# Stanford CS193p

Developing Applications for iOS
Winter 2015

# Today

- `UITextField`

  Bonus Topic!

- Table View

  A `UIView` for displaying long lists or tables of data

# UITextField

- Like UILabel, but editable
  - Typing things in on an iPhone is secondary UI (keyboard is tiny).
  - More of a mainstream UI element on iPad.
  - Don't be fooled by your UI in the simulator (because you can use physical keyboard!).
  - You can set attributed text, text color, alignment, font, etc., just like a UILabel.
- Keyboard appears when UITextField becomes "first responder"
  - It will do this automatically when the user taps on it.
  - Or you can make it the first responder by sending it the `becomeFirstResponder` message.
  - To make the keyboard go away, send `resignFirstResponder` to the UITextField.
- Delegate can get involved with Return key, etc.
  - `func textFieldShouldReturn(sender: UITextField)` // sent when "Return" key is pressed
  - Oftentimes, you will `sender.resignFirstResponder` in this method.
  - Returns <u>whether</u> to do normal processing when Return key is pressed (e.g. target/action).

# UITextField

- Finding out when editing has ended

  Another delegate method ...

  `func textFieldDidEndEditing(sender: UITextField)`

  Sent when the text field resigns being first responder.

- UITextField is a UIControl

  So you can also set up target/action to notify you when things change.

  Just like with a button, there are different UIControlEvents which can kick off an action.

  Right-click on a UITextField in a storyboard to see the options available.

# Keyboard

- Controlling the appearance of the keyboard

  Set the properties defined in the `UITextInputTraits` protocol (which `UITextField` implements).

```
var UITextAutocapitalizationType autocapitalizationType   // words, sentences, etc.
var UITextAutocorrectionType autocorrectionType   // yes or no
var UIReturnKeyType returnKeyType                 // Go, Search, Google, Done, etc.
var BOOL secureTextEntry                          // for passwords, for example
var UIKeyboardType keyboardType                   // ASCII, URL, PhonePad, etc.
```

# Keyboard

- The keyboard comes up over other views
  - So you may need to adjust your view positioning (especially to keep the text field itself visible).
  - You do this by reacting to the UIKeyboard{Will,Did}{Show,Hide}Notifications sent by UIWindow.
  - We have not talked about NSNotifications yet, but it's pretty simple.
  - You register a method to get called when a named "event" occurs like this ...

```
NSNotificationCenter.defaultCenter().addObserver(self,
                                                 selector: "theKeyboardAppeared:",
                                                     name: UIKeyboardDidShowNotification
                                                   object: view.window)
```

  - The event here is UIKeyboardDidShowNotification.
  - The object is the one who is causing the even to happen (our MVC's view's window).
  - func theKeyboardAppeared(notification: NSNotification) will get called when it happens.
  - The notification.userInfo will have details about the appearance.
  - UITableViewController listens for this & scrolls table automatically if a row has a UITextField.

# UITextField

- Other UITextField properties

```
var clearsOnBeginEditing: Bool
var adjustsFontSizeToFitWidth: Bool
var minimumFontSize: CGFloat        // always set this if you set adjustsFontSizeToFitWidth
var placeholder: String             // drawn in gray when text field is empty
var background/disabledBackground: UIImage
var defaultTextAttributes: Dictionary // applies to entire text
```

- Other UITextField functionality

  UITextFields have a "left" and "right" overlays.

  You can control in detail the layout of the text field (border, left/right view, clear button).

- Other Keyboard functionality

  Keyboards can have accessory views that appear above the keyboard (custom toolbar, etc.).

```
var inputAccessoryView: UIView  // UITextField method
```

# UITableView

- Very important class for displaying data in a table
  - One-dimensional table.
  - It's a subclass of `UIScrollView`.
  - Table can be static or dynamic (i.e. a list of items).
  - Lots and lots of customization via a `dataSource` protocol and a `delegate` protocol.
  - Very efficient even with very large sets of data.
- Displaying multi-dimensional tables ...
  - Usually done via a `UINavigationController` with multiple MVC's where View is `UITableView`
- Kinds of `UITableViews`
  - Plain or Grouped
  - Static or Dynamic
  - Divided into sections or not
  - Different formats for each row in the table (including completely customized)

# UITableView

UITableViewStyle.Plain

.Grouped

Dynamic (List) & Plain (ungrouped)

Static & Grouped

# UITableView

## Plain Style

Table Header

```
var tableHeaderView: UIView
```

# UITableView

## Plain Style

Table Header

Table Footer

```
var tableFooterView: UIView
```

# UITableView

## Plain Style

Table Header

Section

Table Footer

# UITableView

## Plain Style

Table Header

Section

Table Footer

Section Header

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

No Sections

Sections

# Cell Type

## Subtitle

`UITableViewCellStyle.Subtitle`

## Basic

`.Default`

## Right Detail

`.Value1`

## Left Detail

`.Value2`

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.

TVCExample
iPhone 6
TVCExample
TVCExample
Table View Controller
Prototype Cells
Table View
Prototype Content
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.
wAny hAny
CS193p
Winter 2015

TVCExample
iPhone 6
TVCExample
TVCExample
Table View Controller
Prototype Cells
Table View
Prototype Content
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.
wAny hAny
CS193p
Winter 2015

Just use
File -> New File …
as usual.

Make sure you set the superclass to UITableViewController

CS193p
Winter 2015

TVCExample 〉 iPhone 6

TVCExample 〉 TVCExample

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

My Table View Controller

Prototype Cells

Table View

Prototype Content

View Controller

**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

wAny hAny

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.

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.

CS193p
Winter 2015

TVCExample 〉 iPhone 6

TVCExample 〉 TVCExample

One important attribute is the Plain vs. Grouped style …

Prototype Cells

Table View
Prototype Content

wAny hAny

**Table View**

Content: Dynamic Prototypes
Prototype Cells: 1
Sty: ✓ Plain / Grouped
Separat
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 Vertical
Zoom: 1 Min, 1 Max

CS193p
Winter 2015

TVCExample 〉 iPhone 6

TVCExample 〉 TVCExample

Prototype Cells

Table View

Prototype Content

wAny hAny

**Table View**

Content: Dynamic Prototypes

Prototype Cells: 1

Sty ✓ Plain

Grouped

Separat

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 Vertical

Zoom: 1 Min — 1 Max

CS193p
Winter 2015

TVCExample 〉 iPhone 6

TVCExample 〉 TVCExample

Grouped

PROTOTYPE CELLS

Table View

Prototype Content

wAny hAny

**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

CS193p
Winter 2015

TVCExample
iPhone 6
TVCExample
TVCExample
Grouped
PROTOTYPE CELLS
Table View
Prototype Content
Another important attribute is Dynamic versus Static ...
Table View
Conte
Dynamic Prototypes
Static Cells
Prototype Cel
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 Vertical
Zoom
1
1
Min
Max
wAny hAny
CS193p
Winter 2015

TVCExample
iPhone 6
TVCExample
TVCExample
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
Zoom
1
1
Min
Max
Grouped
Table View
Static Content
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).
wAny hAny
CS193p
Winter 2015

TVCExample
iPhone 6
TVCExample
TVCExample
A more interesting table, however, is a Dynamic one ...
Grouped
Table View
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 Vertical
Zoom
1
1
Min
Max
Table View
Static Content
wAny hAny
CS193p
Winter 2015

TVCExample
iPhone 6
TVCExample
TVCExample
Grouped
PROTOTYPE CELLS
Table View
Prototype Content
wAny hAny
... which we almost always use in Plain style.
Table View
Content
Dynamic Prototypes
Prototype Cells
Plain
Grouped
Separator
Default
Selection
Single Selection
Editing
No Selection During Editing
Show Selection on Touch
Row Limit
0
Text
Background
Normal
Tracking
Scroll View
Style
Scroll Indicators
Shows Horizontal Indicator
Shows Vertical Indicator
Scrolling
Scrolling Enabled
Paging Enabled
Direction Lock Enabled
Bounce
Bounces
Bounce Horizontally
Zoom
Min
Max
CS193p
Winter 2015

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

CS193p
Winter 2015

TVCExample
iPhone 6
TVCExample
TVCExample
Any cell can be inspected in the Attributes Inspector ...
... to set things like the style of the cell.
Prototype Cells
Table View
Prototype Content
wAny hAny
Table View Cell
Custom
Basic
Right Detail
Left Detail
Subtitle
None
Level
Width
Indent While Editing
Shows Re-order Controls
Default Insets
Scale To Fill
Tag
User Interaction Enabled
Multiple Touch
Alpha
Background
Default
Tint
Default
Opaque
Hidden
Clears Graphics Context
Clip Subviews
Autoresize Subviews
Stretching
X
Y
Width
CS193p
Winter 2015

TVCExample 〉 iPhone 6

TVCExample 〉 TVCExample

Subtitle cell style

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: 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

CS193p
Winter 2015

TVCExample 〉 iPhone 6

TVCExample 〉 TVCExample

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

Prototype Cells

Title

Subtitle

Table View

Prototype Content

wAny hAny

**Table View Cell**

Style: Subtitle

Image:

Identifier: Reuse Identifier

Selection: Default

Accesso ✓ None

Disclosure Indicator

Detail Disclosure

Checkmark

Detail

Editing Ac

Indentatic

Indent While Editing

Shows Re-order Co rols

Separator: Default Insets

This one's sort of special ...

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

CS193p
Winter 2015

TVCExample 〉 iPhone 6

TVCExample 〉 TVCExample

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

Prototype Cells

Title
Subtitle

Table View
Prototype Content

wAny hAny

**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

CS193p
Winter 2015

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: 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

CS193p
Winter 2015

One of the cell styles you can choose is Custom.

CS193p
Winter 2015

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

You can change their size.

CS193p
Winter 2015

TVCExample 〉 iPhone 6

TVCExample 〉 TVCExample

Prototype Cells

Table View

Prototype Content

wAny hAny

**Table View Cell**

Style: Custom

Identifier: Reuse Identifier

Selection: Default

Accessory: None

Editing Acc.: None

Indentation: Level 0, Width 10

Indent While Editing

Shows Re-order Controls

Separator: Default Insets

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

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

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

1 2 — **Segmented Control** - Displays multiple segments, each of which functions as a discrete button.

Text — **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.

CS193p
Winter 2015

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

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 ...

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

- New ▸
  - Tab ⌘T
  - Window ⇧⌘T
  - File... ⌘N
  - Playground... ⌥⇧⌘N
  - Target...
  - Project... ⇧⌘N
  - Workspace... ^⌘N
  - Group ⌥⌘N
  - Group from Selection
- Add Files to "TVCExample"... ⌥⌘A
- Open... ⌘O
- Open Recent ▸
- Open Quickly... ⇧⌘O
- Close Window ⌘W
- Close Tab
- Close "Main.storyboard" ^⌘W
- Close Project ⌥⌘W
- Save ⌘S
- Duplicate... ⇧⌘S
- Revert to Saved...
- Unlock...
- Export...
- Show in Finder
- Open with External Editor
- Save As Workspace...
- Project Settings...
- Create Snapshot... ^⌘S
- Restore Snapshot...
- Page Setup... ⇧⌘P
- Print... ⌘P

Table View
Prototype Content

wAny hAny

**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

CS193p
Winter 2015

TVCExample 〉 iPhone 6

TVCExample 〉 TVCExample

Choose options for your new file:

Class: MyTableViewCell

Subclass of: UITableViewCell

Also create XIB file

iPad

Language: Swift

Cancel

Previous

Next

Choose UITableViewCell as the class to subclass off of.

Lab

Table View

Prototype Content

wAny hAny

**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

TVCExample
iPhone 6
TVCExample
TVCExample
Then set it in the Identity Inspector as usual.
Custom Class
Class
UITableViewCell
Module
MyTableViewCell
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
Prototype Cells
Label
Table View
Prototype Content
wAny hAny
CS193p
Winter 2015

TVCExample 〉 iPhone 6

TVCExample 〉 TVCExample

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

Prototype Cells

Label

Table View

Prototype Content

wAny hAny

**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

TVCExample 〉 iPhone 6

TVCExample 〉 TVCExample

Prototype Cells

Label

Table View

Prototype Content

wAny hAny

Manual 〉 TV...ple 〉 TV...ple 〉 MyTableViewCell.swift 〉 No Selection

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

import UIKit

class MyTableViewCell: UITableViewCell {

}
```

Insert Outlet or Outlet Collection

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

And then ctrl-drag as usual.

TVCExample 〉 iPhone 6

TVCExample 〉 TVCExample

Prototype Cells

Label

Table View

Prototype Content

wAny hAny

Manual 〉 TV...ple 〉 TV...ple 〉 MyTableViewCell.swift 〉 No Selection

```
//
//  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 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]





}
```

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



}
```

```
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: AnyObject = tv.dequeueReusableCellWithIdentifier("MyCell", forIndexPath: indexPath)



}
```

The dequeued UITableViewCell unfortunately comes back as AnyObject. So we have to cast it to the proper subclass of UITableViewCell. Then we can configure it and return it.

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


    return cell
}
```

CS193p
Winter 2015

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: AnyObject = tv.dequeueReusableCellWithIdentifier("MyCell", forIndexPath: indexPath)
    let cell = dequeued as UITableViewCell
    cell.textLabel?.text = "Title"
    cell.detailTextLabel?.text = "Subtitle"
    return cell
}
```

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


    return cell
}
```

TVCExample 〉 iPhone 6

TVCExample 〉 TVCExample

... 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.).

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

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

wAny hAny

# 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 required 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 UITableViewDataSource 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 …

TVCExample 〉 iPhone 6

TVCExample 〉 TVCExample

Prototype Cells

Table View

Prototype Content

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

Selection Segue
- show
- show detail
- present modally
- popover presentation
- custom

Accessory Action
- show
- show detail
- present modally
- popover presentation
- custom

Non-Adaptive Selection Segue
- push (deprecated)
- modal (deprecated)

**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

Mode: Scale To Fill

Tag: 0

☑ User Interaction Enabled

☐ Multiple Touch

Alpha: 1

Default

Tint: Default

☑ Opaque ☐ Hidden

☑ Clears Graphics Context

☐ Clip Subviews

☑ Autoresize Subviews

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

wAny hAny

CS193p
Winter 2015

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

CS193p
Winter 2015

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”:
            let cell = sender as MyTableViewCell




        default: break
        }
    }
}
```

  So you will need to cast sender with as or 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”:
            let cell = sender as MyTableViewCell
            if 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”:
            let cell = sender as MyTableViewCell
            if 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”:
            let cell = sender as MyTableViewCell
            if 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 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"

- 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