CS275
Fall 2020

iOS: Text Input and Delegation

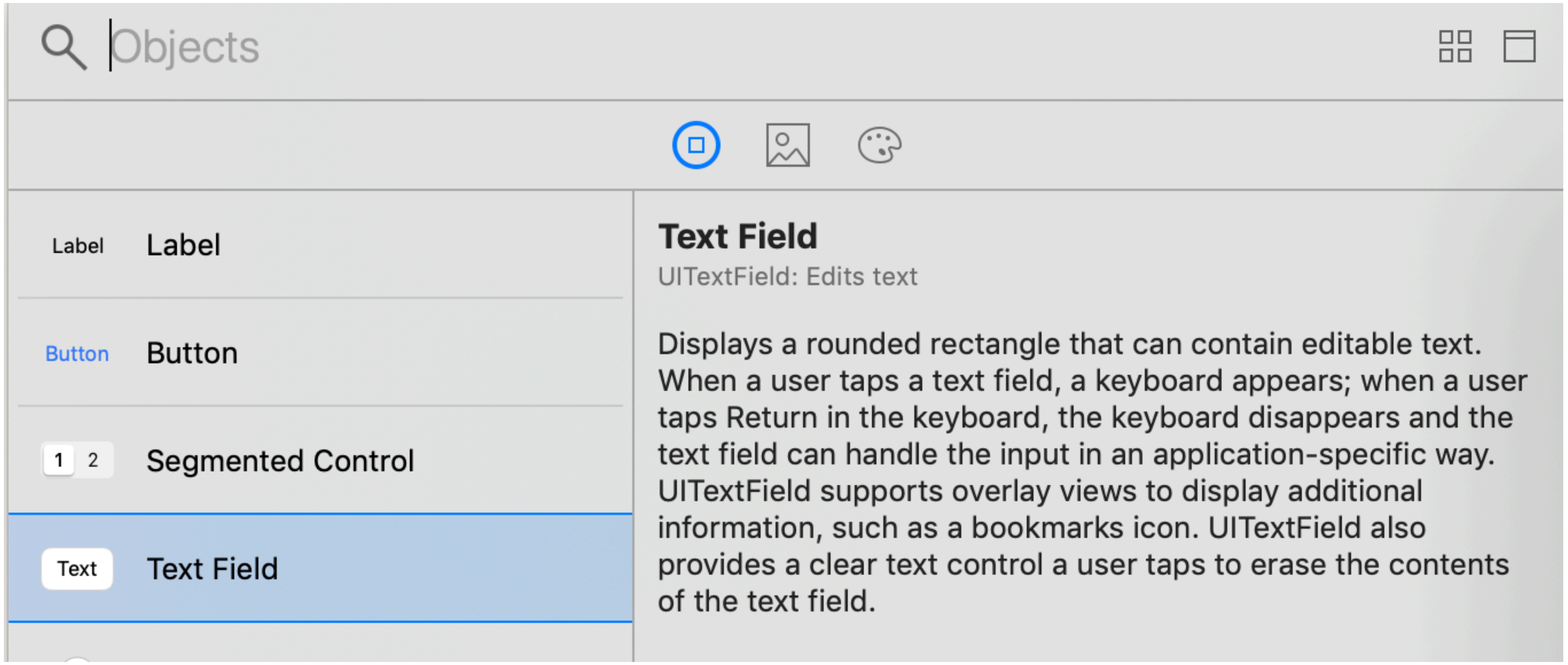
BNRG CHAPTER 4

Topics

- UITextField
- Gesture Recognizers
- Responders
- Number Formatters
- Delegates

UITextField

A `UITextField` is a widget that lets the user type in text



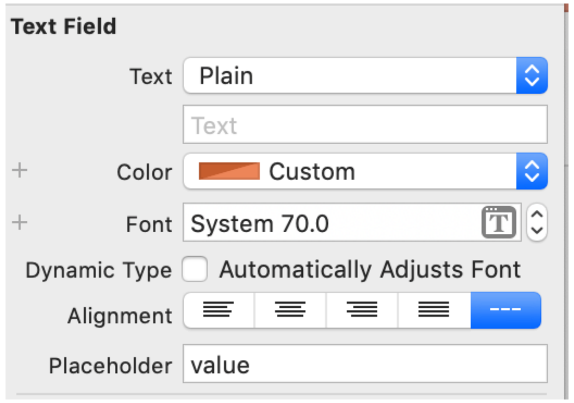
Text Field

UITextField: Edits text

Displays a rounded rectangle that can contain editable text. When a user taps a text field, a keyboard appears; when a user taps Return in the keyboard, the keyboard disappears and the text field can handle the input in an application-specific way. UITextField supports overlay views to display additional information, such as a bookmarks icon. UITextField also provides a clear text control a user taps to erase the contents of the text field.

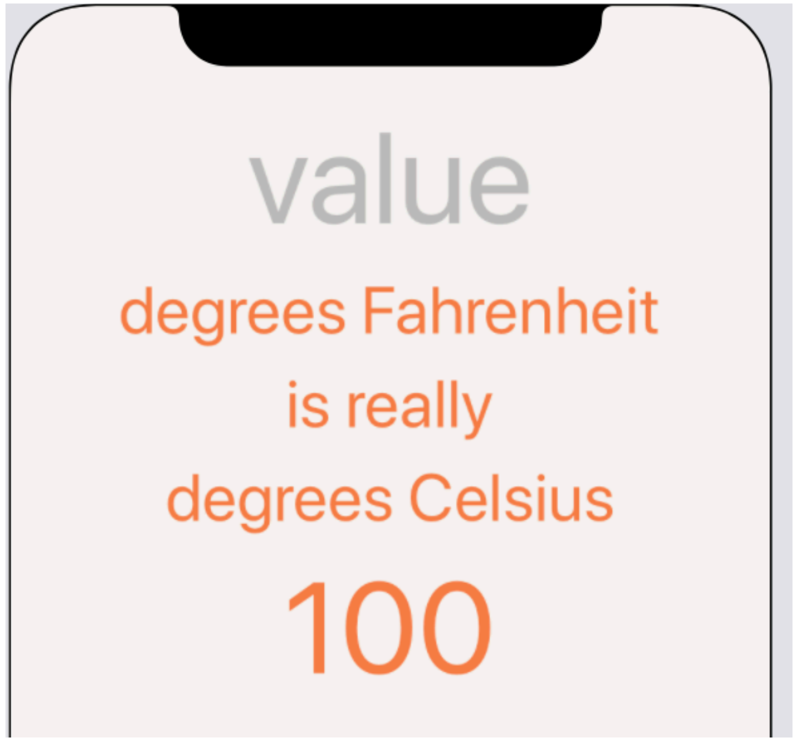
UITextField

- For a text field, we can set all the same kinds of constraints and attributes
  - x, y position
  - centering
  - color
- We can also provide a default text value



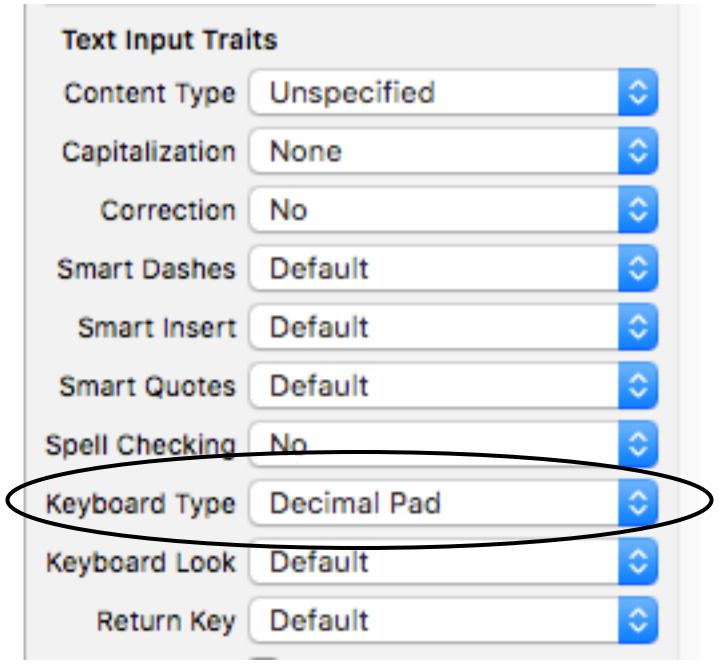
Text Field

Text	Plain
	Text
Color	Custom
Font	System 70.0
Dynamic Type	Automatically Adjusts Font
Alignment	
Placeholder	value



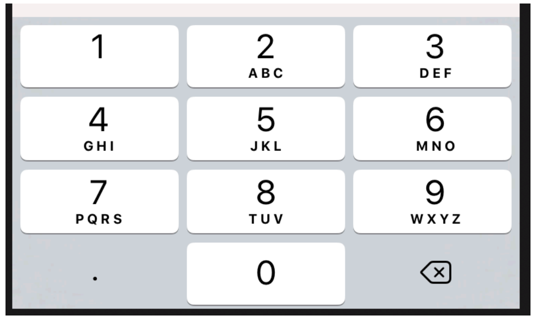
Keyboard Attributes

- When a text field is tapped, a keyboard slides up onto the screen
  - if this doesn't happen, select Keyboard -> Toggle Software Keyboard
- The keyboard's appearance is determined by the text field's `UITextInputTraits`
  - different text fields in an app might need different kinds of input (e.g., numeric vs. full alpha)



Text Input Traits

Content Type	Unspecified
Capitalization	None
Correction	No
Smart Dashes	Default
Smart Insert	Default
Smart Quotes	Default
Spell Checking	No
Keyboard Type	Decimal Pad
Keyboard Look	Default
Return Key	Default



Responding to the Text Field Changes

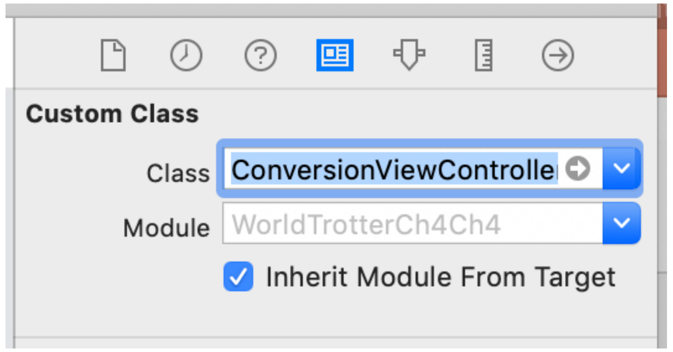
- The work of this app will be to convert the text in the top text field from a Fahrenheit value to a Celsius value and then update the "100" `UILabel` with that value
- To do this, we need a function to listen to the text field and notify us when it has changed
- This listener will go into the `ViewController` class for the app
- When Xcode creates a project, it creates a default Swift file called `ViewController.swift`, containing a class `ViewController`
- It's better to give the file and the class a more meaningful name
  - here, it's `ConversionViewController`, which will go into `ConversionViewController.swift`

Changing the ViewController Class Name

Simplest way to do this

- In the project navigator, delete the `ViewController.swift` that was created for the project
  - it's a stub anyway
- File -> New -> File… and then create a new iOS Swift file `ConversionViewController.swift`

- Then change the custom class in Main.storyboard
  - this is a key step: it tells the storyboard which file contains the ViewController class for the view



Custom Class

Class: ConversionViewControlle

Module: WorldTrotterCh4Ch4

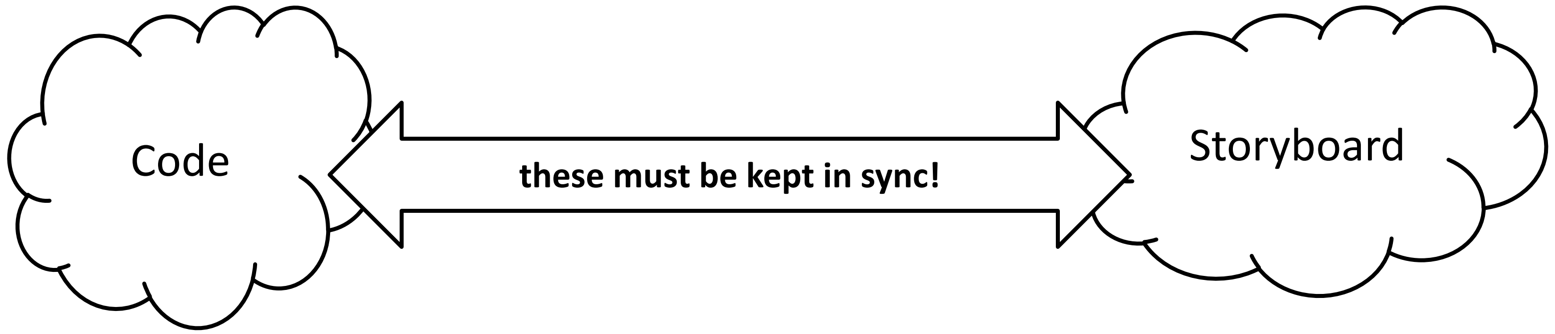
✓ Inherit Module From Target

Storyboard: Pros and Cons

The Storyboard (remember: the Storyboard is an xml file that Xcode renders as a palette) lets us design the UI for an app

- pro: lets us design and see the views for the app and the relations (transitions) between them
- con: must be kept in sync with the code

the two worlds



Hooking up the Text Field and the Label

In `ConversionViewController.swift`:

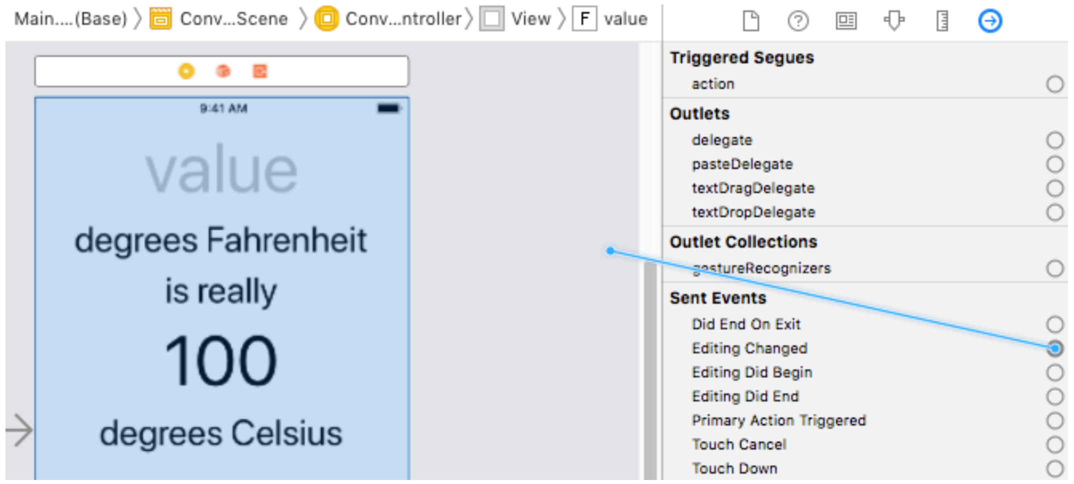
```
class ConversionViewController: UIViewController {
○   @IBOutlet var celsiusLabel: UILabel!

○   @IBAction func fahrenheitFieldEditingChanged(_ textField: UITextField) {
        celsiusLabel.text = textField.text
    }
}
```

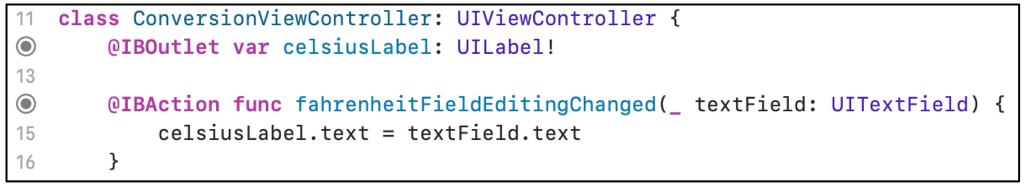
- `@IBOutlet`: a programmatic reference to a UI element
- `@IBAction`: method that will be called in response to user interaction with a UI element

Connecting the ViewController to the GUI

- Control-drag from ConversionViewController to the "100" label
  - and pick celsiusLabel in the pop-up box
- The text listener is a little more complicated
  - select value in the storyboard
  - click the Connections Inspector
  - choose Editing Changed under Sent Events
  - drag to the ConversionViewController
  - pick `fahrenheitFieldEditingChanged(_:)` in the pop-up
- Build and run



Xcode should look like this =>



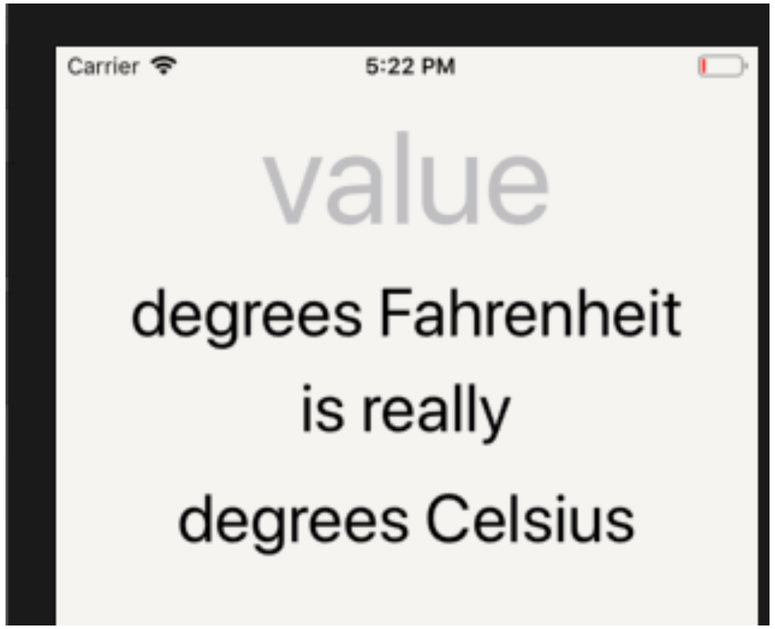
```
class ConversionViewController: UIViewController {
    @IBOutlet var celsiusLabel: UILabel!

    @IBAction func fahrenheitFieldEditingChanged(_ textField: UITextField) {
        celsiusLabel.text = textField.text
    }
```

Setting a Default Value for the Text Field

- Now, the Celsius label text just mimics the text in the Text Field
- If we delete the text in the Text Field, then the Celsius label text becomes "" (the empty string)
- Since the size of "" is zero (horizontally and vertically), the field collapses
- To prevent this, set a default:

```
@IBAction func fahrenheitFieldEditingChanged(_ textField: UITextField) {
    if let text = textField.text, !text.isEmpty {
        celsiusLabel.text = textField.text
    } else {
        celsiusLabel.text = "???"
    }
}
```



`UIResponder`

- A responder is an instance of `UIResponder`
    - subclasses of `UIResponder` include `UIViewController` and `UITextField`
- The responder object receive raw events (such as touches) and either handles the events or forwards them to another responder object
- When a user activates a `UITextField` (by touching in it), the text field becomes the first responder (the responder that is first notified about the event)
    - the method `becomeFirstResponder()` is called on it
- To dismiss the keyboard, we must call `resignFirstResponder()` on the text field
- But to do this, we must first create an outlet (a programmatic reference) to the text field and hook it up to the text field in the UI

Dismissing the Keyboard

One possible behavior in a touch-based UI: dismiss the keyboard by touching somewhere else on the screen

- to do this, we have to be able to receive touch events that aren't tied to a specific UI element
- the way to recognize touch events is through a gesture recognizer

Gesture Recognizer

Gesture recognizer: subclass of `UIGestureRecognizer` that detects a specific touch sequence

- tap, pinch, swipe, etc.

We can use a gesture recognizer to dismiss the keyboard

- the decimal keypad does not have a Return key
- instead, we can add a gesture recognizer to the background view
- this gesture recognizer will call a method that we write
- this method will call `resignFirstResponder()` on the text field

Dismissing the Keyboard

To dismiss the keyboard:

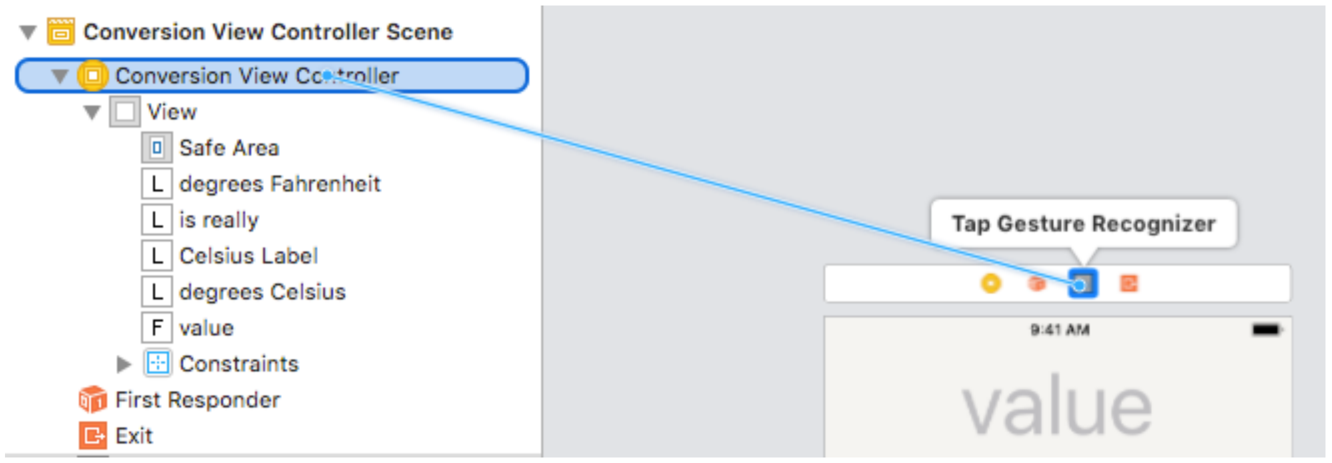
```
@IBAction func dismissKeyboard(_ sender: UITapGestureRecognizer) {
    textField.resignFirstResponder()
}
```

And then add a Tap Gesture Recognizer to the storyboard

- and connect it to the `dismissKeyboard(_:)` method
- more details on next slide

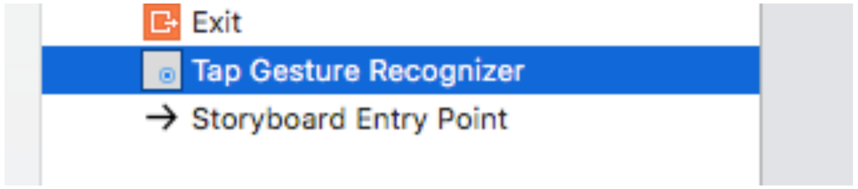
Hooking up the Tap Gesture Recognizer

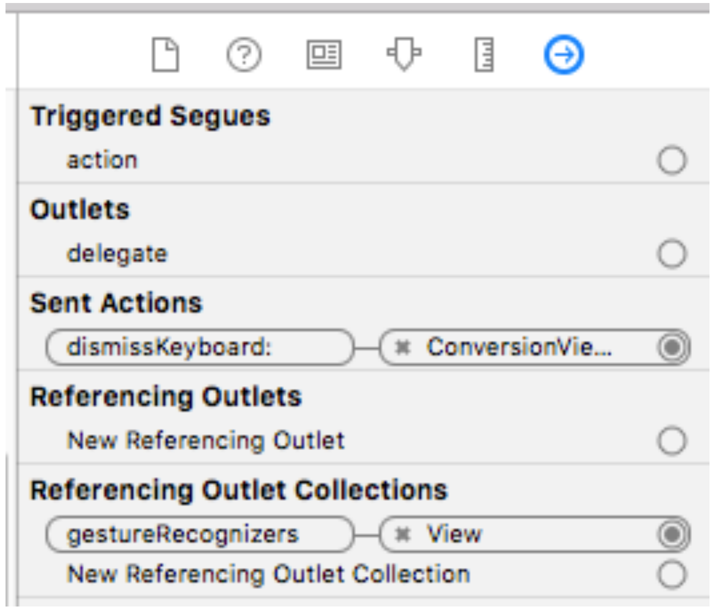
- Select a Tap Gesture Recognizer from the object library
- Drag it to the background of Main.storyboard
- Then control-drag from the Tap Gesture Recognizer in the scene dock to the View Controller
  - and connect it to the `dismissKeyboard(_:)` method



Tap Gesture Recognizer

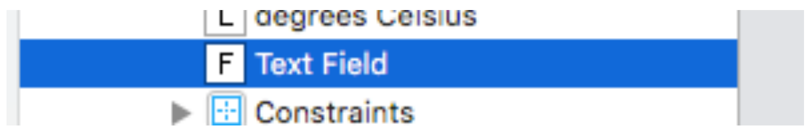
Here are the connections you should see:


Exit
Tap Gesture Recognizer
Storyboard Entry Point

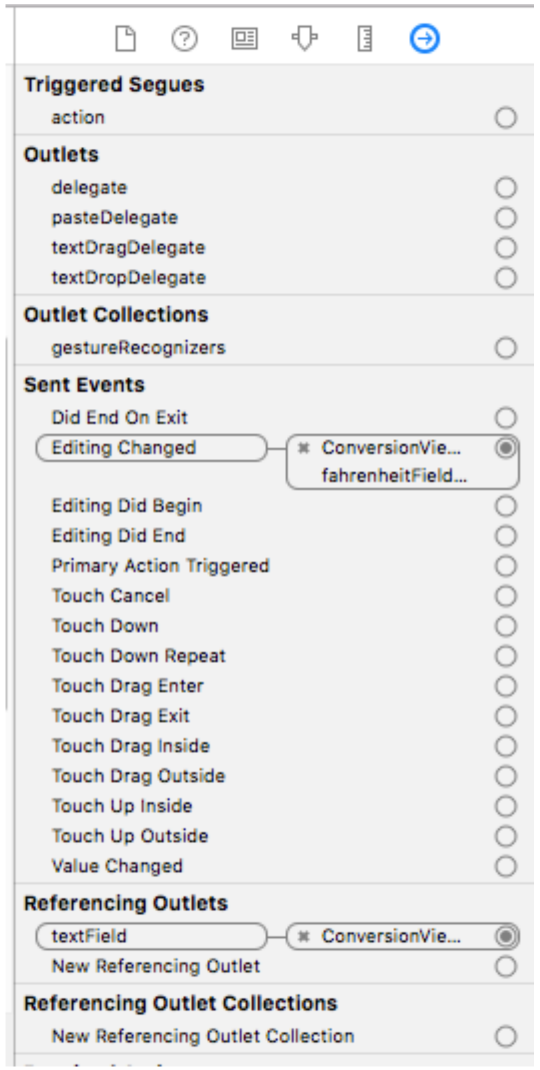

Triggered Segues
action
Outlets
delegate
Sent Actions
dismissKeyboard: — ConversionVie...
Referencing Outlets
New Referencing Outlet
Referencing Outlet Collections
gestureRecognizers — View
New Referencing Outlet Collection

Connections for the Text Field

Here are the connections for the text field:



L degrees Celsius
F Text Field
Constraints



Triggered Segues
action

Outlets
delegate
pasteDelegate
textDragDelegate
textDropDelegate

Outlet Collections
gestureRecognizers

Sent Events
Did End On Exit
Editing Changed — ConversionVie... fahrenheitField...
Editing Did Begin
Editing Did End
Primary Action Triggered
Touch Cancel
Touch Down
Touch Down Repeat
Touch Drag Enter
Touch Drag Exit
Touch Drag Inside
Touch Drag Outside
Touch Up Inside
Touch Up Outside
Value Changed

Referencing Outlets
textField — ConversionVie...
New Referencing Outlet

Referencing Outlet Collections
New Referencing Outlet Collection

Converting F to C

The basic work of the app is to convert a Fahrenheit value to its equivalent Celsius value

- the user keys in the Fahrenheit value in the text field
- the app converts to Celsius and puts the result in the Celsius label

So we'll need a method to do the conversion

`Measurement<UnitType>`

`Measurement<UnitType>` is a part of the iOS Foundation

- Foundation: "The Foundation framework provides a base layer of functionality for apps and frameworks, including data storage and persistence, text processing, date and time calculations, sorting and filtering, and networking."

`Measurement<UnitType>` is a generic data type

- can be an instance of different kinds of units
- provides conversion calculations related to the unit

Converting F to C

Simple, by using built-in `converted(to:)`

```
var fahrenheitValue: Measurement<UnitTemperature>?

var celsiusValue: Measurement<UnitTemperature>? {
    if let fahrenheitValue = fahrenheitValue {
        return fahrenheitValue.converted(to: .celsius)
    } else {
        return nil
    }
}
```

Updating the Celsius Label

Basic idea: whenever the Fahrenheit value changes, update the Celsius label

Elegant way to do this: put a property observer on `fahrenheitValue`

- this property observer should check to see whether `celsiusValue` has a value
- if so, it should update the label with the value

```
var fahrenheitValue: Measurement<UnitTemperature>? {
    didSet {
        updateCelsiusLabel()
    }
}

func updateCelsiusLabel() {
    if let celsiusValue = celsiusValue {
        celsiusLabel.text = "\(celsiusValue.value)"
    } else {
        celsiusLabel.text = "???"
    }
}
```

Responding to the Fahrenheit Field

Last thing: when the user types something in the Fahrenheit field, the `fahrenheitFieldEditingChanged(_:)` method is called

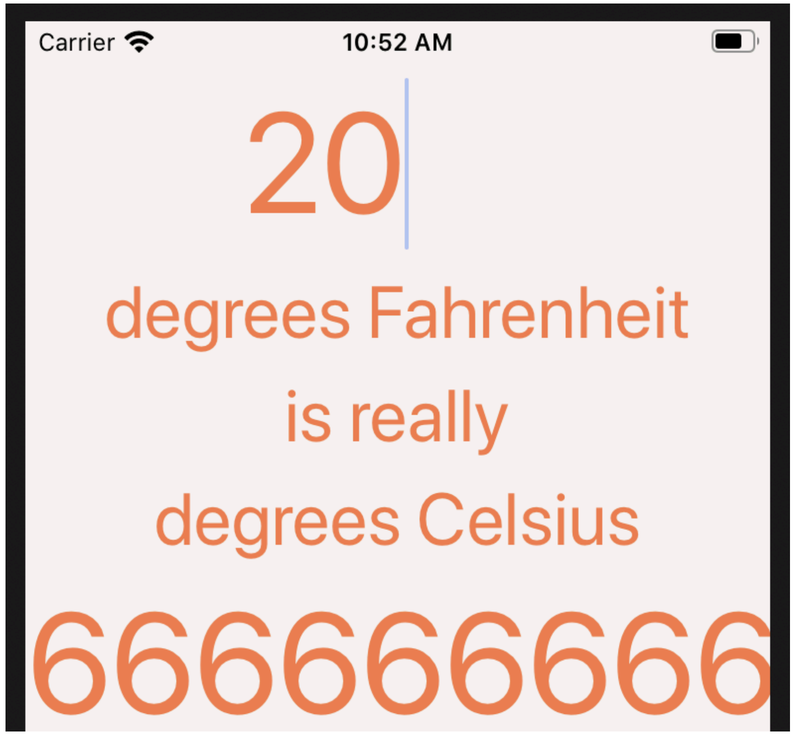
- that method should read the text from the field and create a variable of type `Measurement<UnitTemperature>` from the text

```
@IBAction func fahrenheitFieldEditingChanged(_ textField: UITextField) {
    if let text = textField.text, let value = Double(text) {
        fahrenheitValue = Measurement(value: value, unit: .fahrenheit)
    } else {
        fahrenheitValue = nil
    }
}
```

Input Validation and Decimal Display

There are two slight problems with the app:

1. we need to be able to control the precision with which the Celsius value is displayed


Carrier 10:52 AM

20

degrees Fahrenheit

is really

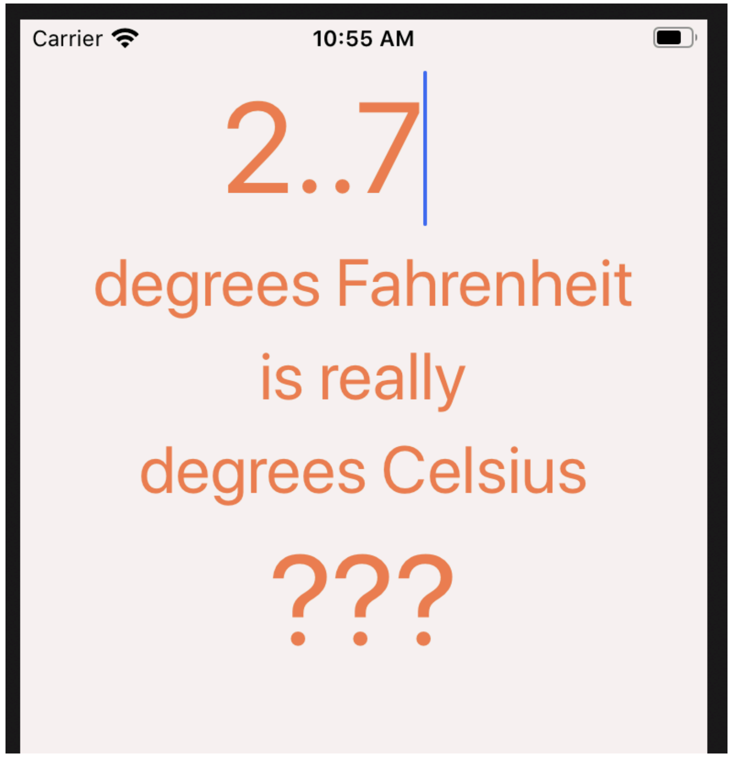
degrees Celsius

666666666

Input Validation and Decimal Display

There are two slight problems with the app:

2. We have to disallow more than one decimal point in the text field



2..7

degrees Fahrenheit

is really

degrees Celsius

???

Number Formatters

Instances of `NumberFormatter` format the textual representation of numbers

```
let numberFormatter: NumberFormatter = {
    let nf = NumberFormatter()
    nf.numberStyle = .decimal
    nf.minimumFractionDigits = 0
    nf.maximumFractionDigits = 1
    return nf
}()
```

this is a closure—it consists of a sequence of statements that are executed, and it then returns a value

this is a little bit of ugliness: the Foundation libraries are written in Objective-C, and so many of their interfaces require Objective-C values

```
celsiusLabel.text = numberFormatter.string(from: NSNumber(value: celsiusValue.value))
```

Delegation

Delegation in Swift is kind of like a generalization of a callback

A callback for an event must be defined ahead of time

Then, whenever the event occurs, the callback function is invoked

Some objects need to make a callback for different kinds of events

- but there is no built-in way for two different callback functions to coordinate and share their information

Delegation

Solution: delegation

Supply a delegate, which will receive all event-related callbacks for a particular object

The delegate object can then store, manipulate, act on, forward the information it receives from the callbacks

Delegation

For iOS, a delegate is a "protocol"

- a specified object to which the first object delegates certain responsibilities

In order for a class to be a delegate, it must implement the set of required interfaces defined in the protocol for that delegate

We will use a delegate to listen to the text field

- if the user enters a decimal point and there is already a decimal, then we'll tell the text field not to accept the second decimal point

Conforming to a Protocol

`UITextFieldDelegate` is the protocol we will use

In order for an object of a particular class to conform to a protocol, the class must implement all required methods

- and it can optionally implement the optional methods

A protocol is not a class

- it's a collection of methods and properties

UITextFieldDelegate

Here's what the `UITextFieldDelegate` protocol looks like:

```
protocol UITextFieldDelegate: NSObjectProtocol {
    optional func textFieldShouldBeginEditing(_ textField: UITextField) -> Bool
    optional func textFieldDidBeginEditing(_ textField: UITextField) -> Bool
    optional func textFieldShouldEndEditing(_ textField: UITextField) -> Bool
    optional func textFieldDidEndEditing(_ textField: UITextField) -> Bool
    optional func textField(_ textField: UITextField,
                            shouldChangeCharactersIn range: NSRange,
                            replacementString string: String) -> Bool
    optional func textFieldShouldClear(_ textField: UITextField) -> Bool
    optional func textFieldShouldReturn(_ textField: UITextField) -> Bool
}
```

Conforming to a Delegate

To conform to one or more delegates, add the delegate(s) after the superclass (if there is one):

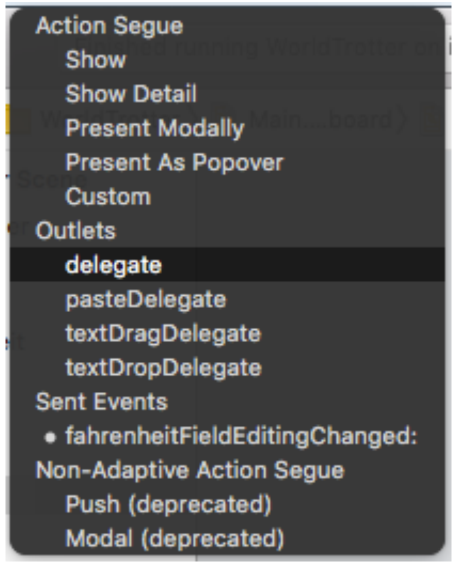
```
class ConversionViewController: UIViewController, UITextFieldDelegate {
   // implementation of the class
}
```

Here, `UITextFieldDelegate` is the delegate, and `UIViewController` is the superclass

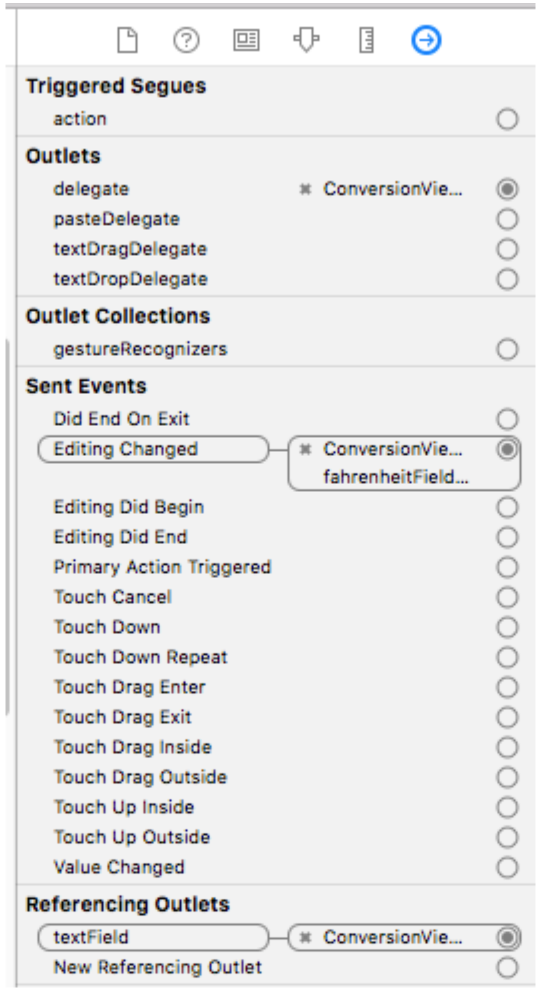
Activating the `UITextFieldDelegate`

Specify the delegate for the text field:

- Control-drag from the text field to the `ConversionViewController`
- and then specify delegate under Outlets


Action Segue
Show
Show Detail
Present Modally
Present As Popover
Custom
Outlets
delegate
pasteDelegate
textDragDelegate
textDropDelegate
Sent Events
• fahrenheitFieldEditingChanged:
Non-Adaptive Action Segue
Push (deprecated)
Modal (deprecated)

text field connections should now look like this:


Triggered Segues
action
Outlets
delegate ✖ ConversionVie...
pasteDelegate
textDragDelegate
textDropDelegate
Outlet Collections
gestureRecognizers
Sent Events
Did End On Exit
Editing Changed ✖ ConversionVie... fahrenheitField...
Editing Did Begin
Editing Did End
Primary Action Triggered
Touch Cancel
Touch Down
Touch Down Repeat
Touch Drag Enter
Touch Drag Exit
Touch Drag Inside
Touch Drag Outside
Touch Up Inside
Touch Up Outside
Value Changed
Referencing Outlets
textField ✖ ConversionVie...
New Referencing Outlet

Preventing Two Decimal Points

The logic is:

```
if the current text in the text field has a decimal
then
  if the new text also has a decimal
  then
    don't accept the new text
  else
    new text is OK
else
  new text is OK
```

Preventing Two Decimal Points

Here's the code:

```
let existingTextHasDecimalSeparator = textField.text?.range(of: ".")
let replacementTextHasDecimalSeparator = string.range(of: ".")

if existingTextHasDecimalSeparator != nil,
    replacementTextHasDecimalSeparator != nil {
    return false
} else {
    return true
}
```

Challenge

- Do the Bronze Challenge: prevent the user from entering non-numeric characters
- You'll have to hunt around on the web to find how to use NSCharacterSet
- In general, the challenges in the book are great practice for extending your knowledge of iOS development