

CSC 471 / 371
Mobile Application
Development for iOS



Prof. Xiaoping Jia
School of Computing, CDM
DePaul University
xjia@cdm.depaul.edu
[@DePaulSWEng](https://twitter.com/DePaulSWEng)

Building UI – Part 2
Stack Views,
Text Input & Popups

Outline

- More auto layout
 - Stack views
- Text input
 - Text Field
 - Text Area
- Popups & alerts



DEPAUL UNIVERSITY 3

Auto Layout with
Stack Views

Stack View

- Introduced in iOS 9
 - To simplify auto layout
- A *stack view* is either a row or a column of UI widgets
 - Light-weight, nested view container
- Attributes
 - **axis:** vertical or horizontal
 - **spacing**
 - **alignment**
 - **distribution**



DEPAUL UNIVERSITY 5

Stack View – Alignment

Top			
Center			
Bottom			
Baseline			

DEPAUL UNIVERSITY 6

Stack View – Alignment

The diagram illustrates four alignment modes for a stack view:

- Fill:** Four identical Jupiter planet icons are shown.
- Leading:** Four identical Jupiter planet icons are shown, with the first one shifted to the left.
- Center:** Four identical Jupiter planet icons are shown, with the second one shifted to the left and the third one shifted to the right.
- Trailing:** Four identical Jupiter planet icons are shown, with the second one shifted to the right and the third one shifted to the left.

DEPAUL UNIVERSITY 7

Stack View – Distribution

The diagram illustrates three distribution modes for a stack view:

- Fill Equally:** Four identical Jupiter planet icons are shown, with equal vertical spacing between them.
- Fill Proportionally:** Four identical Jupiter planet icons are shown, with the first icon being larger than the others.
- Equal Spacing:** Four identical Jupiter planet icons are shown, with equal vertical spacing between them.

DEPAUL UNIVERSITY 8

Using Stack View

- Widgets inside a stack view are managed by the stack view
- A stack view can be treated as a single item in auto layout
- Find stackable regions and use stack view first
- Use constraints to layout the stack view and its siblings

DEPAUL UNIVERSITY 9

Images – Chicago Using Stack Views

- Add 3 Labels and an Image View
- Select all 3 labels
- Click the Stack tool
- The three labels are horizontally stacked in a Stack View

DEPAUL UNIVERSITY 10

Images – Chicago Using Stack Views

- Select the Stack View, in Attribute Inspector,
- Spacing : 20
- Distribution: equal spacing

DEPAUL UNIVERSITY 11

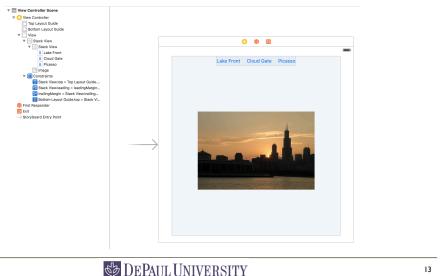
Images – Chicago Using Stack Views

- Select the Stack View and the Image View
- Click the Stack tool
- The Stack View with 3 labels and the Image View are stacked vertically in a new Stack View

DEPAUL UNIVERSITY 12

Images – Chicago Using Stack Views

- Select the the outer *Stack View*, Pin all 4 sides

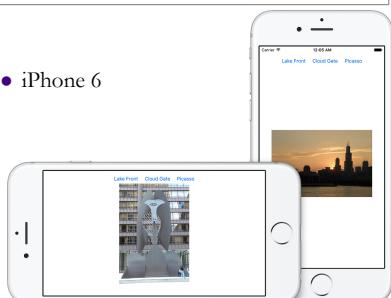


DEPAUL UNIVERSITY

13

Images – Chicago Using Stack Views

- iPhone 6

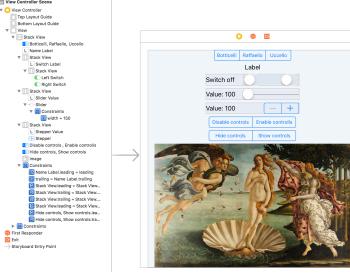


DEPAUL UNIVERSITY

14

Controls – Stack View

- Nested *Stack Views*
- Aligned leading and trailing edges of the three horizontal *Stack Views*
- Lower the content hugging priority of the *Image View*



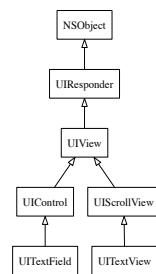
DEPAUL UNIVERSITY

15

Text Input Text Fields & Text Views

Text Input Widgets

- Allow users to input and edit text
- UITextField*
 - Single-line text
 - Subclass of *UIControl*
- UITextView*
 - Multi-line text
 - Subclass of *UIScrollView*
- Support soft keyboards

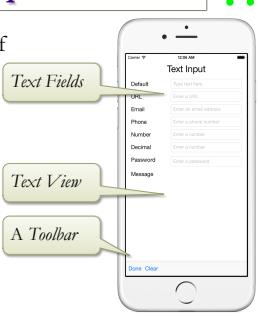


DEPAUL UNIVERSITY

17

The Text Input App

- To demo various types of text input
 - Text field
 - Text view
- Soft keyboard options
 - Keyboard types
 - Prompt
 - Return key
 - Clear key



DEPAUL UNIVERSITY

18

Auto Layout – Add Constraints

- The title label
 - Pin the top, leading (left), and trailing (right) edges
- The text field labels
 - Pin the top and leading (left) edges
- The text fields
 - Pin/align the leading (left) and trailing (right) edges
 - Align with the label, vertical center or baseline



DEPAUL UNIVERSITY 19

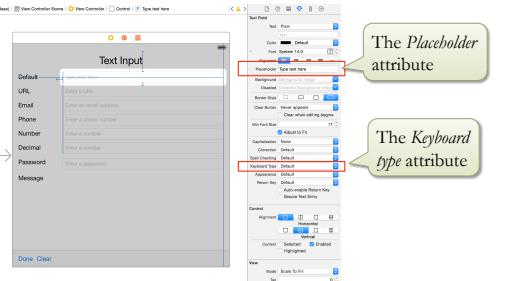
Auto Layout – Add Constraints

- The “Message” label
 - Pin the top and leading (left) edges
- The “Message” text area
 - Pin the top and bottom edges
 - Align the leading (left) and trailing (right) edges
- The toolbar
 - Pin the bottom, the leading (left), and trailing (right) edges



DEPAUL UNIVERSITY 20

The Text Fields – The Keyboard Attributes



The Placeholder attribute
The Keyboard type attribute

DEPAUL UNIVERSITY 21

Text Fields – Soft Keyboard Type

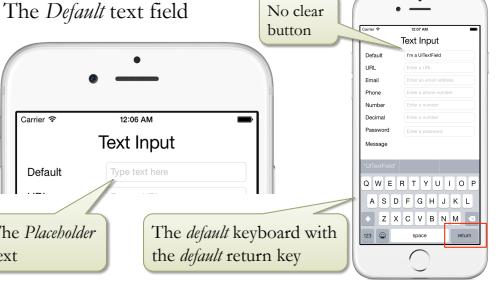
- Types of soft keyboards
- Also support different languages



DEPAUL UNIVERSITY 22

The Keyboard Attributes

- The Default text field
 - No clear button



The Placeholder text
The default keyboard with the default return key

DEPAUL UNIVERSITY 23

The Keyboard Attributes

- The URL & Email keyboard
- The highlighted return key, or the action key
 - Return Key** attribute
- The clear button
 - Clear Button** attribute



DEPAUL UNIVERSITY 24

The Secure Text Input

- For passwords etc.
- Check the *Secure Text Entry* attribute

DEPAUL UNIVERSITY 25

The Text Fields – Handle Key Press

- The soft keyboard will pop up when the text field is touched.
 - Standard behavior handled by the framework
- Typing into a text field (key press events)
 - Standard behavior handled by the framework
 - Nothing to be done in the user code
 - Access the input text using the property: `textField.text`
- Need to dismiss the keyboard**
 - It is the application's responsibility
 - The behavior may depend on application specific logic

2/1/16

DEPAUL UNIVERSITY

26

The First Responder

- A *responder* is an object that can respond to events and handle them.
- The *first responder* is the first responder object to receive user events, including key events.
 - Responders are organized as a *chain*
- When the user taps a view, the system automatically designates that view as the first responder.
- A soft keyboard is presented when a text field or a text view becomes the first responder

DEPAUL UNIVERSITY 27

Dismiss Soft Keyboard

- To dismiss the soft keyboard, stop the text field or text view being the first responder
 - Call the method: `resignFirstResponder`
- When to dismiss the soft keyboard?
 - For text field: when the return key, i.e., the action key, is pressed
- How do we know the return key is pressed?
 - Connect an action to the “Did End On Exit” event of the text field
 - Target-action pattern

DEPAUL UNIVERSITY

28

Dismiss Soft Keyboard – Implementation

- Define an action method: `editEnded`
 - Connect the action method to the `Default`, `URL`, `Email` and `Password` text fields and the event “Did End On Exit”

```

@IBAction func editEnded(sender: UITextField) {
    sender.resignFirstResponder()
}

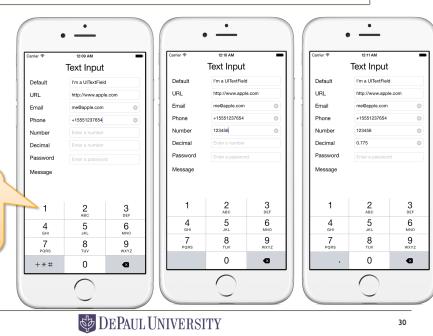
```

DEPAUL UNIVERSITY 29

The Text Fields – The Number Pads

- Phone
- Number
- Decimal

Where is
the return
key?



Dismiss the Number Pads

- The number pad has no return or action key
- Solution A:
 - Provide an action key in the app UI
- Solution B:
 - Touch the background, i.e., the container view, to dismiss the keyboard.
 - Change the container view class from `UIView` to `UIControl`
 - Connect an action to the “*Touch Down*” event of the container view.
 - Target-action pattern

DEPAUL UNIVERSITY

31

Outlet Collections

- How do we know which text field is the first responder?
- Resign each one of them
- Connect an *outlet connection*, i.e., an array of outlets
 - Containing all the text fields
- Similar to connect an outlet
 - Choose *outlet collection* instead
 - Add each view to the outlet connection

```
@IBOutlet var textFields: [UITextField]!
@IBOutlet weak var textView: UITextView!
```

DEPAUL UNIVERSITY

33

Background Tap Event

- Connect an action to the “*Touch Down*” event of the container view: `backgroundTouched`
- Resign first responder for each text field and the text view.

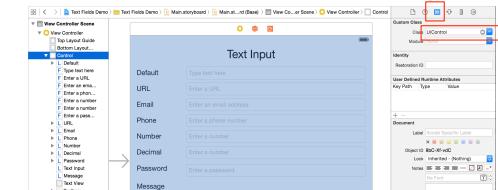
```
@IBAction func backgroundTouched(sender: UIControl) {
    for tf in textFields {
        tf.resignFirstResponder()
    }
    textView.resignFirstResponder()
}
```

DEPAUL UNIVERSITY

35

Change the Class of a View

- Select the root container view
- Select the *Identity Inspector*
- Select a class in the *Class* drop-down list



DEPAUL UNIVERSITY

32

Add Views to an Outlet Collection

- Drag from the dot next to the outlet collection
- to the view to be added



DEPAUL UNIVERSITY

34

Toolbar

- Class: `UIToolbar`
- A horizontal bar containing action buttons, known as *bar items*
 - Bar items behave similarly to buttons
- Typically appear at the top or bottom of the screen.



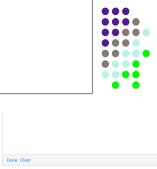
DEPAUL UNIVERSITY

36

Toolbar

- A toolbar at the bottom
 - Contains two bar items: *Done* and *Clear*
- Connect each bar item to an action
 - Ctrl-drag to the view controller
- The clear action


```
@IBAction func clearAction() {
    for tf in textfields {
        tf.text = ""
    }
    textView.text = ""
}
```



DEPAUL UNIVERSITY 37

The Popups

- Two styles of popups are supported
 - Action Sheet*
 - Alert*
- Both are modal
- Consists of
 - Title
 - Message
 - One or more buttons

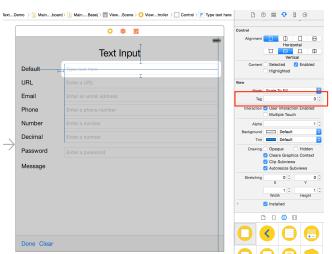


An *Alert* popup

DEPAUL UNIVERSITY 38

Using the Tag Attribute

- Each view object has an integer tag attribute
 - Can be used to identify each view object
- Set the tags of text fields from 0 – 6, from top to bottom



DEPAUL UNIVERSITY 39

The Text Field Labels

- The labels of the text fields, in the order from top to bottom

```
let labels = [
    "Default",
    "URL",
    "Email",
    "Phone",
    "Number",
    "Decimal",
    "Password"
]
```

The positions of the labels in the array correspond the tag values of the associated text fields.



DEPAUL UNIVERSITY 40

Action to Display a Popup

```
@IBAction func doneAction() {
    var input : [Int:String] = [:]
    for tf in textFields {
        tf.resignFirstResponder()
        input[tf.tag] = tf.text ?? ""
    }
}
```

Formulate a message

Display an *Alert* popup

A dictionary of input values

DEPAUL UNIVERSITY 41

Action to Display a Popup

```
@IBAction func doneAction() {
    var input : [Int:String] = [:]
    ...
    var message = ""
    for (i, label) in labels.enumerate() {
        if let text = input[i] {
            message += "\n\((label)): \(text)"
        }
    }
    if !textView.text.isEmpty {
        message += "\nMessage: \(textView.text)"
    }
}
```

Display an *Alert* popup

DEPAUL UNIVERSITY 42

Action to Display a Popup

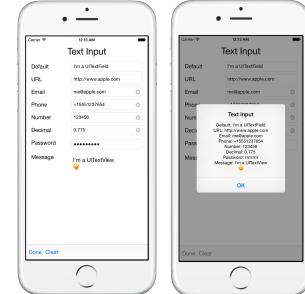
```
@IBAction func doneAction() {
    ...
    var message = ""

    let title = "Text Input"
    let alertController = UIAlertController(title: title,
                                           message: message, preferredStyle: .Alert)
    let cancelAction = UIAlertAction(title: "OK",
                                    style: .Cancel, nil)
    alertController.addAction(cancelAction)
    presentViewController(alertController, animated: true,
                      completion: nil)
}
```

DEPAUL UNIVERSITY

43

The Alert Popup



DEPAUL UNIVERSITY

44

Sample Code

- Images – Stack View.zip
- Controls – Stack View.zip
- Text Input.zip

DEPAUL UNIVERSITY

45

Next ...

- Multiple views
- Segues
- Tabbed views

❖ iOS is a trademark of Apple Inc.

DEPAUL UNIVERSITY

46