

# Stanford CS193p

Developing Applications for iOS  
Winter 2017



CS193p  
Winter 2017

# Today

- **Table View**

Way to display large data sets

Demo: Twitter Client



# UITableView

UITableViewController.PLAIN

.grouped

Carrier Groups All Contacts +

Carrier C Search DISTANCES A Ryan Clady D Eric Decker M Peyton Manning T Knowshon Moreno Demarius Thomas W Wes Welker

In Miles  
In Kilometers

MAP LABELS

Always in English

PREFERRED DIRECTIONS

Driving

Walking

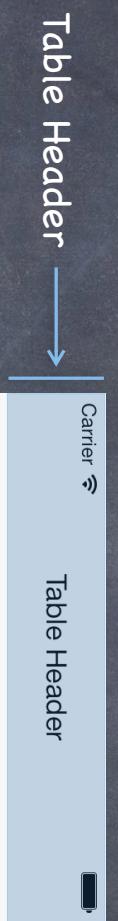
Dynamic (List)  
& Plain  
(ungrouped)

Static  
&  
Grouped



# UITableView

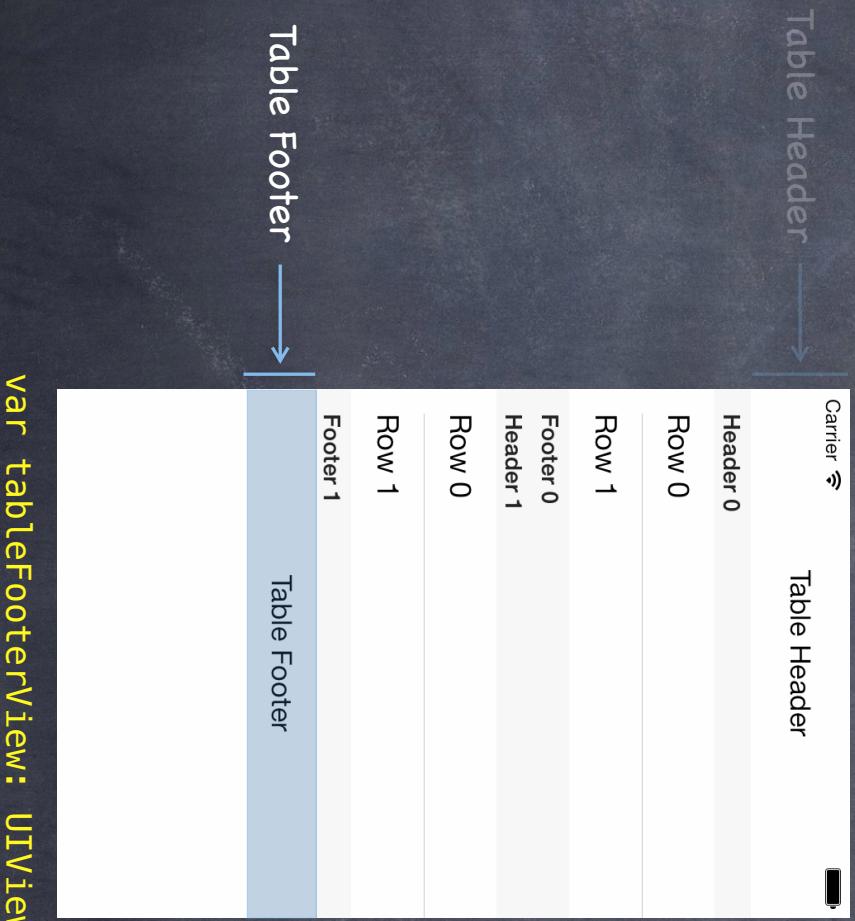
## Plain Style



```
var tableHeaderView: UIView
```

# UITableViewController

## Plain Style



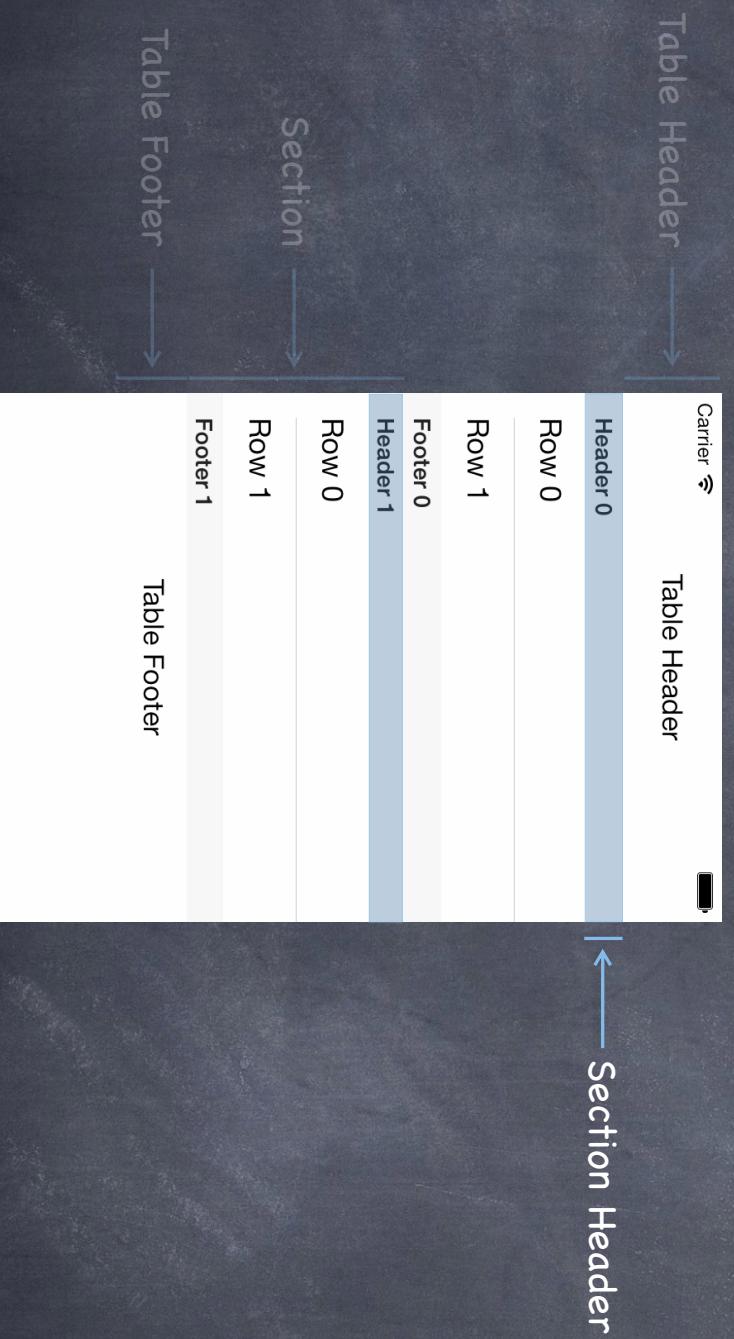
# UITableViewController

## 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			

# UITableViewController

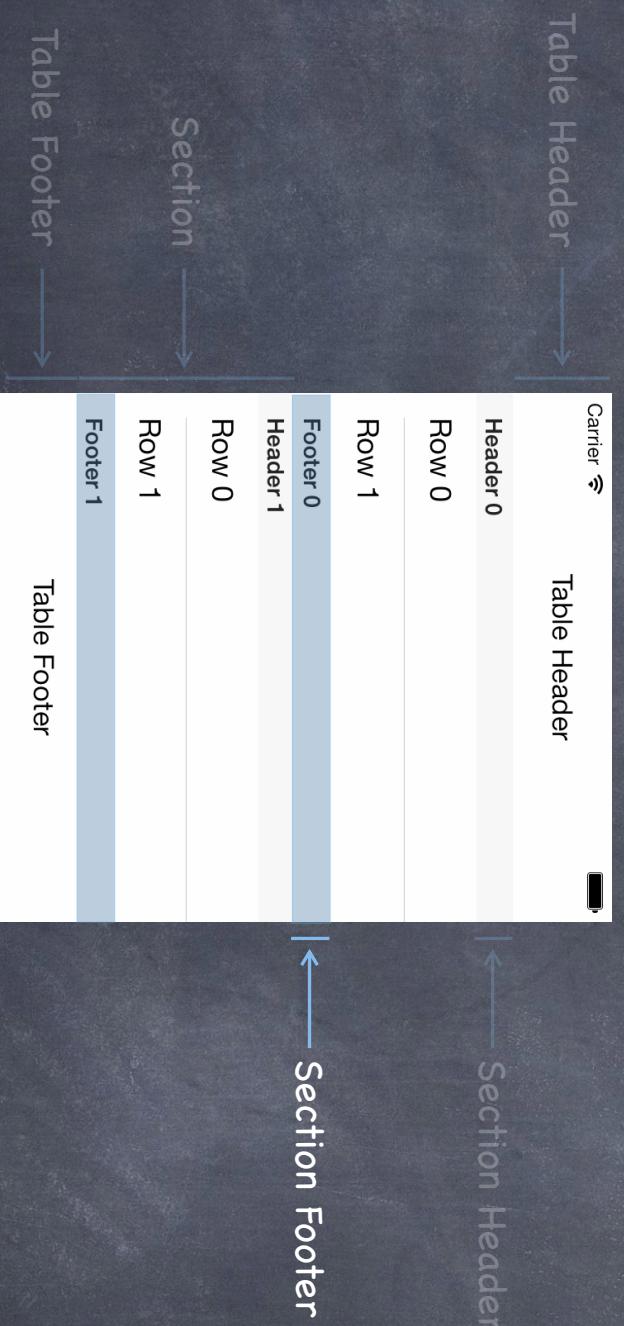
## Plain Style



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

# UITableViewController

## Plain Style

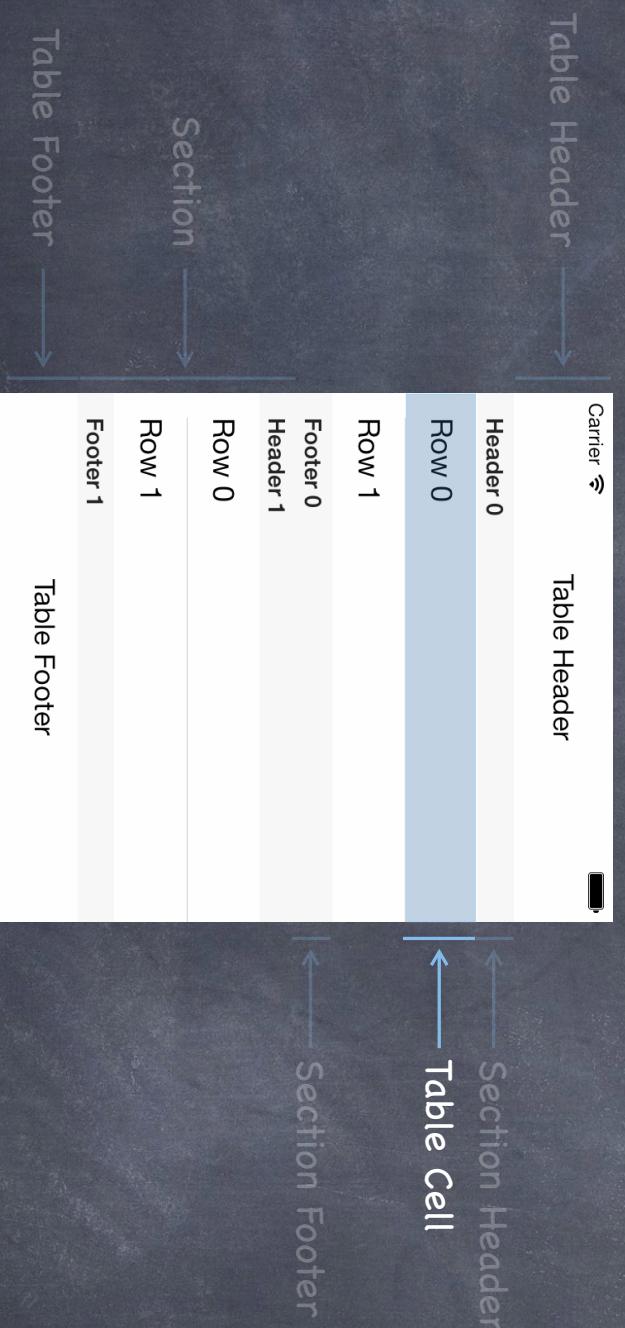


`UITableViewDataSource's tableView(UITableView, titleForFooterInSection: Int)`



# UITableView

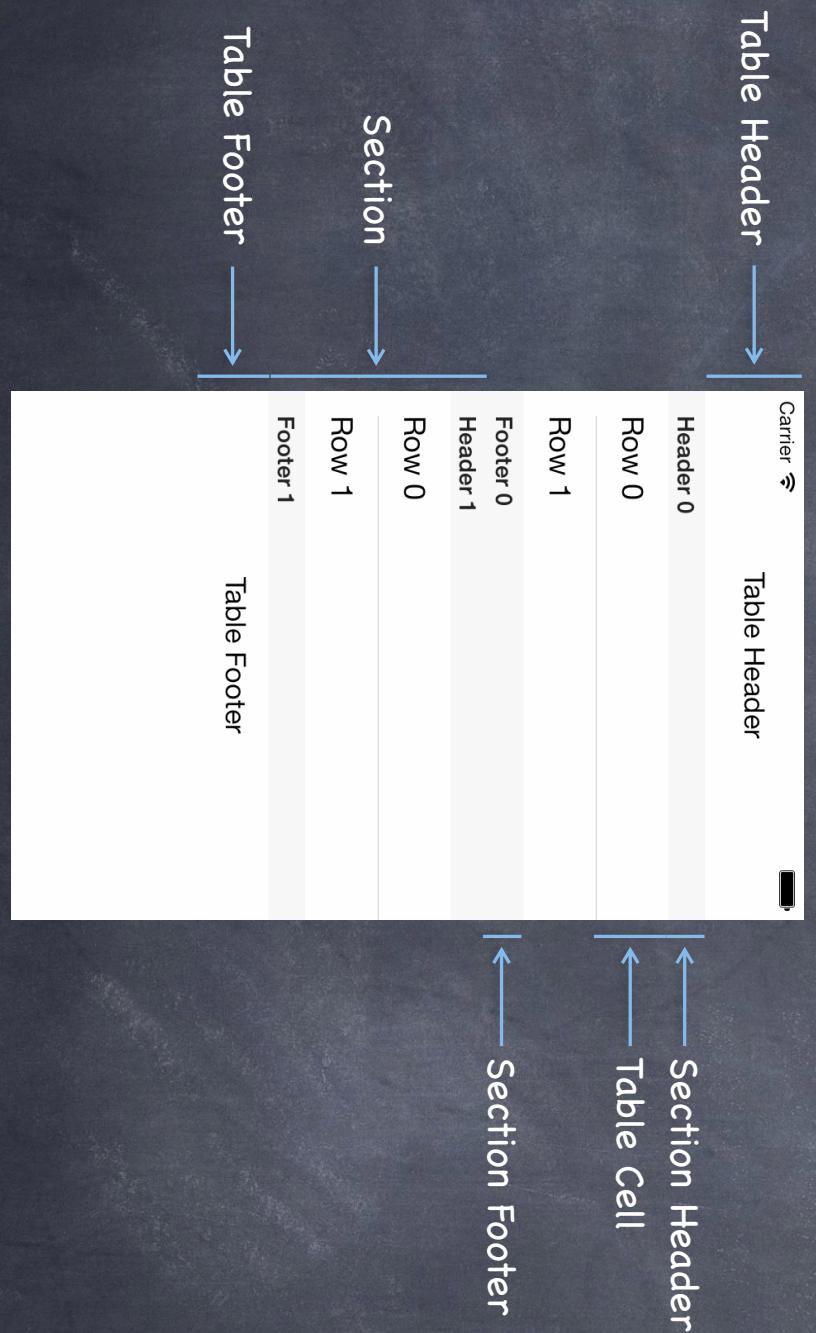
## Plain Style



`UITableViewDataSource's tableView(tableView, cellForRowAtIndexPath indexPath:)`

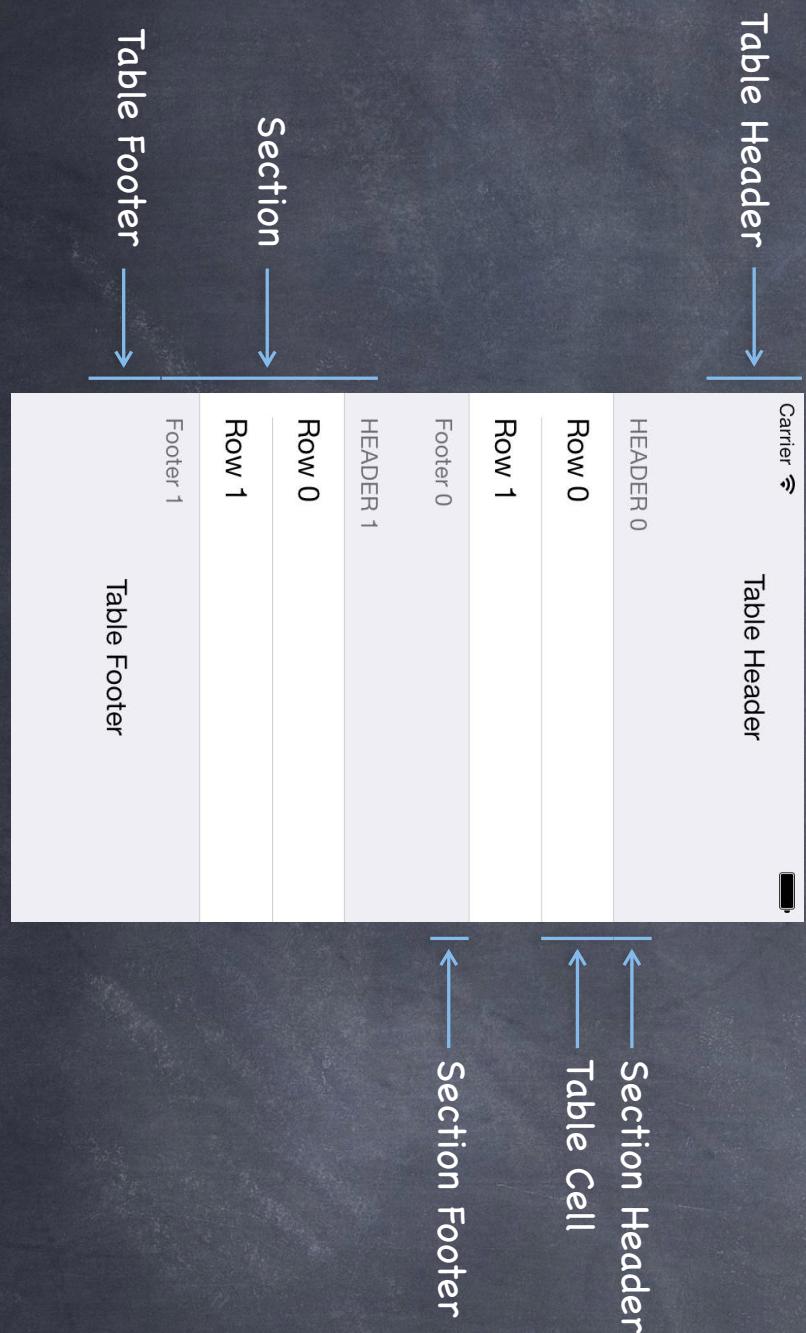
# UITableView

## Plain Style



# UITableViewController

## Grouped Style



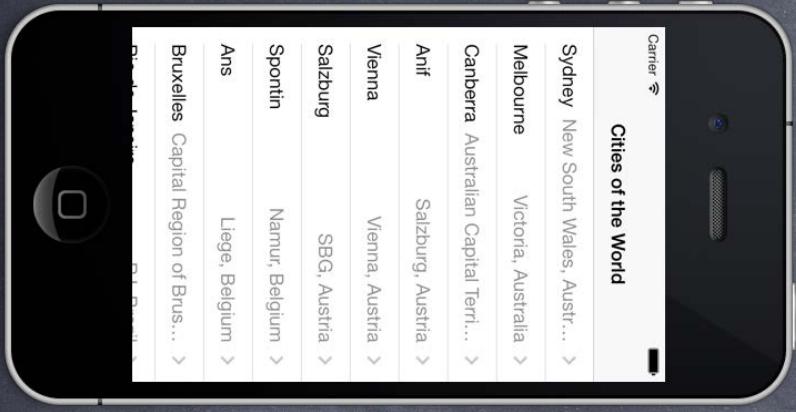
# Sections or Not

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 Cuitláhuacán	Distrito Federal, Mexico	
Netherlands		
Amsterdam	North Holland, Netherlands	

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	Milan, Italy	

# Cell Type



Right Detail

.value1

Left Detail

.value2



CS193P  
Winter 2017

Basic

.default

Subtitle

UITableViewCellStyle.subtitle

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.

No Selection

	Table View Controller - A controller that manages a UITableView through a hierarchy of views.
	Collection View Controller - A controller that manages a UICollectionView.
	Tab Bar Controller - A controller that manages a UITabBarController.

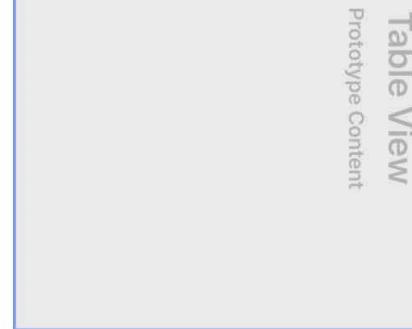
Controller: (subclass of) UITableViewcontroller  
Controller's view property: the UITableView

The screenshot shows the Xcode Interface Builder storyboard editor. A blue callout bubble points from the text above to a Table View controller in the storyboard. The storyboard contains a single Table View with a prototype cell labeled "Prototype Content". Below the storyboard, the Table View Controller inspector is open, showing the following settings:

Table View Controller
Selection <input checked="" type="checkbox"/> Clear on Appearance
Refreshing <input type="checkbox"/> Disabled
<b>View Controller</b>
Title <input type="text"/>
<input type="checkbox"/> Is Initial View Controller
<b>Layout</b> <input checked="" type="checkbox"/> Adjust Scroll View Insets
<input type="checkbox"/> Hide Bottom Bar on Push
<input checked="" type="checkbox"/> Resize View From NIB
<input type="checkbox"/> Use Full Screen (Deprecated)
<b>Extend Edges</b> <input checked="" type="checkbox"/> Under Top Bars
<input checked="" type="checkbox"/> Under Bottom Bars
<input type="checkbox"/> Under Opaque Bars

At the bottom of the screen, there are several interface icons: a magnifying glass, a square, a circle, a triangle, a rectangle, and a document.

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



File Edit View Find Navigate Editor Product Debug Source Control Window Help

New

Add Files to "TVCExample"...

Open... Open Recent Open Quickly...

Close Window Close Tab Close "Main.storyboard"

Close Project

Save Duplicate... Revert to Saved... Unlock... Export...

Show in Finder Open with External Editor Save As Workspace... Project Settings...

Page Setup... Print...

File... Playground... Target... Project... Workspace... Group Group from Selection

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

X Object ID ab2-9w-rly Lock Inherited - (Nothing)

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

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

Tab Bar Controller - A controller that manages a tab bar view.

CS193P Winter 2017

Just use File -> New -> File ... as usual.

Choose a template for your new file:

iOS

watchOS tvOS macOS

## Source



Cocoa Touch Class



UI Test Case Class



Unit Test Case Class



Playground



Swift File

## Objective-C File

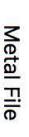
Header File



C File



C++ File



Metal File

Cancel

Previous

Next

Filter

## Custom Class

Class



UITableViewController



None



## Identity



Storyboard ID



Restoration ID



Use Storyboard ID

## Document

Label



Xcode Specific Label



X



Red



Yellow



Green



Blue



Purple



Grey

Object ID ab2-9w-wly



X



Red



Yellow



Green



Blue



Purple



Grey



▼



▼



▼

Lock

Inherited - (Nothing)

▼

Storyboard



View



Empty



Launch Screen



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

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

Tab Bar Controller - A controller for tab bar interfaces.

CS193P Winter 2017

Q E H H

Q E H H

Q

E

H

H

H

H

H

H

H

H



&lt;

&gt;



TVCEExample



TVCEExample



iPhone 7



Ready

— 75% +

Choose options for your new file:

Class: UITableViewController

Subclass of: UITableViewController

UIView

UIViewController

UITableViewViewController

UITableViewCell

UICollectionViewViewController

Cancel

Previous

Next

Make sure you set the  
superclass to  
UITableViewController

User Defined Runtime Attributes

Label Xcode specific Label

X

Object ID ab2-9w-rly

Lock Inherited - (Nothing)

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

Table View

Prototype Content

Custom Class

Class: MyTableViewCellController

Module: Current - TVCEExam...

Identity

Storyboard ID:

Restoration ID:

Use Storyboard ID

User Defined Runtime Attributes

Key Path	Type	Value
+	-	

Document

Label: Xcode Specific Label

X:

Object ID: ab2-9w-rly

Lock: Inherited - (Nothing)

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

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

Tab Bar Controller - A controller

Your UITableViewContoller 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.

The screenshot shows the Xcode interface with the storyboard open. In the Utilities panel, the 'Identity' tab is selected for the 'MyTableViewController' class. The 'Custom Class' dropdown is set to 'MyTableViewController'. Below it, the 'Module' dropdown shows 'Current - TVCEExam...'. The storyboard view shows a table view with a single prototype cell containing three icons: a yellow square, a red circle, and a blue rectangle. A callout bubble points from the bottom of the storyboard towards the 'Identity' tab in the Utilities panel.

Custom Class  
Class MyTableViewController  
Module Current - TVCEExam...

Prototype Cells

Identity

Table View

Prototype Content

User Defined Runtime Attributes

Key Path Type Value

+ -

Document

Label Xcode Specific Label

X

Object ID ab2-9w-rly

Lock Inherited - (Nothing)

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

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

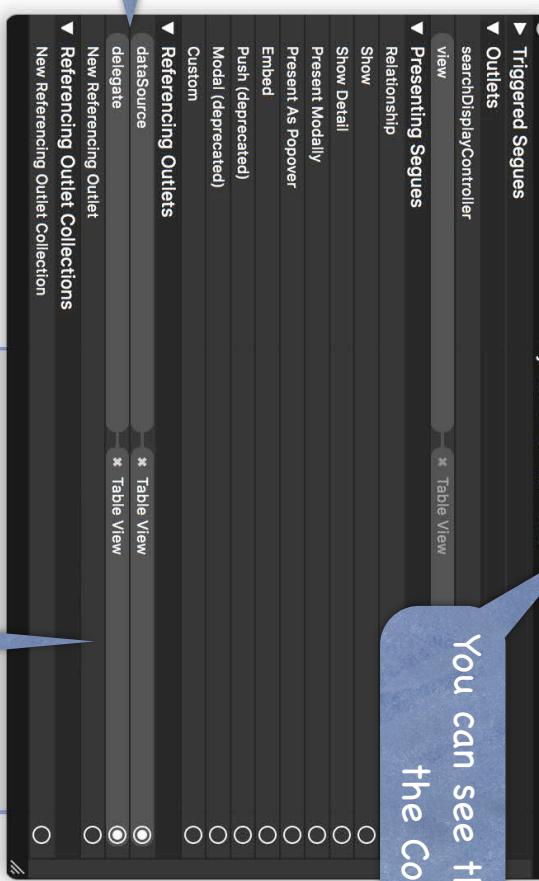
Tab Bar Controller - A controller for tab bar navigation.

CS193P Winter 2017

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.



### Document

User Defined Runtime Attributes
Key Path Type Value

### Identity

Class	Module
MyUITableViewContro...	Current - TVCEExam...



### Table View Controller

controller that manages a table view through a hierarchy of views.

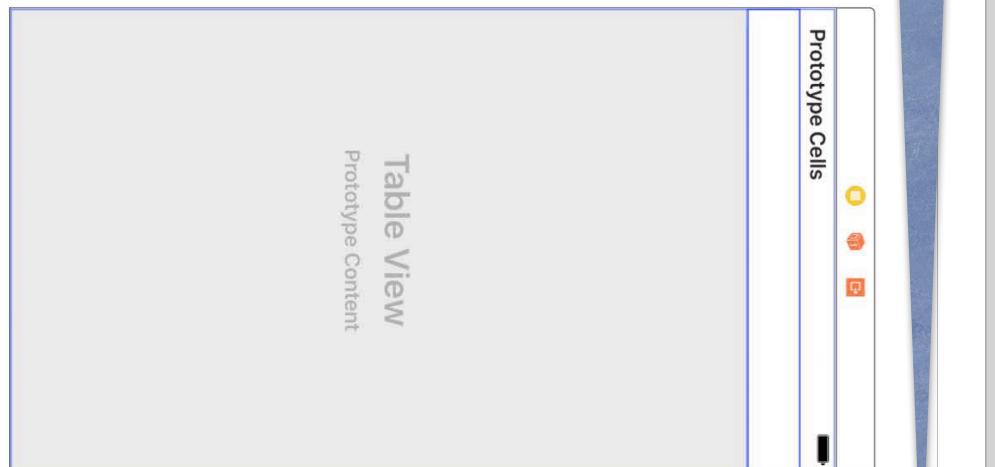


### Collection View Controller

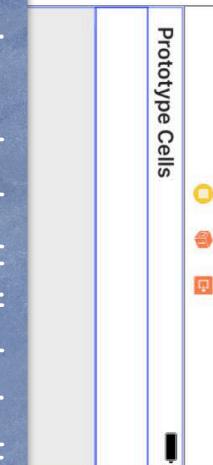
A controller that manages a collection view.

You can edit attributes of the UITableView by inspecting it.

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 ...



## Table View

Prototype Content

Style

Default

controller that manages navigation through a hierarchy of views.



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

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



CS193P  
Winter 2017

View as: iphone 7 (wC hR)

— 75% +

Table View

Prototype Content

Section Index

Display Limit 0

+ Text Default

+ Background Default

+ Tracking Default

Editing No Selection During Ed...

Scroll View

Style Default

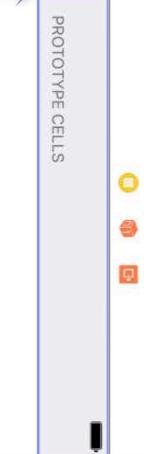
Control Indicators  Show Horizontal Indicators

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

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

Tab Bar Controller - A controller that manages a tab bar view.

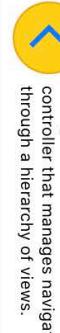
Grouped



## Table View

Prototype Content

Content	Dynamic Prototypes
Prototype Cells	1
Style	Grouped
Separator	Default
+ Separator Inset	Default
Selection	Single Selection
Editing	No Selection During Ed...
Section Index	
Display Limit	0
+ Text	Default
+ Background	Default
+ Tracking	Default
Scroll View	
Style	Default
+	Horizontal Indicators



controller that manages navigation through a hierarchy of views.



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



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



**Tab Bar Controller** - A controller that manages a tab bar view.

CS193P  
Winter 2017

— 75% +

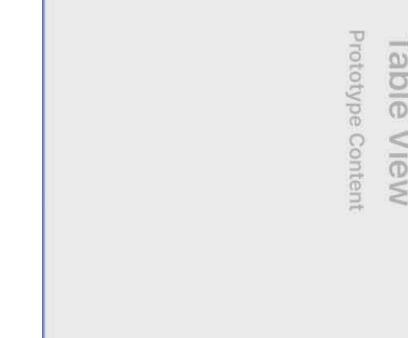


Filter

View as: iphone 7 (wC hR)

Grouped

Another important attribute is  
Dynamic versus Static ...



— 75% +



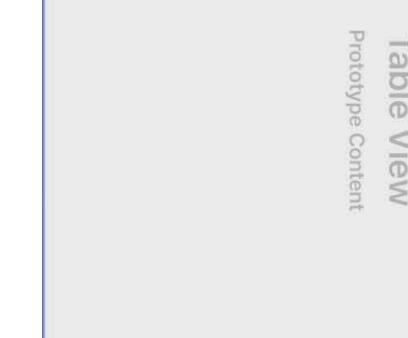
Filter

CS193P  
Winter 2017

View as: iphone 7 (wC hR)

Grouped

Another important attribute is  
Dynamic versus Static ...



Grouped

"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 to the Controller).

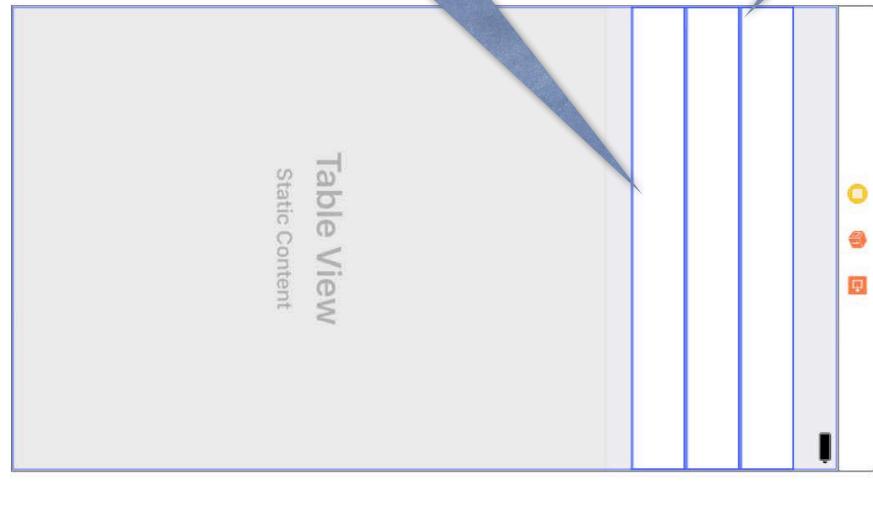


Table View

Static Content

Section Index

Display Limit 0

+ Text Default

+ Background Default

+ Tracking Default

Scroll View

Style Default

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

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

Tab Bar Controller - A controller for tab bar navigation.

CS193P Winter 2017

View as: iPhone 7 (wC hR)

75%

View

Content Mode	Center
Semantic	Unspecified
Tag	0

Interaction

<input checked="" type="checkbox"/> User Interaction Enabled
<input checked="" type="checkbox"/> Multiple Touch

Alpha

1
---

Background

Default
---------

Drawing

<input type="checkbox"/> Opaque
<input type="checkbox"/> Hidden
<input checked="" type="checkbox"/> Clears Graphics Context
<input checked="" type="checkbox"/> Clip To Bounds
<input checked="" type="checkbox"/> Autoresizes Subviews

Label

Label - A variably sized amount of static text.

Button

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

Segmented Control

Segmented Control - Displays multiple segments, each of which

View

Stretching

0	0
---	---

Width

1	1
---	---

Height

Static Content

Table View

— 75% +

TVCEExample > TVCEExample

TVCEExample: Ready

Label

Text: Plain

Color: Default

Font: System 17.0

Alignment: Center

Lines: 1

Behavior:  Enabled

Highlighted

Baseline: Align Baselines

Line Break: Truncate Tail

Autoshrink: Fixed Font Size

Highlighted: Default

Shadow: Default

Shadow Offset: 0 -1

Width: Height:

Table View

Static Content

View

Button

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

Segmented Control - Displays multiple segments, each of which

Label Label - A variably sized amount of static text.

Button available in Interface Builder.

Filter Winter 2017

View as: iphone 7 (wC hR)

Table View

Static Content

View

Label

Text: Plain

Color: Default

Font: System 17.0

Alignment: Center

Lines: 1

Behavior:  Enabled

Baseline: Align Baselines

Line Break: Truncate Tail

Autoshrink: Fixed Font Size

+ Highlighted: Default

+ Shadow: Default

Shadow Offset: 0 -1

Width: Height:

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

**Label** - A variably sized amount of static text.

**Segmented Control** - displays multiple segments, each of which



## View

Content Mode Center

Semantic Unspecified

Tag 0

Interaction  User Interaction Enabled

Multiple Touch

## Alpha

1

## Background

+

## Tint

+

## Drawing

 Opaque Hidden Clears Graphics Context Clip To Bounds Autoresizes Subviews

## Stretching

X

0

Y

0

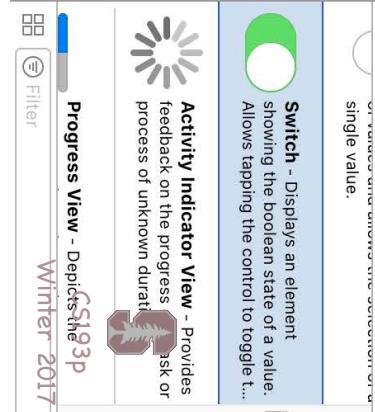
## Width

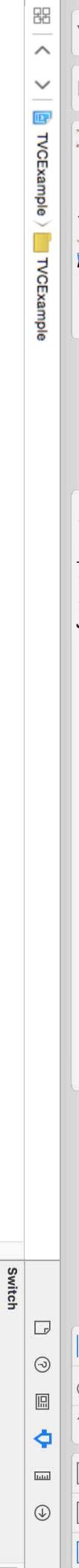
1

Height

## Table View

Static Content





**Switch**

Value	On
+ On Tint	Default
+ Thumb Tint	Default
+ On Image	On Image
+ Off Image	Off Image

**Control**

Alignment	Horizontal
+ Vertical	Vertical

State

<input type="checkbox"/> Selected
<input checked="" type="checkbox"/> Enabled
<input type="checkbox"/> Highlighted

**View**

Content Mode	Scale To Fill
Semantic	Unspecified
Tag	0

Single value.

**Switch** - Displays an element showing the boolean state of a value. Allows tapping the control to toggle t...

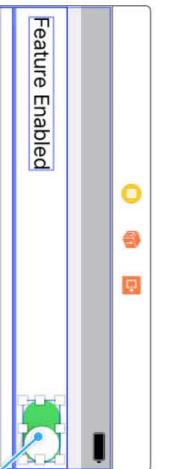
**Activity Indicator View** - Provides feedback on the progress task or process of unknown duration.

**Progress View** - Depicts the progress of a task or process.

&lt; &gt; TVCEExample &gt; TVCEExample

&lt; &gt; Automatic &gt; MyTableViewController.swift &gt; MyTableViewController

≡ ⌂ ↻ ↺ ↻ ↻ ↻



```
1 //////////////////////////////////////////////////////////////////  
2 //////////////////////////////////////////////////////////////////  
3 //////////////////////////////////////////////////////////////////  
4 //////////////////////////////////////////////////////////////////  
5 //////////////////////////////////////////////////////////////////  
6 //////////////////////////////////////////////////////////////////  
7 //////////////////////////////////////////////////////////////////  
8  
9 import UIKit  
10  
11 class MyTableViewController: UITableViewController  
12 {  
13     override func viewDidLoad()  
14     {  
15         super.viewDidLoad()  
16     }  
17  
18 }
```

Insert Outlet, Action, or Outlet Collection

## Table View

Static Content

View as: iphone 7 (wC hR)

TVCEExample > TVCEExample > MyTableViewCellController.swift > MyTableViewCellController

```
1 //////////////////////////////////////////////////////////////////  
2 //////////////////////////////////////////////////////////////////  
3 //////////////////////////////////////////////////////////////////  
4 //////////////////////////////////////////////////////////////////  
5 //////////////////////////////////////////////////////////////////  
6 //////////////////////////////////////////////////////////////////  
7 //////////////////////////////////////////////////////////////////  
8  
import UIKit  
9  
class MyTableViewCellController: UITableViewcontroller  
{  
    @IBOutlet weak var featureEnabledSwitch: UISwitch!  
}
```



Connection  
Outlet  
Object  
Name  
Type  
Storage

featureEnabledSwitch  
UISwitch  
Weak

Cancel Connect

## Table View

Static Content



View as: iphone 7 (wC hR)





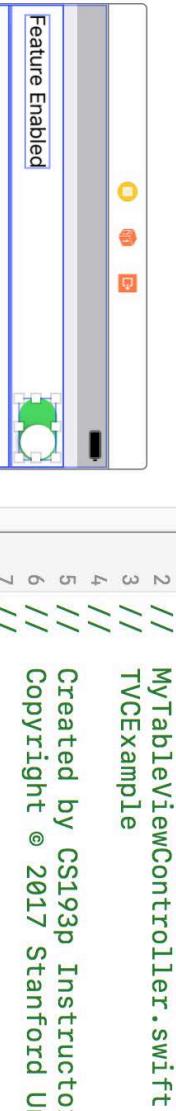
TVCEExample &gt; iPhone 7

TVCEExample: Ready



TVCEExample < TVCEExample > TVCEExample > < > Automatic > MyTableViewController.swift > MyTableViewController

+ X



MyTableViewController.swift  
TVCEExample  
Created by CS193p Instructor.  
Copyright © 2017 Stanford University. All rights reserved.

```
1 //////////////////////////////////////////////////////////////////  
2 //////////////////////////////////////////////////////////////////  
3 //////////////////////////////////////////////////////////////////  
4 //////////////////////////////////////////////////////////////////  
5 //////////////////////////////////////////////////////////////////  
6 //////////////////////////////////////////////////////////////////  
7 //////////////////////////////////////////////////////////////////  
8  
9 import UIKit  
10  
11 class MyTableViewController: UITableViewController  
12 {  
13     @IBOutlet weak var featureEnabledSwitch: UISwitch!  
14  
15 }  
16  
17  
18
```

## Table View

Static Content

Let's clear this UI out  
and look at another way  
to do table views.



View

Content Mode	Scale To Fill
Semantic	Unspecified
Tag	0
Interaction	<input checked="" type="checkbox"/> User Interaction Enabled
Alpha	1
+ Background	<input type="color"/>
+ Tint	<input type="color"/> Default
Drawing	<input type="checkbox"/> Opaque
	<input type="checkbox"/> Hidden
	<input checked="" type="checkbox"/> Clears Graphics Ext
	<input type="checkbox"/> Clip To Bounds
	<input checked="" type="checkbox"/> Autoresize Subviews

A different way to populate the UI of your table view is by dynamically providing the data at runtime.

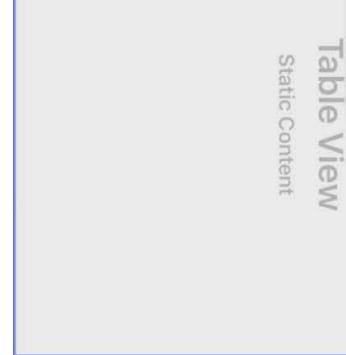


Table View

Static Content

Scroll View

Style Default

- Display Limit 0
- + Text Default
- + Background Default
- + Tracking Default

Section Index

Sections 1

- Style Grouped
- + Separator Default
- + Separator Inset Default
- Selection Single Selection
- Editing No Selection During Ed...

A different way to populate the UI of your table view is by dynamically providing the data at runtime.



#### Scroll View

Style Default

Display Limit 0

Text Default

+ Background Default

+ Tracking Default

Scrolling  Shows Horizontal Indicator...

Shows Vertical Indicator

Scrolling Enabled

Paging Enabled

Direction Lock Enabled

Bounce

Bounces

Bounce Horizontally

Bounce Vertically

Zoom 1

Min 1

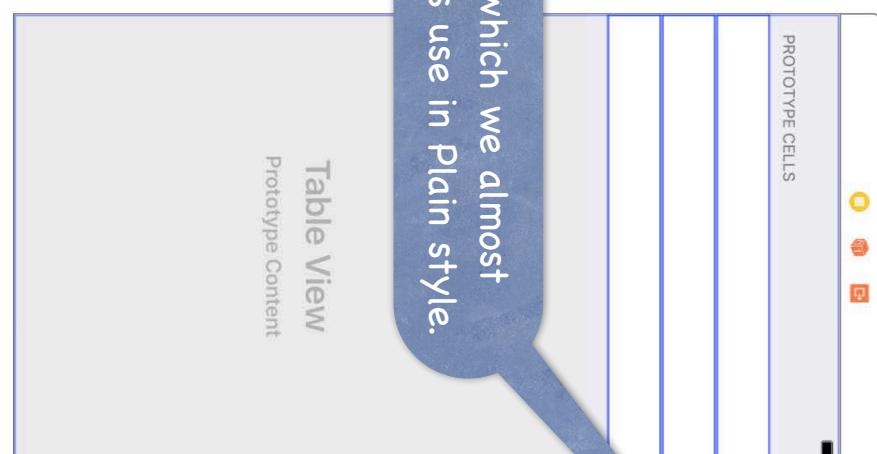
Max 1

Touch  Bounces Zoom

Delays Content Touches

Winter 2017

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



### ScrollView

Style Default

Display Limit 0

Text Default

Background Default

Tracking Default

Shows Horizontal Indicator

Shows Vertical Indicator

Scrolling Enabled

Paging Enabled

Direction Lock Enabled

Bounce Enabled

Bounces Enabled

Bounce Horizontally

Bounce Vertically

Zoom Min 1

Bounces Enabled

Touch Enabled

Bounces Zoom CS193P

Delays ContentTouches

Winter 2017

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

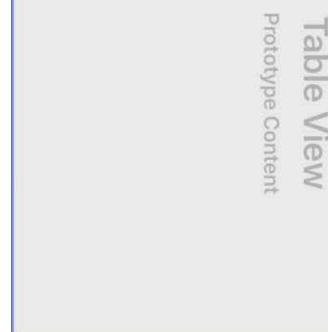
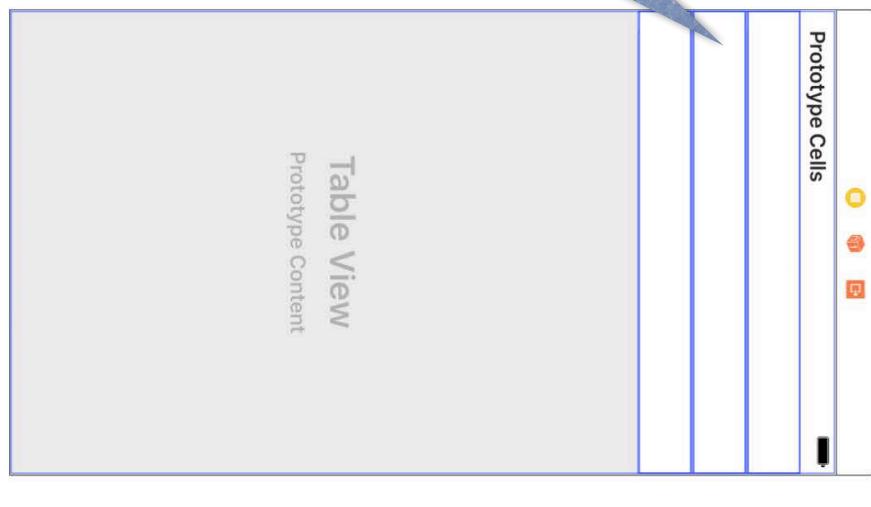


Table View	
Prototype Content	
Content	Dynamic Prototypes
Prototype Cell	Plain
Style	Plain
Separator	Default
Separator Inset	Default
Selection	Single Selection
Editing	No Selection During Ed...
Section Index	
Display Limit	0
+ Text	Default
+ Background	Default
+ Tracking	Default
Scroll View	
Style	Default
Scroll Indicat...	<input checked="" type="checkbox"/> Shows Horizontal Indicat...
Background	Default
Tracking	Default
Scanning	<input checked="" type="checkbox"/> Scrolling Enabled
Paging Enabled	<input type="checkbox"/>
Direction Lock Enabled	<input type="checkbox"/>
Bounce	<input checked="" type="checkbox"/> Bounces
Bounce Horizontally	<input type="checkbox"/>
Bounce Vertically	<input checked="" type="checkbox"/> Bounce Vertically
Zoom	1
Min	1
Touch	<input checked="" type="checkbox"/> Bounces Zoom
Zoom	CS193P
Content	Delayed Content Touches
Touches	Winter 2017

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



Any cell can be clicked on ...

... and inspected in the  
Attributes Inspector ...

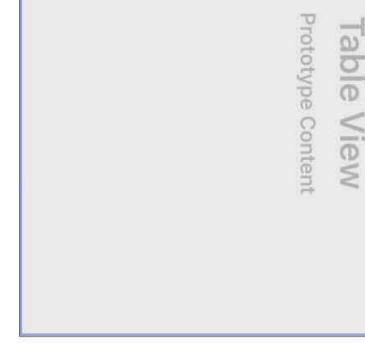


Table View Cell

Style: Custom

Identifier: Reuse Identifier

Selection: Default

Accessory: None

Editing Acc.: None

Focus Style: Default

Indentation: 0 Level: 10 Width:

Indent While Editing

Shows Re-order Controls

Separator: Default Insets

View

Content Mode: Scale To Fill

Semantic: Unspecified

Tag: 0

Interaction:  User Interaction Enabled

Alpha: 1

+ Background:  Default

+ Tint:

Drawing:  Opaque

Hidden

Clears Graphics Context

Clip To Bounds

Autoresizes Subviews

Stretching: 0 X 0 CS193p

Here's where you can set the style of the cell.

The screenshot shows the Xcode storyboard editor with a "Table View Cell" configuration panel open. The panel has tabs for "Style" (selected), "Identifier", "Basic", "Right Detail", and "Left Detail". Under "Style", there are sections for "Prototype Cells" (with three icons: yellow, red, and blue), "Accessory" (with options like "None", "Disclosure", "Detail Disclosure", and "Subtitle" which is selected), and "Focus Style" (with "Default" selected). There are also "Indentation" (Level 0, Width 10) and "Separator" (Shows Re-order Controls off, Default Insets selected) settings. A callout bubble points from the text "Here's where you can set the style of the cell." to the "Accessory" section of the panel. The storyboard itself shows a "Table View" with a single "Prototype Content" cell.

## Subtitle cell style

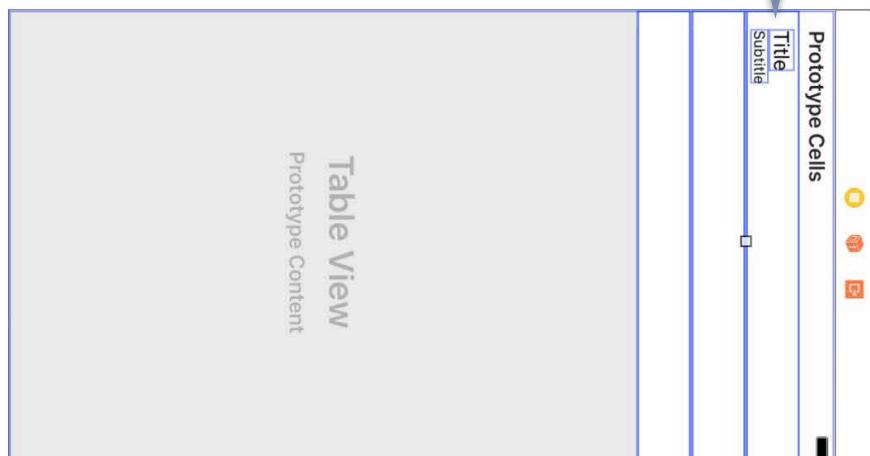


Table View Cell

Style	Subtitle
Image	Image
Identifier	Reuse Identifier
Selection	Default
Accessory	None
Editing Acc.	None
Focus Style	Default
Indentation	0
Level	10
Width	10
<input type="checkbox"/> Shows Re-order Controls	
Separator	Default Insets

Table View

Prototype Content

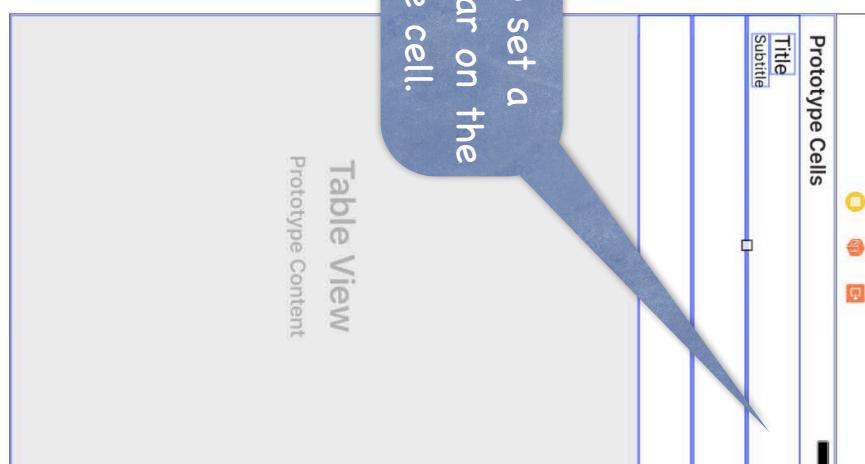
View

Content Mode	Scale To Fill
Semantic	Unspecified
Tag	0
Interaction	<input checked="" type="checkbox"/> User Interaction Enabled
Alpha	1
Background	+
Tint	+
Drawing	<input checked="" type="checkbox"/> Opaque
Hidden	<input type="checkbox"/>
Clears Graphics	<input checked="" type="checkbox"/>
Clip To Bounds	<input checked="" type="checkbox"/>
Autoresizing Subviews	<input checked="" type="checkbox"/>
Stretching	0

CS193P Winter 2017

View as: iphone 7 (wC hR)

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



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

This one's sort of special ...

The screenshot shows the 'TVCEExample' Xcode project with the 'iPhone 7' storyboard selected. A callout bubble points from the text 'This one's sort of special ...' to the 'Detail Disclosure' button in the 'Table View Cell' settings. The 'Table View Cell' settings panel is open, showing various options like 'Style' (set to 'Subtitle'), 'Image' (set to 'Image'), 'Identifier' (set to 'Reuse Identifier'), 'Selection' (set to 'Default'), 'Accessory' (set to 'None'), and 'Editing Accessory' (set to 'Disclosure Indicator'). The 'Detail Disclosure' button is highlighted. Below the cell settings, the 'Table View' settings panel is visible, showing 'Content Mode' (set to 'Scale To Fill'), 'Semantic' (set to 'Unspecified'), 'Tag' (set to 0), 'Interaction' (checkbox checked for 'User Interaction Enabled' and 'Multiple Touch'), and 'Drawing' (checkbox checked for 'Opaque'). At the bottom, there are sections for 'Background' (set to 'Alpha') and 'Tint' (set to 'Default'). The status bar at the top shows the Xcode interface with file navigation and search icons.

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

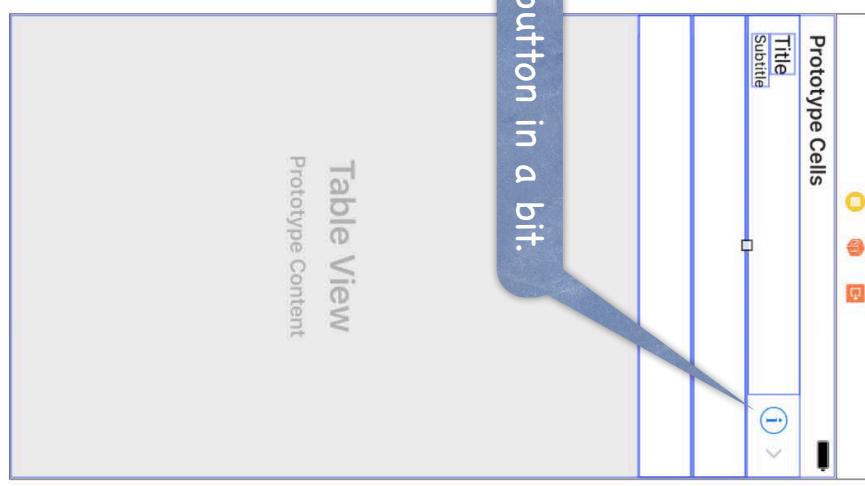


Table View Cell

Style	Subtitle
Image	Image
Identifier	Reuse Identifier
Selection	<input checked="" type="checkbox"/> None
Accesso	<input checked="" type="checkbox"/> Disclosure Indicator
Editing Ac	<input checked="" type="checkbox"/> Checkmark
Focus Style	<input checked="" type="checkbox"/> Detail

Prototype Cells

Title	<input checked="" type="checkbox"/>
Subtitle	<input checked="" type="checkbox"/>

Table View

Prototype Content

View

Content Mode	Scale To Fill
Semantic	Unspecified
Tag	0
Interaction	<input checked="" type="checkbox"/> User Interaction Enabled
Alpha	1
Background	
Tint	
Drawing	<input checked="" type="checkbox"/> Opaque
	<input type="checkbox"/> Hidden
	<input checked="" type="checkbox"/> Clears Graphics Ext
	<input checked="" type="checkbox"/> Clip To Bounds
	<input checked="" type="checkbox"/> Autoresize Subviews

Stretching 0

CS193P Winter 2017

— 75% +

View as: iphone 7 (wC hR)

TVCEExample > TVCEExample

Table View C

Custom

Basic Right Detail Left Detail Subtitle

Image

Identifier Reuse Identifier

Selection Default

Accessory None

Editing Acc. None

Focus Style Default

Indentation 0 Level 10 Width

Indent While Editing

Shows Re-order Controls

Separator Default Insets

Table View

Prototype Content

One of the cell styles you can choose is Custom.

View

Content Mode Scale To Fill

Semantic Unspecified

Tag 0

Interaction  User Interaction Enabled

Multiple Touch

Alpha 1

Background

Tint Default

Drawing Opaque

Hidden

Clears Graphics

Clip To Bounds

Autoresize Subviews

Stretching 0

CS193P Winter 2017

View as: iphone 7 (wC hR)

TVCEExample > TVCEExample

Table View Cell

Style: Custom

Identifier: Reuse Identifier

Selection: Default

Accessory: None

Editing Acc.: None

Focus Style: Default

Indentation: Level 0 Width 10

Show Re-order Controls:

Separator: Default Insets

**View**

Content Mode: Scale To Fill

Semantic: Unspecified

Tag: 0

Interaction:  User Interaction Enabled

Alpha: 1

+ Background:  Default

+ Tint:  Default

Drawing:  Opaque

Hidden:

Clears Graphics Context:

Clip To Bounds:

Autoresize Subviews:

Stretching: X 0 Y 0 CS193p

Table View

Prototype Content

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

The UI inside these cells is going to get replicated for each row  
(because this is a Dynamic table and thus these are "prototype" cells).

— 75% +

View as: iPhone 7 (wC hR)

You can change their size.

Table View

Prototype Content

Table View Cell

Style: Custom

Identifier: Reuse Identifier

W: 315.0 H: 125.0

Accessory: None

Editing Acc.: None

Focus Style: Default

Indentation: 0 Level: 10 Width: 10

Shows Re-order Controls:

Separator: Default Insets

View

Content Mode: Scale To Fill

Semantic: Unspecified

Tag: 0

Interaction:  User Interaction Enabled

Multiple Touch:

Alpha: 1

Background:  Default

Tint:  Default

Drawing:  Opaque

Hidden:

Clears Graphics Context:

Clip To Bounds:

Autoresize Subviews:

Stretching: 0 X: 0 Y: 0 CS193p

— 75% +

□ View as: iPhone 7 (wC hR)

**Table View Cell**

Style	Custom
Identifier	Reuse Identifier
Selection	Default
Accessory	None
Editing Acc.	None
Focus Style	Default
Indentation	0
Level	10
Width	
<input checked="" type="checkbox"/> Shows Re-order Controls	
Separator	Default Insets

**Prototype Cells**

Photo	
Title	
Description	

**Table View**

Prototype Content

**Interaction**  User Interaction Enabled

Multiple Touch

 available in Interface Builder.

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

**Label** Label - A variably sized amount of static text.

**Segmented Control** - Displays a segmented control with three segments labeled "Winter", "Spring", and "Summer".

View as: iPhone 7 (wC hR)

— 75% +

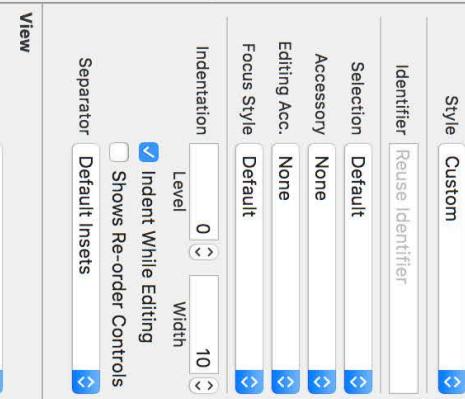
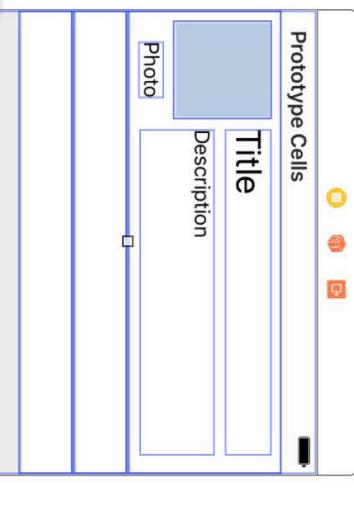
You can't wire up outlets from your UITableViewContoller to these UI elements though.

Why not? Because there could be 100's or 1000's of these rows, each with a copy of this UI!

So, instead, we wire the outlets up to the UITableView that is containing these UI elements.

The kind of UIView that is containing these elements is a UITableViewCell.

Just like with a Controller, we have to subclass UITableView in order to have outlets in it.



So we must select the cell ...

And use the Identity Inspector  
to set the subclass.

The screenshot shows the Xcode interface with a storyboard file open. A UITableView is selected, and its prototype cell is highlighted with a blue border. A callout bubble points from the text "So we must select the cell ..." to the prototype cell. Another callout bubble points from the text "And use the Identity Inspector to set the subclass." to the Identity tab of the Identity Inspector. The Identity Inspector shows the custom class `UITableViewCell` selected under the "Custom Class" section. The "Identity" section contains fields for Restoration ID, User Defined Runtime Attributes, Key Path, and Value. The "Accessibility" section includes fields for Enabled, Label, Hint, Identifier, Traits, and Static Text. The bottom of the screen shows the Xcode toolbar and a portion of the navigation bar.

**Xcode**

**File Edit View Find Navigate Editor Product Debug Source Control Window Help**

**New**

Add Files to "TVCEExample"...

Open... Open Recent Open Quickly...

Close Window Close Tab Close "Main.storyboard"

Close Project Save Duplicate... Revert to Saved... Unlock... Export...

Show in Finder Open with External Editor

Save As Workspace... Project Settings...

Page Setup... Print...

Tab Window

File... Playground... Target... Project... Workspace...

Group Group from Selection

Custom Class

Class UITableViewTableViewCell

Module None

Identity

Restoration ID

Prototype Cells

Title Description Photo

Document

Label Xcode Specific Label

Object ID siz-fu-BC6

Lock Inherited - (Nothing)

Notes

No Font

Comment For Localizer

Accessibility

Accessibility Enabled

Label

Hint Hint

Identifier Identifier

Traits

Button

Image

Static Text

CS193p Winter 2017

View as: iPhone 7 (wC hR)

— 75% +

Choose options for your new file:

Class: UITableViewController

Subclass of:

UITableViewController  
UITableViewController  
UITableViewController  
UITableViewController  
UITableViewController  
UITableViewController

Usage:  
Choose UITableViewController as  
the class to subclass off of.

Cancel

Previous

Next

Object ID siz-fu-BC6

Lock Inherited - (Nothing)

Notes

No Font

Comment For Localizer

#### Accessibility

Accessibility  Enabled

Label

Hint

Identifier

Traits

Button

Image

Static Text

CS193p Winter 2017

Choose options for your new file:

## Custom Class

Class	UITableViewCell
Module	None

## Identity

Restoration ID

Class:  
MyTableViewCellCellSubclass of:  
UITableViewCell Also create XIB fileLanguage:  
Swift

Cancel

Previous

Next

## Accessibility

Accessibility  Enabled

Label

Hint

Identifier

Traits

Button

Image

Static Text

CS193p

Winter 2017

View as: iPhone 7 (wC hR)

— 75% +



Custom Class	<input type="button" value=""/>	<input type="button" value="?"/>	<input type="button" value=""/>				
Class	<input type="text" value="UITableViewCell"/>						
Module	<input type="text" value="MyTableViewCell"/>						

Prototype Cells	
<input type="button" value=""/>	<input type="button" value=""/>
<input type="button" value=""/>	<input type="button" value=""/>
<input type="button" value=""/>	<input type="button" value=""/>
<input type="button" value=""/>	<input type="button" value=""/>

### Title

Description

Photo

Restoration ID

Key Path

Type

Value

Inherited - (Nothing)

Lock

Notes

No Font

Comment For Localizer

Then set it in the Inspector as usual.

-fu-BC6

Accessibility	<input type="checkbox"/> Enabled
Label	<input type="text" value="Label"/>
Hint	<input type="text" value="Hint"/>
Identifier	<input type="text" value="Identifier"/>

Traits	<input type="checkbox"/> Button
Image	<input type="checkbox"/> Image
Static Text	<input type="checkbox"/> Static Text

Table View  
Prototype Content

Accessibility

Enabled

Label

Hint

Identifier

Traits  Button

Image

Static Text  CS193p

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

Static cell UI elements outlets are wired to the UITableViewContoller.

Dynamic cell UI elements outlets are wired to the UITableViewCell containing them.

The storyboard shows a prototype cell with three sections: Photo, Title, and Description. The Photo section contains a placeholder image. The Title section contains a label with the text 'Title'. The Description section contains a label with the text 'Description'.

Custom Class: MyTableViewCell

Identity: Restoration ID: (empty)

User Defined Runtime Attributes:

- Key Path: (empty)
- Type: (empty)
- Value: (empty)

Document:

- Label: Xcode Specific Label
- Object ID: sz-fu-BC6
- Lock: Inherited - (Nothing)
- Notes: (empty)
- Right: (empty)
- Want For Localizer: (empty)

Accessibility: Enabled

Label: (empty)

Hint: Hint

Identifier: Identifier

Traits: (empty)

Button: (empty)

Image: (empty)

Selected: (empty)

Static Text: CS193p

Winter 2017

TVCEExample < > TVCEExample > TVCEExample  
TVCEExample: Ready  
< > Automatic > MyTableViewCellController.swift > MyTableViewCellController



```
1 // MyTableViewCellController.swift
2 // TVCEExample
3 // Created by CS193p Instructor.
4 // Copyright © 2017 Stanford University. All rights reserved.
5 // 
6 // 
7 // 
8 // 
9 import UIKit
10 
11 class MyTableViewCellController: UITableViewController
12 {
```

Use the Assistant Editor to get the  
UITableViewCell code on screen at the  
same time as your UI.

## Table View

Prototype Content

TVCEExample < TVCEExample > TVCEExample

MyTableViewController.swift  
TVCEExample

≡ ⌂ ↻ □ □ □

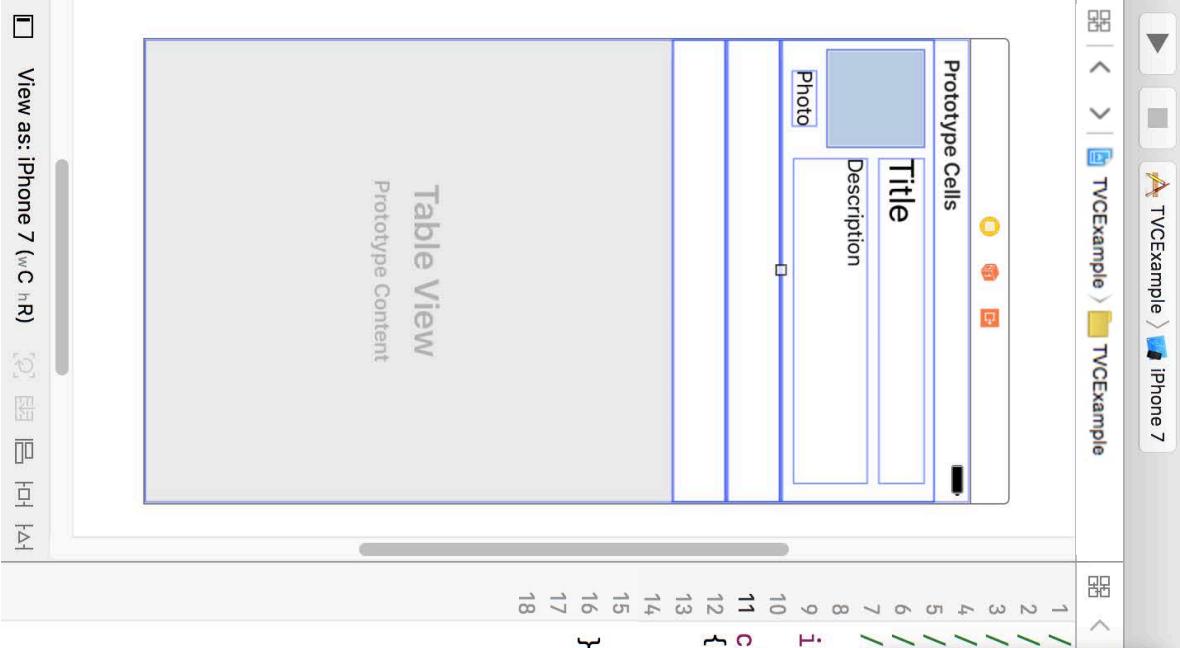


```
1 // MyTableViewController.swift
2 // TVCEExample
3 // Created by CS193p Instructor.
4 // Copyright © 2017 Stanford University. All rights reserved.
5 //
6 //
7 //
8 import UIKit
9
10 class MyTableViewController: UITableViewController
11 {
12
13
14
15
16
17
18 }
```

But when you do, if you're in Automatic mode, it will show you the UITableViewcontroller instead.

## Table View

Prototype Content





... switch to Manual ...

```
1 import UIKit
2
3 class MyTableViewController: UITableViewController {
4
5     // ...
6
7     // ...
8
9
10
11
12
13
14
15
16
17
18 }
```

The screenshot shows the Xcode interface with the following details:

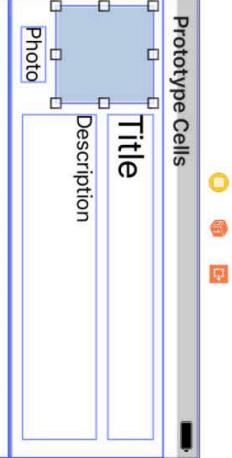
- Top Bar:** Shows the project name "TVCEExample" and target "iPhone 7".
- File Navigator:** Shows files like AppDelegate.swift, ViewController.swift, Main.storyboard, MyTableViewCell.swift, Assets.xcassets, LaunchScreen.storyboard, Info.plist, and TVCEExample.xcdatamodeld.
- Outline View:** Shows "Prototype Cells" and "Title" and "Description" cells.
- Table View Controller:** Shows "Photo" and "Title" cells.
- Code Editor:** Displays the following Swift code:

```
import UIKit

class MyTableViewCell: UITableViewCell {
}
```
- Callout Bubble:** A blue callout bubble points from the "Title" cell in the storyboard to the "Title" variable in the code editor, containing the text "... and choose your UITableViewCell subclass instead."

TVCEExample < > TVCEExample > TVCEExample < > Manual > TVCEExample > TVCEExample > MyTableViewCell.swift > MyTableViewCell

```
1 // MyTableViewCell.swift  
2 // TVCEExample  
3 // Created by CS193p Instructor.  
4 // Copyright © 2017 Stanford University. All rights reserved.  
5 //  
6 //  
7 //  
8 //  
9 import UIKit  
10  
11 class MyTableViewCell: UITableViewCell {  
12  
13  
14  
15  
16  
17  
18}
```



Now you're ready to ctrl-drag!

## Table View

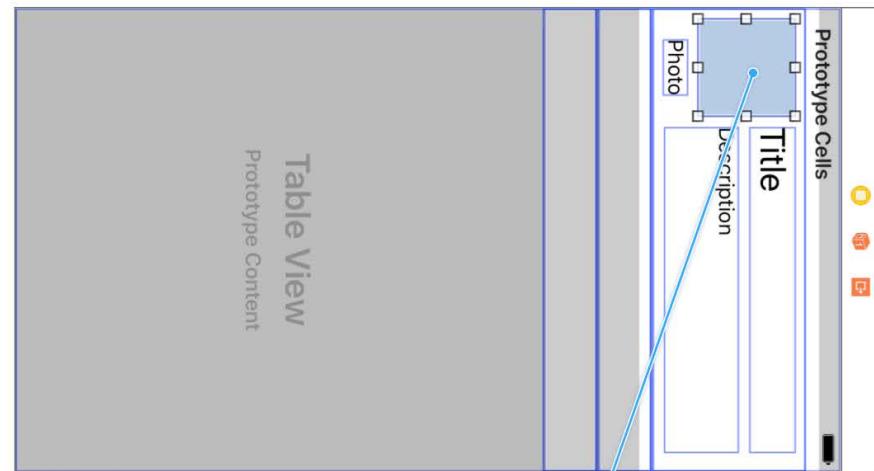
Prototype Content

View as: iphone 7 (wC hR) □

TVCEExample | < > TVCEExample > TVCEExample

1 // MyTableViewCell.swift  
2 // TVCEExample  
3 // Created by CS193p Instructor.  
4 // Copyright © 2017 Stanford University. All rights reserved.  
5 //  
6 //  
7 //  
8 //  
9 import UIKit  
10  
11 class MyTableViewCell: UITableViewCell  
12 {  
13 }  
14  
15  
16  
17  
18

Insert Outlet or Outlet Collection



## Table View

Prototype Content

< > TVCExample > TVCExample

MyTableViewCell.swift  
TVCExample  
Created by CS193p Instructor.  
Copyright © 2017 Stanford University. All rights reserved.



Connection      Outlet  
Object      My TableView Cell  
Name      photoImageView  
Type      UIImageView  
Storage      Weak

Connect

Cancel

```
import UIKit

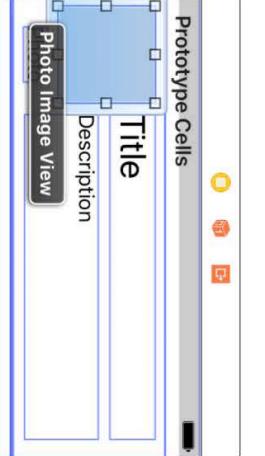
class MyTableViewCell: UITableViewCell {
    @IBOutlet weak var photoImageView: UIImageView!
}
```

## Table View

Prototype Content

TVCEExample < > TVCEExample > TVCEExample

MyTableViewCell.swift  
TVCEExample  
Created by CS193p Instructor.  
Copyright © 2017 Stanford University. All rights reserved.



```
1 //////////////////////////////////////////////////////////////////  
2 //////////////////////////////////////////////////////////////////  
3 //////////////////////////////////////////////////////////////////  
4 //////////////////////////////////////////////////////////////////  
5 //////////////////////////////////////////////////////////////////  
6 //////////////////////////////////////////////////////////////////  
7 //////////////////////////////////////////////////////////////////  
8 //////////////////////////////////////////////////////////////////  
9 import UIKit  
10  
11 class MyTableViewCell: UITableViewCell {  
12     @IBOutlet weak var photoImageView: UIImageView!  
13 }  
14  
15  
16  
17  
18
```

This outlet is not in the Controller!  
It's in your UITableViewCell subclass.  
Every row in the table will have its own  
photoImageView.

The screenshot shows the Xcode interface with two main panes. The top pane displays a storyboard titled 'TVCEExample' containing a table view with three prototype cells. The first cell is labeled 'Title', the second 'Description', and the third 'Photo Image View'. The bottom pane shows the 'MyTableViewCell.swift' file with the following code:

```
1 //////////////////////////////////////////////////////////////////
2 // MyTableViewCell.swift
3 // TVCEExample
4 //////////////////////////////////////////////////////////////////
5 // Created by CS193p Instructor.
6 // Copyright © 2017 Stanford University. All rights reserved.
7 //
8
9 import UIKit
10
11 class MyTableViewCell: UITableViewCell
12 {
13     @IBOutlet weak var photoImageView: UIImageView!
14
15     var infoShownByThisCell: Type { didSet { updateUI() } }
16
17
18 }
```

A callout bubble points from the text 'A UITableViewCell subclass has to have some public API that gives it the information it needs to load up its outlet views.' to the line 'var infoShownByThisCell: Type { didSet { updateUI() } }'.

**Table View**

Prototype Content

We'll see where you set this var in code in a moment.

TVCExample: Ready

TVCEExample > iPhone 7

View as: iphone 7 (wC hR)

CS193p Winter 2017

# 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, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
```

IndexPath 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, cellForRowAt indexPath: IndexPath) -> UITableViewCell
{
    let data = myInternalDataStructure[indexPath.section][indexPath.row]
    // myInternalDataStructure is conceptual here: it doesn't have to be an Array of Arrays
}
```



# 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, cellForRowAt indexPath: IndexPath) -> 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 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, cellForRowAt indexPath: IndexPath) -> UITableViewCell
{
    let data = myInternalDataStructure[indexPath.section][indexPath.row]

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

    return cell
}
```



Table View Cell

Style: Subtitle

Image: Image

Identifier: Reuse Identifier

Selection: Default

Accessory: None

Editing Acc.: None

Focus Style: Default

Indentation: Level 0, Width 10

Indent While Editing

Shows Re-order Controls

Separator: Default Insets

```
func tableView(_ tv: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let data = myInternalDataStructure[indexPath.section][indexPath.row]
    ...
}
```

To Fill

specified

0

Interaction Enabled

multiple Touch

1

Default

que

jen

Clears Graphics

Clip To Bounds

Autoresizing Subviews

Stretching 0

CS193P Winter 2017

Table View Cell

Style Subtitle  
Image Image

Identifier Reuse Identifier

Selection Default

Accessory None

Editing Acc. None

Focus Style Default

Indentation Level 0 Width 10  
 Indent While Editing  
 Shows Re-order Controls

Separator Default Insets

View

Content Mode Scale To Fill

Semantic Unspecified

Tag 0

Interaction  User Interaction Enabled  
 Multiple Touch

Alpha 1

Background

Tint Default

Drawing Opaque  
 Hidden  
 Clears Graphics  
 Clip To Bounds  
 AutoresizesSubviews

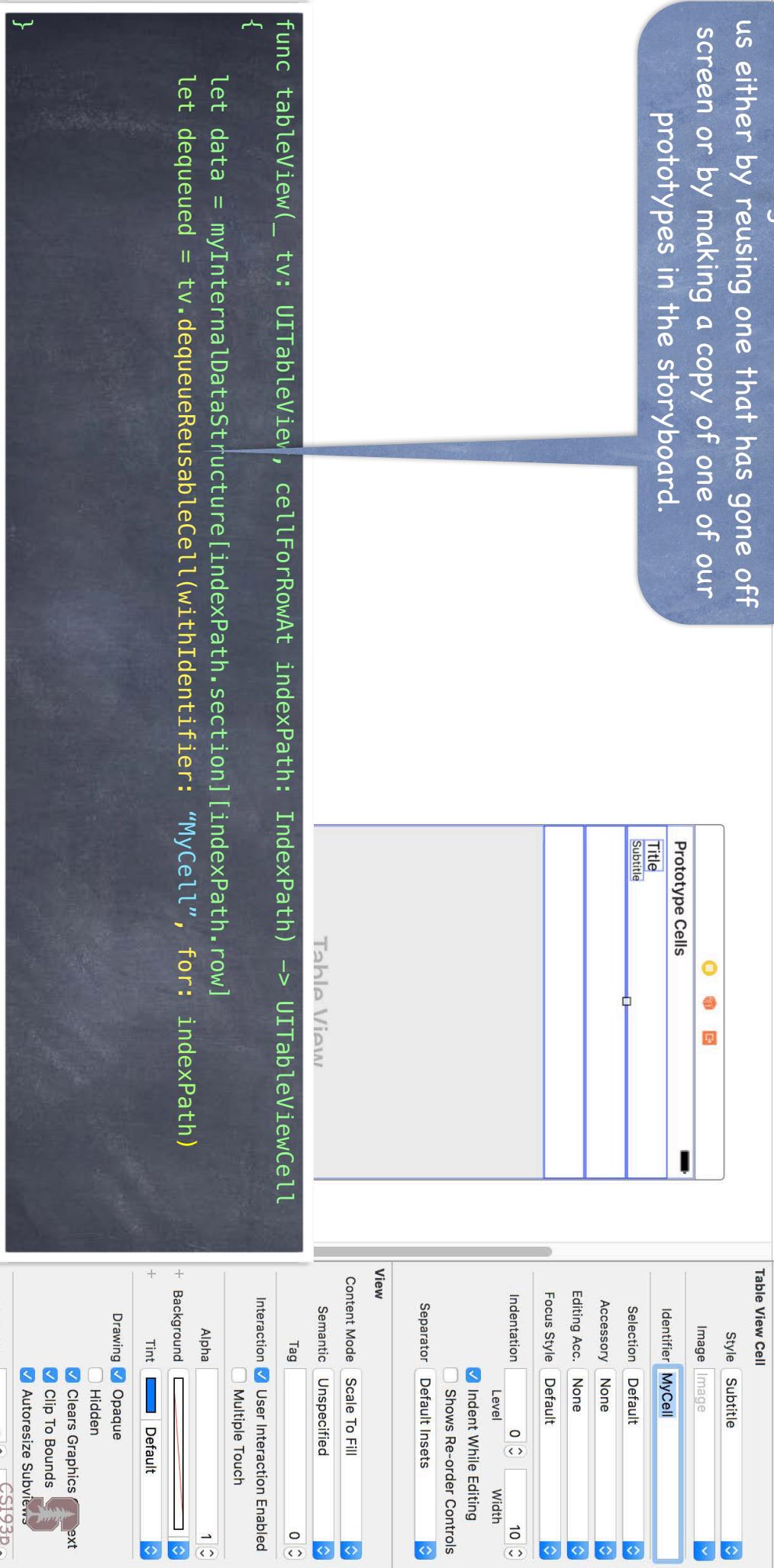
Stretching 0

CS193p Winter 2017

Table View

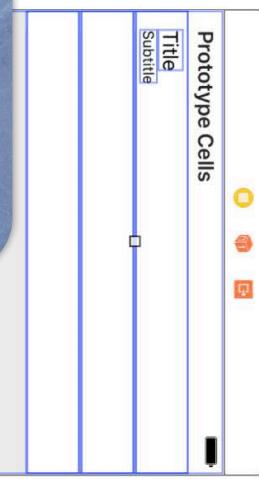
```
func tableView(_ tv: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let data = myInternalDataStructure[indexPath.section][indexPath.row]
    ...
}
```

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 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, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let data = myInternalDataStructure[indexPath.section][indexPath]
    let dequeued = tv.dequeueReusableCell(withIdentifier: "MyCell", for: indexPath)
```

For a non-Custom cell ...

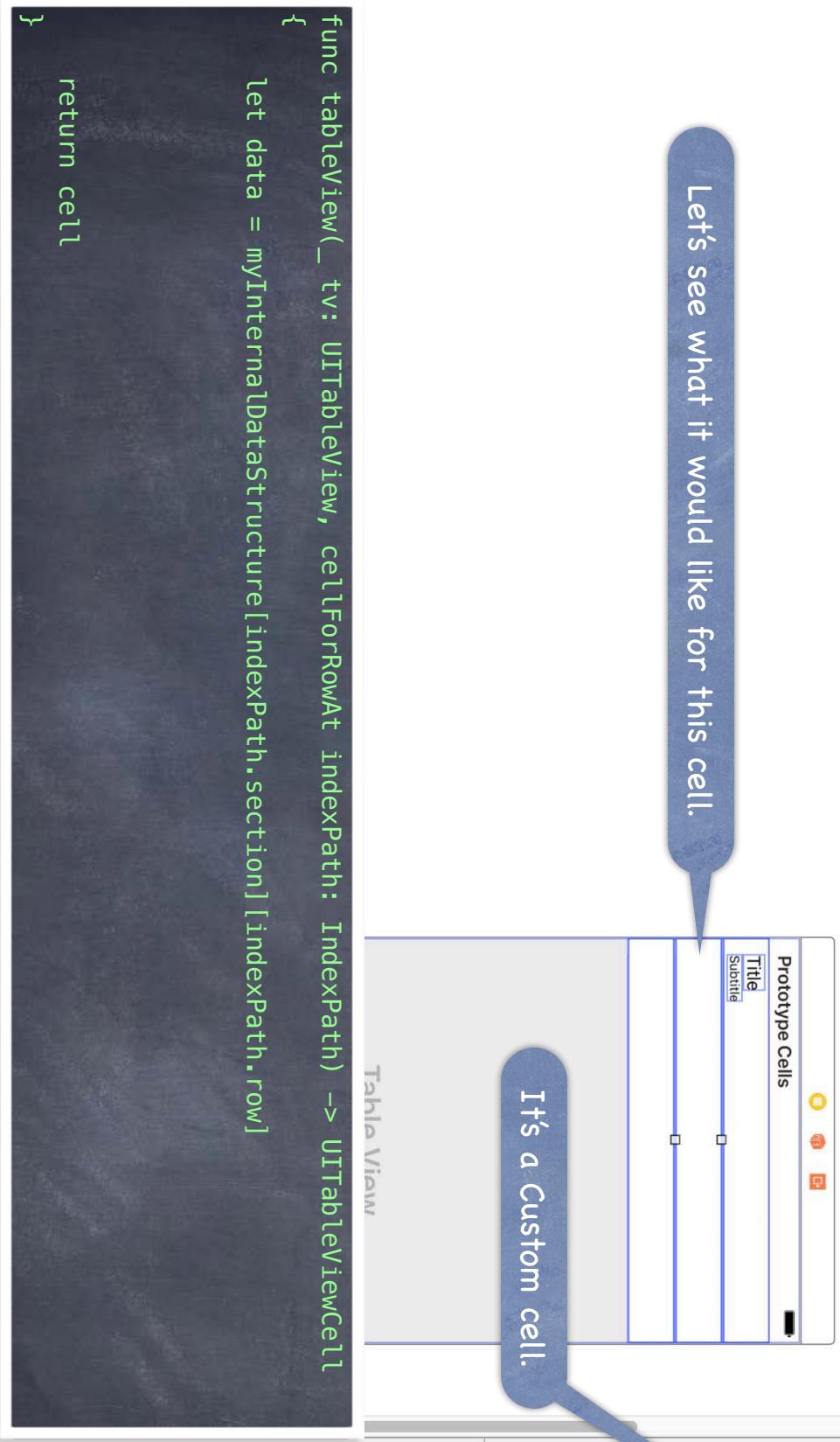
... the dequeued thing will be a generic UITableView/TableViewCell.  
You can look up its API to see what sort of configuration  
options are available for it.

```
func tableView(_ tv: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let data = myInternalDataStructure[indexPath.section][indexPath.row]
    let dequeued = tv.dequeueReusableCell(withIdentifier: "MyCell", for: indexPath)

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

Let's see what it would like for this cell.

It's a Custom cell.



Let's see what it would like for this cell.

The screenshot shows the Xcode interface for a UITableView configuration. A blue callout points to the 'Identifier' field, which contains the value 'MyCustomCell'. The 'Prototype Cells' section shows three rows: 'Title' (selected), 'Subtitle', and 'None'. The 'Table View Cell' settings panel is open, showing the following configuration:

Style	Custom
Identifier	MyCustomCell
Selection	Default
Accessory	None
Editing Accessory	None
Focus Style	Default
Indentation Level	0
Width	10
Shows Re-order Controls	<input type="checkbox"/>
Separator	Default Insets

The 'View' section includes:

- Content Mode: Multiple Touch
- Alpha: 1
- Background: Unspecified
- Tag: 0
- Interaction: User Interaction Enabled
- Drawing: Opaque
- Background Tint: Default

The 'Drawing' section includes:

- Hidden:
- Clears Graphics Context:
- Clip To Bounds:
- Autoresize Subviews:

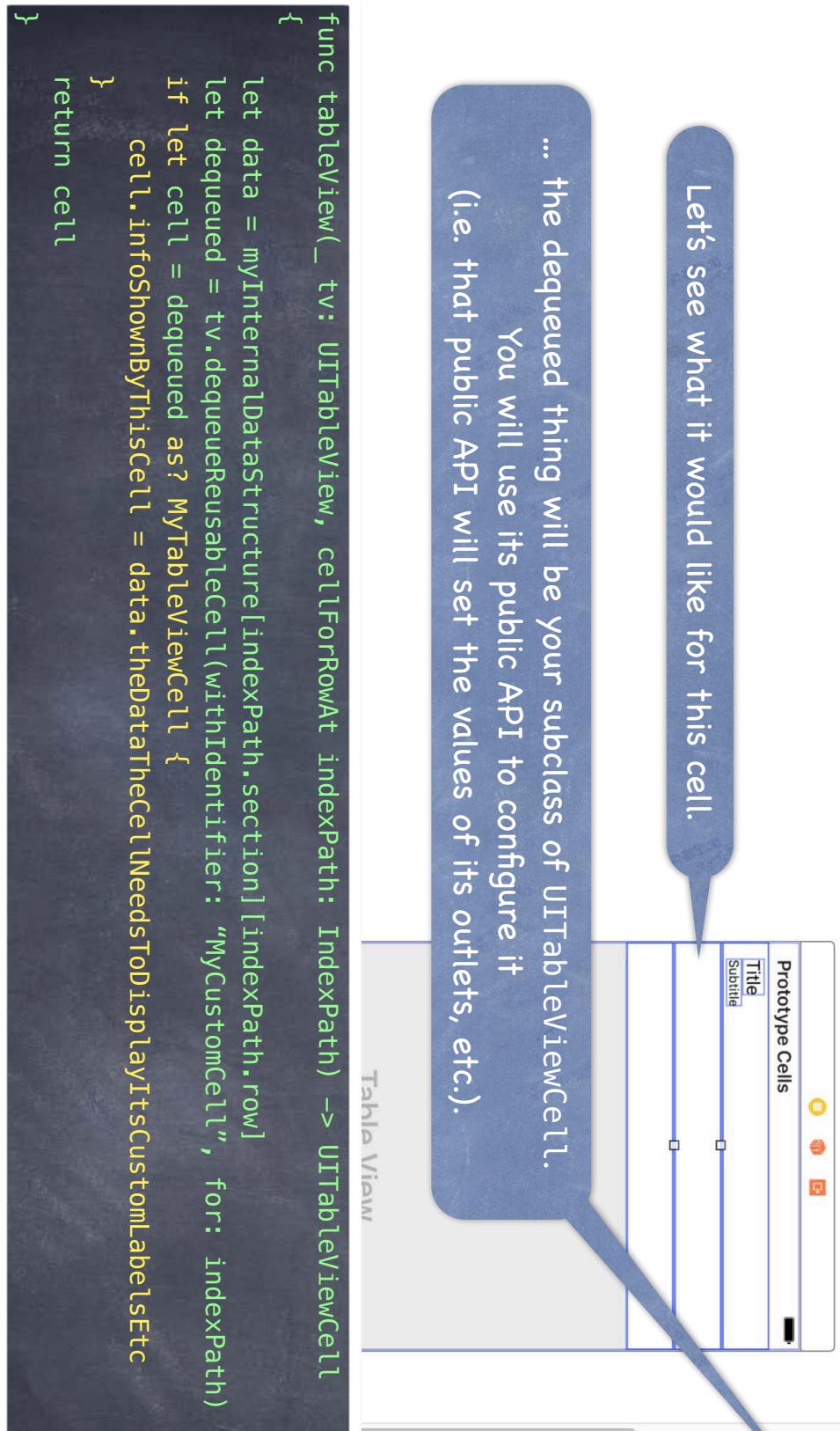
The 'Stretching' section shows values: X: 0, Y: 0, CS: 193, CP: 0.

The code for the UITableView delegate method is as follows:

```
func tableView(_ tv: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {    let data = myInternalDataStructure[indexPath.section][indexPath.row]    let dequeued = tv.dequeueReusableCell(withIdentifier: "MyCustomCell", for: indexPath)    return dequeued}
```

## Identity Inspector

... 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.).



TVCEExample > TVCEExample > TVCEExample > MyTableViewCell.swift > MyTableViewCell

```
// MyTableViewCell.swift  
// TVCEExample
```

```
/// Created by CS193p Instructor.  
/// Copyright © 2017 Stanford University. All rights reserved.
```

```
import UIKit
```

```
class MyTableViewCell: UITableViewCell
```

```
@IBOutlet weak var photoImageView: UIImageView!  
var infoShownByThisCell: Type { didSet { updateUI() } }
```

```
func tableView(_ tv: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell  
{  
    let data = myInternalDataStructure[indexPath.section][indexPath.row]  
    let dequeued = tv.dequeueReusableCell(withIdentifier: "MyCustomCell", for: indexPath)  
    if let cell = dequeued as? MyTableViewCell  
        cell.infoShownByThisCell = 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 numberOfSections(in tv: UITableView) -> Int
func tableView(_ tv: 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 required method in this protocol (as is `cellForRowAt`)
- ⦿ 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 `numberOfSections` and `numberOfRowsInSection`
3. implement `cellForRowAt` to return loaded-up `UITableViewCellCells`

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

So you return this information via `UITableViewDataSource` methods ...

```
func tableView(UITableView, titleForHeaderInSection,FooterInSection: 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

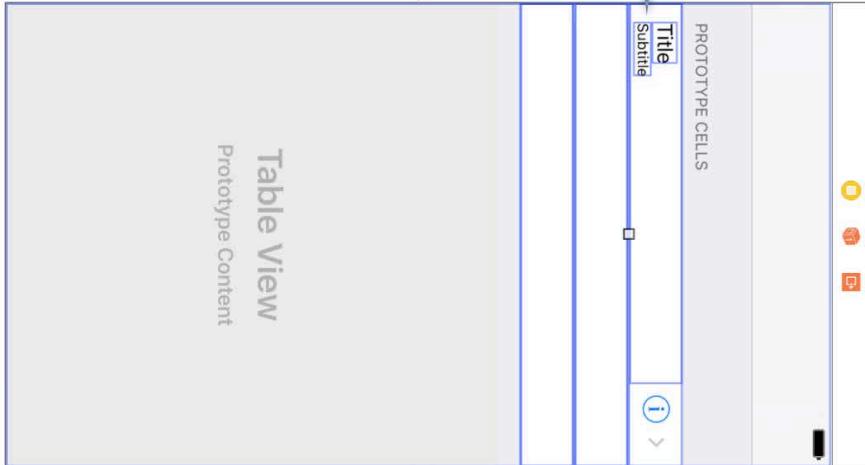
But we're not going to cover them in lecture

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 `UITableViewDataSource` protocol here)



How to segue  
from a table  
view row ...

Navigation Controller



— 75% +



Q E H A

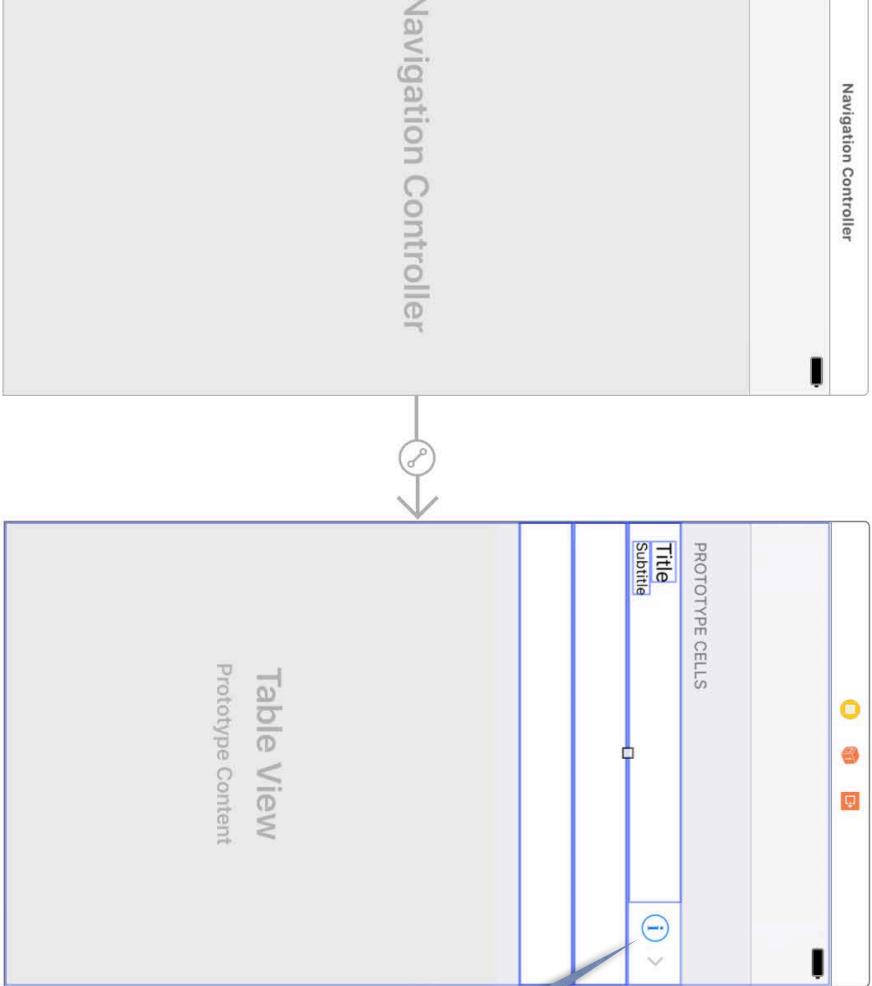
Stretching 0 CS193P Winter 2017

Table View Cell

Style	Subtitle
Image	Image
Identifier	MyCell
Selection	Default
Accessory	Detail Disclosure
Editing Acc.	None
Focus Style	Default
Indentation	0
Level	10
Width	10
<input checked="" type="checkbox"/> Indent While Editing	
<input type="checkbox"/> Shows Re-order Controls	
Separator	Default Insets

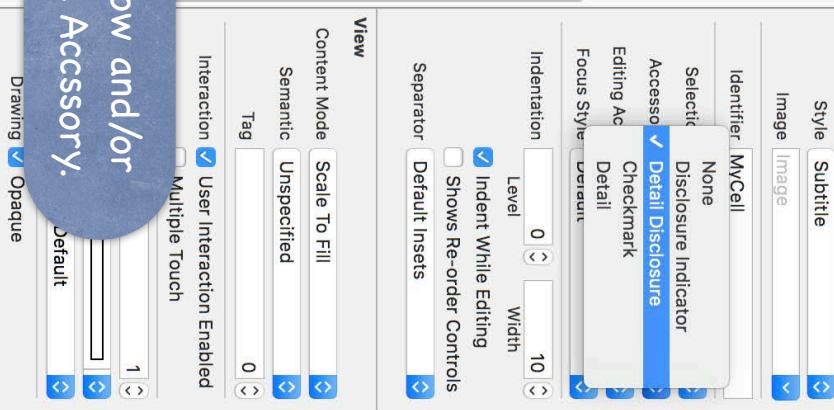
View

Content Mode	Scale To Fill
Semantic	Unspecified
Tag	0
Interaction	<input checked="" type="checkbox"/> User Interaction Enabled
	<input type="checkbox"/> Multiple Touch
Alpha	1
+ Background	
+ Tint	Default
Drawing	<input checked="" type="checkbox"/> Opaque
	<input type="checkbox"/> Hidden
	<input checked="" type="checkbox"/> Clears Graphics Ext
	<input checked="" type="checkbox"/> Clip To Bounds
	<input checked="" type="checkbox"/> Autoresizes Subviews



Note that this row has a Detail Disclosure Accessory.

We can segue from the row and/or from the Detail Disclosure Accessory.



Navigation Controller

View Controller

Table View Cell

Style Subtitle

Image Image

Identifier MyCell

Selection Default

Accessory Detail Disclosure

Editing Acc. None

Focus Style Default

Indentation Level Width

Indent While Editing

Shows Re-order Controls

Separator Default Insets

Navigation Controller

Just ctrl-drag as usual!

Table View

Prototype Content

View Controller

Content Mode Scale To Fill

Semantic Unspecified

Tag 0

Interaction  User Interaction Enabled

Multiple Touch

Alpha 1

Background

Tint Default

Drawing Opaque

Hidden

Clears Graphics

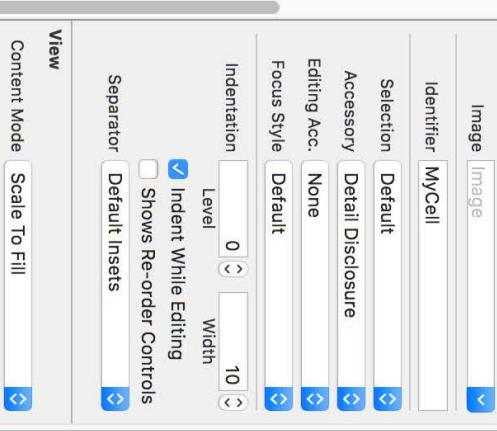
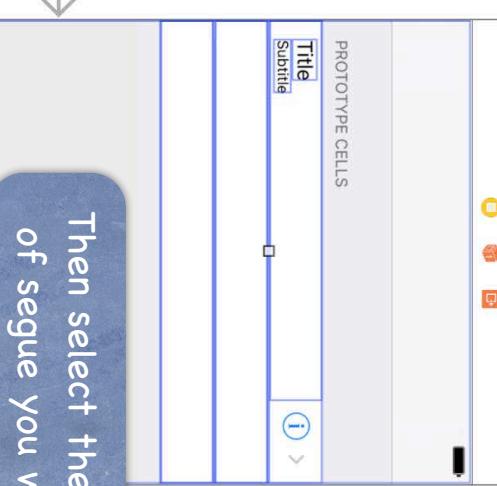
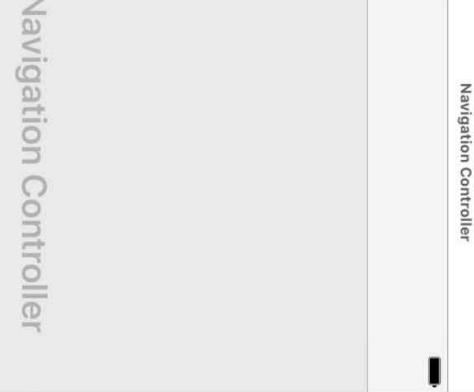
Clip To Bounds

Autoresizes Subviews

Stretching 0

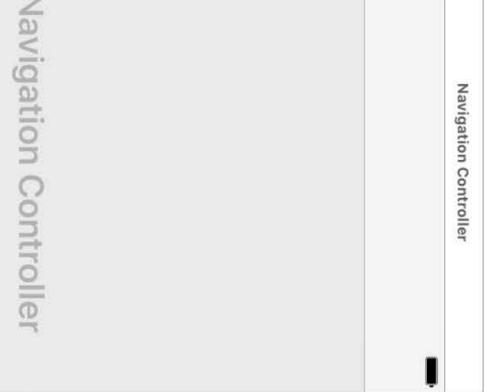
CS193P Winter 2017

View as: iPhone 7 (wC hR)



Navigation Controller

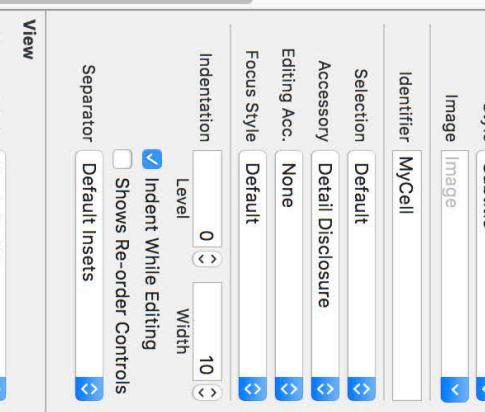
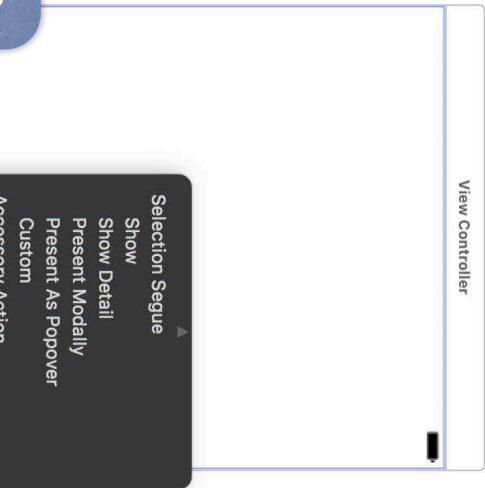
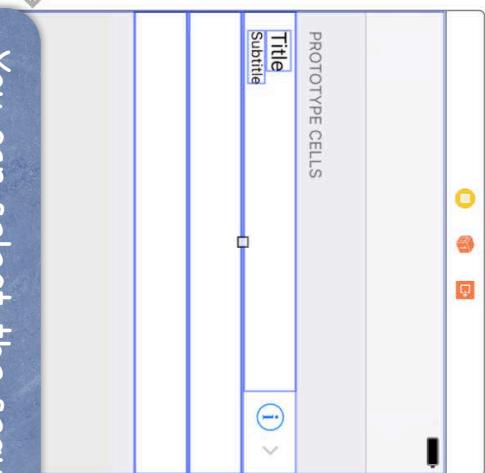
Then select the kind  
of segue you want.

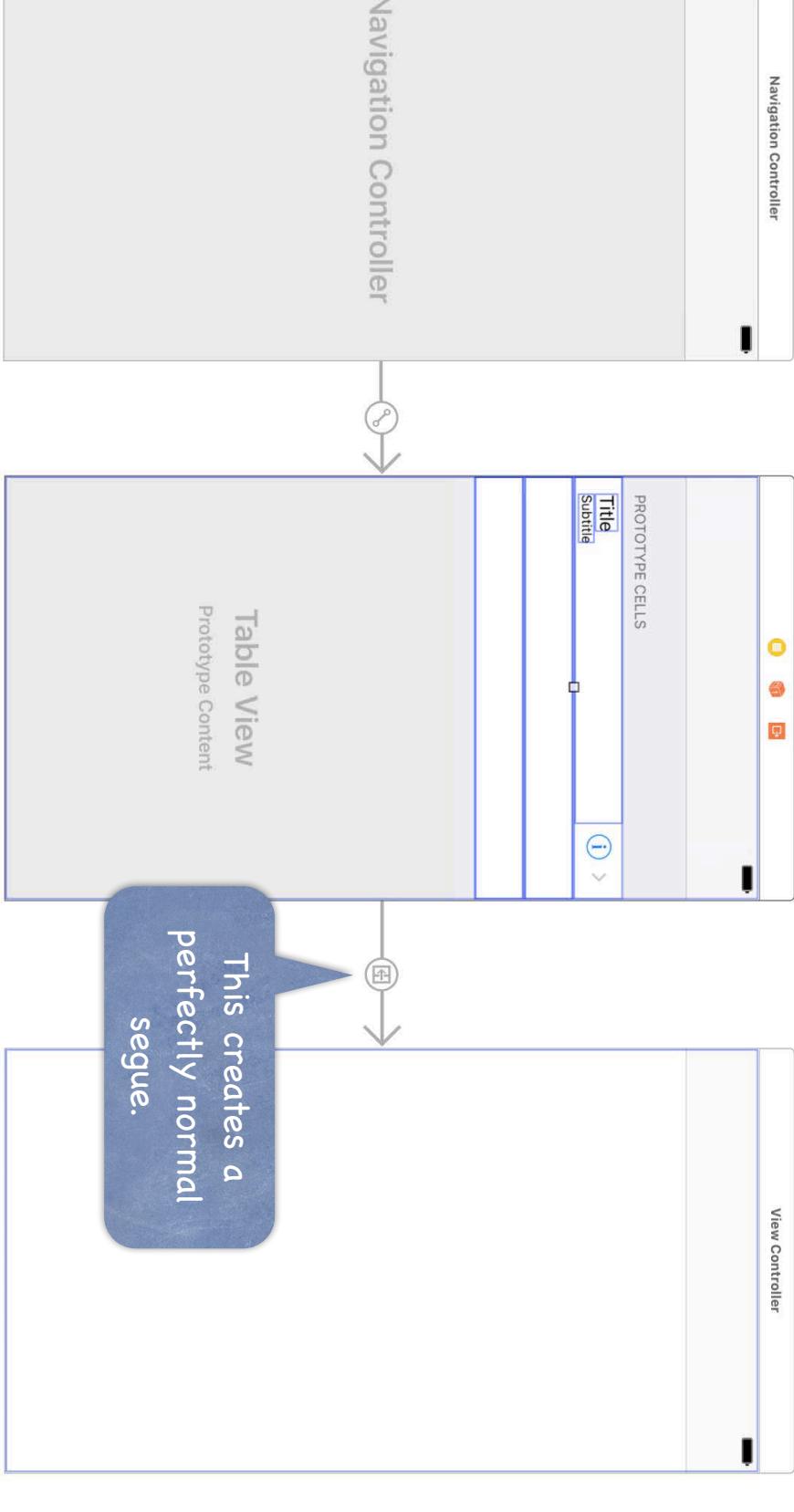


You can select the segue for the Detail Disclosure Accessory too.

Table View  
Prototype Content

Accessory Action  
Show  
Show Detail  
Present Modally  
Present As Popover  
Custom  
Non-Adaptive Selection Segue  
Push (deprecated)  
Modal (deprecated)







<

>

TVCEExample

TVCEExample



<

>

TVCEExample

TVCEExample

Navigation Controller



View Controller



PROTOTYPE CELLS

Title

Subtitle



>

You can inspect it.

Storyboard Segue

Identifier

Class

Module

Kind

Show (e.g. Push)

Animates

Peek & Pop

Preview & Commit Sequences



View as: iphone 7 (wC hR)

— 75% +



□ { } ⓘ Winter 2017



CS193p



<

>

↶

↷



Navigation Controller

Navigation Controller

PROTOTYPE CELLS

Title

Subtitle

i >

And set its identifier.

Storyboard Segue

Identifier: **AbcSegue**

Class: UIStoryboardSegue

Module: None

Kind: Show (e.g. Push)

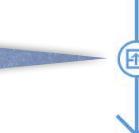
Animates

Peek & Pop  Preview & Commit Sequences

Navigation Controller



Table View  
Prototype Content



Let's take a look at  
prepare(for segue:sender:) ...



# 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 prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if let identifier = segue.identifier {
        switch identifier {
            case "XyzSegue": // handle Xyzsegue here
            case "AbcSegue":
                ...
            default: break
        }
    }
}
```

You can see now why `sender` is `Any`

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 prepare(for segue: UIStoryboardSegue, sender: Any?) {
    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 UITableViewCells 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 prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    if let identifier = segue.identifier {  
        switch identifier {  
            case "XyzSegue": // handle XyzSegue here  
            case "AbcSegue":  
                if let cell = sender as? MyTableViewCell,  
                    let indexPath = tableView.indexPath(for: cell) {  
                    // do something with indexPath  
                }  
            default:  
                break  
        }  
    }  
}
```

indexPath(for cell:) does not accept Any.  
It has to be a UITableViewCell of some sort.

Usually we will need the IndexPath 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 prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if let identifier = segue.identifier {
        switch identifier {
            case "XyzSegue": // handle XyzSegue here
            case "AbcSegue":
                if let cell = sender as? MyTableViewCell,
                    let indexPath = tableView.indexPath(for: cell),
                    let seguedToMVC = segue.destination as? MyMVC {
                    ...
                }
            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 prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if let identifier = segue.identifier {
        switch identifier {
            case "XyzSegue": // handle XyzSegue here
            case "AbcSegue":
                if let cell = sender as? MyTableViewCell,
                    let indexPath = tableView.indexPath(for: cell),
                    let seguedToMVC = segue.destination as? MyMVC {
                    seguedToMVC.publicAPI = data[indexPath.section][indexPath.row]
                }
            default: break
        }
    }
}
```

and then get data from our internal data structure using the IndexPath'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 prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if let identifier = segue.identifier {
        switch identifier {
            case "XyzSegue": // handle XyzSegue here
            case "AbcSegue":
                if let cell = sender as? MyTableViewCell,
                    let indexPath = tableView.indexPath(for: cell),
                    let seguedToVC = segue.destination as? MyVC {
                    seguedToVC.publicAPI = data[indexPath.section][indexPath.row]
                }
            default: break
        }
    }
}
```

and then get data from our internal data structure using the IndexPath'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 the data it displays (that's the dataSource's job), how 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”

- **UITableViewController 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(UITableView, didSelectRowAtIndexPath indexPath: IndexPath) {  
    // 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, accessoryButtonTappedForRowWith indexPath: IndexPath)
```

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 UITableView objects to draw section headers and footers
  - Handling editing rows (moving them around with touch gestures)
  - willBegin/didEnd notifications for editing (i.e. deleting, inserting, moving rows)
  - Copying/pasting rows



# UITableView

- What if your Model changes?

```
func reloadData()
```

Causes the UITableView to call numberOfRowsInSection and numberOfRowsInSection all over again and then cellForRowAt 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 reloadRows(at indexPaths: [IndexPath], with animation: 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(tableView, heightForRowAtIndexPath: IndexPath) -> 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 (UITableView is a subclass of UIScrollView).

- 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

