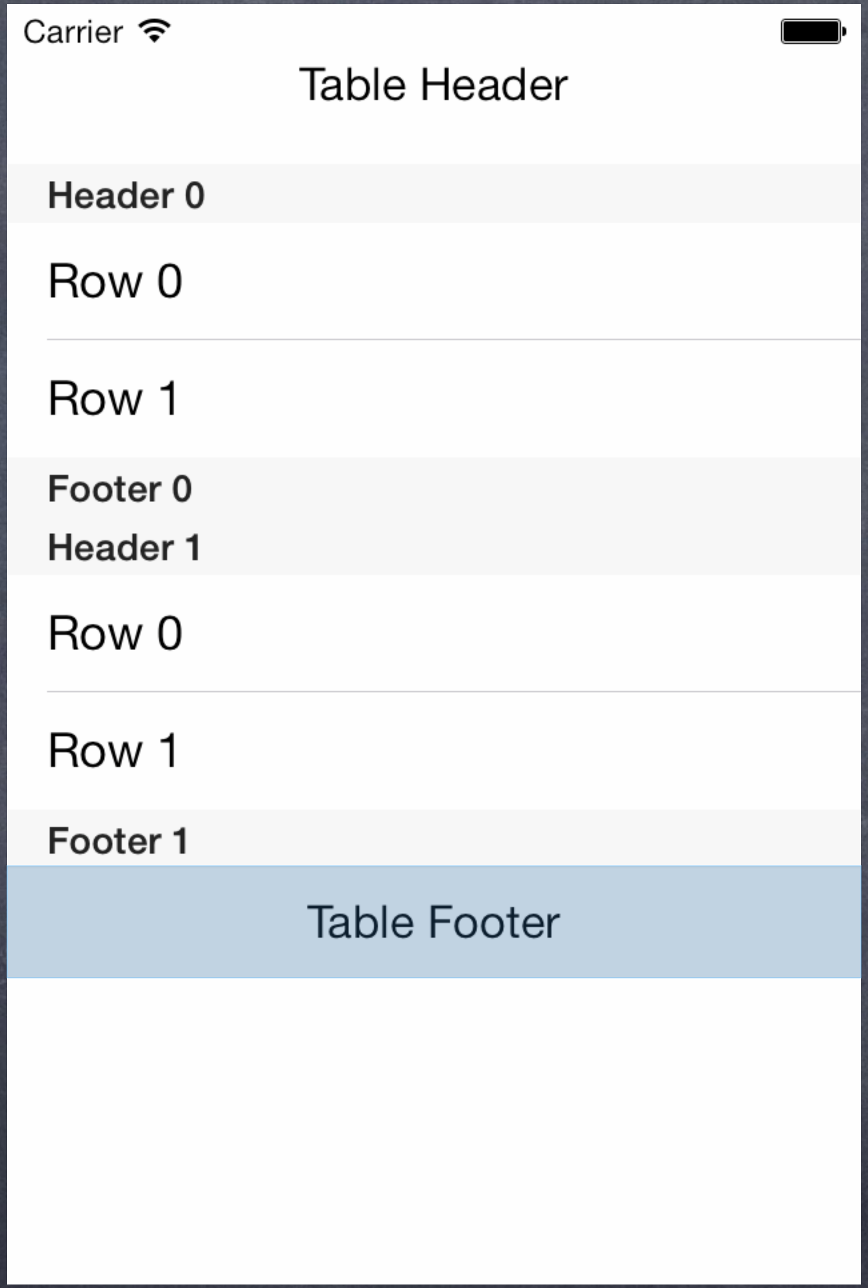
```
var tableHeaderView: UIView
```

UITableView

Plain Style

Table Header

Table Footer



Carrier

Table Header

Header 0

Row 0

Row 1

Footer 0

Header 1

Row 0

Row 1

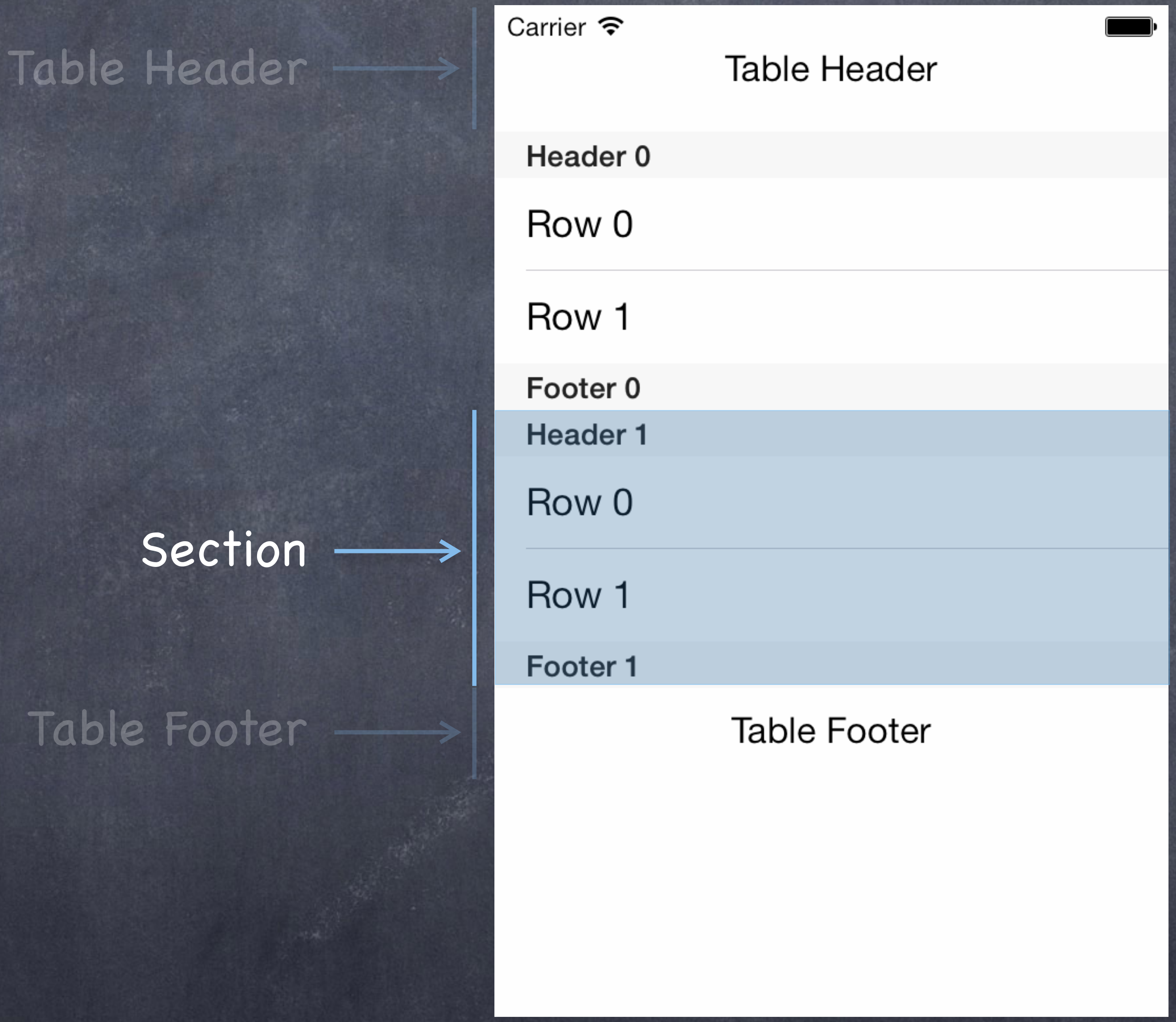
Footer 1

Table Footer

`var tableFooterView: UIView`

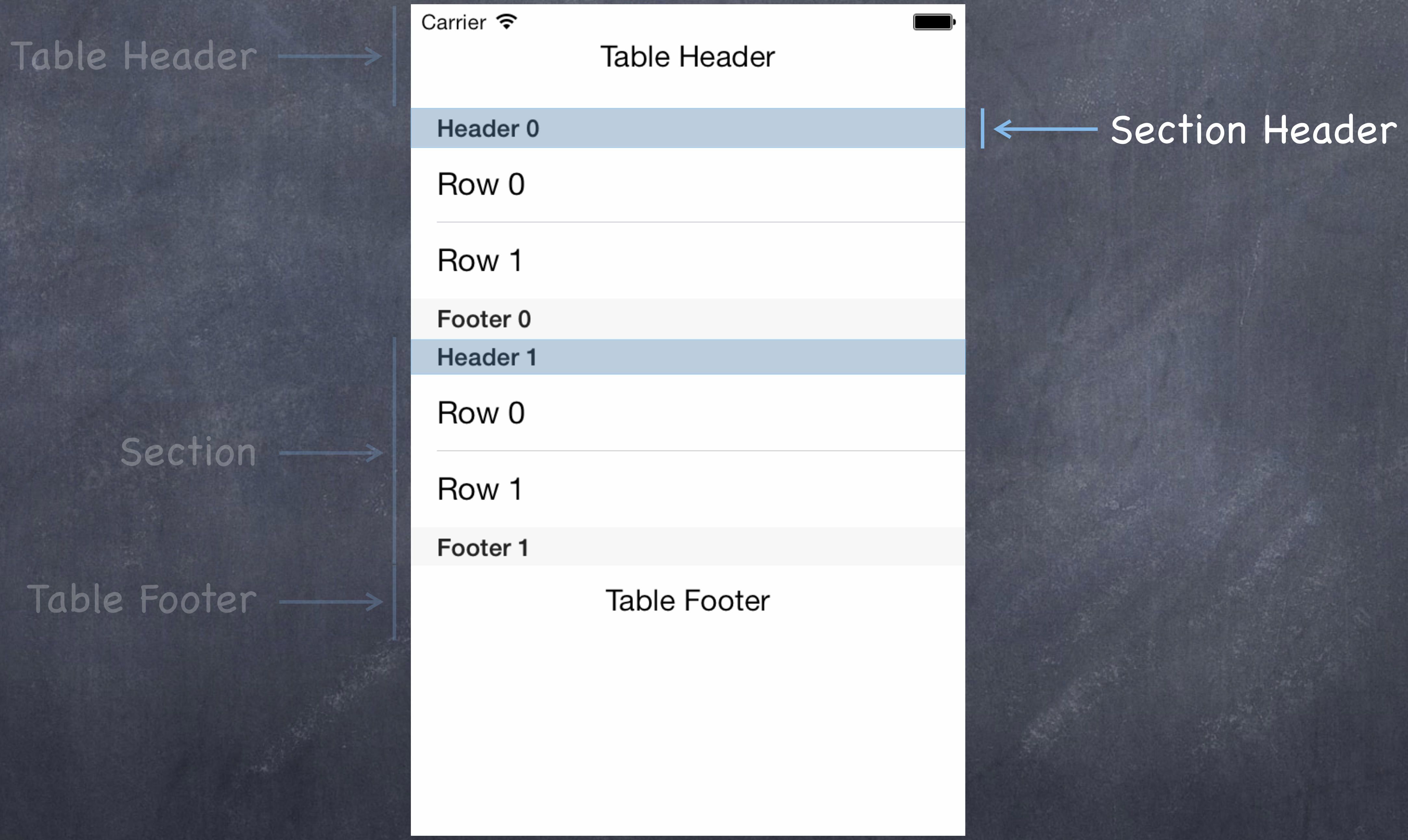
UITableView

Plain Style



UITableView

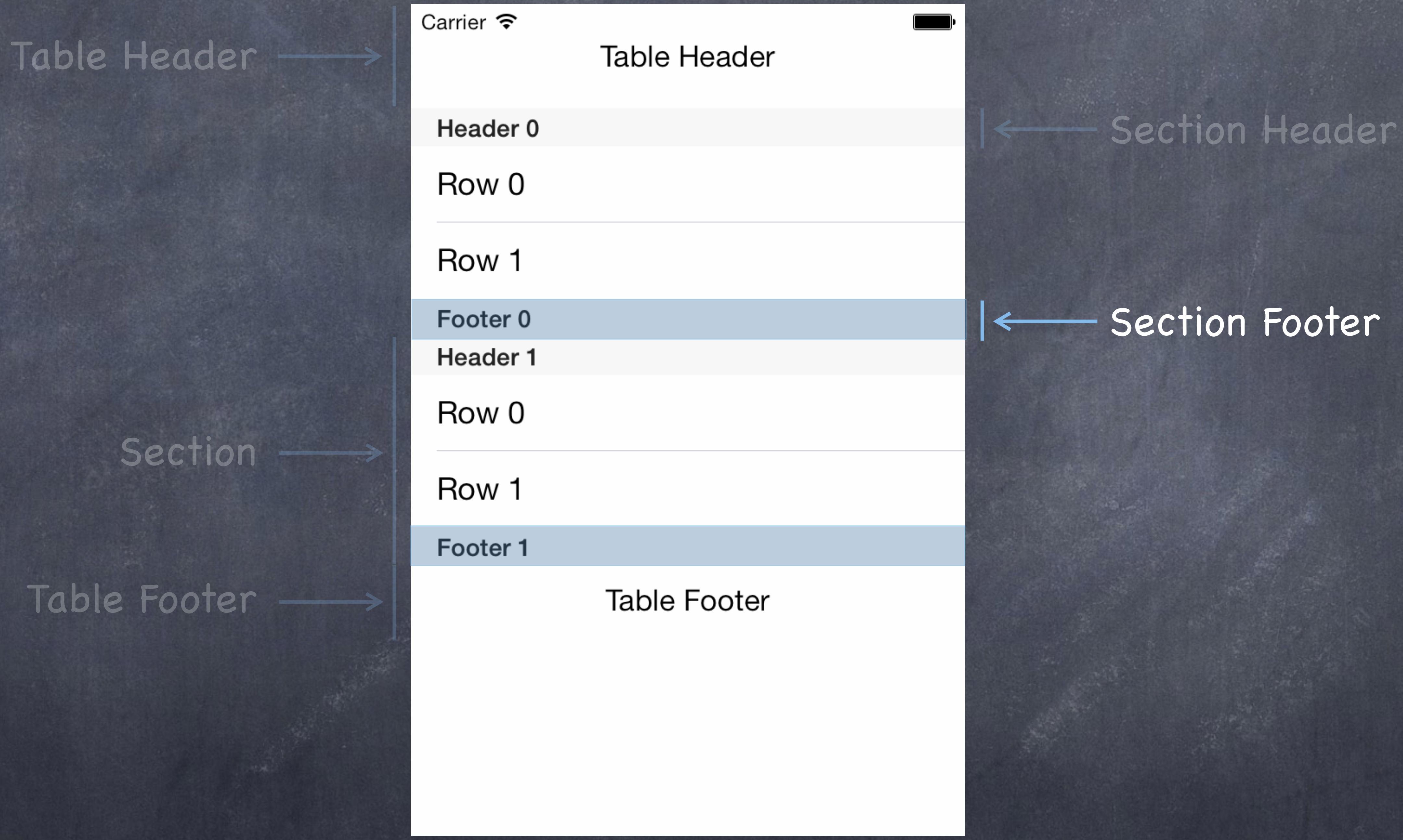
Plain Style



UITableViewDataSource's tableView(UITableView, titleForHeaderInSection: Int)

UITableView

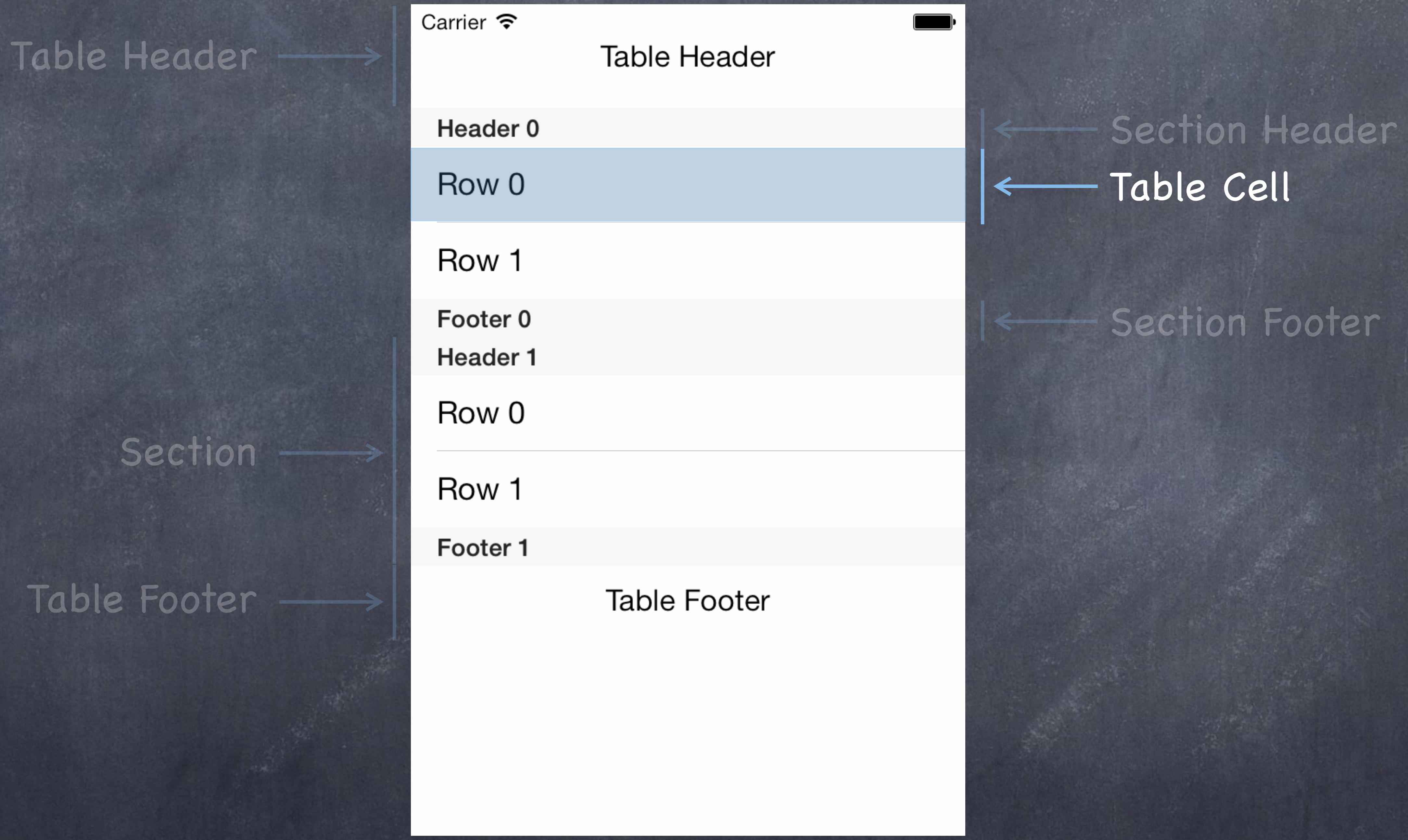
Plain Style



`UITableViewDataSource's tableView(UITableView, titleForFooterInSection: Int)`

UITableView

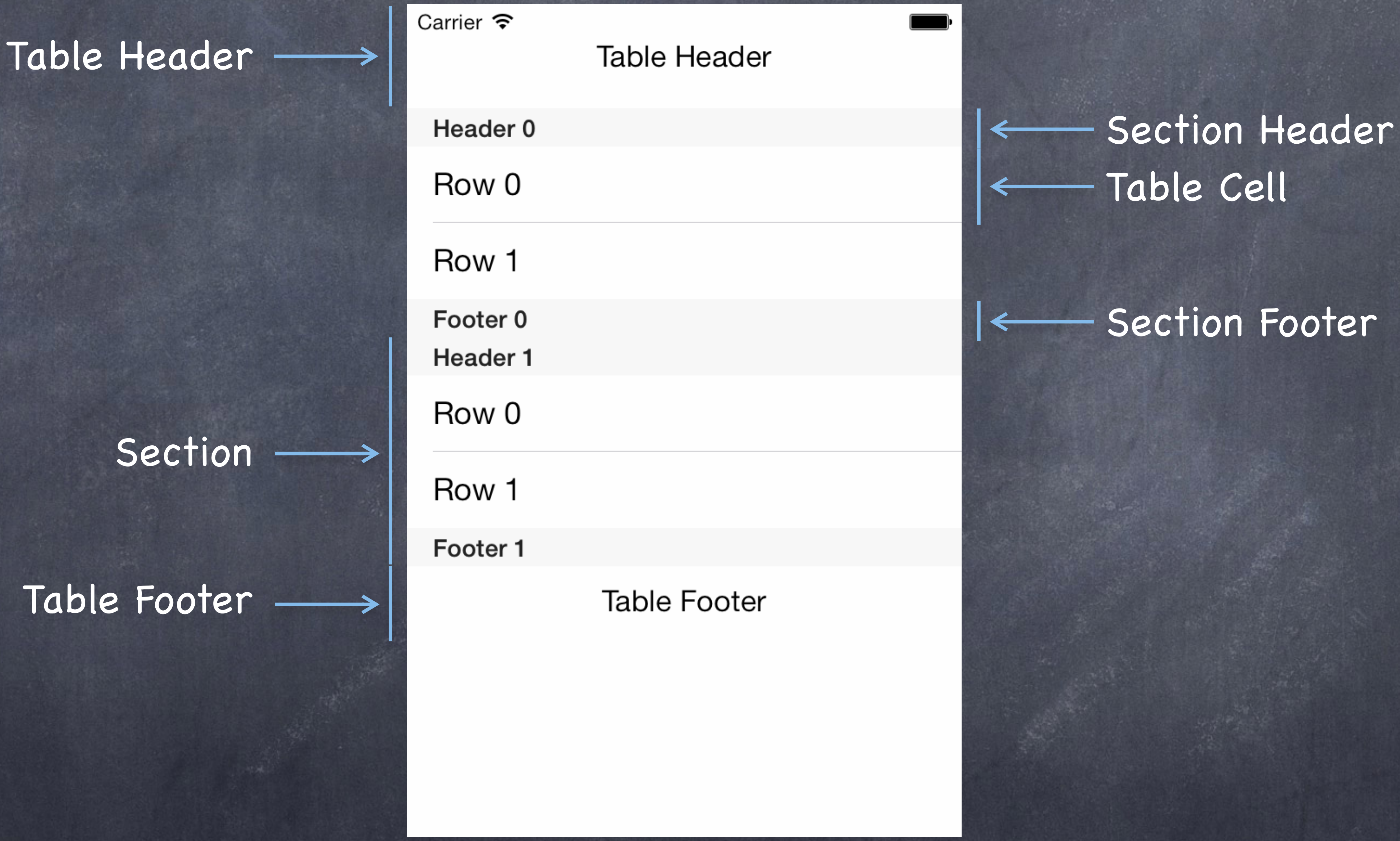
Plain Style



`UITableViewDataSource's tableView(UITableView, cellForRowAtIndexPath: NSIndexPath)`

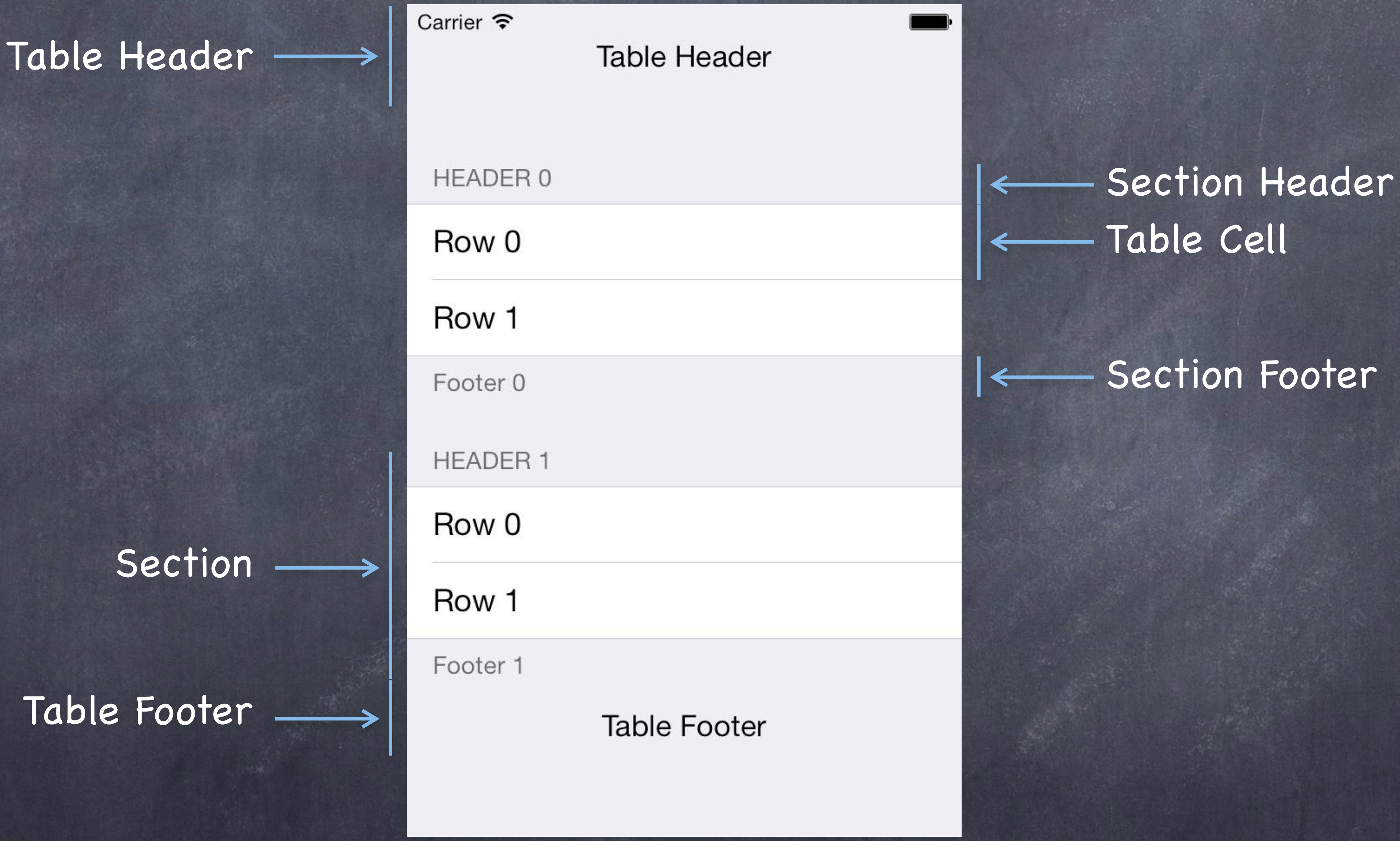
UITableView

Plain Style

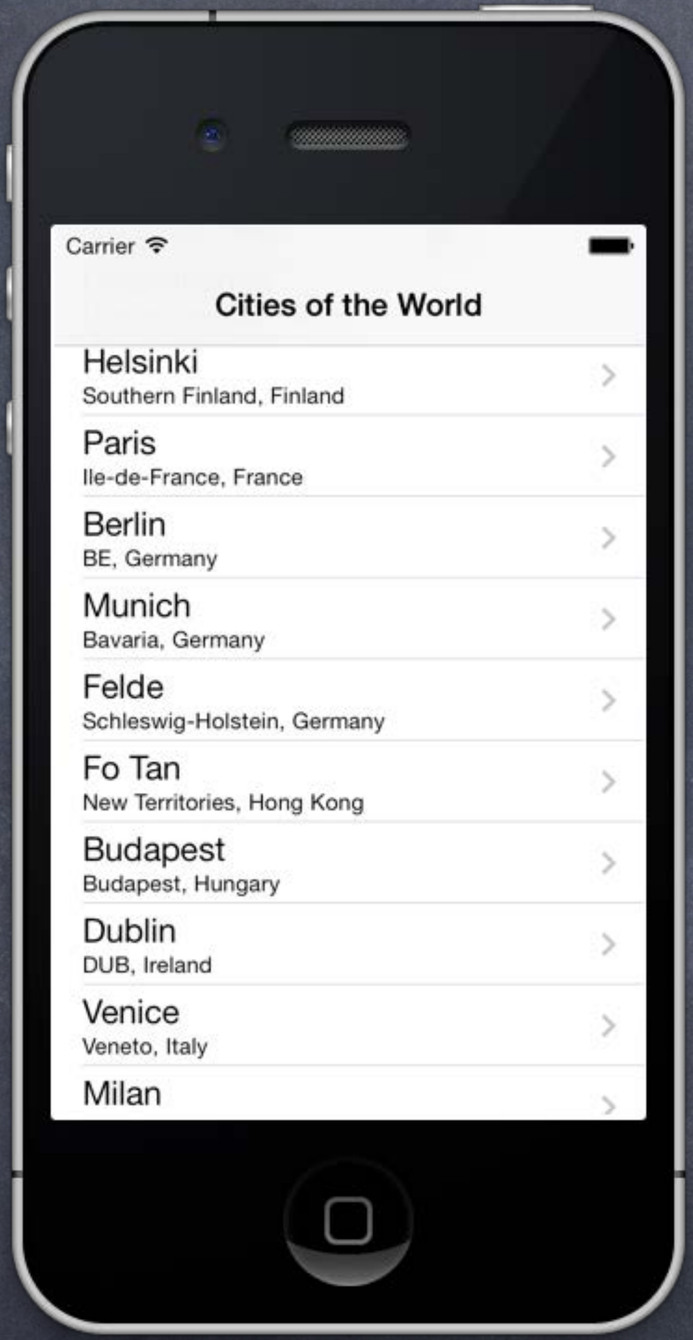


UITableView

Grouped Style



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

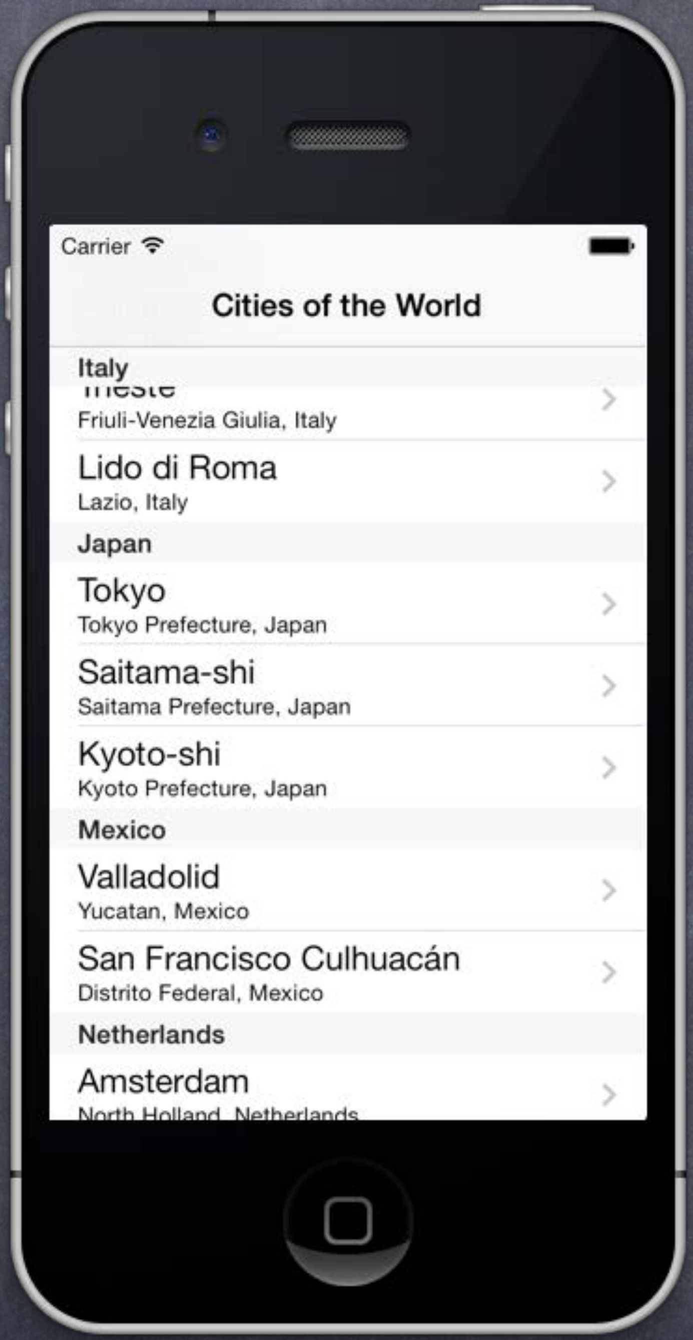
Budapest
Budapest, Hungary

Dublin
DUB, Ireland

Venice
Veneto, Italy

Milan

No Sections



Carrier

Cities of the World

Italy

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

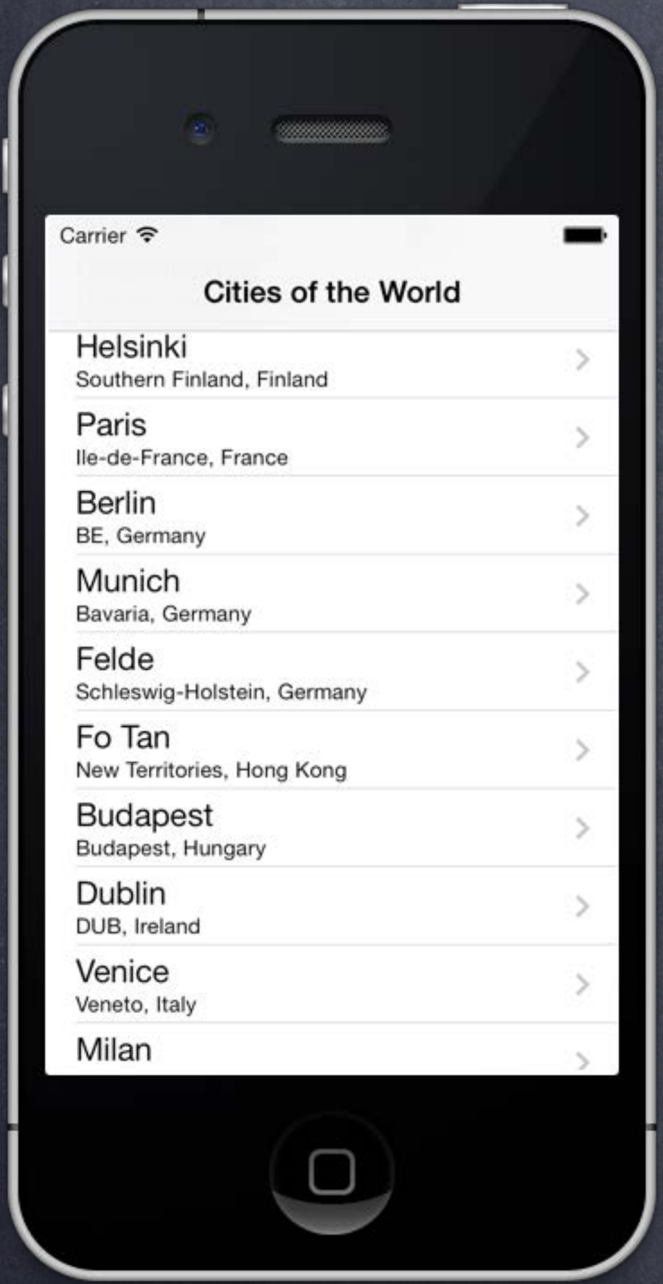
San Francisco Culhuacán
Distrito Federal, Mexico

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

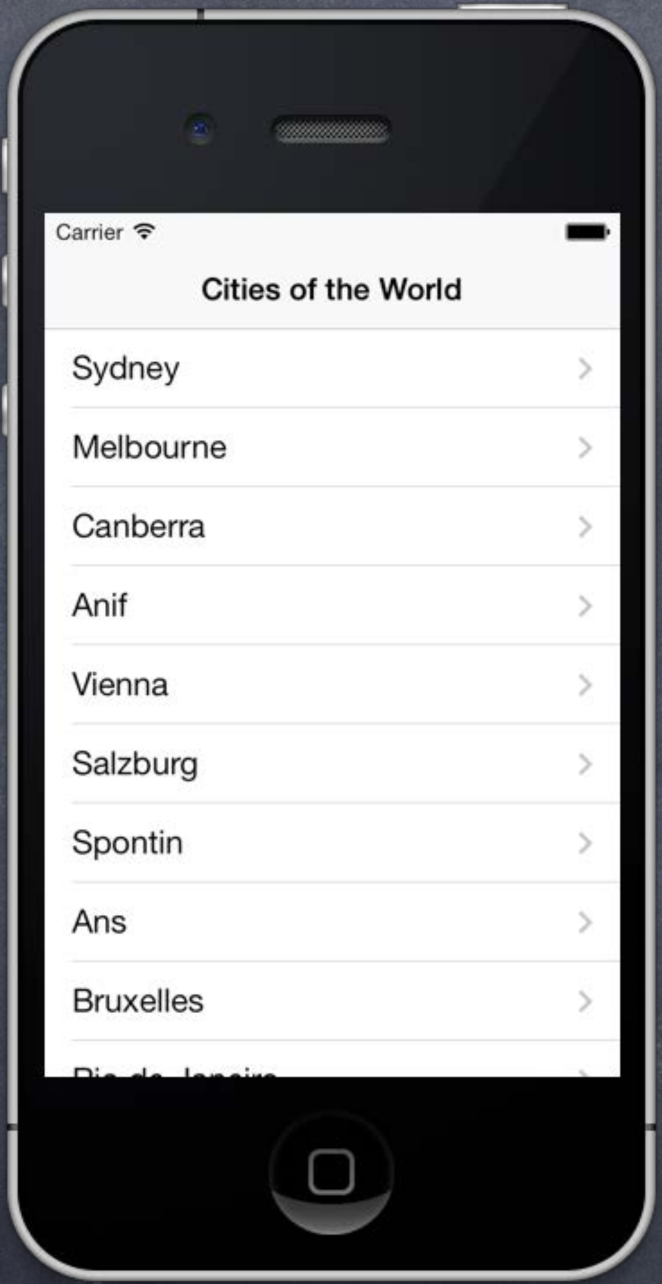
Dublin
DUB, Ireland

Venice
Veneto, Italy

Milan

Subtitle

`UITableViewCellStyle.Subtitle`


Carrier

Cities of the World

Sydney

Melbourne

Canberra

Anif

Vienna

Salzburg

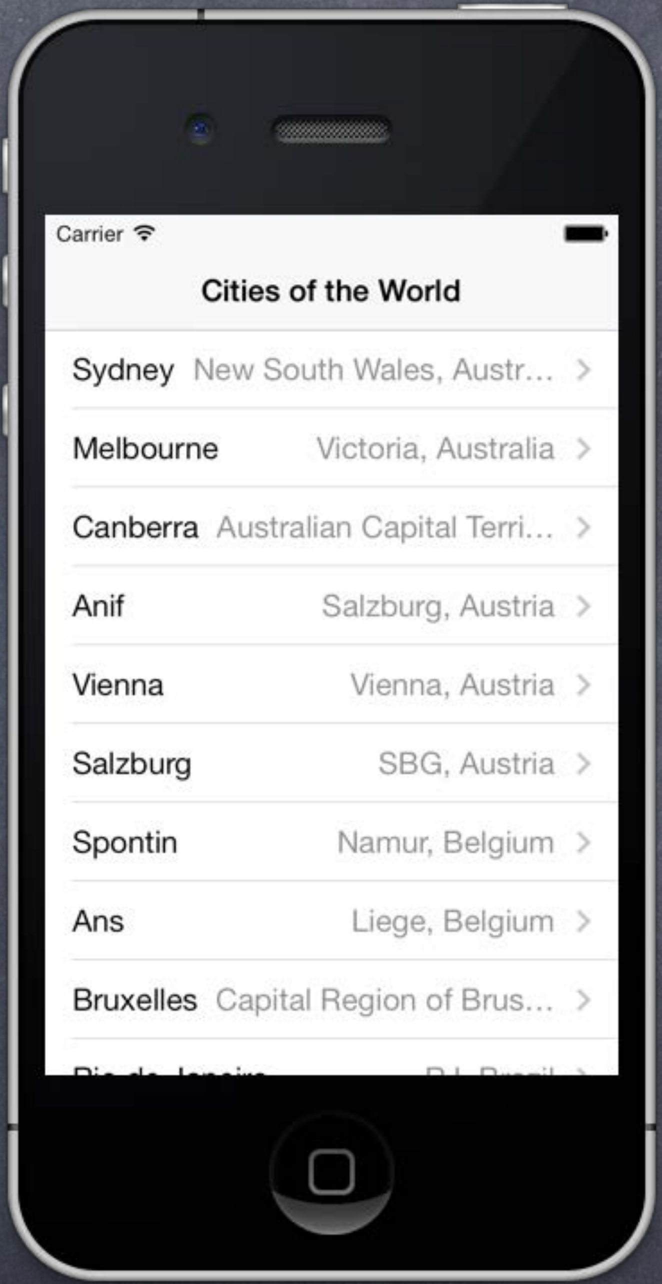
Spontin

Ans

Bruxelles

Basic

`.Default`

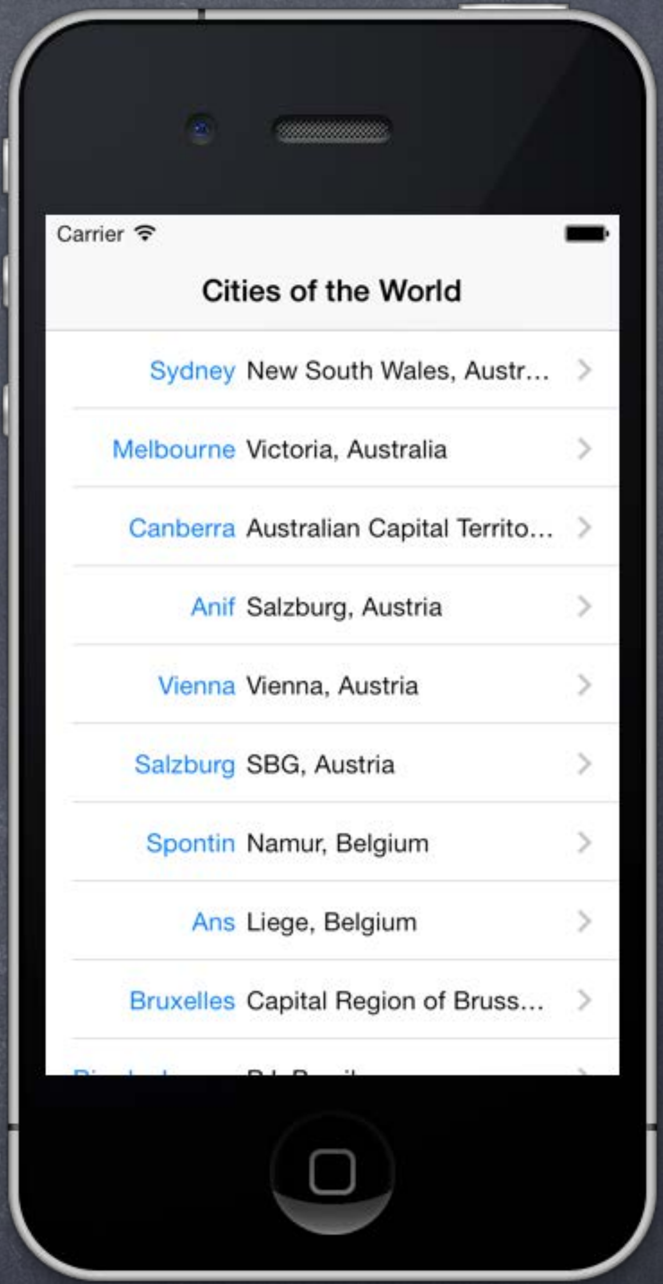

Carrier

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...

Right Detail

`.Value1`

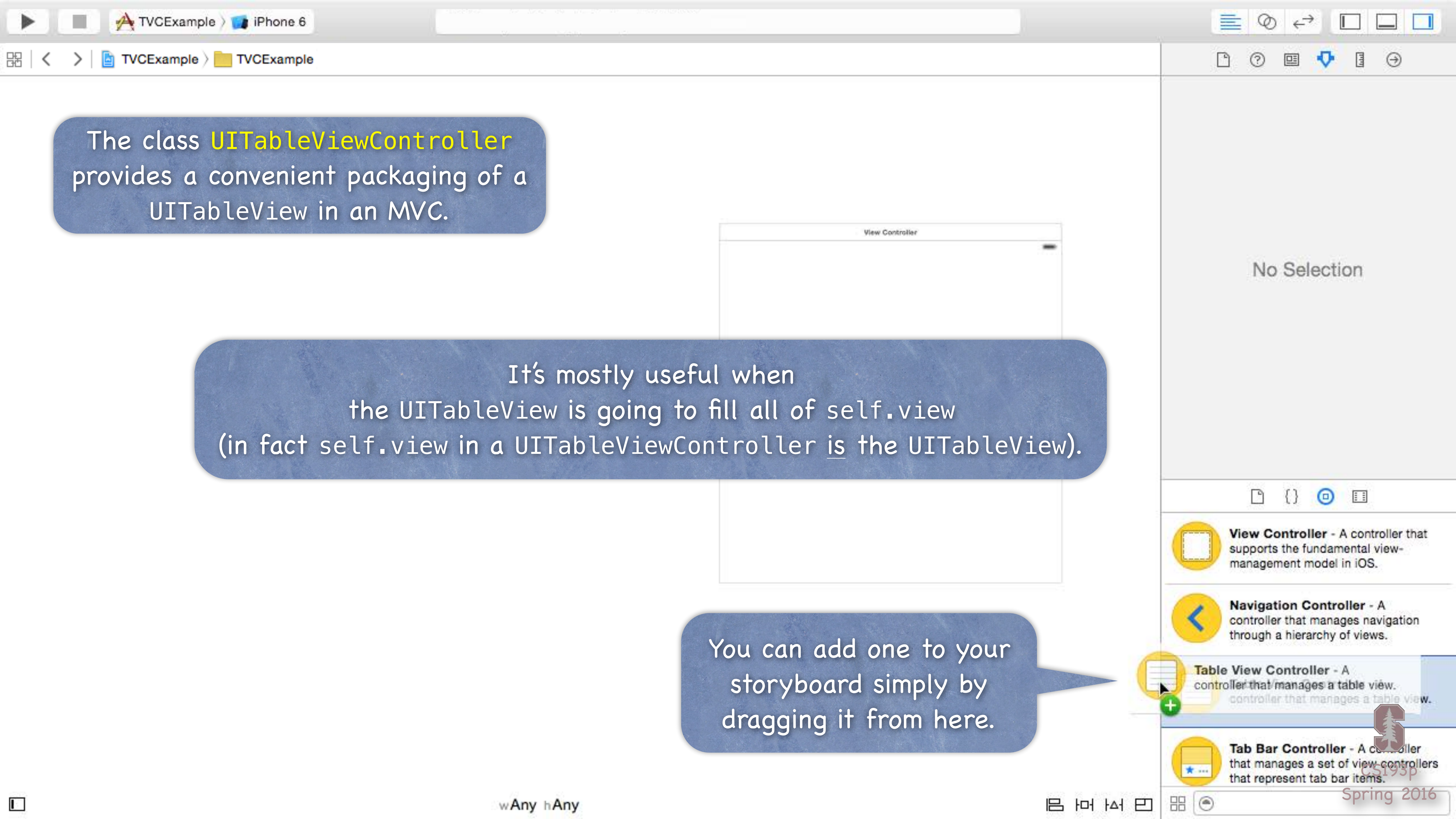

Carrier

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

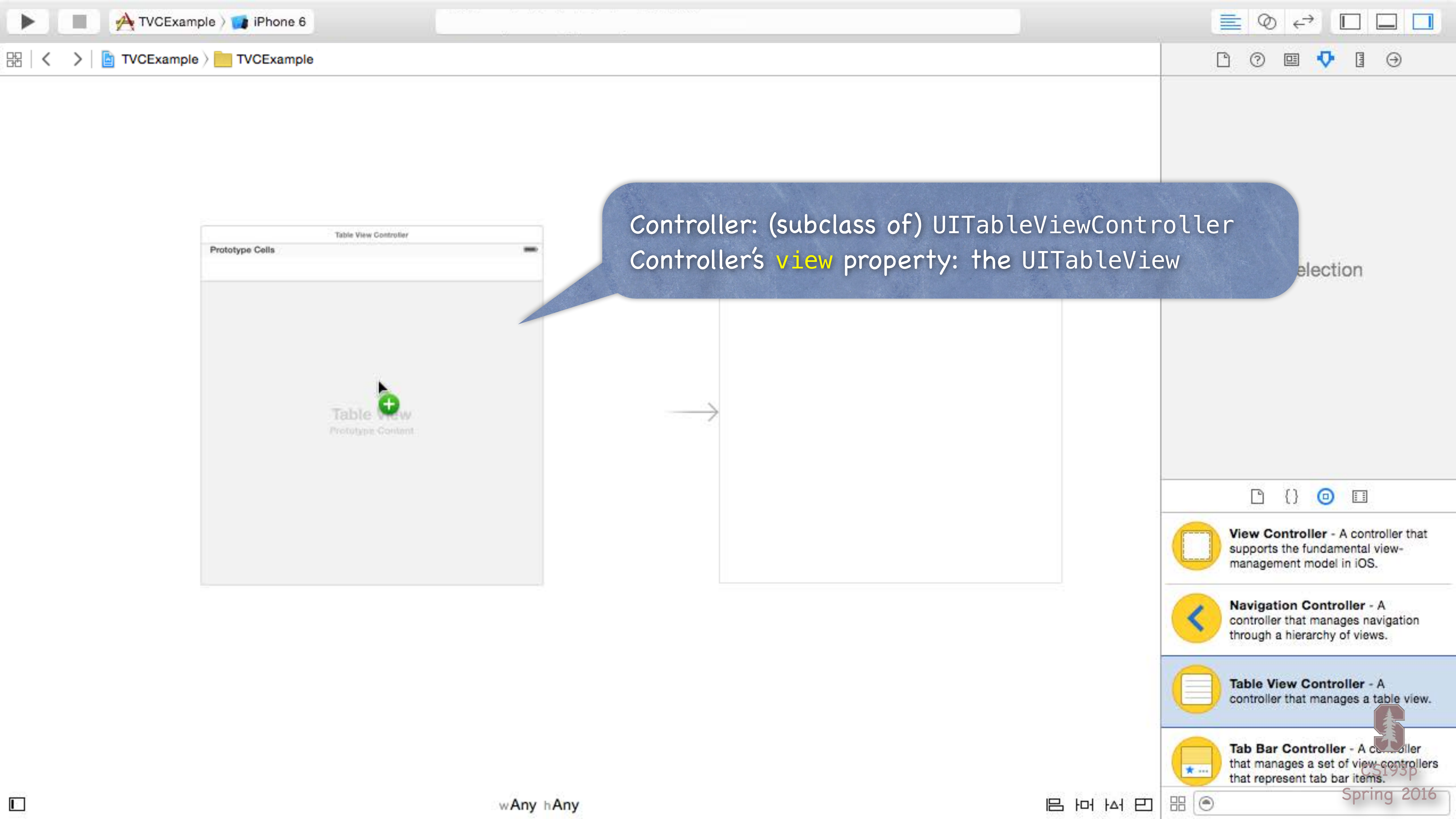
`.Value2`


TVCExample
iPhone 6
TVCExample
TVCExample
View Controller
No Selection
View Controller - A controller that supports the fundamental view-management model in iOS.
Navigation Controller - A controller that manages navigation through a hierarchy of views.
Table View Controller - A controller that manages a table view.
Tab Bar Controller - A controller that manages a set of view controllers that represent tab bar items.
wAny hAny

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).

You can add one to your storyboard simply by dragging it from here.



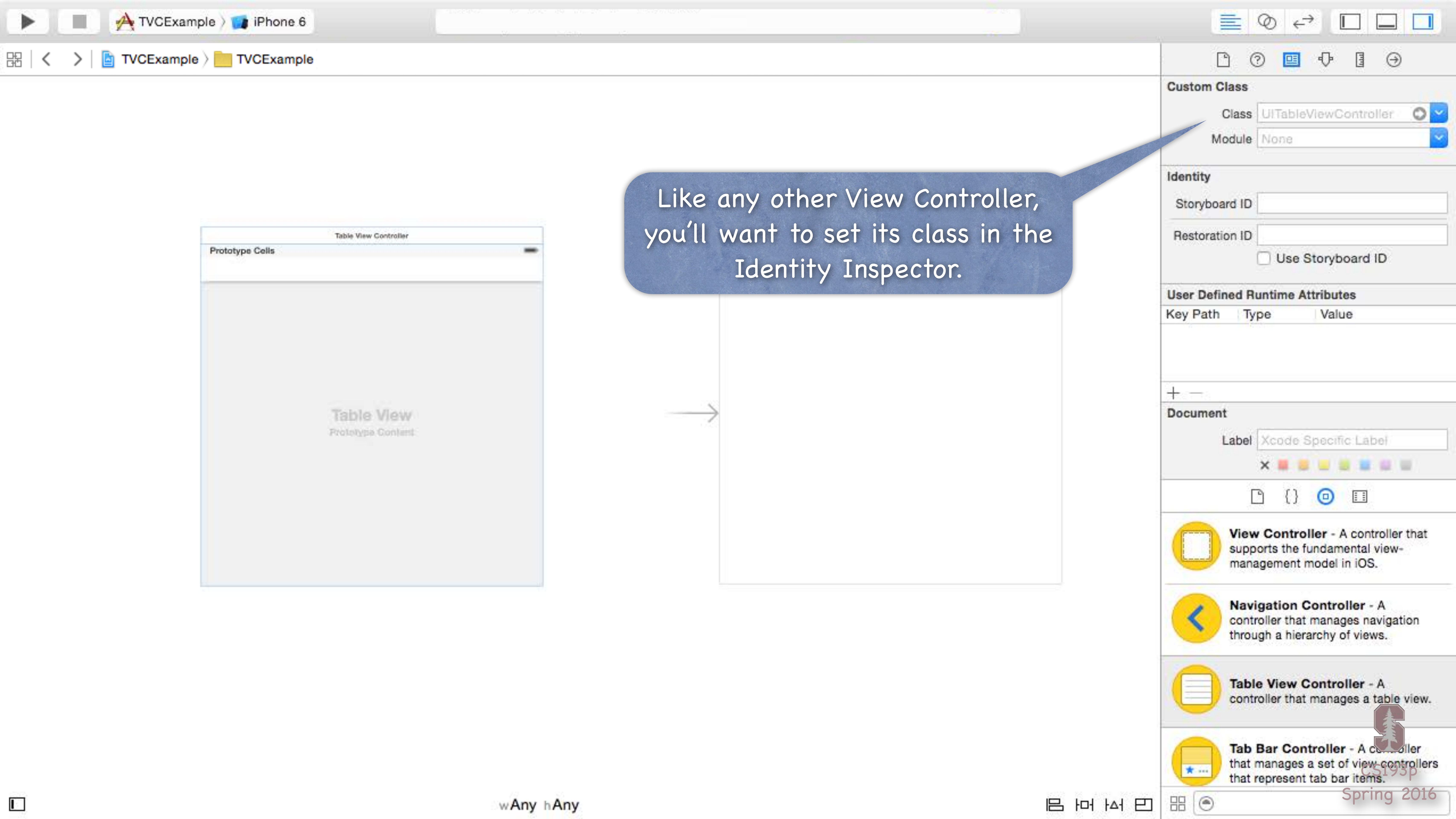
Controller: (subclass of) UITableViewController
Controller's view property: the UITableView

View Controller - A controller that supports the fundamental view-management model in iOS.

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

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

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



Like any other View Controller, you'll want to set its class in the Identity Inspector.

Custom Class

Class UITableViewController

Module None

Identity

Storyboard ID

Restoration ID

Use Storyboard ID

User Defined Runtime Attributes

Key Path	Type	Value

Document

Label Xcode Specific Label

View Controller - A controller that supports the fundamental view-management model in iOS.

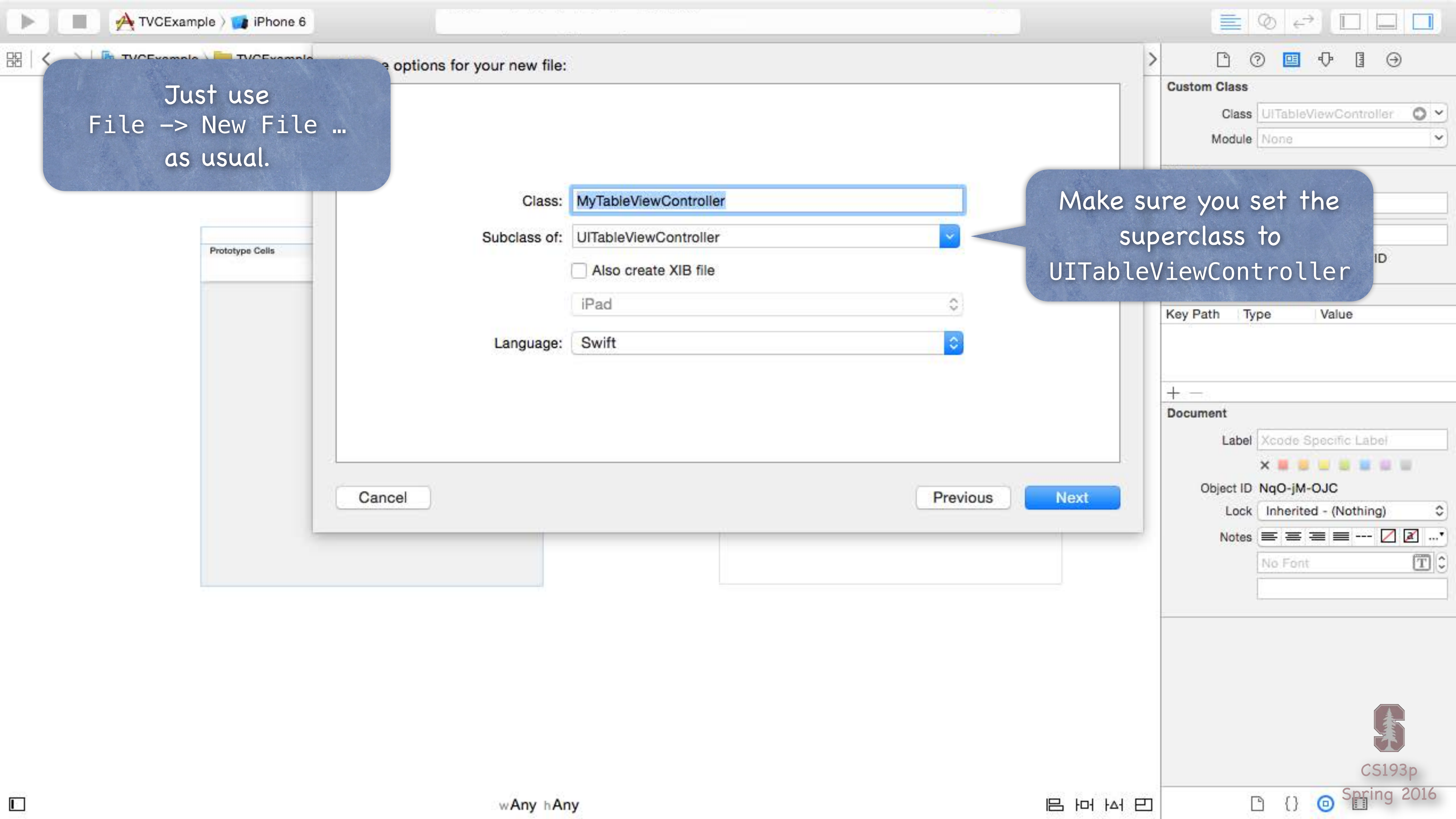
Navigation Controller - A controller that manages navigation through a hierarchy of views.

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

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

Just use
File -> New File …
as usual.

Make sure you set the superclass to UITableViewController



…e options for your new file:

Class: MyTableViewController

Subclass of: UITableViewController

☐ Also create XIB file

iPad

Language: Swift

Cancel | Previous | Next

Custom Class

Class: UITableViewController

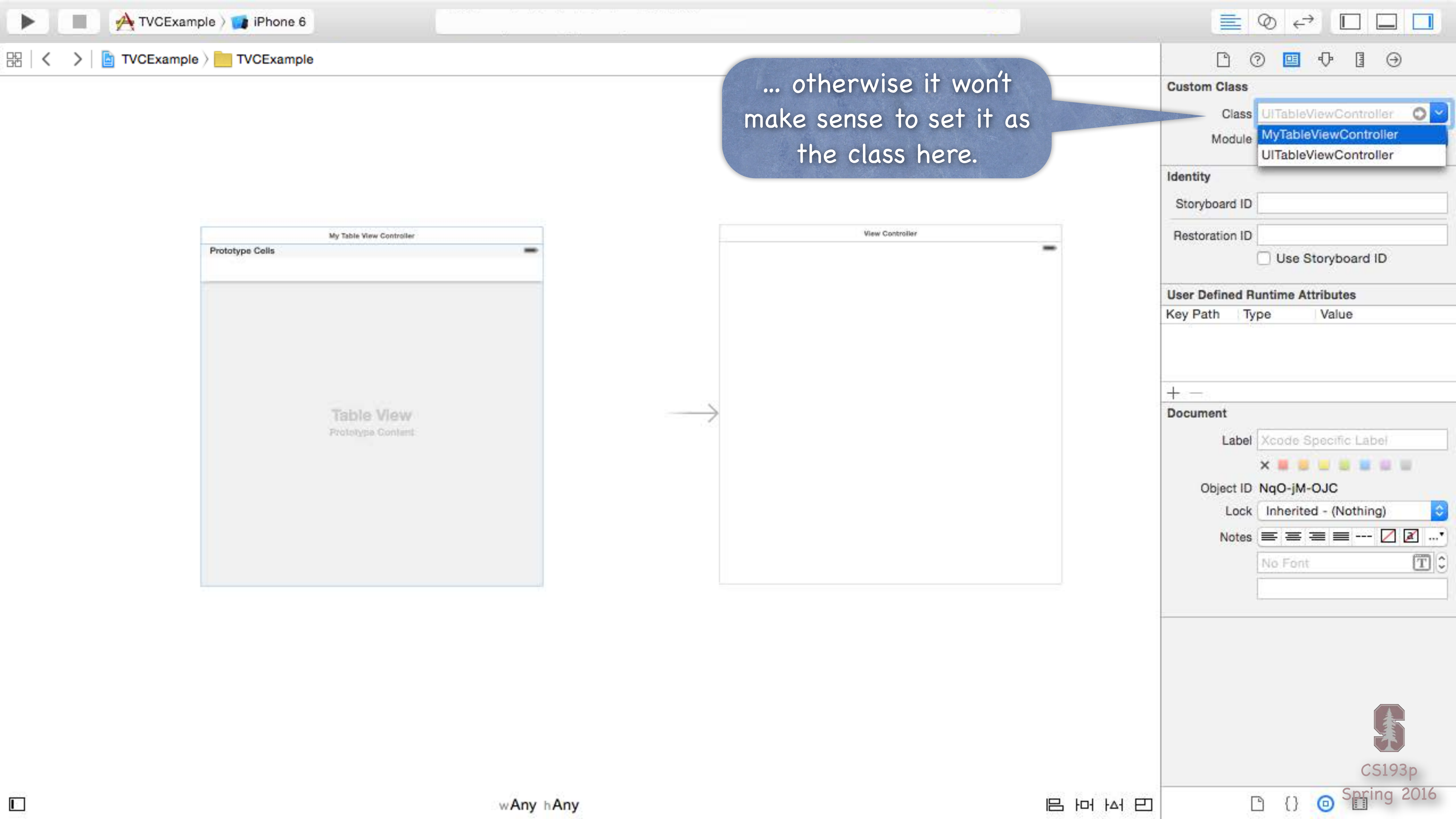
Module: None

Document

Label: Xcode Specific Label

Object ID: NqO-jM-OJC

Lock: Inherited - (Nothing)



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

Custom Class

Class: UITableViewController

Module

MyTableViewController

UITableViewController

Identity

Storyboard ID

Restoration ID

Use Storyboard ID

User Defined Runtime Attributes

Key Path	Type	Value

Document

Label: Xcode Specific Label

Object ID: NqO-jM-OJC

Lock: Inherited - (Nothing)

Notes

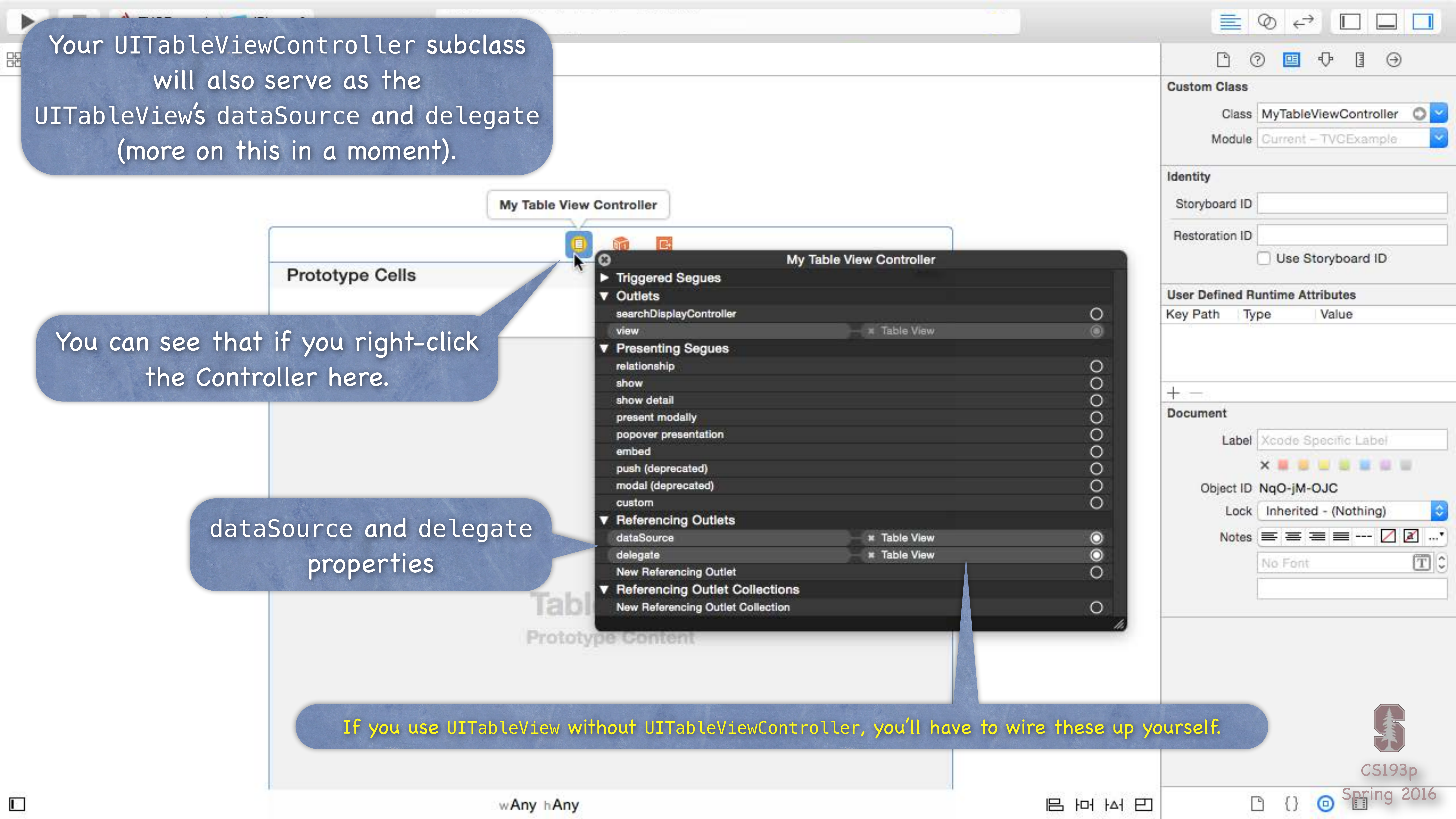
No Font

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 properties

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



My Table View Controller

Prototype Cells

My Table View Controller
- ▶ Triggered Segues
- ▼ Outlets
  - searchDisplayController
  - view — Table View
- ▼ Presenting Segues
  - relationship
  - show
  - show detail
  - present modally
  - popover presentation
  - embed
  - push (deprecated)
  - modal (deprecated)
  - custom
- ▼ Referencing Outlets
  - dataSource — Table View
  - delegate — Table View
  - New Referencing Outlet
- ▼ Referencing Outlet Collections
  - New Referencing Outlet Collection

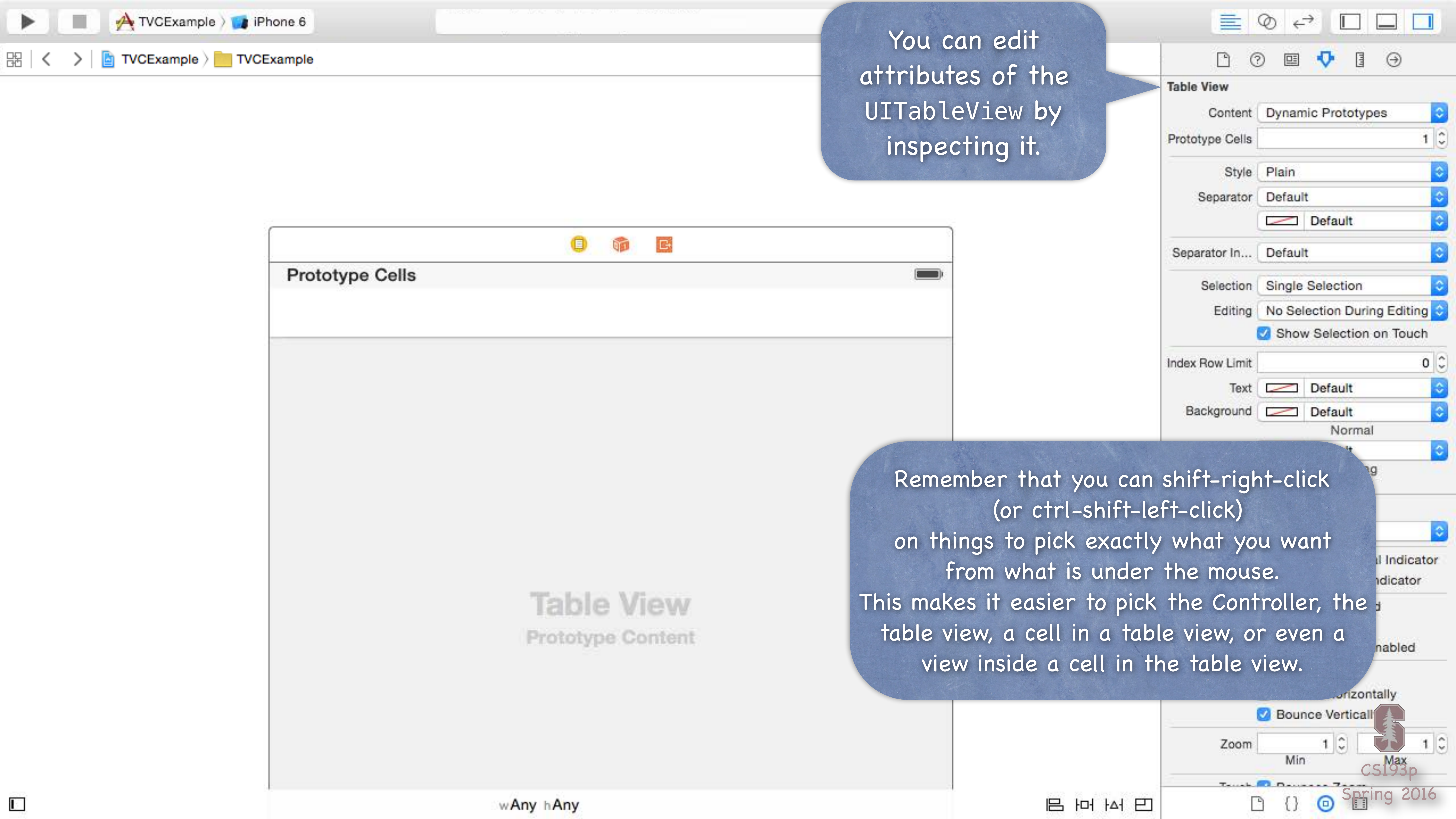
Custom Class
Class: MyTableViewController
Module: Current – TVCExample

Identity
Storyboard ID:
Restoration ID:
Use Storyboard ID

User Defined Runtime Attributes

Key Path	Type	Value

Document
Label: Xcode Specific Label
Object ID: NqO-jM-OJC
Lock: Inherited - (Nothing)
Notes: No Font

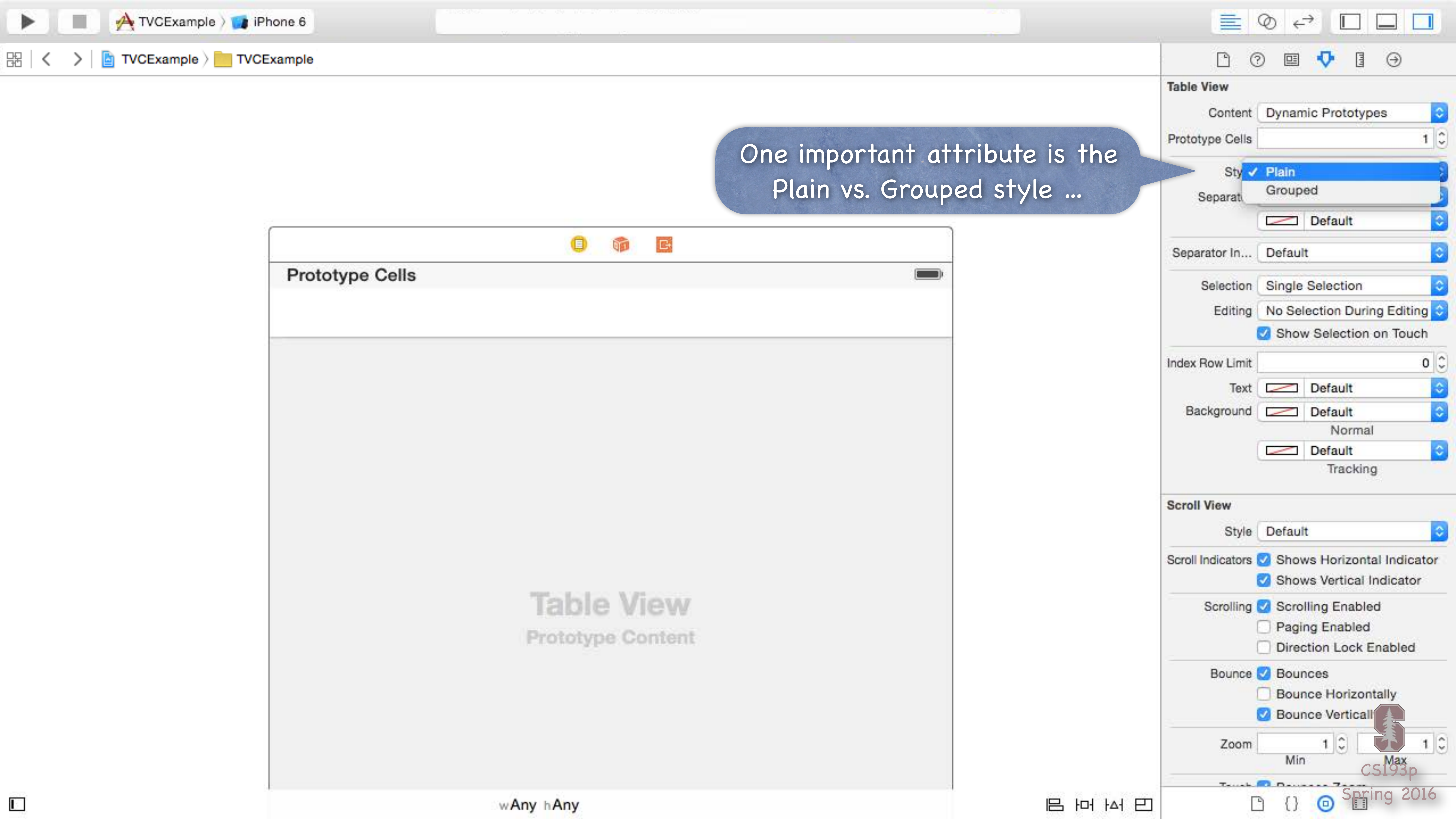


You can edit attributes of the UITableView by inspecting it.

Table View

Content: Dynamic Prototypes
Prototype Cells: 1
Style: Plain
Separator: Default
Default
Separator In...: Default
Selection: Single Selection
Editing: No Selection During Editing
✓ Show Selection on Touch
Index Row Limit: 0
Text: Default
Background: Default
Normal
✓ Bounce Verticall
Zoom: 1 Min, 1 Max

Remember that you can shift-right-click (or ctrl-shift-left-click) on things to pick exactly what you want from what is under the mouse.
This makes it easier to pick the Controller, the table view, a cell in a table view, or even a view inside a cell in the table view.



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

TVCExample > iPhone 6

TVCExample > TVCExample

Prototype Cells

Table View
Prototype Content

wAny hAny

Table View

Content: Dynamic Prototypes

Prototype Cells: 1

Style: [✓ Plain] Grouped

Separator: Default

Separator Inset: Default

Selection: Single Selection

Editing: No Selection During Editing

[✓] Show Selection on Touch

Index Row Limit: 0

Text: Default

Background: Default (Normal)

Default (Tracking)

Scroll View

Style: Default

Scroll Indicators: [✓] Shows Horizontal Indicator, [✓] Shows Vertical Indicator

Scrolling: [✓] Scrolling Enabled, [ ] Paging Enabled, [ ] Direction Lock Enabled

Bounce: [✓] Bounces, [ ] Bounce Horizontally, [✓] Bounce Vertically

Zoom: Min 1, Max 1

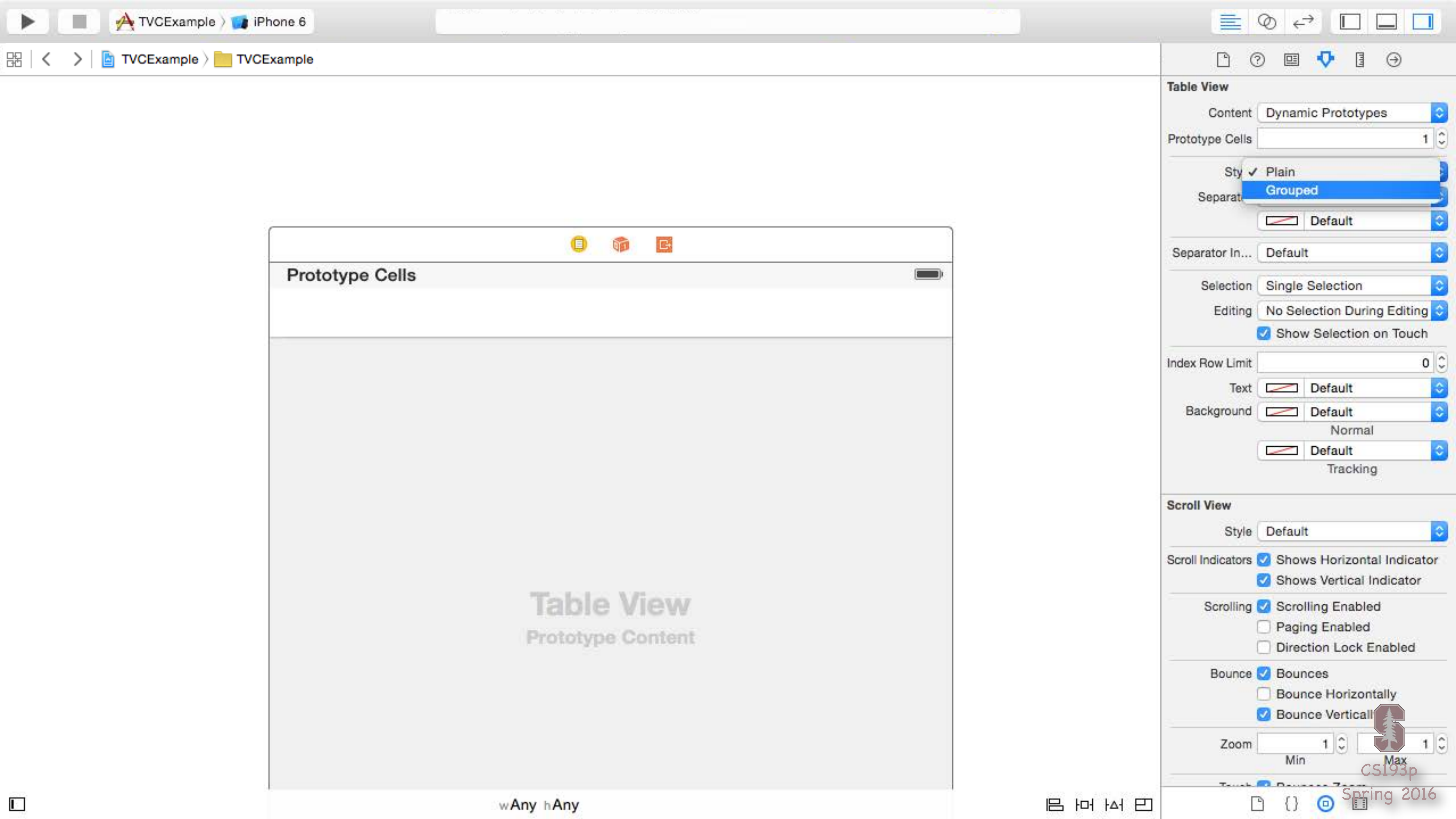


Table View

Content: Dynamic Prototypes

Prototype Cells: 1

Style: ✓ Plain / Grouped

Separator: Default

Separator Inset: Default

Selection: Single Selection

Editing: No Selection During Editing

[✓] Show Selection on Touch

Index Row Limit: 0

Text: Default

Background: Default (Normal)

Default (Tracking)

Scroll View

Style: Default

Scroll Indicators: [✓] Shows Horizontal Indicator [✓] Shows Vertical Indicator

Scrolling: [✓] Scrolling Enabled [ ] Paging Enabled [ ] Direction Lock Enabled

Bounce: [✓] Bounces [ ] Bounce Horizontally [✓] Bounce Vertically

Zoom: 1 Min, 1 Max

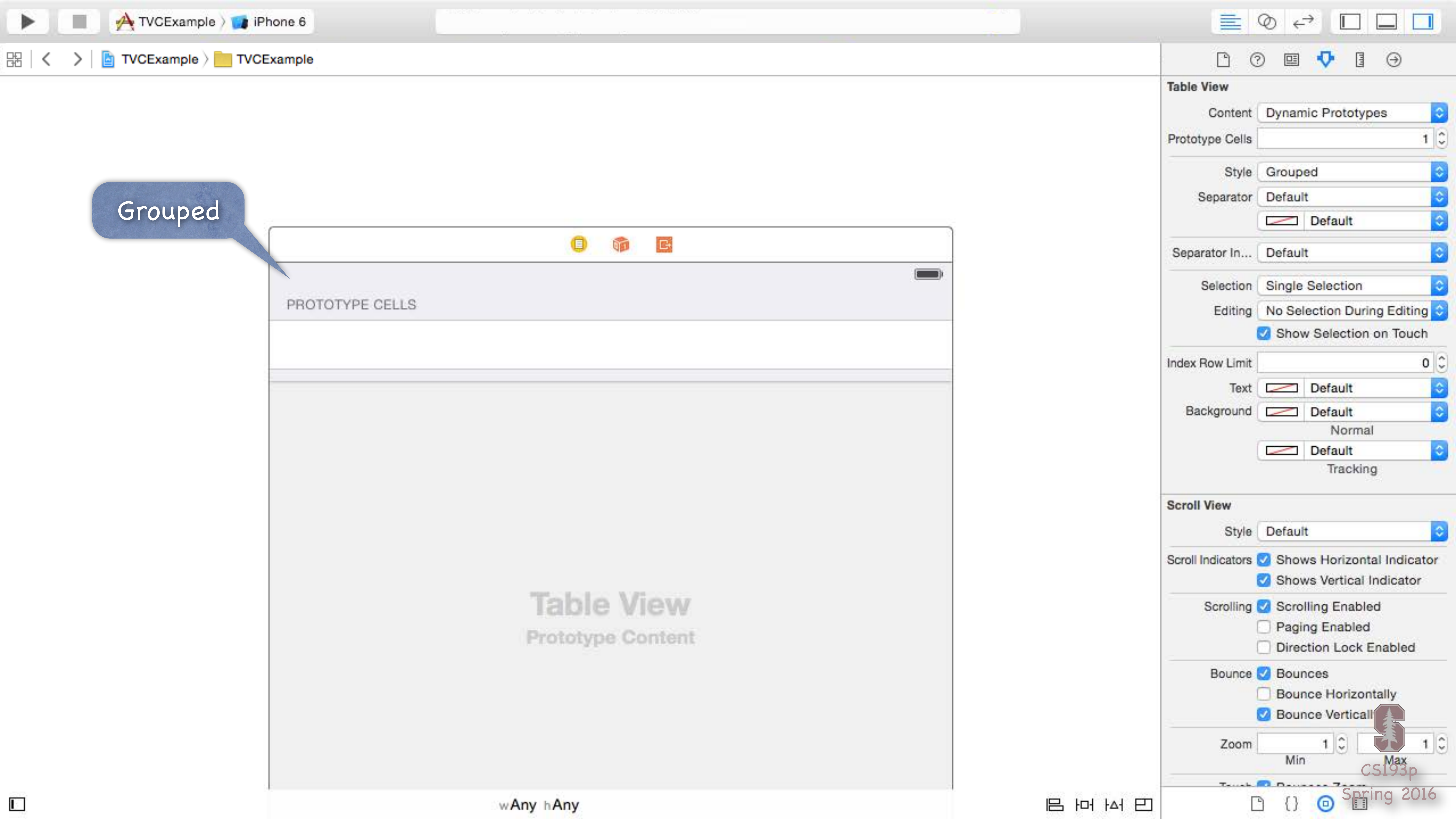
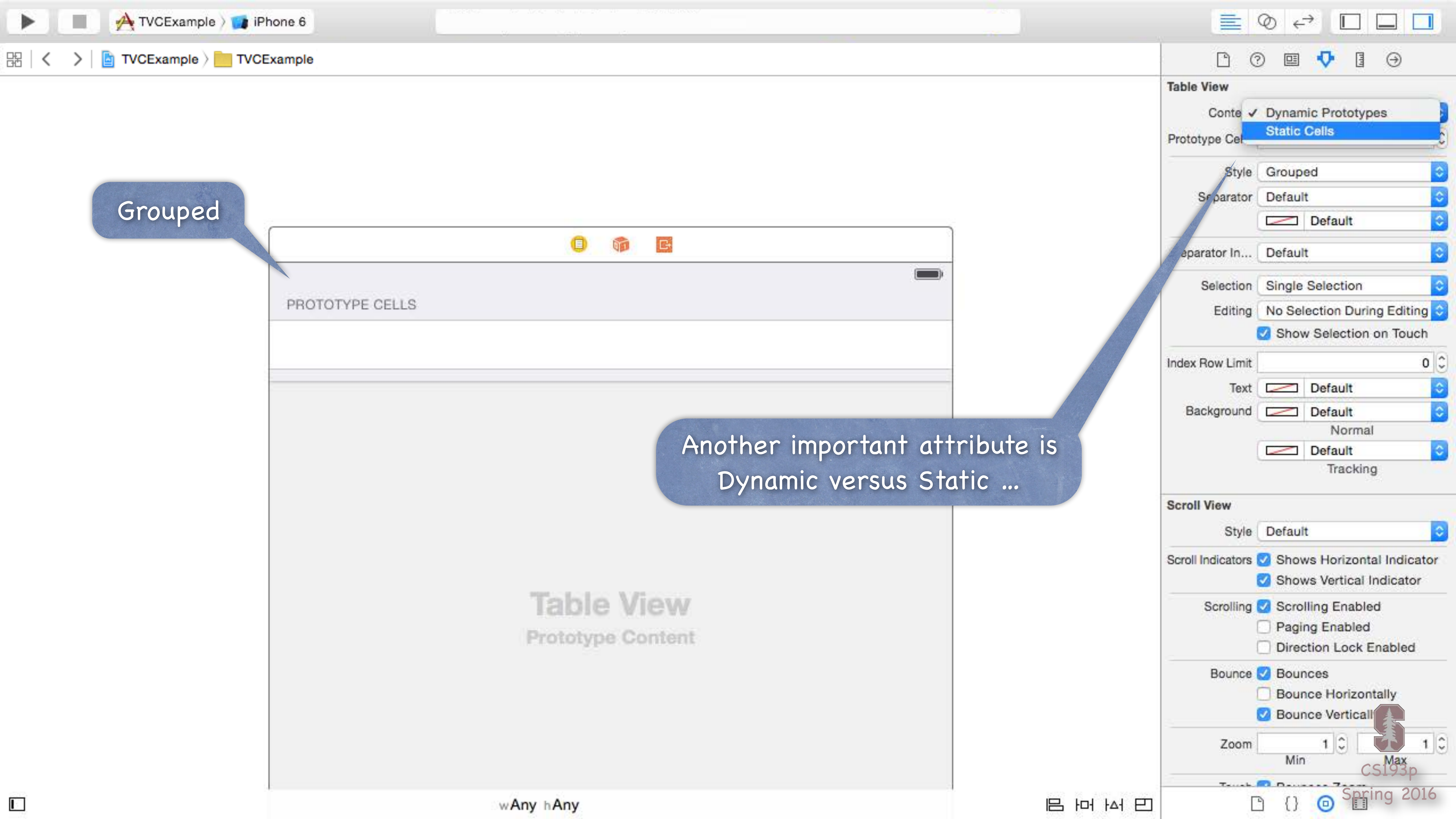


Table View

Content: Dynamic Prototypes
Prototype Cells: 1
Style: Grouped
Separator: Default
Default
Separator In...: Default
Selection: Single Selection
Editing: No Selection During Editing
[✓] Show Selection on Touch
Index Row Limit: 0
Text: Default
Background: Default
Normal
Default
Tracking

Scroll View

Style: Default
Scroll Indicators: [✓] Shows Horizontal Indicator
[✓] Shows Vertical Indicator
Scrolling: [✓] Scrolling Enabled
[ ] Paging Enabled
[ ] Direction Lock Enabled
Bounce: [✓] Bounces
[ ] Bounce Horizontally
[✓] Bounce Vertically
Zoom: 1 Min, 1 Max



Grouped

Another important attribute is Dynamic versus Static …

TVCExample 〉 iPhone 6

TVCExample 〉 TVCExample

PROTOTYPE CELLS

Table View
Prototype Content

wAny hAny

Table View

Content: ✓ Dynamic Prototypes / Static Cells

Prototype Cells

Style: Grouped

Separator: Default

Default

Separator Inset: Default

Selection: Single Selection

Editing: No Selection During Editing

[✓] Show Selection on Touch

Index Row Limit: 0

Text: Default

Background: Default

Normal

Default

Tracking

Scroll View

Style: Default

Scroll Indicators: [✓] Shows Horizontal Indicator

[✓] Shows Vertical Indicator

Scrolling: [✓] Scrolling Enabled

[ ] Paging Enabled

[ ] Direction Lock Enabled

Bounce: [✓] Bounces

[ ] Bounce Horizontally

[✓] Bounce Vertically

Zoom: 1 Min, 1 Max

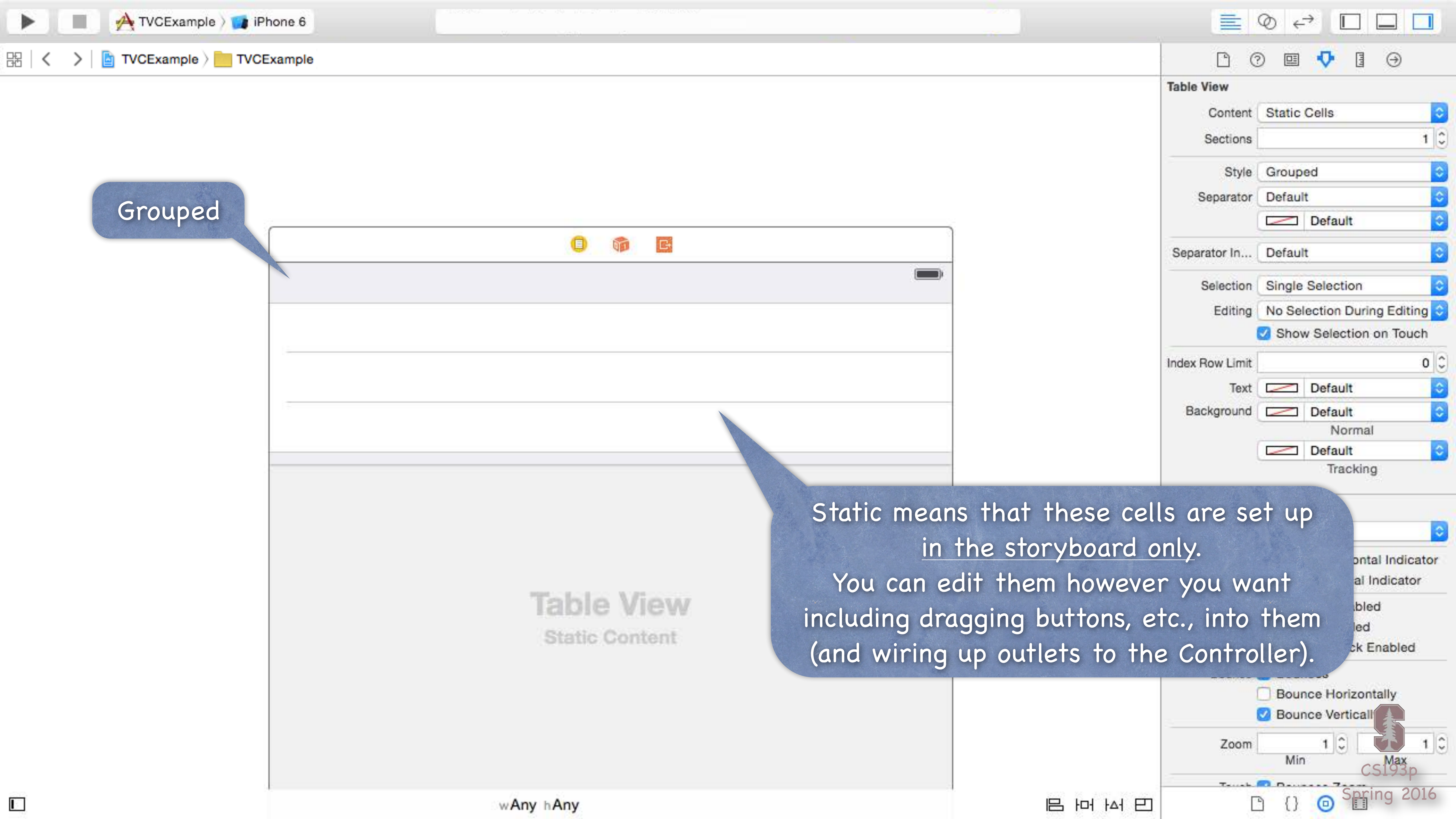


Table View

Content: Static Cells
Sections: 1
Style: Grouped
Separator: Default
Default
Separator In...: Default
Selection: Single Selection
Editing: No Selection During Editing
✓ Show Selection on Touch
Index Row Limit: 0
Text: Default
Background: Default
Normal
Default
Tracking
Bounce Horizontally
✓ Bounce Vertically
Zoom: 1 Min, 1 Max

Grouped

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

Grouped

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

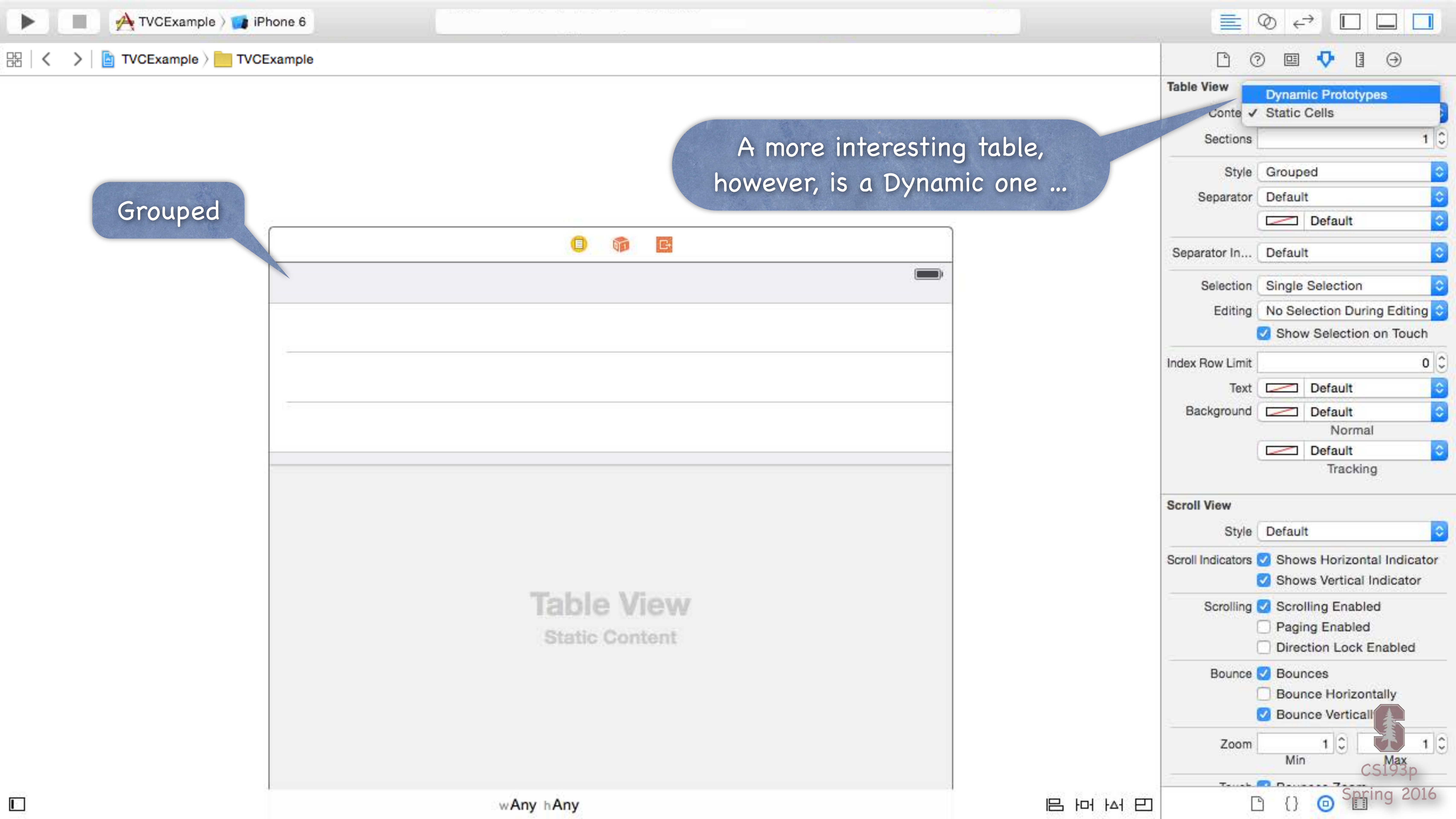
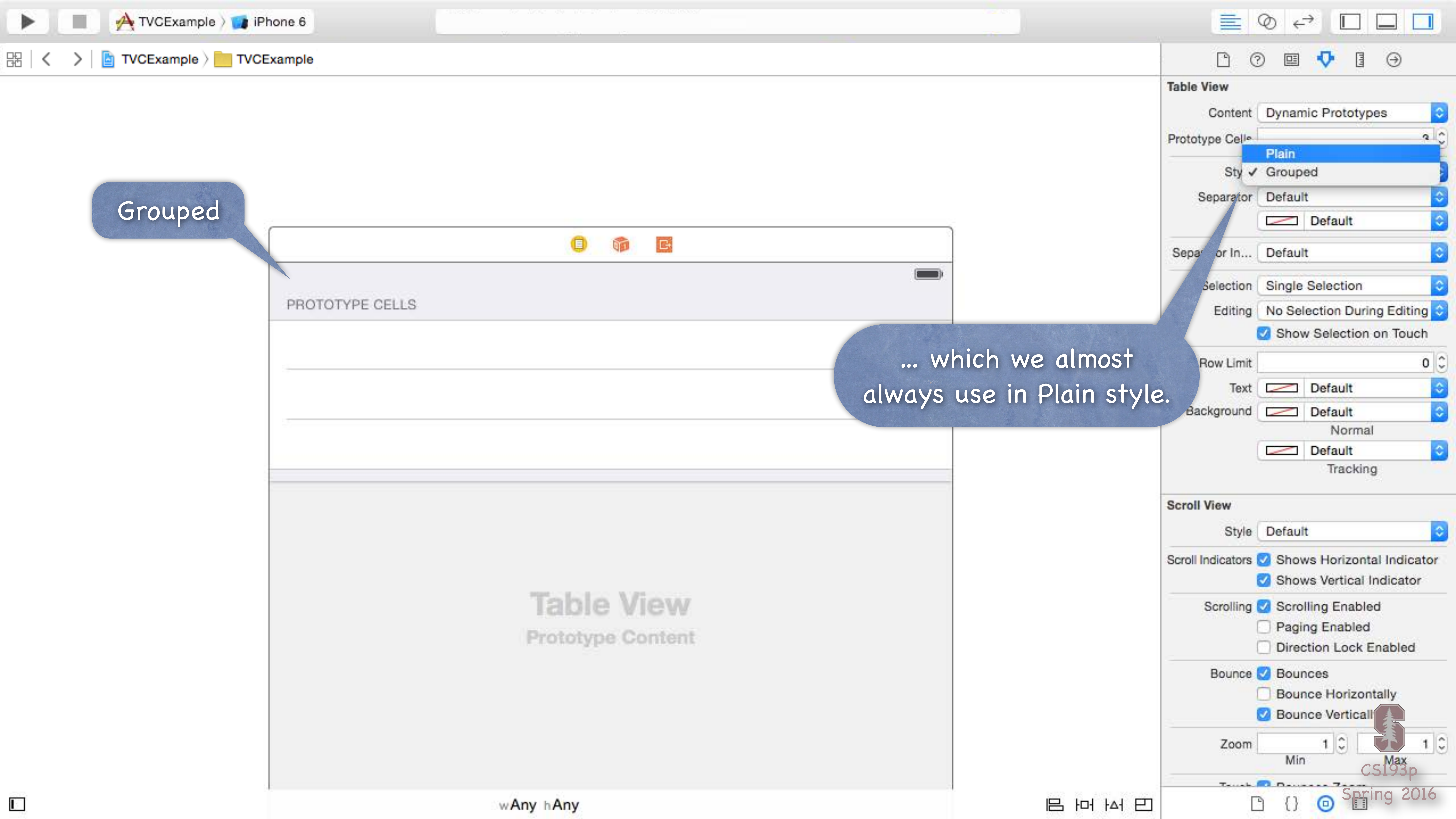


Table View
Content: Dynamic Prototypes / ✓ Static Cells
Sections 1
Style Grouped
Separator Default
Default
Separator In… Default
Selection Single Selection
Editing No Selection During Editing
✓ Show Selection on Touch
Index Row Limit 0
Text Default
Background Default
Normal
Default
Tracking

Scroll View
Style Default
Scroll Indicators ✓ Shows Horizontal Indicator
✓ Shows Vertical Indicator
Scrolling ✓ Scrolling Enabled
☐ Paging Enabled
☐ Direction Lock Enabled
Bounce ✓ Bounces
☐ Bounce Horizontally
✓ Bounce Vertically
Zoom 1 Min 1 Max



Grouped

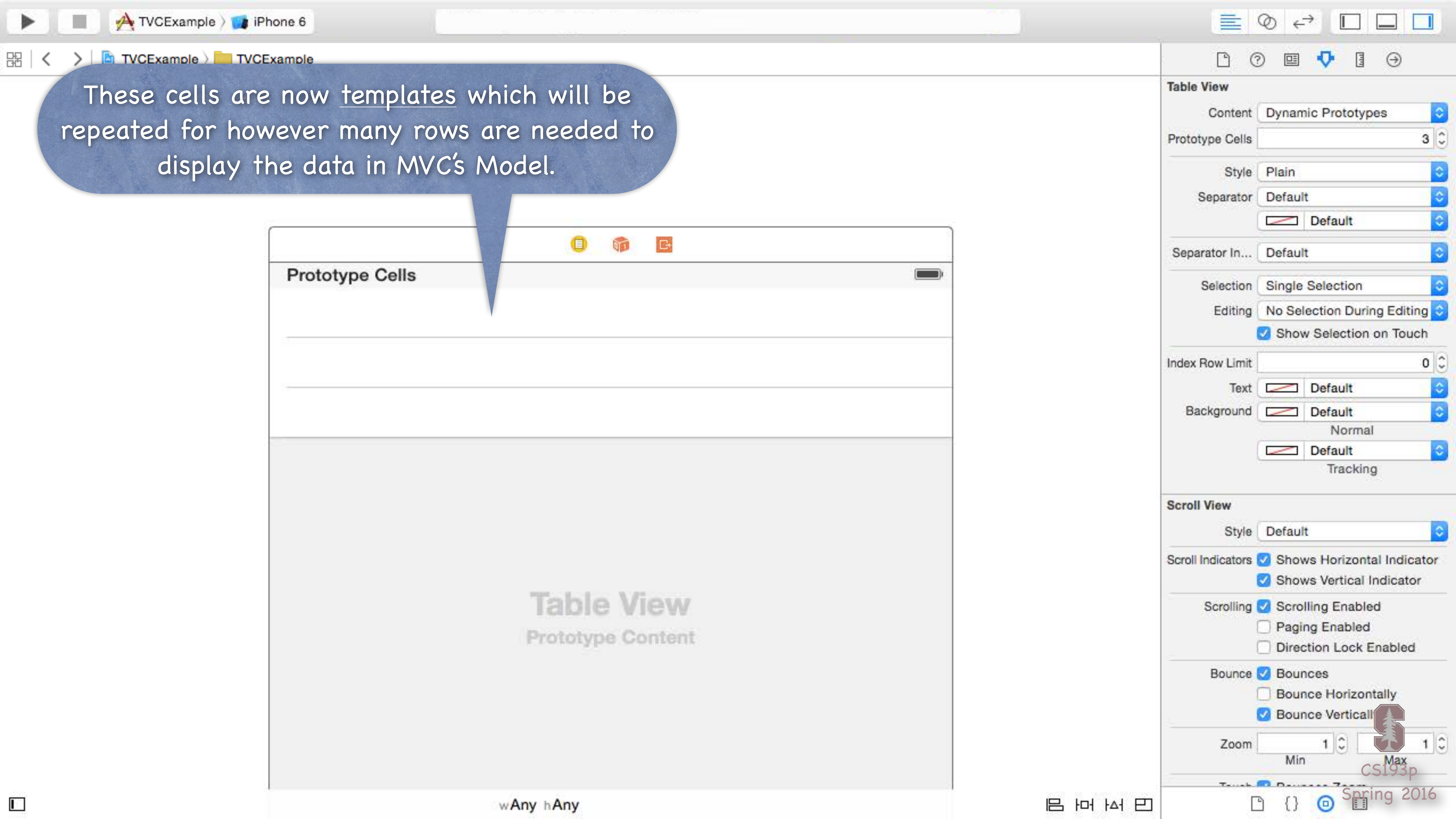
… which we almost always use in Plain style.

Table View

Content: Dynamic Prototypes
Prototype Cells: 3
Style: Plain / ✓ Grouped
Separator: Default
Default
Separator Inset: Default
Selection: Single Selection
Editing: No Selection During Editing
✓ Show Selection on Touch
Row Limit: 0
Text: Default
Background: Default
Normal
Default
Tracking

Scroll View

Style: Default
Scroll Indicators: ✓ Shows Horizontal Indicator
✓ Shows Vertical Indicator
Scrolling: ✓ Scrolling Enabled
☐ Paging Enabled
☐ Direction Lock Enabled
Bounce: ✓ Bounces
☐ Bounce Horizontally
✓ Bounce Vertically
Zoom: 1 Min, 1 Max



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

TVCExample 〉 iPhone 6

TVCExample 〉 TVCExample

Prototype Cells

Table View
Prototype Content

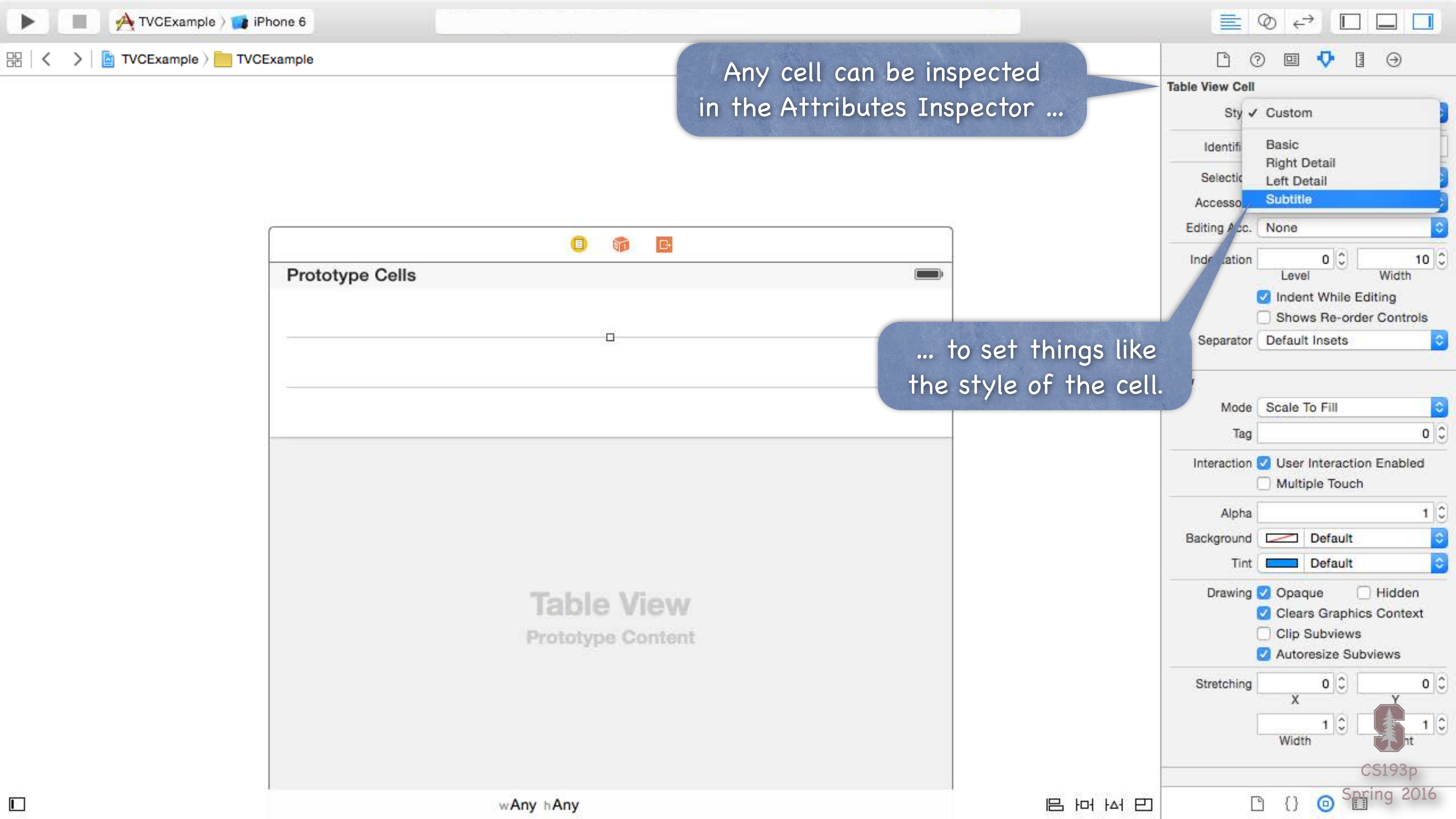
wAny hAny

Table View

Content	Dynamic Prototypes
Prototype Cells	3
Style	Plain
Separator	Default
	Default
Separator In...	Default
Selection	Single Selection
Editing	No Selection During Editing
	✓ Show Selection on Touch
Index Row Limit	0
Text	Default
Background	Default
	Normal
	Default
	Tracking

Scroll View

Style	Default
Scroll Indicators	✓ Shows Horizontal Indicator
	✓ Shows Vertical Indicator
Scrolling	✓ Scrolling Enabled
	☐ Paging Enabled
	☐ Direction Lock Enabled
Bounce	✓ Bounces
	☐ Bounce Horizontally
	✓ Bounce Vertically
Zoom	1 (Min) 1 (Max)



Any cell can be inspected in the Attributes Inspector …

… to set things like the style of the cell.

Table View Cell

Style: ✓ Custom, Basic, Right Detail, Left Detail, [✓ Subtitle]

Editing Acc.: None

Indentation: Level 0, Width 10

☑ Indent While Editing

☐ Shows Re-order Controls

Separator: Default Insets

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

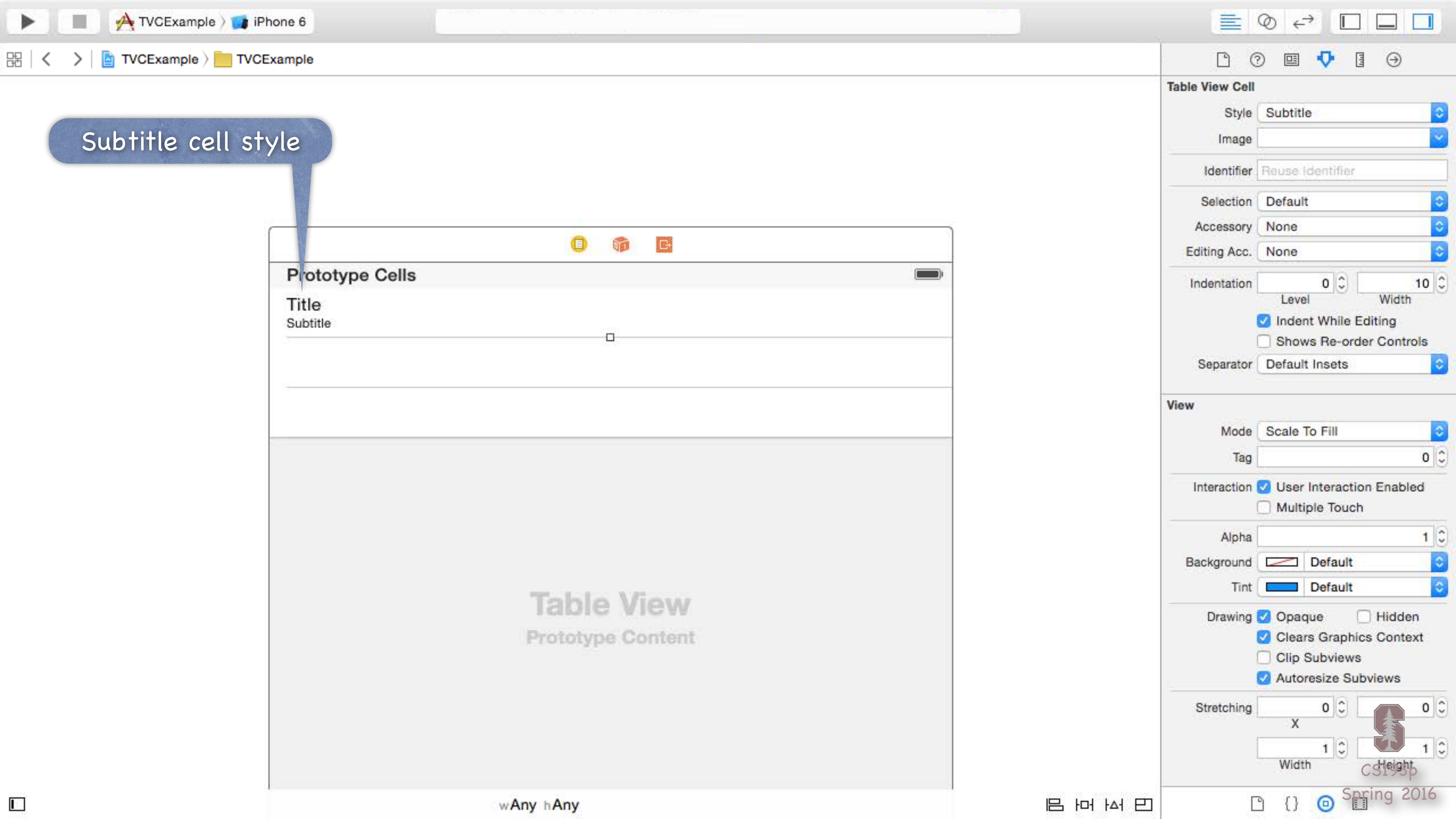
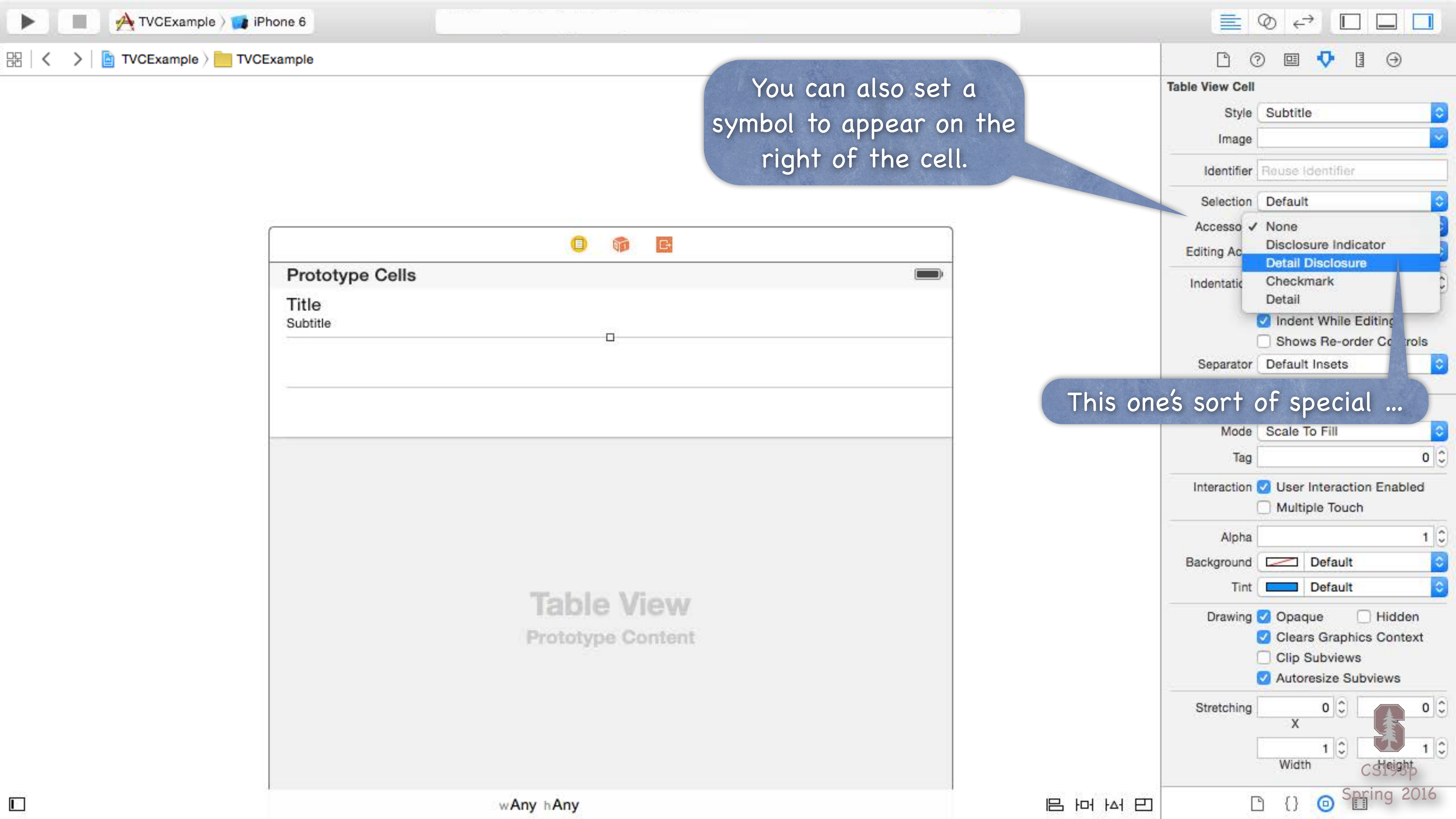


Table View Cell

Style: Subtitle
Image:
Identifier: Reuse Identifier
Selection: Default
Accessory: None
Editing Acc.: None
Indentation: 0 Level, 10 Width
[✓] Indent While Editing
[ ] Shows Re-order Controls
Separator: Default Insets

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 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: Default

Accessory: ✓ None / Disclosure Indicator / Detail Disclosure / Checkmark / Detail

Editing Acc

Indentation

✓ Indent While Editing

Shows Re-order Controls

Separator: Default Insets

Mode: Scale To Fill

Tag: 0

Interaction: ✓ User Interaction Enabled

Multiple Touch

Alpha: 1

Background: Default

Tint: Default

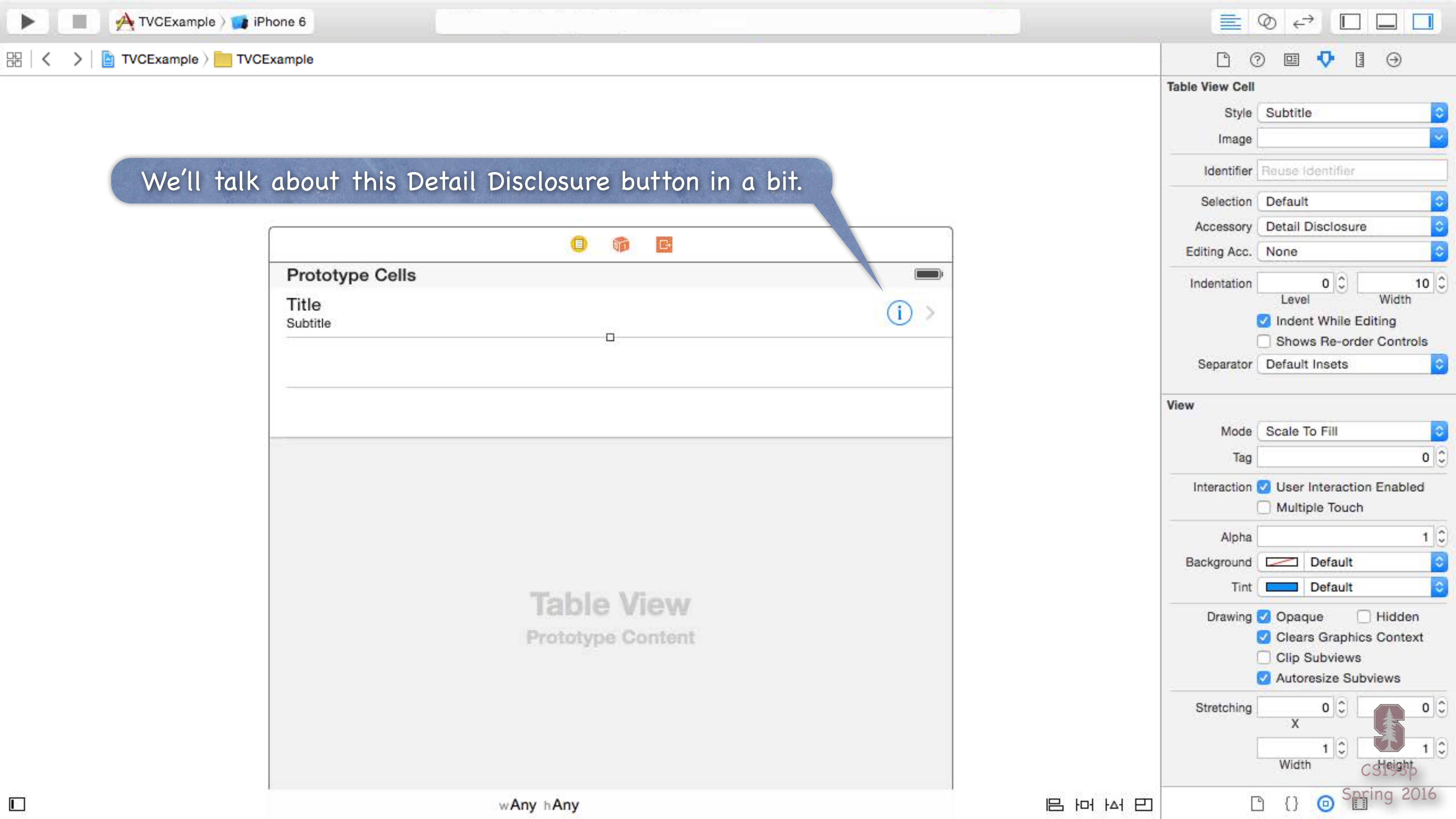
Drawing: ✓ Opaque, Hidden

✓ Clears Graphics Context

Clip Subviews

✓ Autoresize Subviews

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



We'll talk about this Detail Disclosure button in a bit.

Prototype Cells

Title
Subtitle

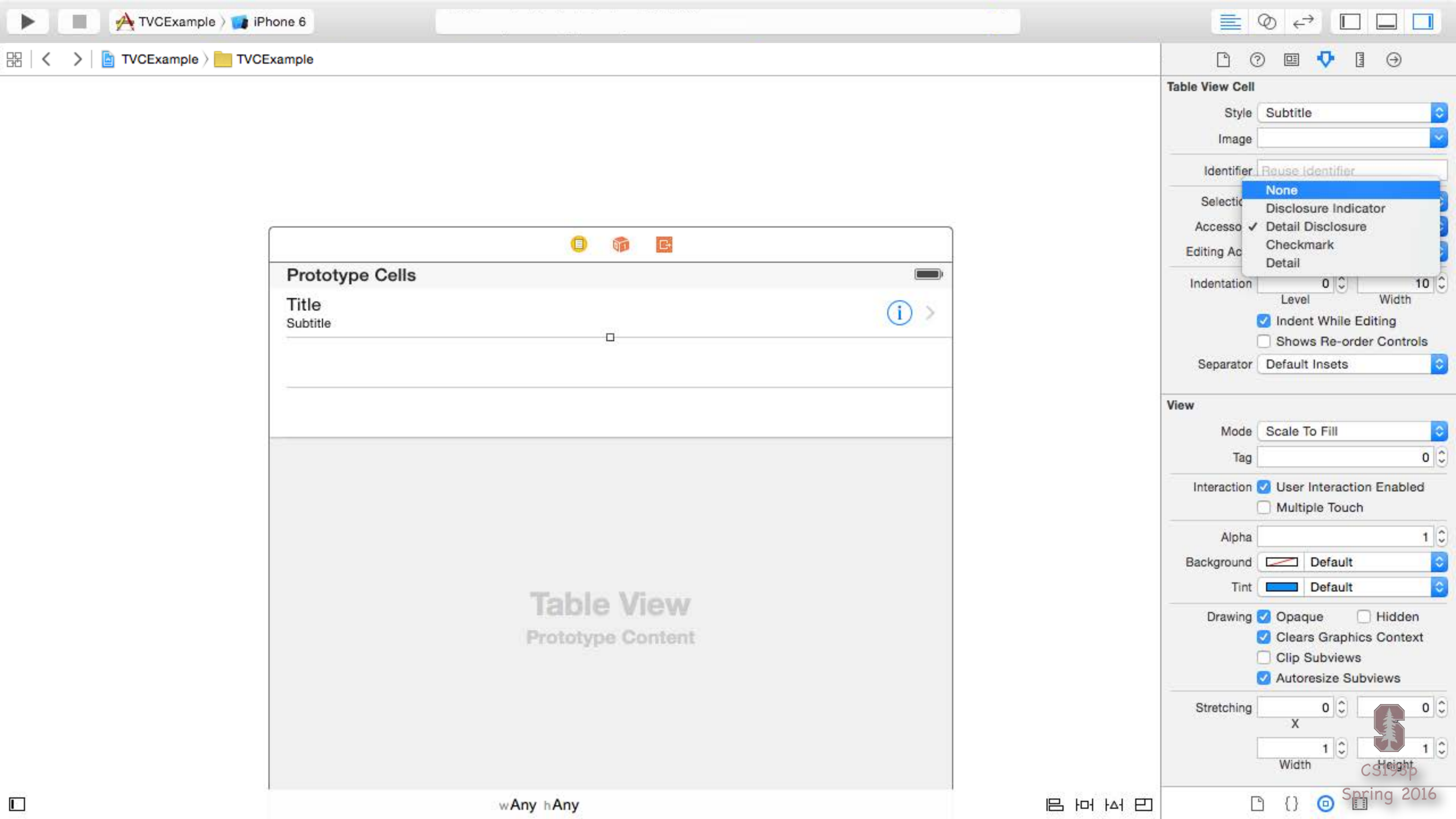
Table View
Prototype Content

Table View Cell

Style: Subtitle
Image:
Identifier: Reuse Identifier
Selection: Default
Accessory: Detail Disclosure
Editing Acc.: None
Indentation: 0 Level, 10 Width
☑ Indent While Editing
☐ Shows Re-order Controls
Separator: Default Insets

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



TVCExample › iPhone 6

TVCExample › TVCExample

Prototype Cells

Title
Subtitle

Table View
Prototype Content

wAny hAny

Table View Cell

Style: Subtitle

Image:

Identifier: Reuse Identifier

Selectio

Accesso

Editing Ac

- None
- Disclosure Indicator
- ✓ Detail Disclosure
- Checkmark
- Detail

Indentation: 0 Level, 10 Width

[✓] Indent While Editing

[ ] Shows Re-order Controls

Separator: Default Insets

View

Mode: Scale To Fill

Tag: 0

Interaction: [✓] User Interaction Enabled, [ ] Multiple Touch

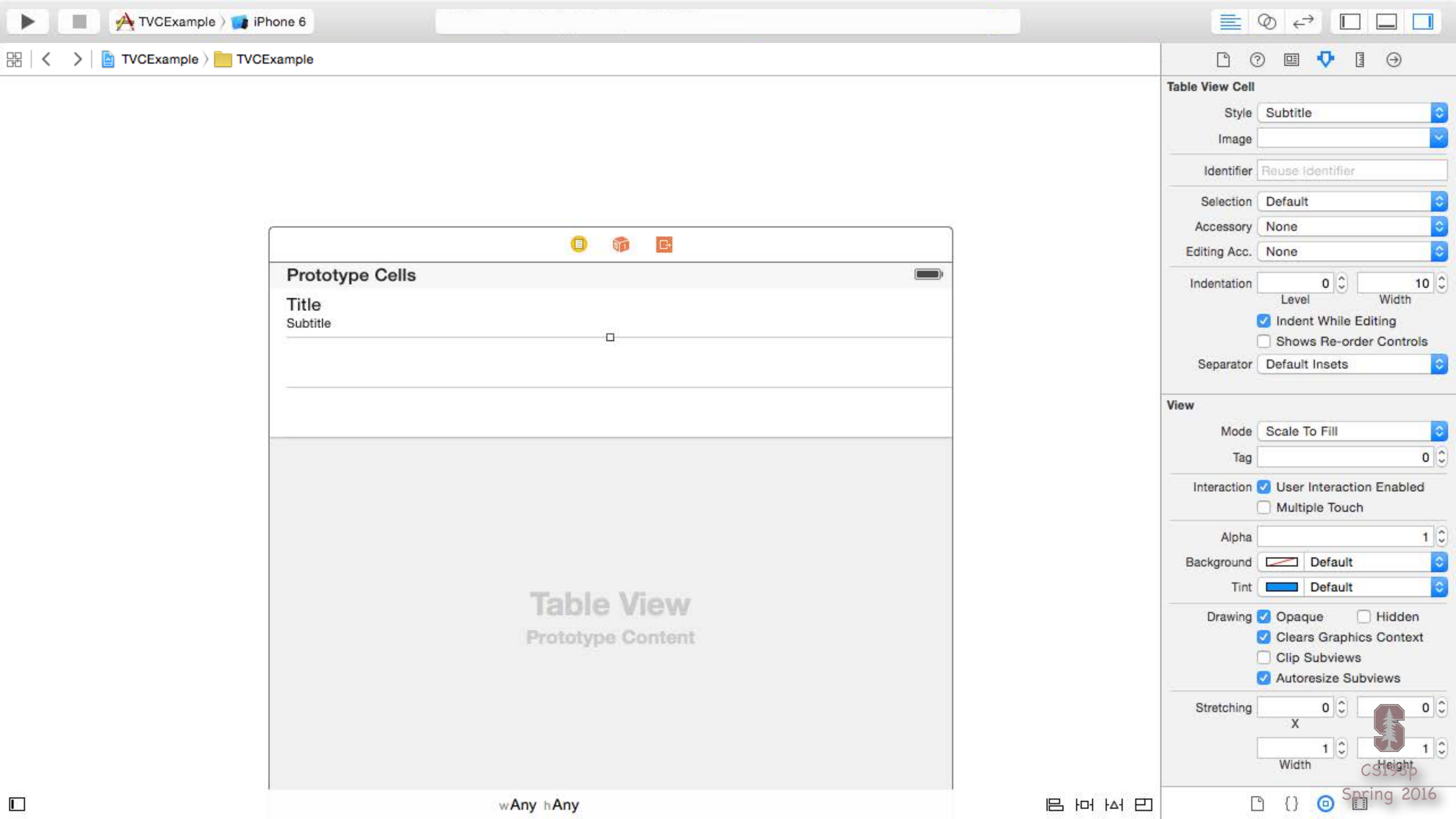
Alpha: 1

Background: Default

Tint: Default

Drawing: [✓] Opaque, [ ] Hidden, [✓] Clears Graphics Context, [ ] Clip Subviews, [✓] Autoresize Subviews

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



TVCExample 〉 iPhone 6

TVCExample 〉 TVCExample

Prototype Cells

Title
Subtitle

Table View
Prototype Content

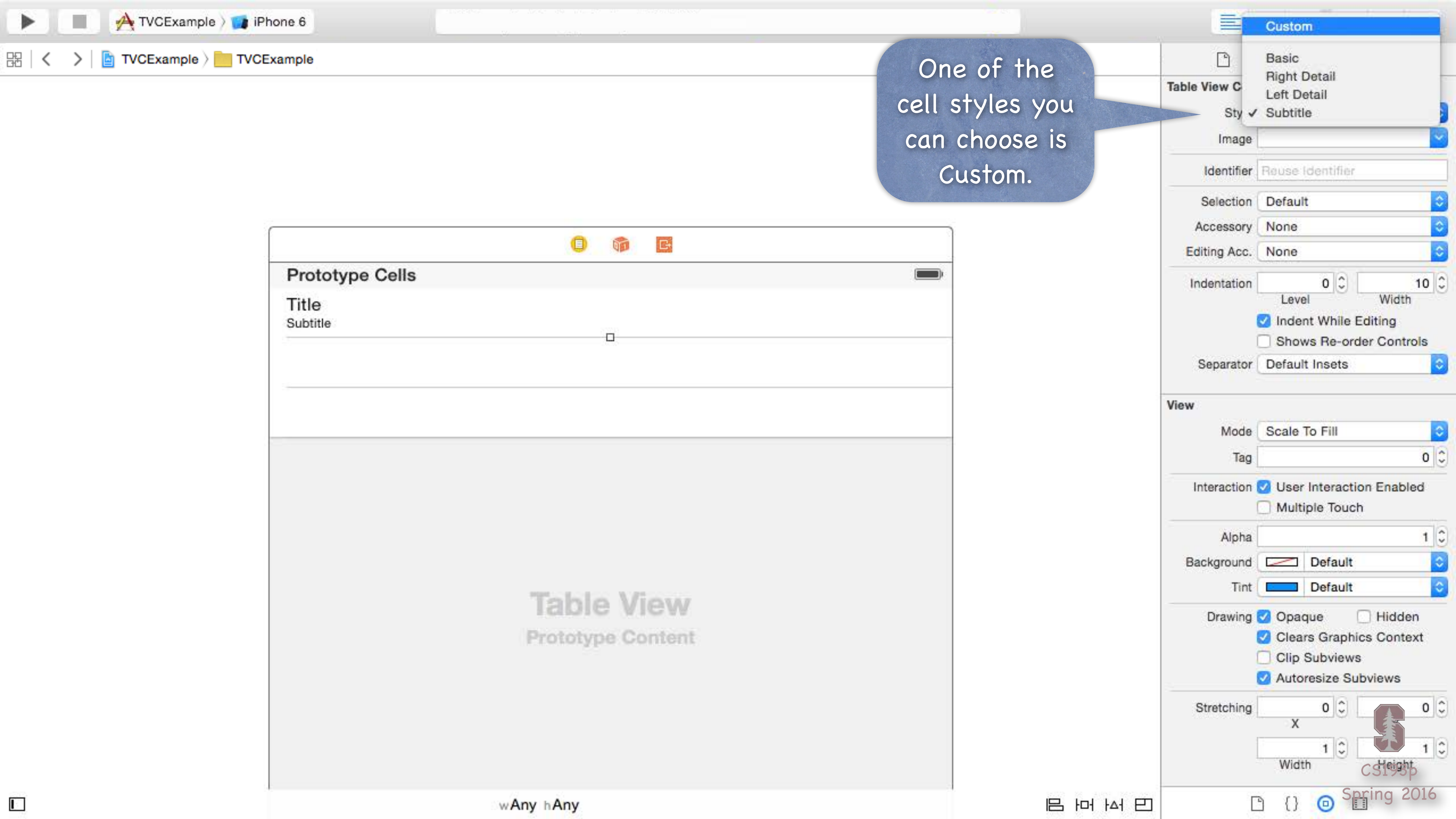
wAny hAny

Table View Cell

Style: Subtitle
Image:
Identifier: Reuse Identifier
Selection: Default
Accessory: None
Editing Acc.: None
Indentation: 0 (Level) 10 (Width)
[✓] Indent While Editing
[ ] Shows Re-order Controls
Separator: Default Insets

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
1 (Width) 1 (Height)



One of the cell styles you can choose is Custom.

TVCExample 〉 iPhone 6

TVCExample 〉 TVCExample

Prototype Cells

Title
Subtitle

Table View
Prototype Content

wAny hAny

Table View C

Style: ✓ Subtitle (menu: Custom, Basic, Right Detail, Left Detail, Subtitle)

Image:

Identifier: Reuse Identifier

Selection: Default

Accessory: None

Editing Acc.: None

Indentation: 0 Level, 10 Width

☑ Indent While Editing
☐ Shows Re-order Controls

Separator: Default Insets

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

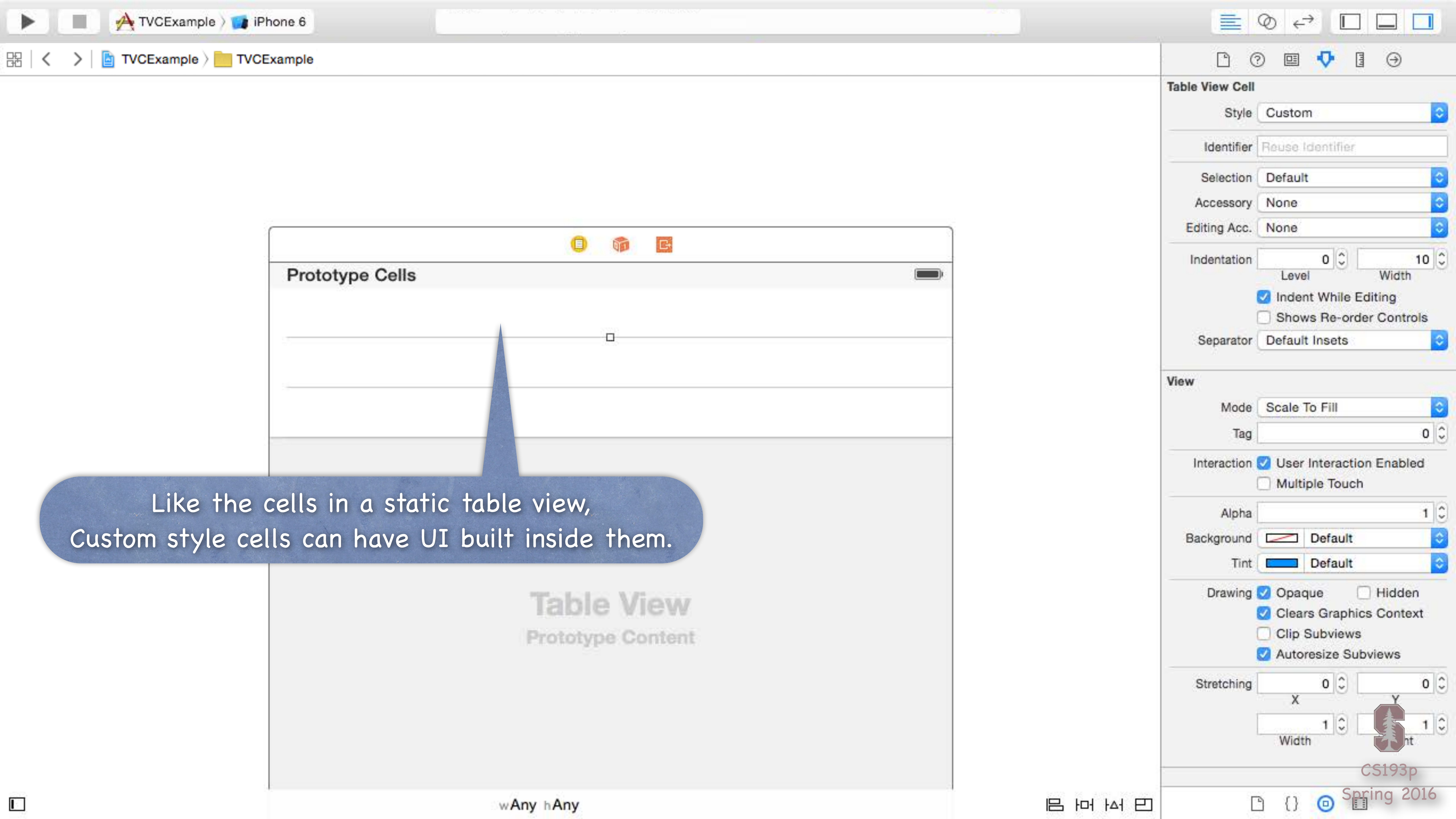


Table View Cell

Style: Custom
Identifier: Reuse Identifier
Selection: Default
Accessory: None
Editing Acc.: None
Indentation: 0 Level, 10 Width
☑ Indent While Editing
☐ Shows Re-order Controls
Separator: Default Insets

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

Like the cells in a static table view,
Custom style cells can have UI built inside them.

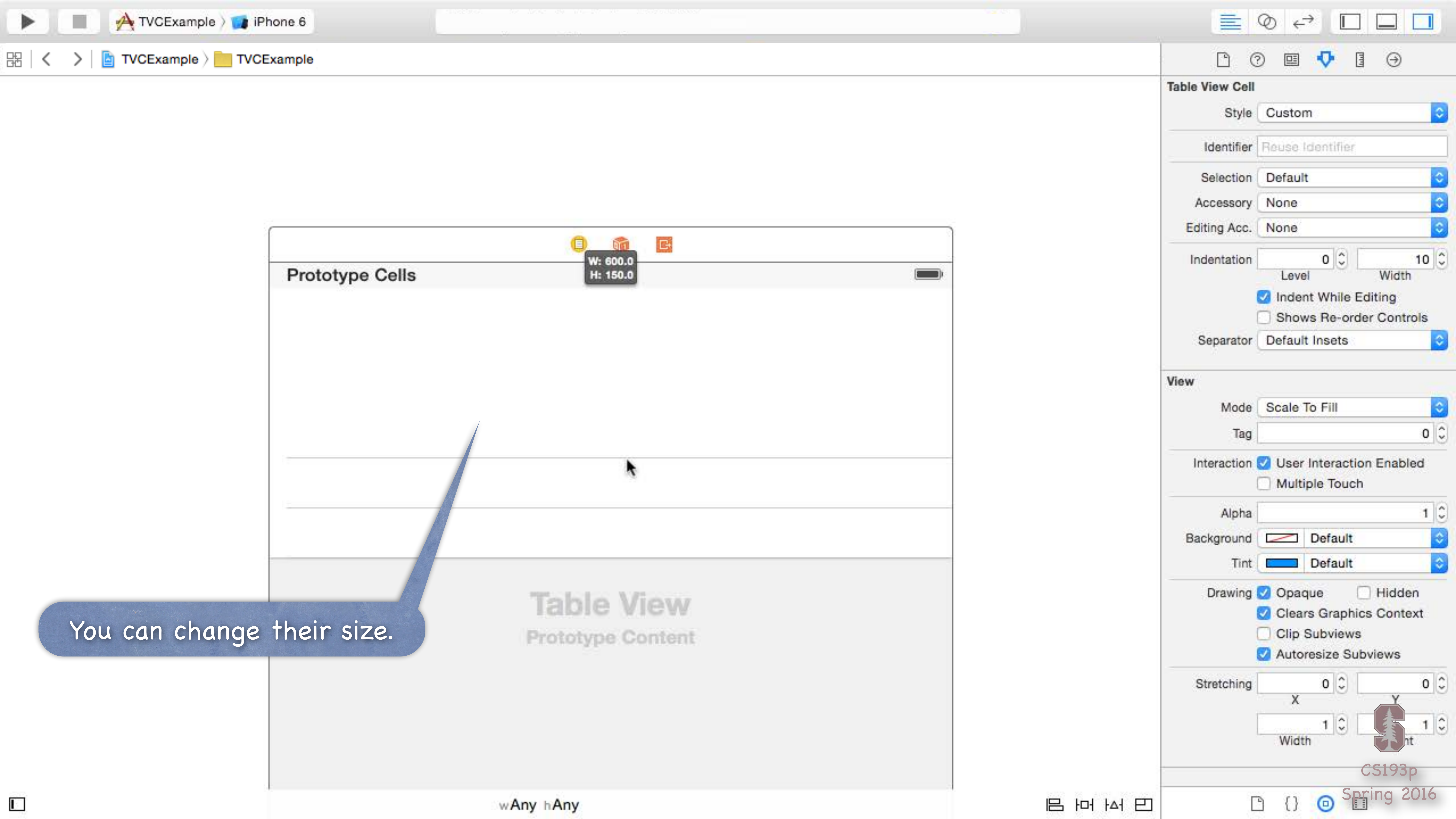


Table View Cell

Style: Custom
Identifier: Reuse Identifier
Selection: Default
Accessory: None
Editing Acc.: None
Indentation: 0 Level, 10 Width
☑ Indent While Editing
☐ Shows Re-order Controls
Separator: Default Insets

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 change their size.

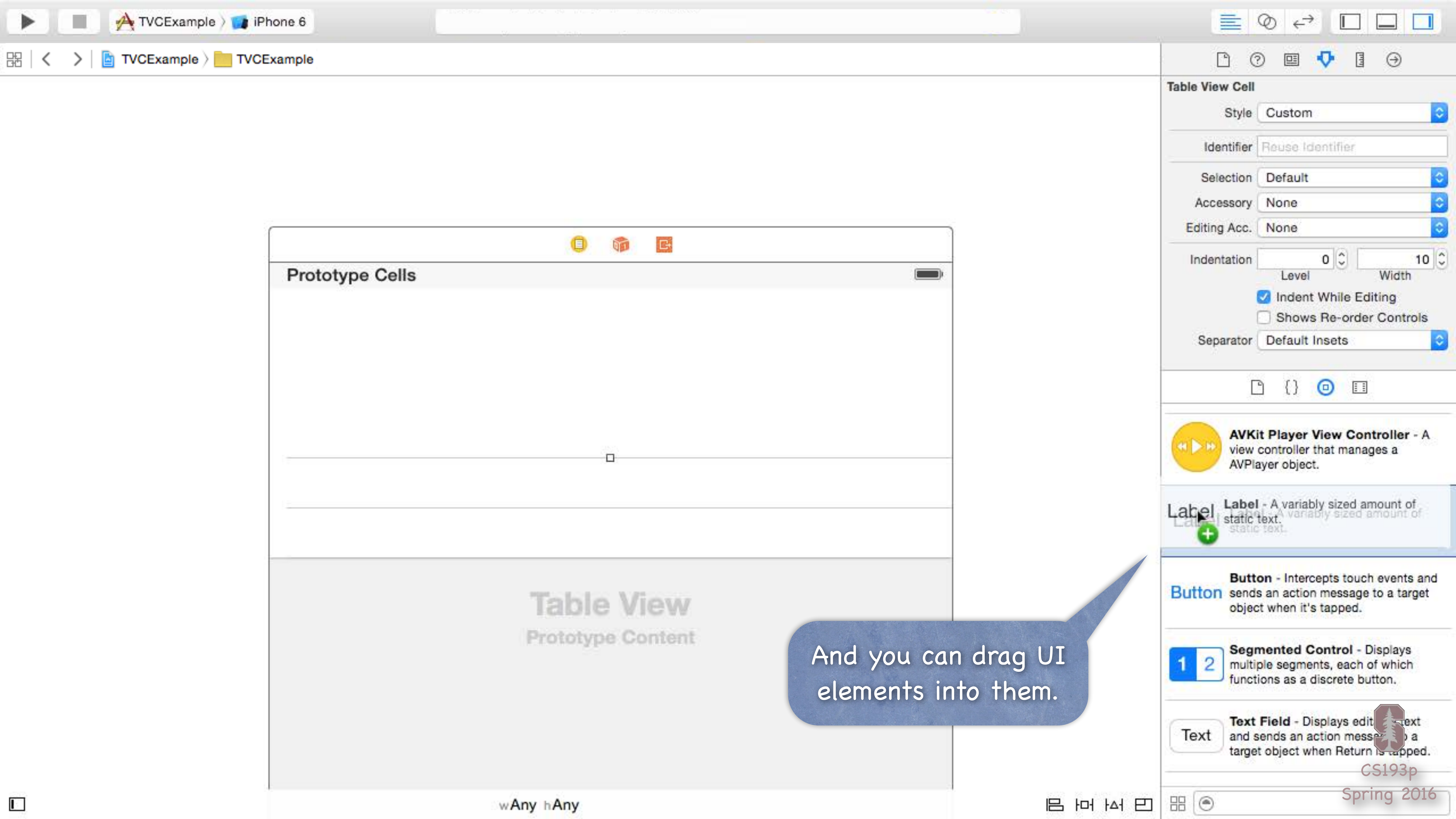


Table View Cell

Style: Custom
Identifier: Reuse Identifier
Selection: Default
Accessory: None
Editing Acc.: None
Indentation: 0 (Level), 10 (Width)
[✓] Indent While Editing
[ ] Shows Re-order Controls
Separator: Default Insets

AVKit Player View Controller - A view controller that manages a AVPlayer object.

Label - A variably sized amount of static text.

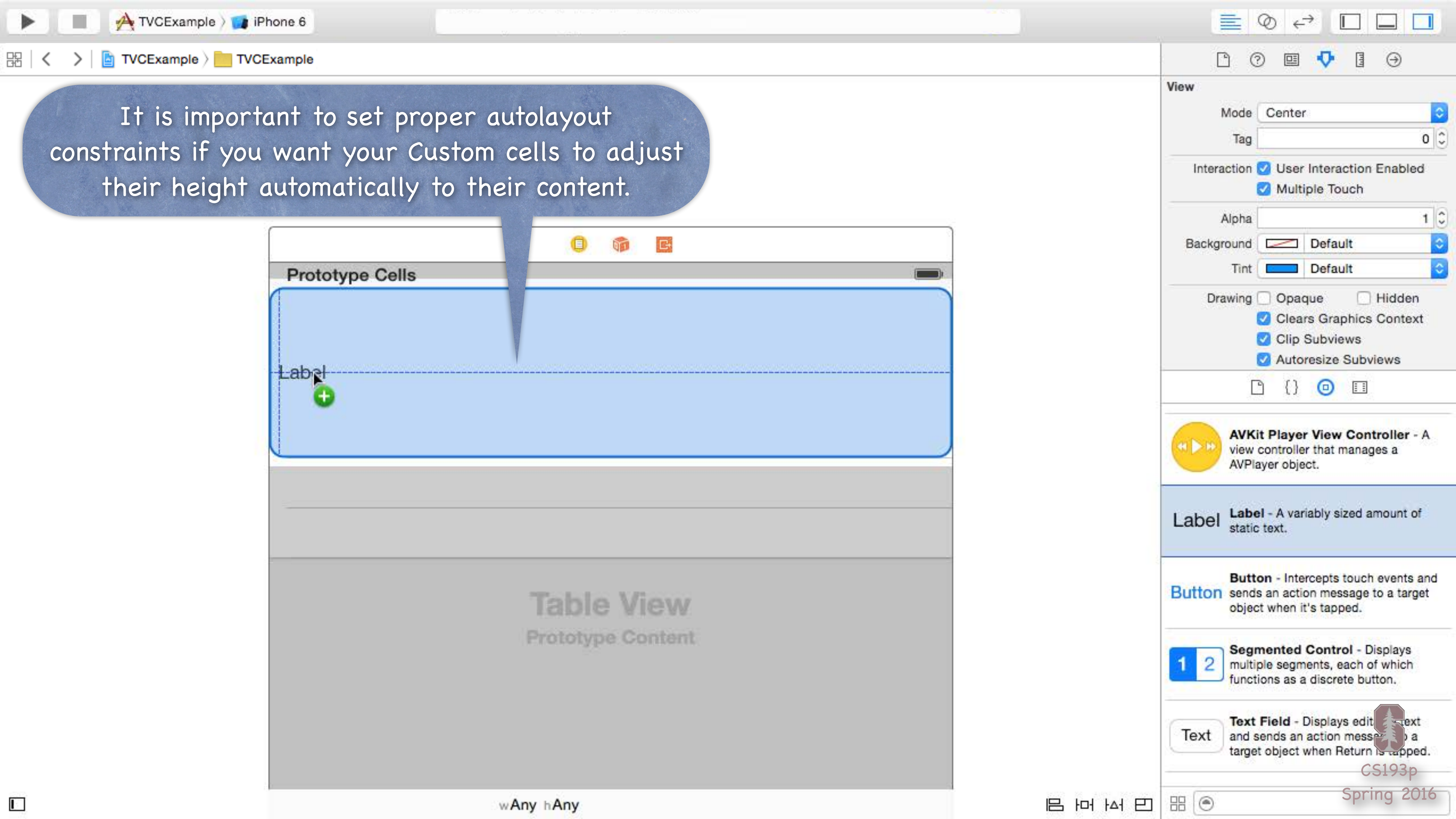
Button - Intercepts touch events and sends an action message to a target object when it's tapped.

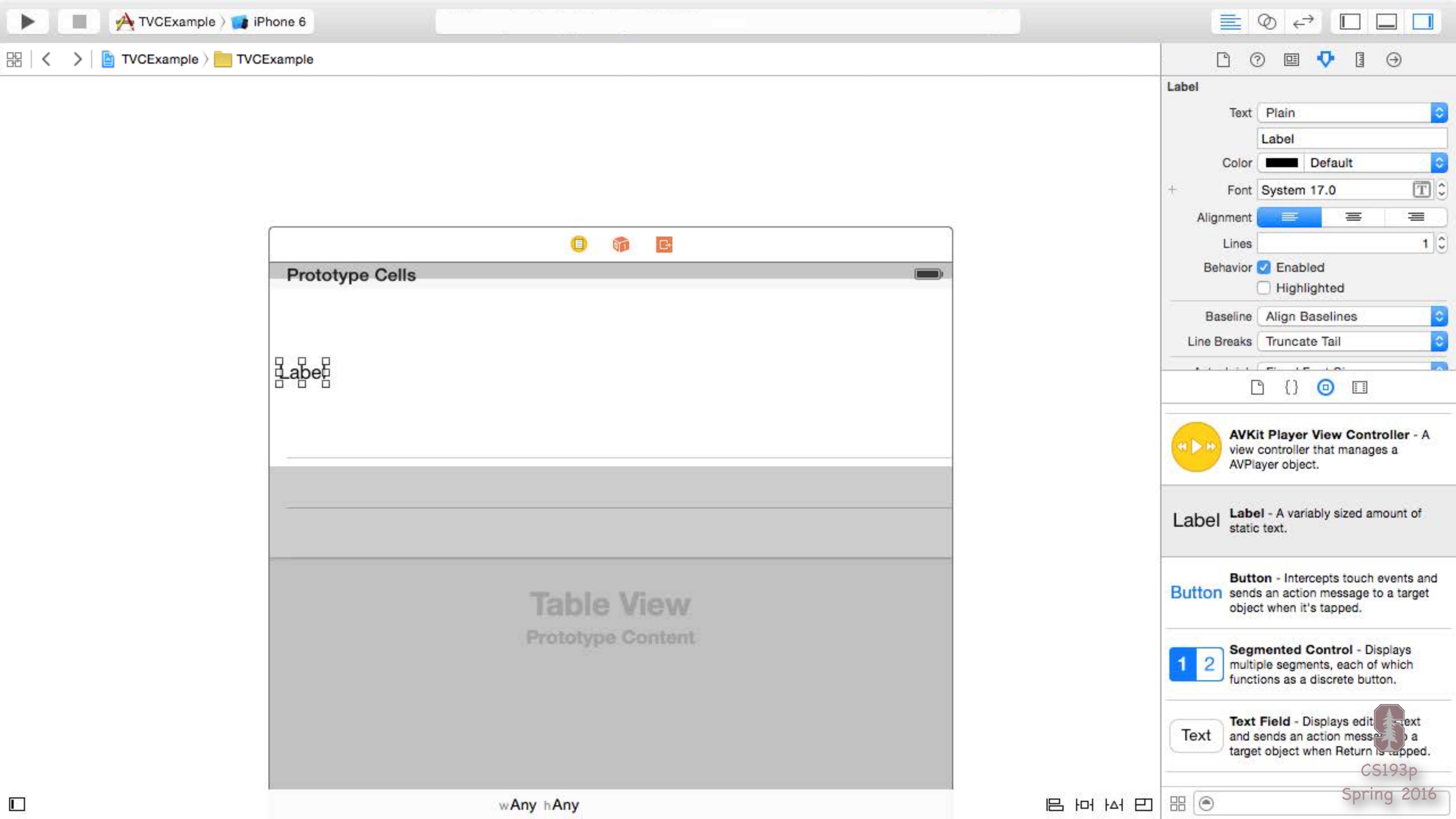
Segmented Control - Displays multiple segments, each of which functions as a discrete button.

Text Field - Displays edit[illegible] text and sends an action mess[illegible] a target object when Return is tapped.

And you can drag UI elements into them.

It is important to set proper autolayout constraints if you want your Custom cells to adjust their height automatically to their content.


TVCExample
iPhone 6
TVCExample
TVCExample
Prototype Cells
Label
Table View
Prototype Content
wAny hAny
View
Mode Center
Tag 0
Interaction
User Interaction Enabled
Multiple Touch
Alpha 1
Background Default
Tint Default
Drawing
Opaque
Hidden
Clears Graphics Context
Clip Subviews
Autoresize Subviews
AVKit Player View Controller - A view controller that manages a AVPlayer object.
Label - A variably sized amount of static text.
Button - Intercepts touch events and sends an action message to a target object when it's tapped.
Segmented Control - Displays multiple segments, each of which functions as a discrete button.
Text Field - Displays editable text and sends an action message to a target object when Return is tapped.



TVCExample 〉 iPhone 6

TVCExample 〉 TVCExample

Label

Text: Plain

Label

Color: Default

Font: System 17.0

Alignment

Lines: 1

Behavior: [✓ Enabled] Highlighted

Baseline: Align Baselines

Line Breaks: Truncate Tail

Prototype Cells

Label

Table View
Prototype Content

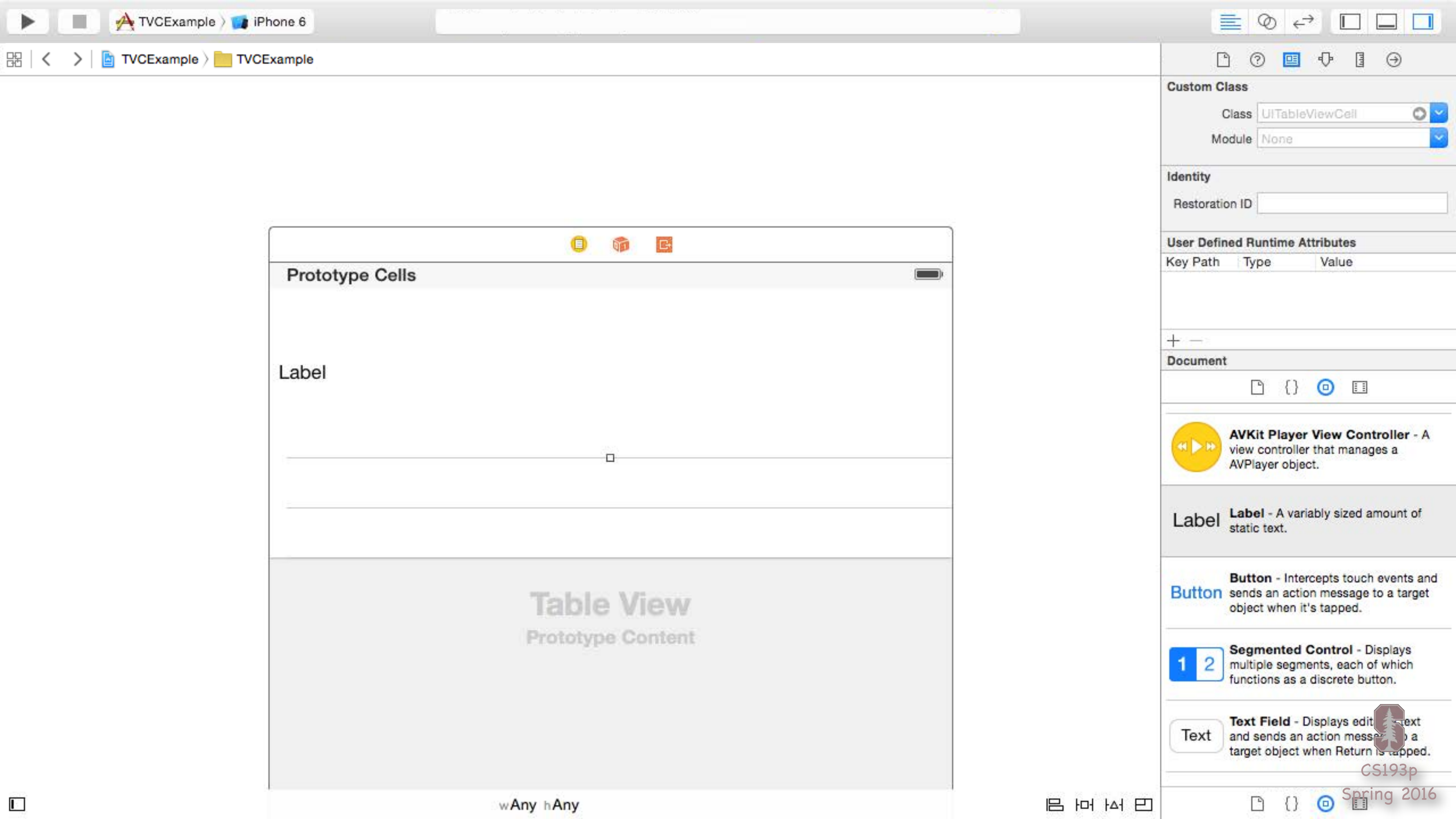
AVKit Player View Controller - A view controller that manages a AVPlayer object.

Label - A variably sized amount of static text.

Button - Intercepts touch events and sends an action message to a target object when it's tapped.

Segmented Control - Displays multiple segments, each of which functions as a discrete button.

Text Field - Displays editable text and sends an action message to a target object when Return is tapped.



Custom Class

Class UITableViewCell

Module None

Identity

Restoration ID

User Defined Runtime Attributes

Key Path	Type	Value

Document

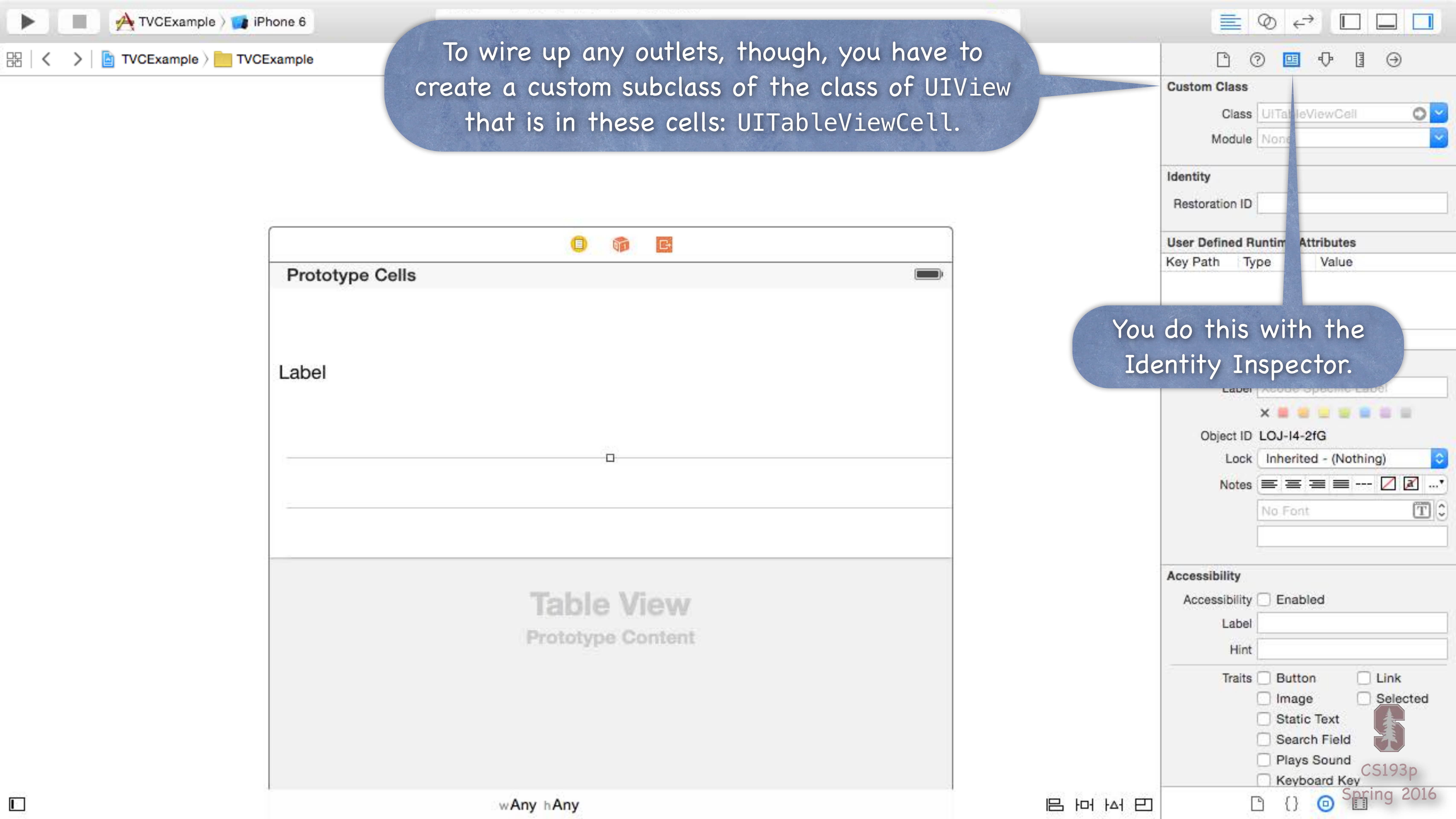
AVKit Player View Controller - A view controller that manages a AVPlayer object.

Label - A variably sized amount of static text.

Button - Intercepts touch events and sends an action message to a target object when it's tapped.

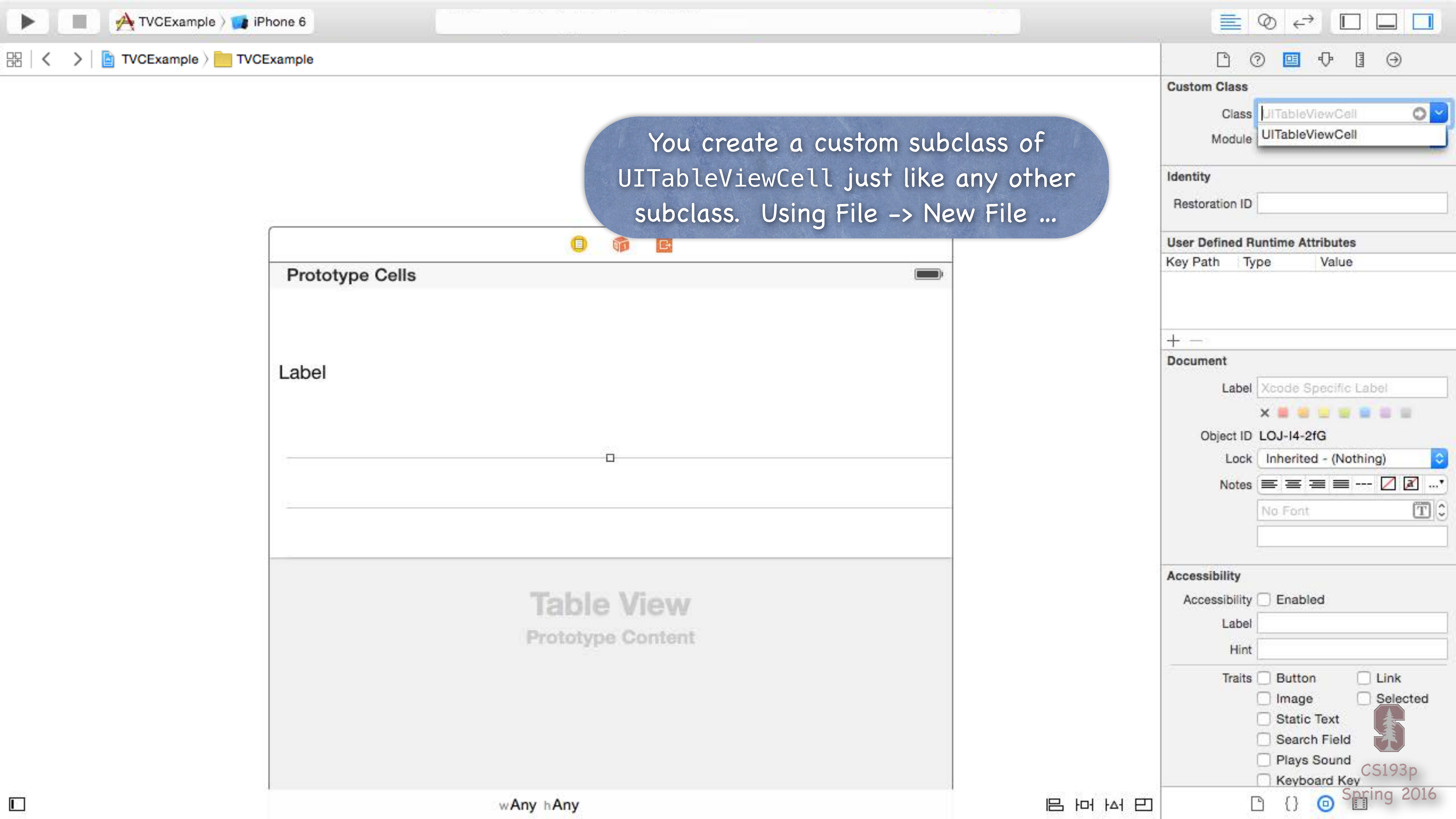
Segmented Control - Displays multiple segments, each of which functions as a discrete button.

Text Field - Displays edit[illegible]text and sends an action mess[illegible] a target object when Return is tapped.



To wire up any outlets, though, you have to create a custom subclass of the class of UIView that is in these cells: UITableViewCell.

You do this with the Identity Inspector.



You create a custom subclass of UITableViewCell just like any other subclass. Using File -> New File ...

TVCExample 〉 iPhone 6

TVCExample 〉 TVCExample

Prototype Cells

Label

Table View
Prototype Content

wAny hAny

Custom Class

Class: UITableViewCell

Module: UITableViewCell

Identity

Restoration ID

User Defined Runtime Attributes

Key Path	Type	Value

Document

Label: Xcode Specific Label

Object ID: LOJ-I4-2fG

Lock: Inherited - (Nothing)

Notes

No Font

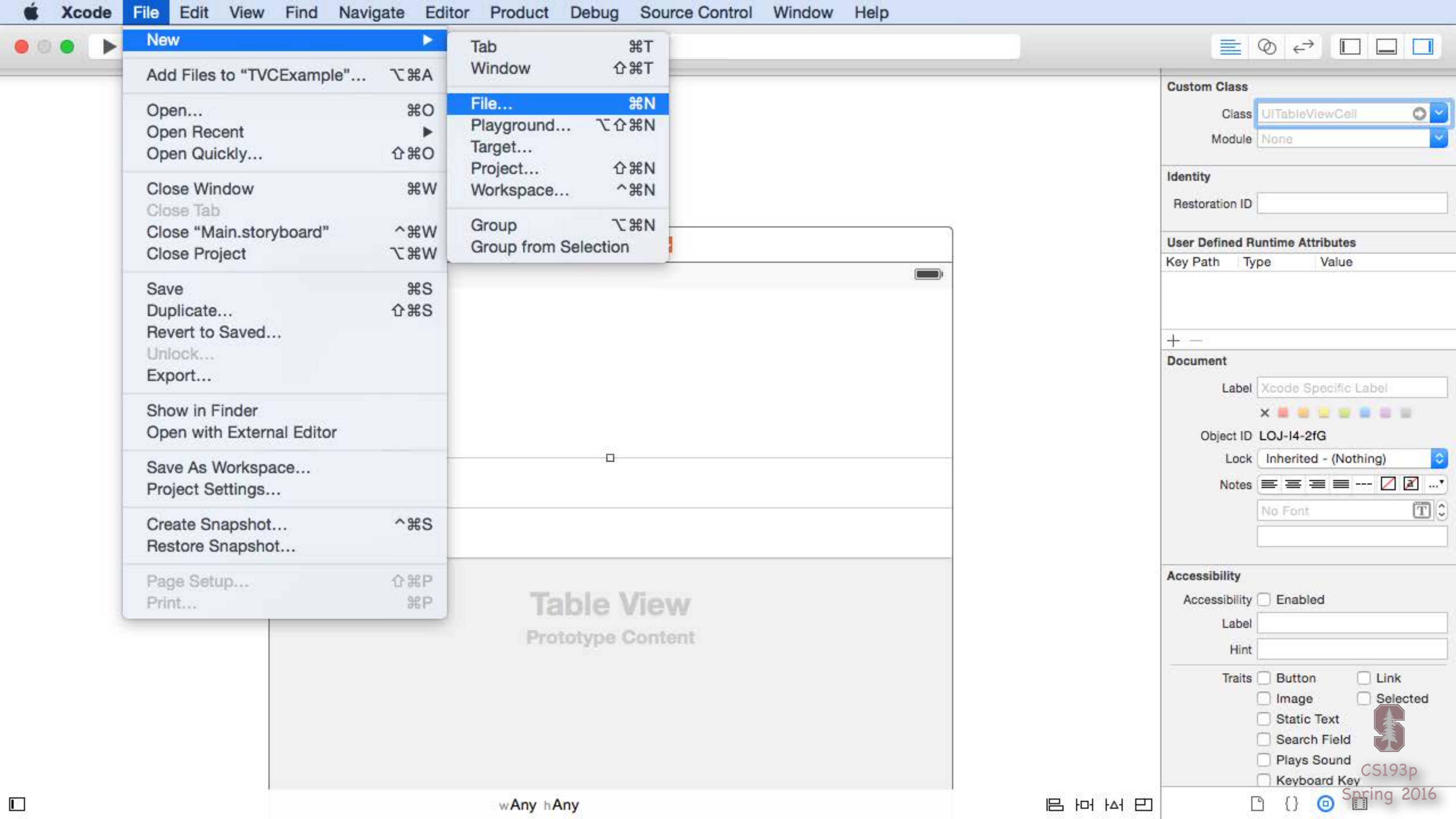
Accessibility

Accessibility: Enabled

Label

Hint

Traits: Button, Link, Image, Selected, Static Text, Search Field, Plays Sound, Keyboard Key



Custom Class
Class: UITableViewCell
Module: None

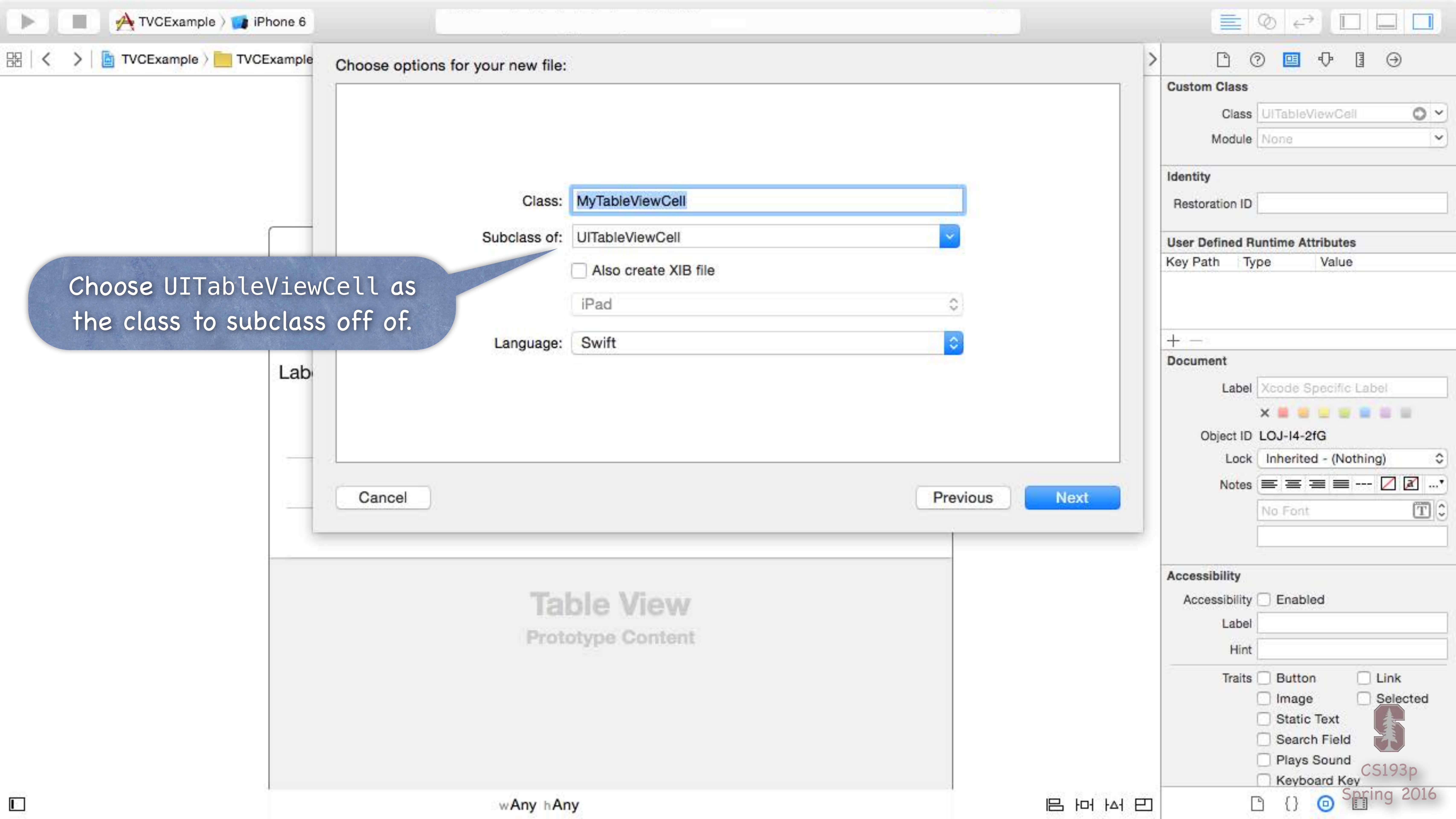
Identity
Restoration ID:

User Defined Runtime Attributes

Key Path	Type	Value

Document
Label: Xcode Specific Label
Object ID: LOJ-I4-2fG
Lock: Inherited - (Nothing)
Notes: No Font

Accessibility
Accessibility: ☐ Enabled
Label:
Hint:
Traits: ☐ Button ☐ Link ☐ Image ☐ Selected ☐ Static Text ☐ Search Field ☐ Plays Sound ☐ Keyboard Key



Choose options for your new file:

Class: MyTableViewCell

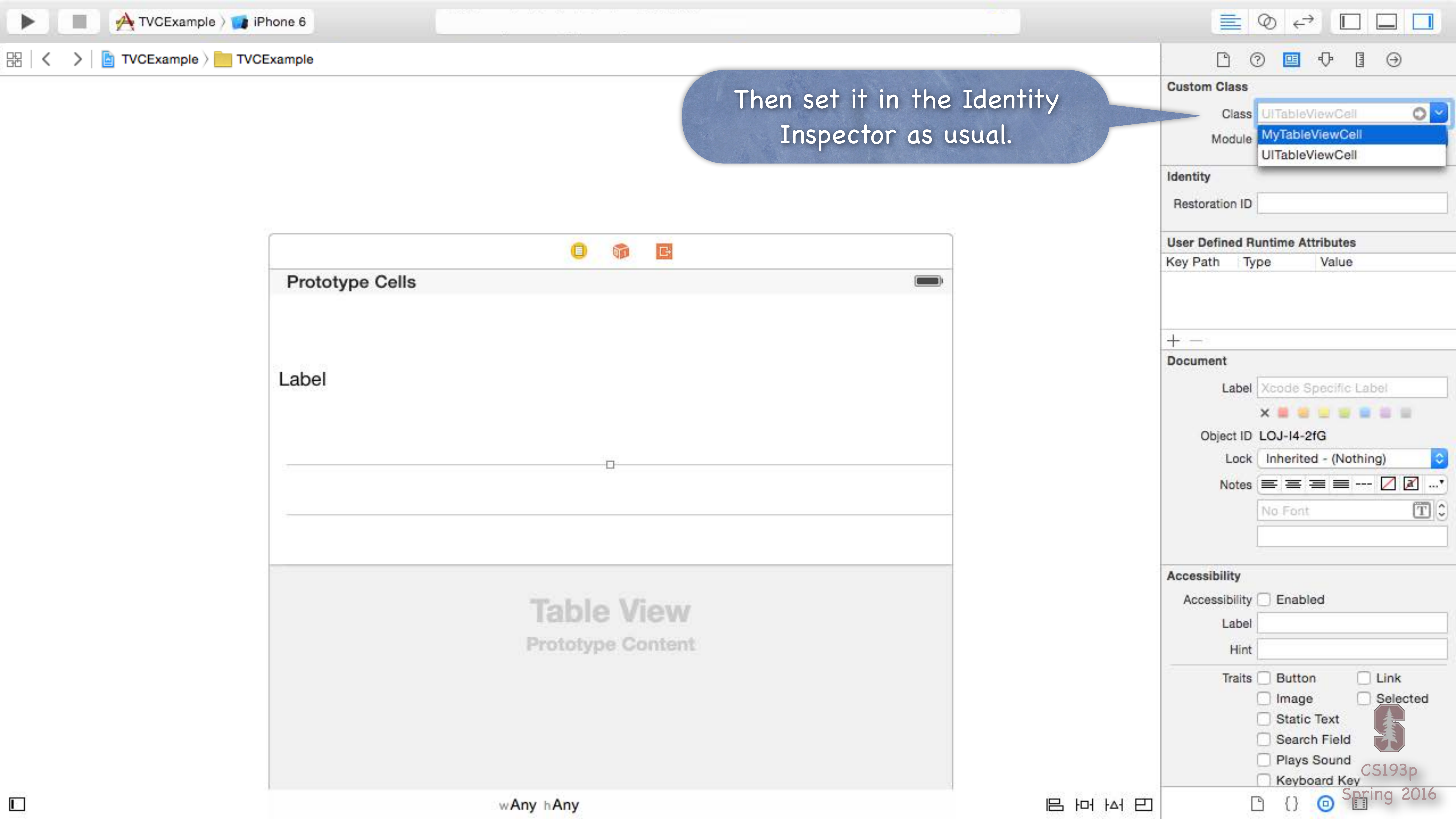
Subclass of: UITableViewCell

Also create XIB file

iPad

Language: Swift

Choose UITableViewCell as the class to subclass off of.



Then set it in the Identity Inspector as usual.

TVCExample 〉 iPhone 6

TVCExample 〉 TVCExample

Custom Class

Class: UITableViewCell

MyTableViewCell

UITableViewCell

Module

Identity

Restoration ID

User Defined Runtime Attributes

Key Path	Type	Value

Document

Label: Xcode Specific Label

Object ID: LOJ-I4-2fG

Lock: Inherited - (Nothing)

Notes

No Font

Accessibility

Accessibility: Enabled

Label

Hint

Traits: Button, Link, Image, Selected, Static Text, Search Field, Plays Sound, Keyboard Key

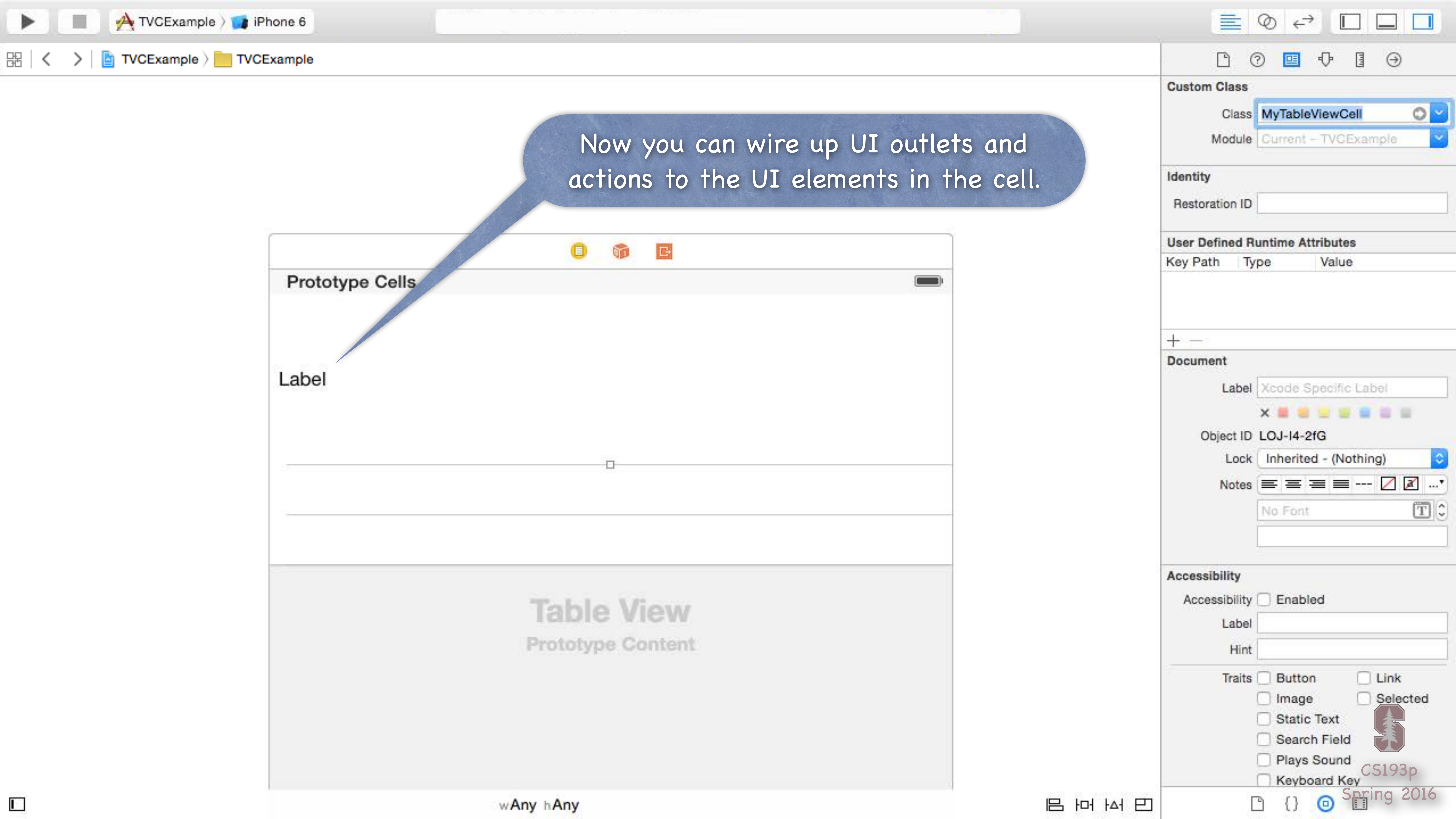
Prototype Cells

Label

Table View

Prototype Content

wAny hAny



Now you can wire up UI outlets and actions to the UI elements in the cell.

Custom Class

Class MyTableViewCell

Module Current – TVCExample

Identity

Restoration ID

User Defined Runtime Attributes

Key Path	Type	Value

Document

Label Xcode Specific Label

Object ID LOJ-I4-2fG

Lock Inherited - (Nothing)

Notes

No Font

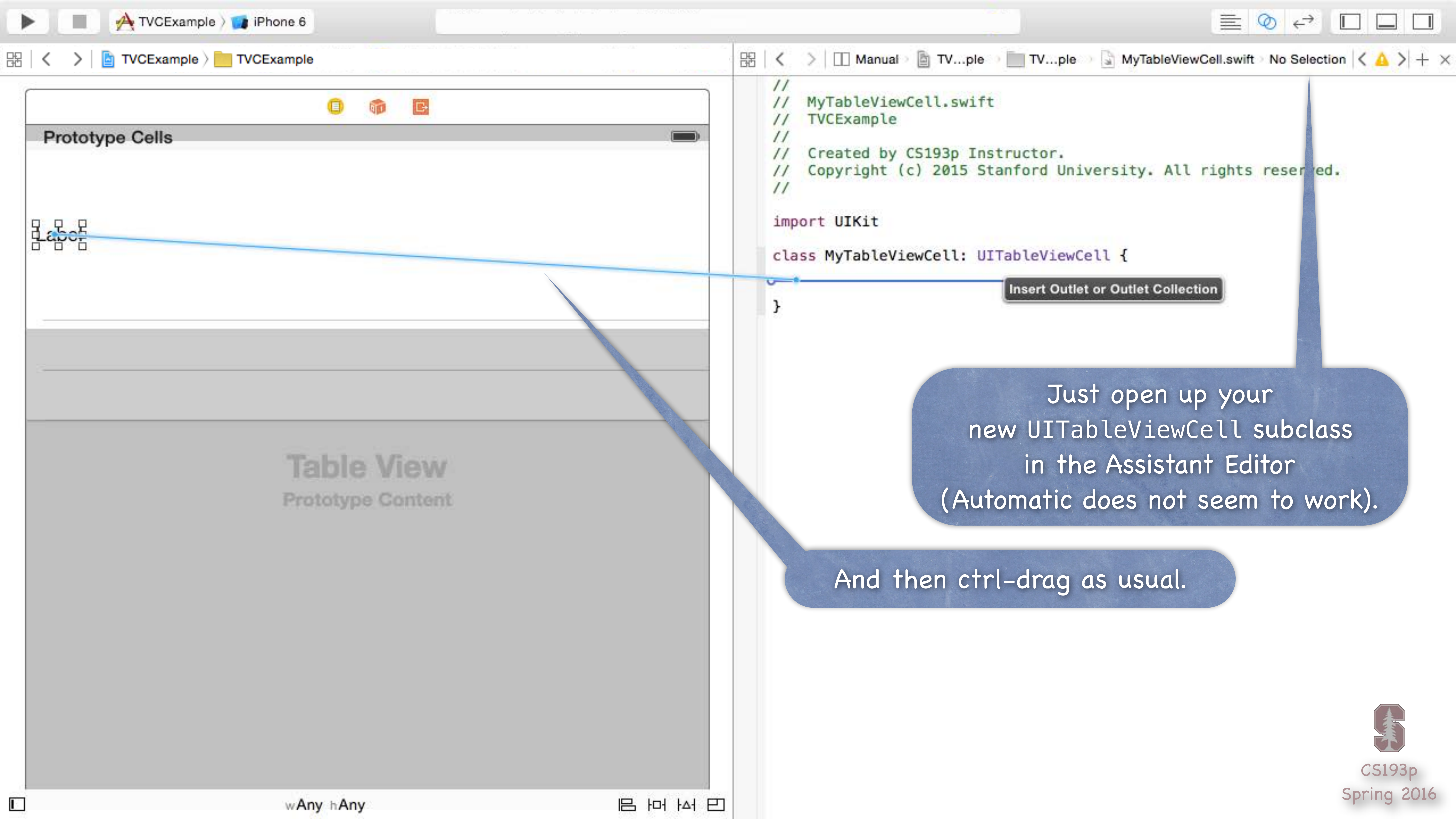
Accessibility

Accessibility [ ] Enabled

Label

Hint

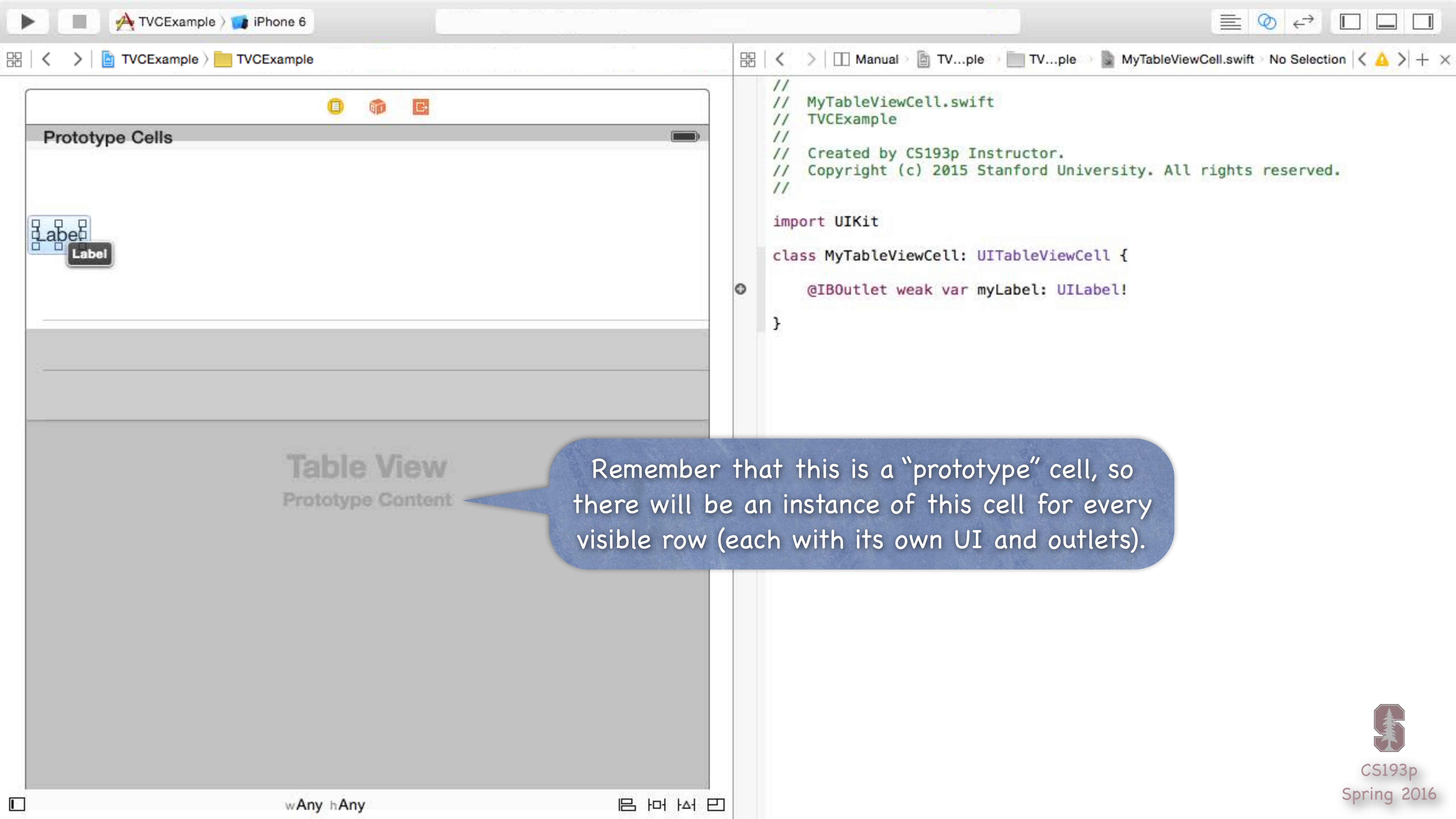
Traits [ ] Button [ ] Link [ ] Image [ ] Selected [ ] Static Text [ ] Search Field [ ] Plays Sound [ ] Keyboard Key



```
//
//  MyTableViewCell.swift
//  TVCExample
//
//  Created by CS193p Instructor.
//  Copyright (c) 2015 Stanford University. All rights reserved.
//

import UIKit

class MyTableViewCell: UITableViewCell {

}
```

Just open up your new UITableViewCell subclass in the Assistant Editor (Automatic does not seem to work).

And then ctrl-drag as usual.



```
//
//  MyTableViewCell.swift
//  TVCExample
//
//  Created by CS193p Instructor.
//  Copyright (c) 2015 Stanford University. All rights reserved.
//

import UIKit

class MyTableViewCell: UITableViewCell {

    @IBOutlet weak var myLabel: UILabel!

}
```

Remember that this is a "prototype" cell, so there will be an instance of this cell for every visible row (each with its own UI and outlets).

UITableView Protocols

- How to connect all this stuff up in code?
  - Connections to code are made using the UITableView's dataSource and delegate
    - The delegate is used to control how the table is displayed (it's look and feel)
    - The dataSource provides the data that is displayed inside the cells
  - UITableViewController automatically sets itself as the UITableView's delegate & dataSource
  - Your UITableViewController subclass will also have a property pointing to the UITableView …
  - `var tableView: UITableView // self.view in UITableViewController`
- When do we need to implement the dataSource?
  - Whenever the data in the table is dynamic (i.e. not static cells)
  - There are three important methods in this protocol …
    - How many sections in the table?
    - How many rows in each section?
    - Give me a view 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) …

Customizing Each Row

- Providing a UIView to draw each row …

  It has to be a UITableViewCell (which is a subclass of UIView) or subclass thereof
  Don't worry, if you have 10,000 rows, only the visible ones will have a UITableViewCell
  But this means that UITableViewCells are reused as rows appear and disappear
  This has ramifications for multithreaded situations, so be careful in that scenario

  The UITableView will ask its UITableViewDataSource for the UITableViewCell for a row …

```
func tableView(tv: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
{



}
```

NSIndexPath is just a container to pass you the section and row in question.

Customizing Each Row

- Providing a UIView to draw each row …

  It has to be a UITableViewCell (which is a subclass of UIView) or subclass thereof
  Don't worry, if you have 10,000 rows, only the visible ones will have a UITableViewCell
  But this means that UITableViewCells are reused as rows appear and disappear
  This has ramifications for multithreaded situations, so be careful in that scenario

  The UITableView will ask its UITableViewDataSource for the UITableViewCell for a row …

```
func tableView(tv: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
{
    let data = myInternalDataStructure[indexPath.section][indexPath.row]




}
```

Customizing Each Row

- Providing a UIView to draw each row …

  It has to be a UITableViewCell (which is a subclass of UIView) or subclass thereof
  Don't worry, if you have 10,000 rows, only the visible ones will have a UITableViewCell
  But this means that UITableViewCells are reused as rows appear and disappear
  This has ramifications for multithreaded situations, so be careful in that scenario

  The UITableView will ask its UITableViewDataSource for the UITableViewCell for a row …

```
func tableView(tv: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
{
    let data = myInternalDataStructure[indexPath.section][indexPath.row]

    let cell = . . . // create a UITableViewCell and load it up with data

    return cell
}
```

Customizing Each Row

- Providing a UIView to draw each row ...

  It has to be a UITableViewCell (which is a subclass of UIView) or subclass thereof
  Don't worry, if you have 10,000 rows, only the <u>visible</u> ones will have a UITableViewCell
  But this means that UITableViewCells are <u>reused</u> as rows appear and disappear
  This has ramifications for multithreaded situations, so be careful in that scenario

  The UITableView will ask its UITableViewDataSource for the UITableViewCell for a row ...

```
func tableView(tv: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
{
    let data = myInternalDataStructure[indexPath.section][indexPath.row]






}
```

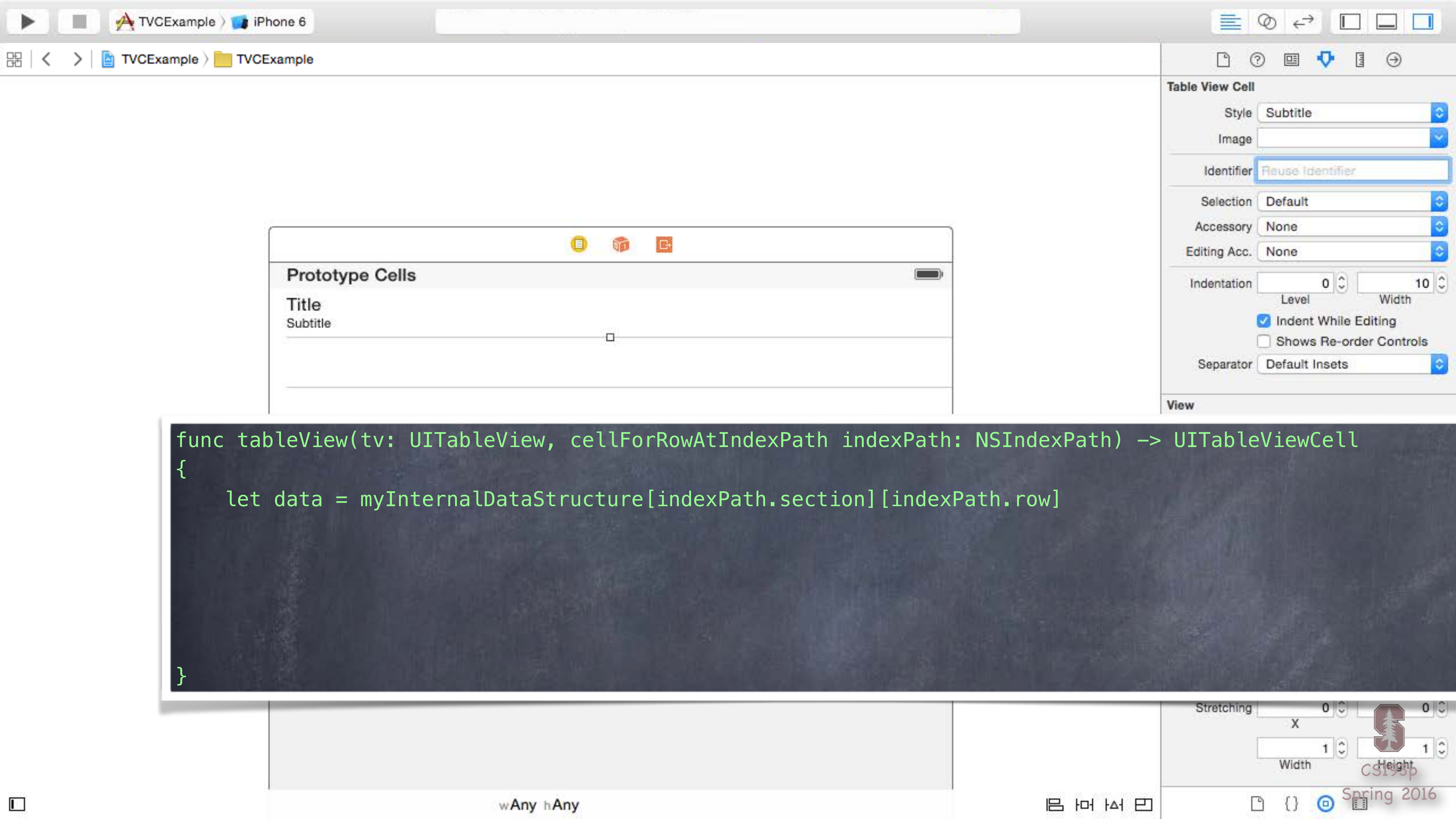


Table View Cell

Style: Subtitle
Image:
Identifier: Reuse Identifier
Selection: Default
Accessory: None
Editing Acc.: None
Indentation: 0 Level, 10 Width
[✓] Indent While Editing
[ ] Shows Re-order Controls
Separator: Default Insets
Stretching: 0 X, 0 Y
1 Width, 1 Height

```
func tableView(tv: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
{
    let data = myInternalDataStructure[indexPath.section][indexPath.row]



}
```

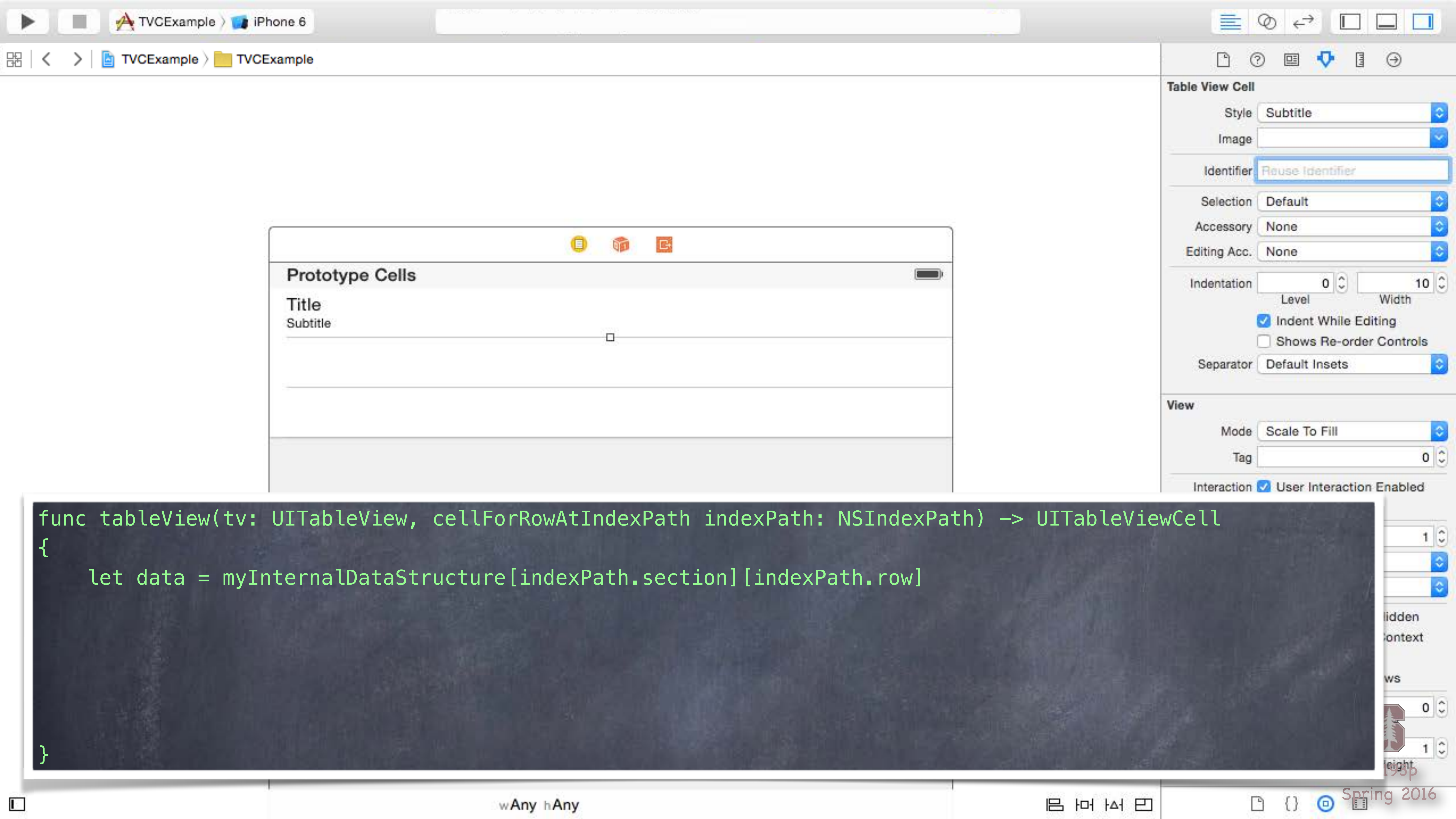


Table View Cell

Style: Subtitle

Image:

Identifier: Reuse Identifier

Selection: Default

Accessory: None

Editing Acc.: None

Indentation: 0 Level, 10 Width

☑ Indent While Editing

☐ Shows Re-order Controls

Separator: Default Insets

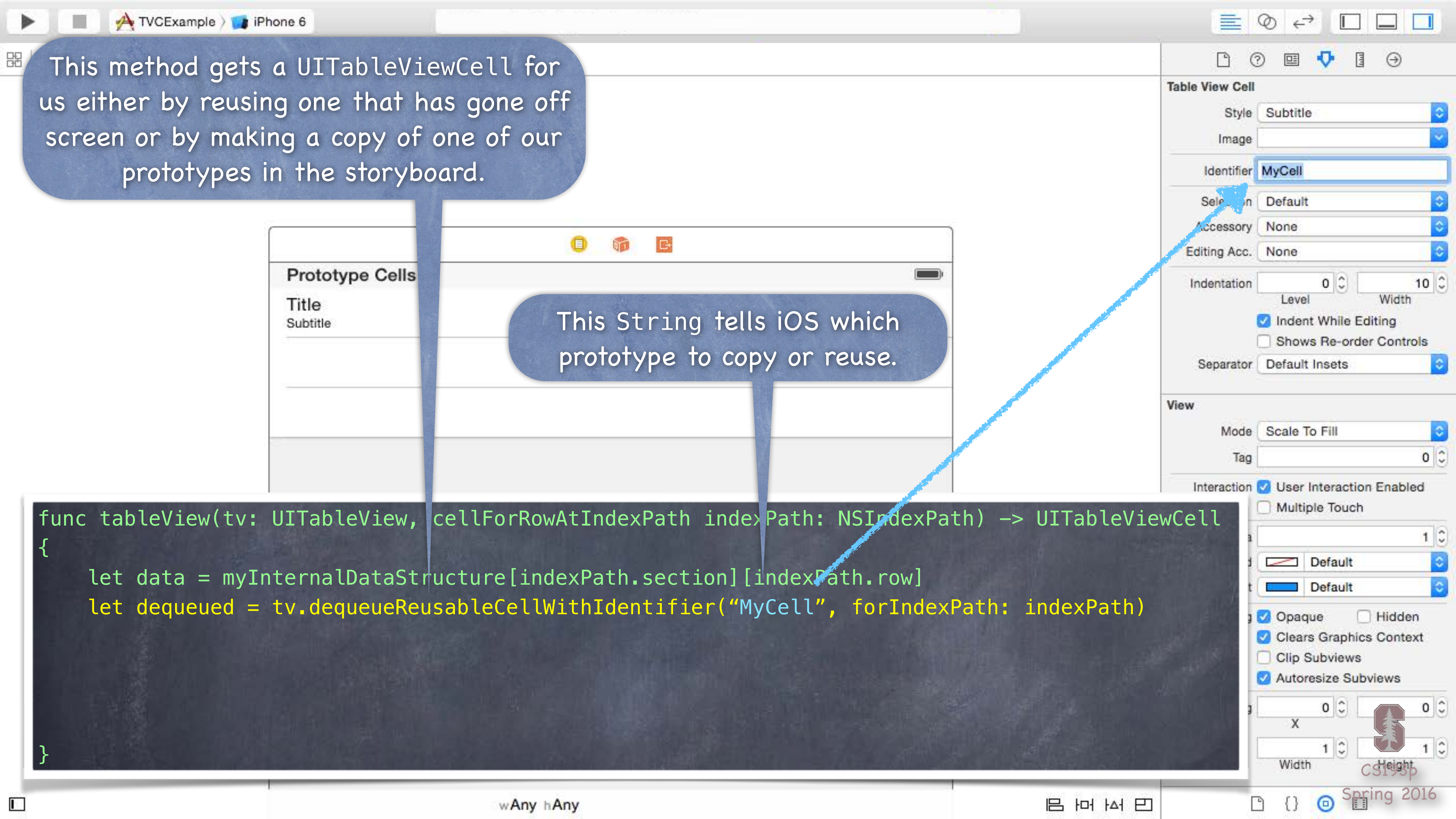
View

Mode: Scale To Fill

Tag: 0

Interaction ☑ User Interaction Enabled

```
func tableView(tv: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
{
    let data = myInternalDataStructure[indexPath.section][indexPath.row]



}
```



This method gets a UITableViewCell for us either by reusing one that has gone off screen or by making a copy of one of our prototypes in the storyboard.

This String tells iOS which prototype to copy or reuse.

```
func tableView(tv: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
{
    let data = myInternalDataStructure[indexPath.section][indexPath.row]
    let dequeued = tv.dequeueReusableCellWithIdentifier("MyCell", forIndexPath: indexPath)



}
```

Table View Cell

Style	Subtitle
Image	
Identifier	MyCell
Selection	Default
Accessory	None
Editing Acc.	None
Indentation	0 Level, 10 Width
	✓ Indent While Editing
	☐ Shows Re-order Controls
Separator	Default Insets

View

Mode	Scale To Fill
Tag	0
Interaction	✓ User Interaction Enabled
	☐ Multiple Touch
	1
	Default
	Default
	✓ Opaque ☐ Hidden
	✓ Clears Graphics Context
	☐ Clip Subviews
	✓ Autoresize Subviews
	0 X, 0
	1 Width, 1 Height

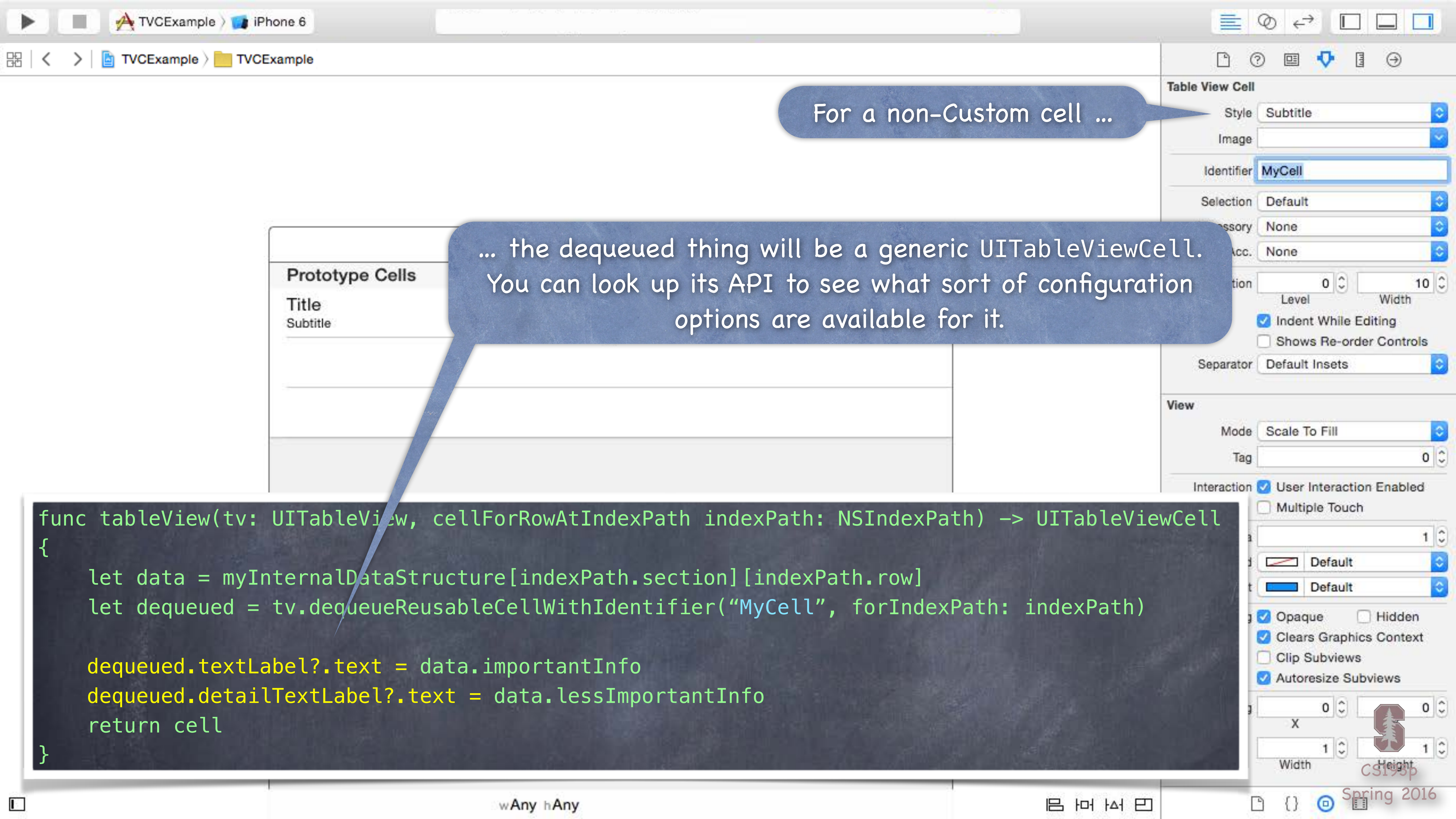


Table View Cell

Style: Subtitle
Image:
Identifier: MyCell
Selection: Default
…ssory: None
…cc.: None
…tion: 0 Level, 10 Width
✓ Indent While Editing
☐ Shows Re-order Controls
Separator: Default Insets

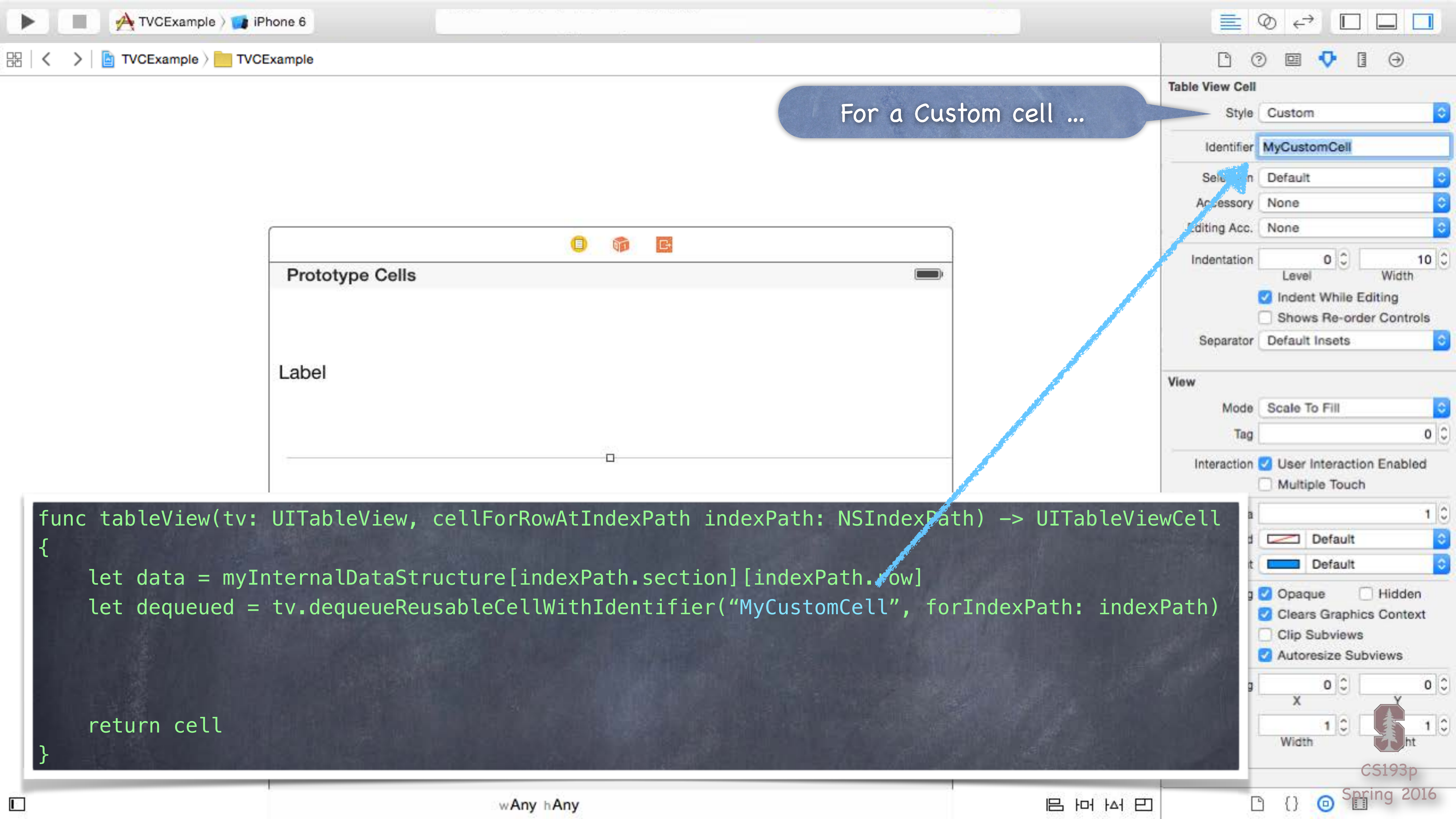
View

Mode: Scale To Fill
Tag: 0
Interaction: ✓ User Interaction Enabled
☐ Multiple Touch
1
Default
Default
✓ Opaque ☐ Hidden
✓ Clears Graphics Context
☐ Clip Subviews
✓ Autoresize Subviews
0 X, 0
1 Width, 1 Height

For a non-Custom cell …

… the dequeued thing will be a generic UITableViewCell.
You can look up its API to see what sort of configuration options are available for it.

```
func tableView(tv: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
{
    let data = myInternalDataStructure[indexPath.section][indexPath.row]
    let dequeued = tv.dequeueReusableCellWithIdentifier("MyCell", forIndexPath: indexPath)

    dequeued.textLabel?.text = data.importantInfo
    dequeued.detailTextLabel?.text = data.lessImportantInfo
    return cell
}
```



For a Custom cell …

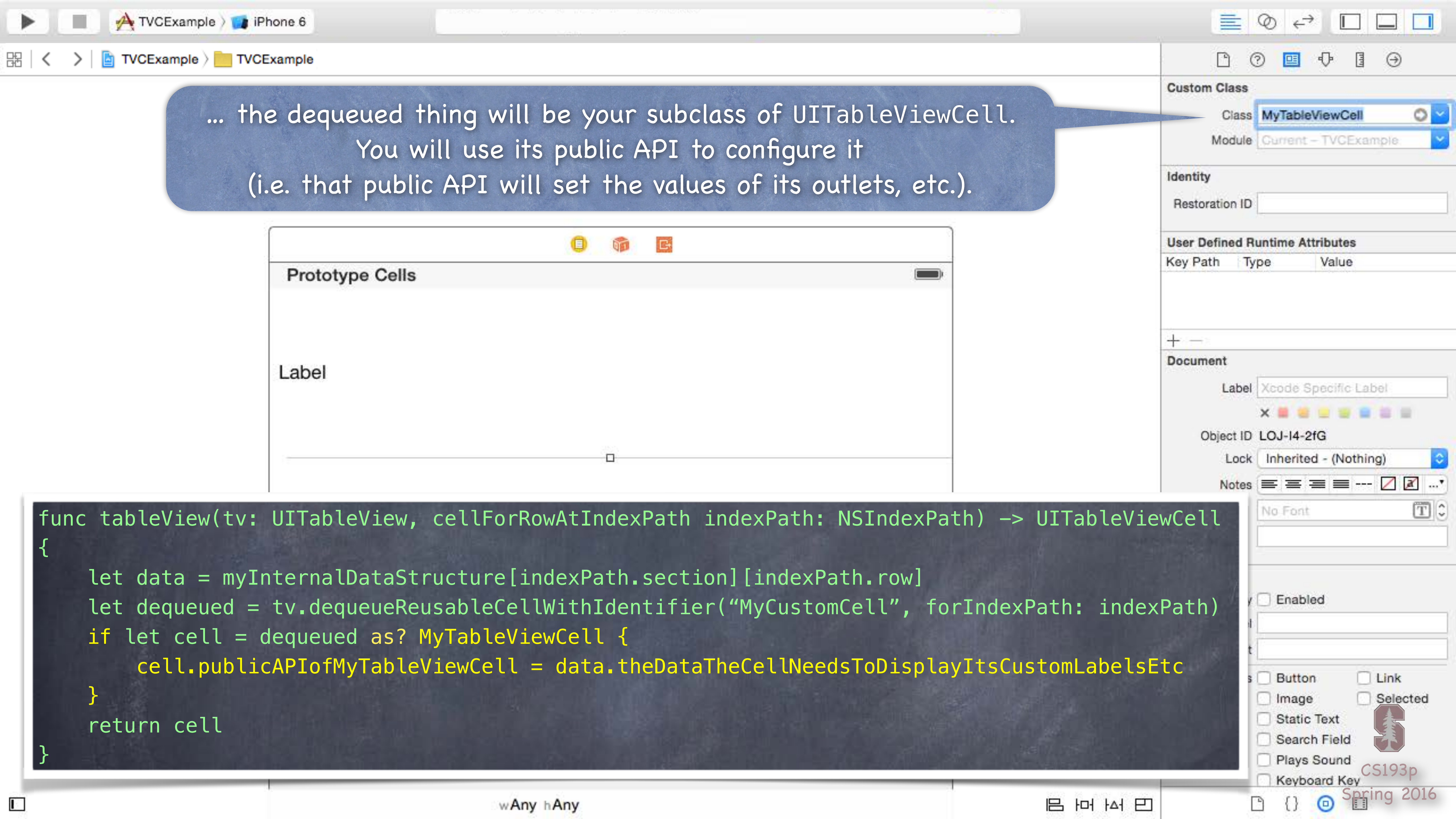
Table View Cell

- Style: Custom
- Identifier: MyCustomCell
- Selection: Default
- Accessory: None
- Editing Acc.: None
- Indentation: Level 0, Width 10
- ☑ Indent While Editing
- ☐ Shows Re-order Controls
- Separator: Default Insets

View

- Mode: Scale To Fill
- Tag: 0
- Interaction: ☑ User Interaction Enabled, ☐ Multiple Touch
- 1
- Default
- Default
- ☑ Opaque ☐ Hidden
- ☑ Clears Graphics Context
- ☐ Clip Subviews
- ☑ Autoresize Subviews
- X 0, Y 0
- Width 1, Height 1

```swift
func tableView(tv: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
{
    let data = myInternalDataStructure[indexPath.section][indexPath.row]
    let dequeued = tv.dequeueReusableCellWithIdentifier("MyCustomCell", forIndexPath: indexPath)


    return cell
}
```



… the dequeued thing will be your subclass of UITableViewCell.
You will use its public API to configure it
(i.e. that public API will set the values of its outlets, etc.).

```
func tableView(tv: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
{
    let data = myInternalDataStructure[indexPath.section][indexPath.row]
    let dequeued = tv.dequeueReusableCellWithIdentifier("MyCustomCell", forIndexPath: indexPath)
    if let cell = dequeued as? MyTableViewCell {
        cell.publicAPIofMyTableViewCell = data.theDataTheCellNeedsToDisplayItsCustomLabelsEtc
    }
    return cell
}
```

UITableViewDataSource

- How does a dynamic table know how many rows there are?
  And how many sections, too, of course?
  Via these `UITableViewDataSource` protocol methods …

```
func numberOfSectionsInTableView(sender: UITableView) -> Int
func tableView(sender: UITableView, numberOfRowsInSection: Int) -> Int
```

- Number of sections is 1 by default
  In other words, if you don't implement `numberOfSectionsInTableView`, it will be 1
- No default for `numberOfRowsInSection`
  This is a <u>required</u> method in this protocol (as is `cellForRowAtIndexPath`)
- What about a static table?
  Do not implement these `dataSource` methods for a static table
  `UITableViewController` will take care of that for you
  You edit the data directly in the storyboard

UITableViewDataSource

- Summary

  Loading your table view with data is simple …

  1. set the table view's `dataSource` to your Controller (automatic with `UITableViewController`)
  2. implement `numberOfSectionsInTableView` and `numberOfRowsInSection`
  3. implement `cellForRowAtIndexPath` to return loaded-up `UITableViewCells`

- Section titles are also considered part of the table's "data"

  So you return this information via `UITableViewDataSource` methods …

  `func tableView(UITableView, titleFor{Header,Footer}InSection: Int) -> String`

  If a `String` is not sufficient, the `UITableView`'s `delegate` can provide a `UIView`

- There are a number of other methods in this protocol

  But we're not going to cover them in lecture

  They are mostly about dealing with editing the table by deleting/moving/inserting rows

  That's because when rows are deleted, inserted or moved, it would likely modify the Model

  (and we're talking about the `UITableView`<u>`DataSource`</u> protocol here)

How do you segue when a row is touched?

Just ctrl-drag from a prototype (or static) row to another MVC of course ...

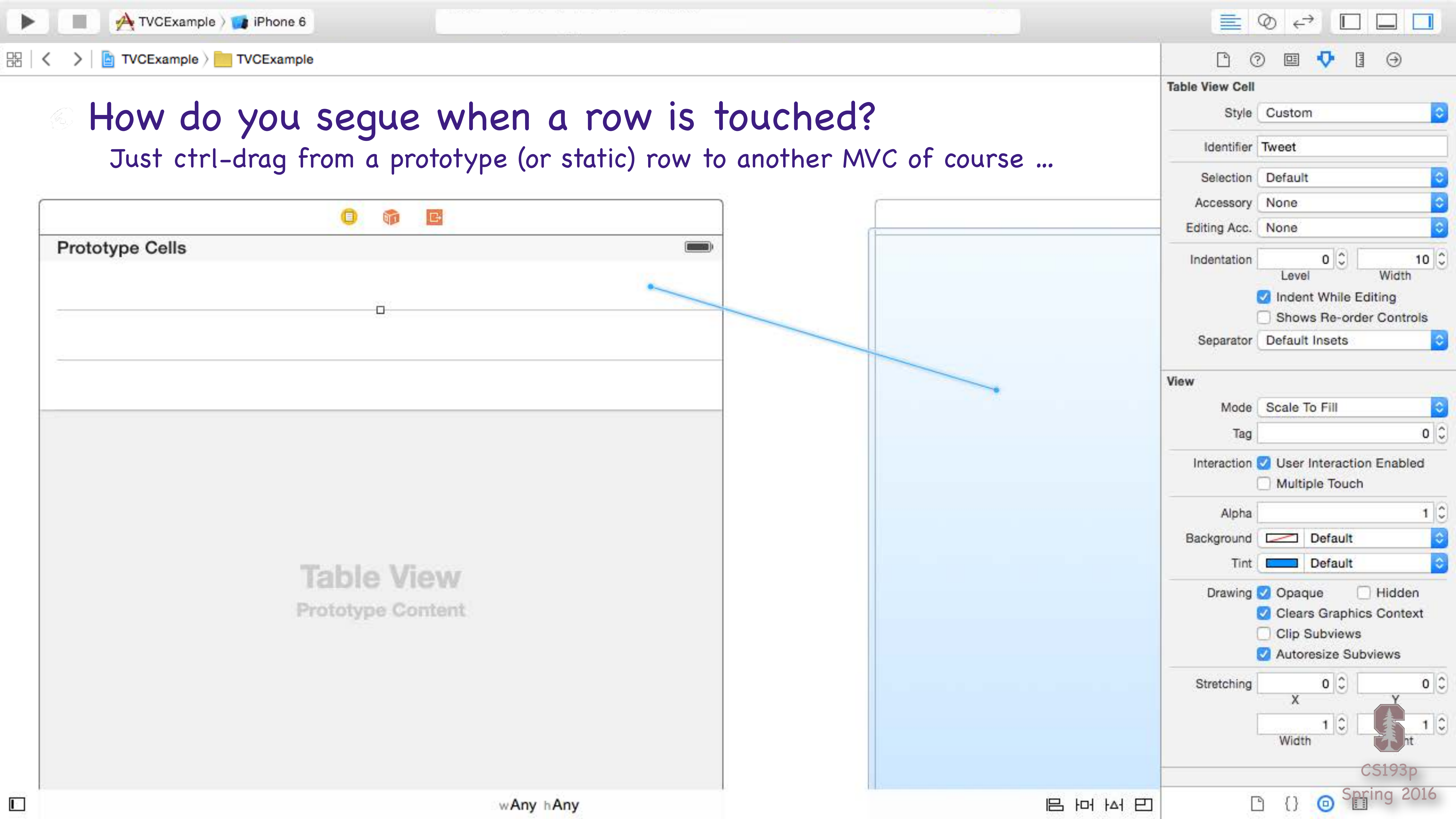
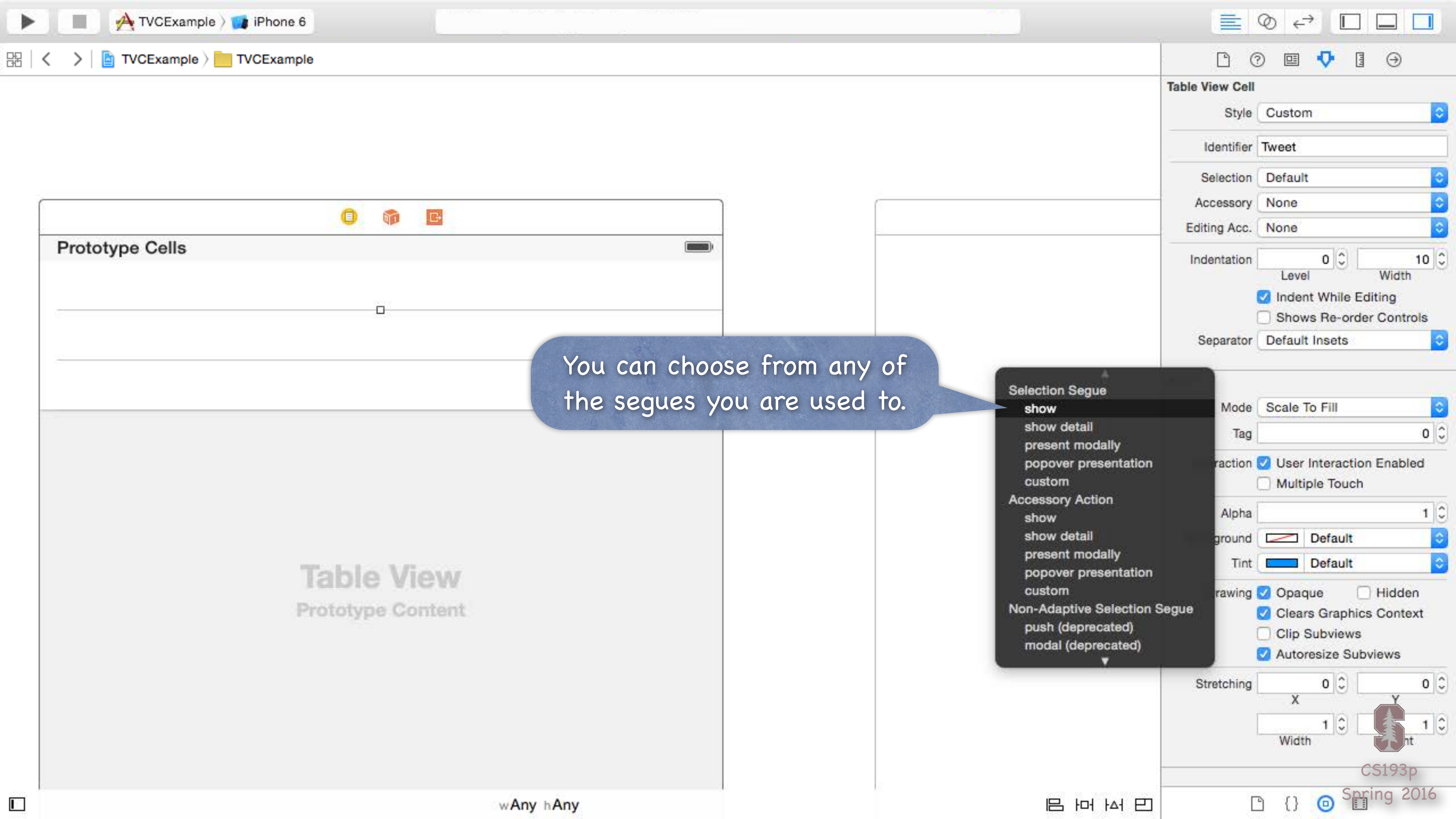


Table View Cell

Style: Custom
Identifier: Tweet
Selection: Default
Accessory: None
Editing Acc.: None
Indentation: Level 0, Width 10
☑ Indent While Editing
☐ Shows Re-order Controls
Separator: Default Insets

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



You can choose from any of the segues you are used to.

Table View Cell

Style: Custom
Identifier: Tweet
Selection: Default
Accessory: None
Editing Acc.: None
Indentation: 0 (Level), 10 (Width)
[✓] Indent While Editing
[ ] Shows Re-order Controls
Separator: Default Insets

Mode: Scale To Fill
Tag: 0
[✓] User Interaction Enabled
[ ] Multiple Touch
Alpha: 1
Background: Default
Tint: Default
[✓] Opaque [ ] Hidden
[✓] Clears Graphics Context
[ ] Clip Subviews
[✓] Autoresize Subviews
Stretching: 0 (X), 0 (Y), 1 (Width), 1

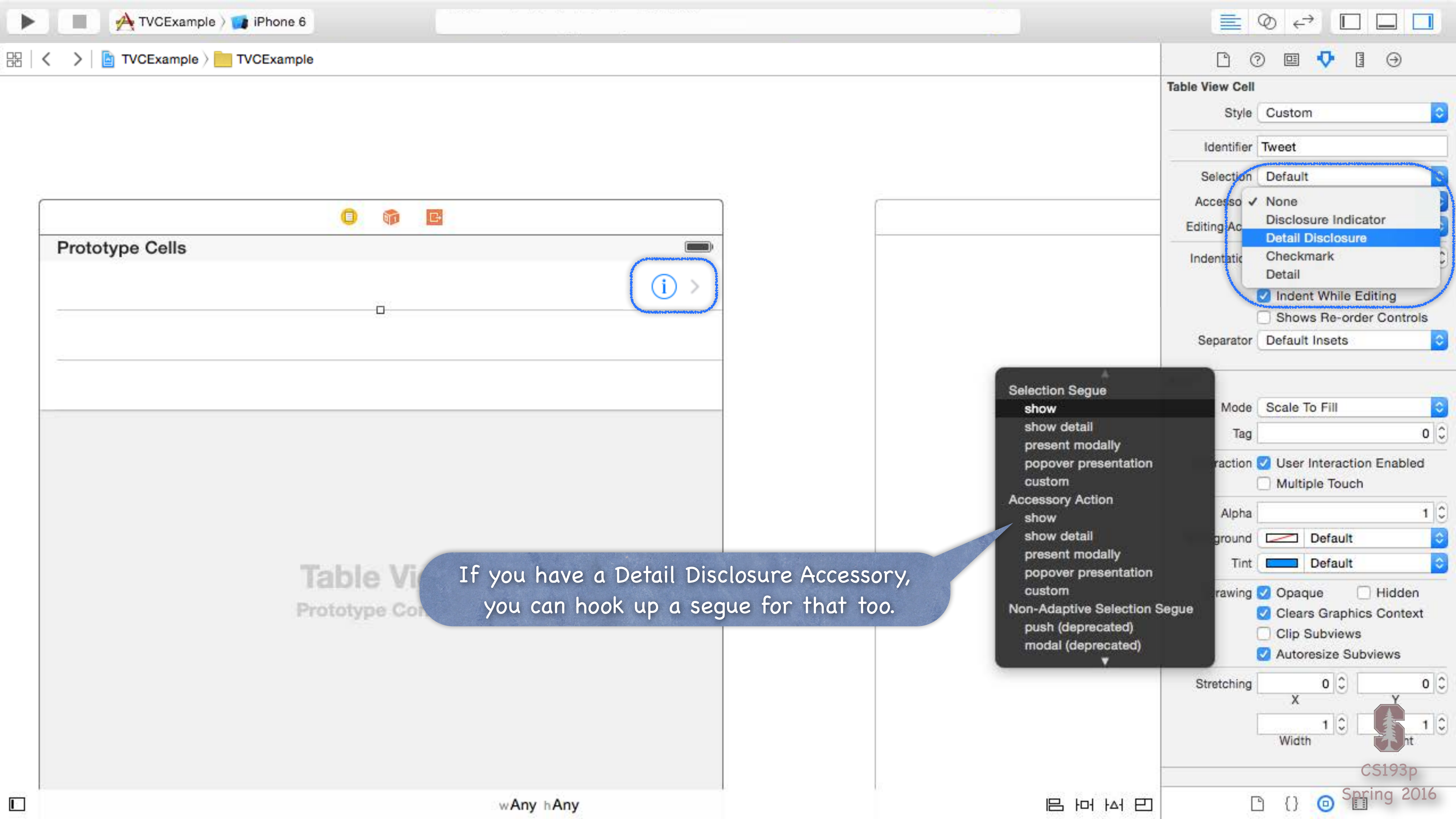


Table View Cell

Style: Custom

Identifier: Tweet

Selection: Default

Accessory: ✓ None / Disclosure Indicator / [✓ Detail Disclosure] / Checkmark / Detail

☑ Indent While Editing

☐ Shows Re-order Controls

Separator: Default Insets

Mode: Scale To Fill

Tag: 0

☑ User Interaction Enabled

☐ Multiple Touch

Alpha: 1

Background: Default

Tint: Default

☑ Opaque ☐ Hidden

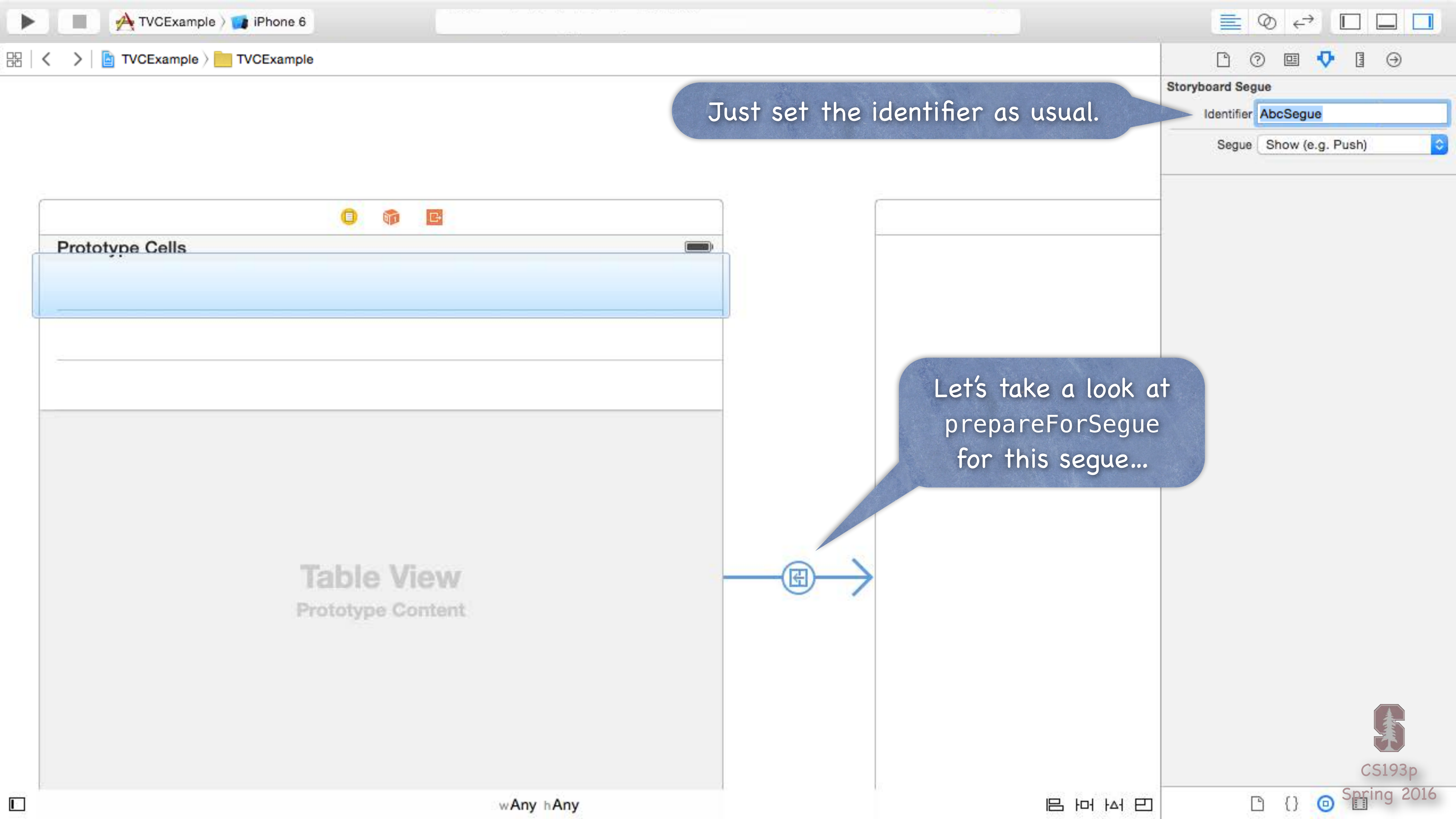
☑ Clears Graphics Context

☐ Clip Subviews

☑ Autoresize Subviews

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

If you have a Detail Disclosure Accessory, you can hook up a segue for that too.



Storyboard Segue

Identifier AbcSegue

Segue Show (e.g. Push)

Just set the identifier as usual.

Let's take a look at prepareForSegue for this segue...

Table View Segues

- Preparing to segue from a row in a table view

  The sender argument to prepareForSegue is the UITableViewCell of that row …

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
        case "XyzSegue": // handle XyzSegue here
        case "AbcSegue":






        default: break
        }
    }
}
```

  You can see now why sender is AnyObject

  Sometimes it's a UIButton, sometimes it's a UITableViewCell

Table View Segues

- Preparing to segue from a row in a table view

  The sender argument to prepareForSegue is the UITableViewCell of that row …

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
        case “XyzSegue”: // handle XyzSegue here
        case “AbcSegue”:
            if let cell = sender as? MyTableViewCell {



            }
        default: break
        }
    }
}
```

  So you will need to cast sender with as? to turn it into a UITableViewCell

  If you have a custom UITableViewCell subclass, you can cast it to that if it matters

Table View Segues

- Preparing to segue from a row in a table view

  The sender argument to prepareForSegue is the UITableViewCell of that row …

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
        case "XyzSegue": // handle XyzSegue here
        case "AbcSegue":
            if let cell = sender as? MyTableViewCell,
               let indexPath = tableView.indexPathForCell(cell) {


            }
        default: break
        }
    }
}
```

indexPathForCell does not accept AnyObject.
It has to be a UITableViewCell of some sort.

Usually we will need the NSIndexPath of the UITableViewCell
Because we use that to index into our internal data structures

Table View Segues

- Preparing to segue from a row in a table view

  The sender argument to prepareForSegue is the UITableViewCell of that row …

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
        case "XyzSegue": // handle XyzSegue here
        case "AbcSegue":
            if let cell = sender as? MyTableViewCell,
               let indexPath = tableView.indexPathForCell(cell),
               let seguedToMVC = segue.destinationViewController as? MyVC {

            }
        default: break
        }
    }
}
```

  Now we just get our destination MVC as the proper class as usual …

Table View Segues

- Preparing to segue from a row in a table view

  The sender argument to prepareForSegue is the UITableViewCell of that row …

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
        case “XyzSegue”: // handle XyzSegue here
        case “AbcSegue”:
            if let cell = sender as? MyTableViewCell,
               let indexPath = tableView.indexPathForCell(cell),
               let seguedToMVC = segue.destinationViewController as? MyVC {
                seguedToMVC.publicAPI = data[indexPath.section][indexPath.row]
            }
        default: break
        }
    }
}
```

  and then get data from our internal data structure using the NSIndexPath’s section and row

Table View Segues

- Preparing to segue from a row in a table view

  The sender argument to prepareForSegue is the UITableViewCell of that row …

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
        case "XyzSegue": // handle XyzSegue here
        case "AbcSegue":
            if let cell = sender as? MyTableViewCell,
               let indexPath = tableView.indexPathForCell(cell),
               let seguedToMVC = segue.destinationViewController as? MyVC {
                seguedToMVC.publicAPI = data[indexPath.section][indexPath.row]
            }
        default: break
        }
    }
}
```

  and then get data from our internal data structure using the NSIndexPath's section and row
  and use that information to prepare the segued-to API using its public API

UITableViewDelegate

- So far we've only talked about the UITableView's dataSource
  - But UITableView has another protocol-driven delegate called its delegate
- The delegate controls how the UITableView is displayed
  - Not <u>the data</u> it displays (that's the dataSource's job), <u>how</u> it is displayed
- Common for dataSource and delegate to be the same object
  - Usually the Controller of the MVC containing the UITableView
  - Again, this is set up automatically for you if you use UITableViewController
- The delegate also lets you observe what the table view is doing
  - Especially responding to when the user selects a row
  - Usually you will just segue when this happens, but if you want to track it directly …

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)

  Example: if the master in a split view wants to update the detail without segueing to a new one

```
func tableView(sender: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {
    // go do something based on information about my Model
    // corresponding to indexPath.row in indexPath.section
    // maybe directly update the Detail if I'm the Master in a split view?
}
```

- Delegate method sent when Detail Disclosure button is touched

```
func tableView(UITableView, accessoryButtonTappedForRowWithIndexPath: NSIndexPath)
```

  Again, you can just segue from that Detail Disclosure button if you prefer

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
  - Copying/pasting rows

UITableView

- What if your Model changes?

  ```
  func reloadData()
  ```

  Causes the `UITableView` to call `numberOfSectionsInTableView` and `numberOfRowsInSection`
  all over again and then `cellForRowAtIndexPath` on each visible row

  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 …

  ```
  func reloadRowsAtIndexPaths(indexPaths: [NSIndexPath],
                              withRowAnimation: UITableViewRowAnimation)
  ```

UITableView

- Controlling the height of rows

  Row height can be fixed (`UITableView`'s `var rowHeight: CGFloat`)

  Or it can be determined using autolayout (`rowHeight = UITableViewAutomaticDimension`)

  If you do automatic, help the table view out by setting `estimatedRowHeight` to something

  The `UITableView`'s `delegate` can also control row heights …

  `func tableView(UITableView, {estimated}heightForRowAtIndexPath: NSIndexPath) -> CGFloat`

  Beware: the non-estimated version of this could get called A LOT if you have a big table

UITableView

- There are dozens of other methods in `UITableView` itself
  - 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.
  - As always, part of learning the material in this course is studying the documentation