CSC 471 / 371 Mobile Application Development for iOS

Prof. Xiaoping Jia
School of Computing, CDM
DePaul University
xjia@cdm.depaul.edu
@DePaulSWEng

1

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

2

Outline

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

3

Auto Layout with Stack Views

4

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`

5

Stack View – Alignment

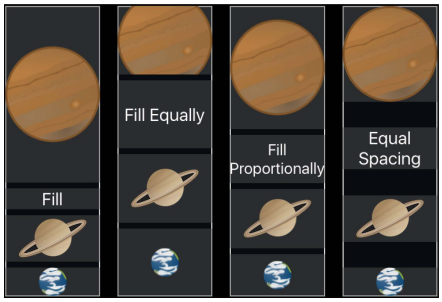
6

Stack View – Alignment

DePaul University 7

7

Stack View – Distribution



DePaul University 8

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

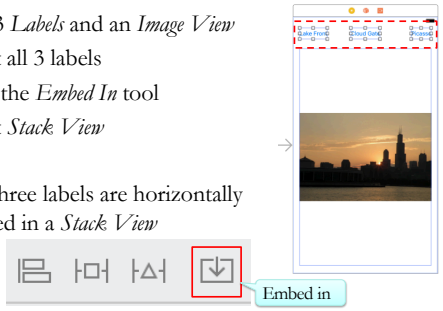
DePaul University 9

9

Images – Chicago Using Stack Views

- Add 3 *Labels* and an *Image View*
- Select all 3 labels
- Click the *Embed In* tool
- Select *Stack View*

- The three labels are horizontally stacked in a *Stack View*



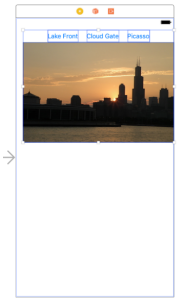
DePaul University 10

10

Images – Chicago Using Stack Views

- Select the *Stack View* and the *Image View*
- Click the *Embed In* tool
- Select *Stack View*

- The *Stack View* with 3 labels and the *Image View* are stacked vertically in a new *Stack View*
  - Alignment: *Center*

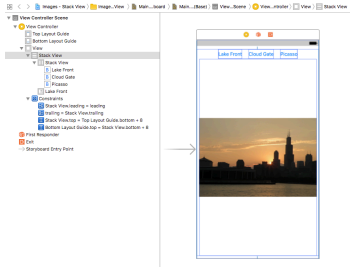


DePaul University 11

11

Images – Chicago Using Stack Views

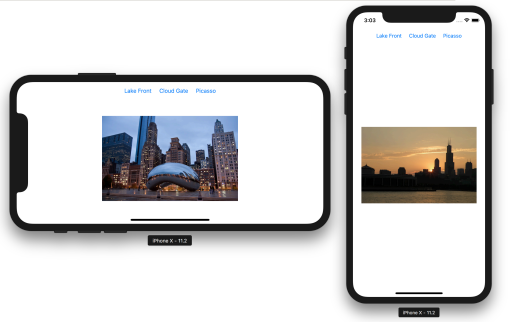
- Select the the outer *Stack View*, Pin all 4 sides



DePaul University 12

12

Images – Chicago Using Stack Views

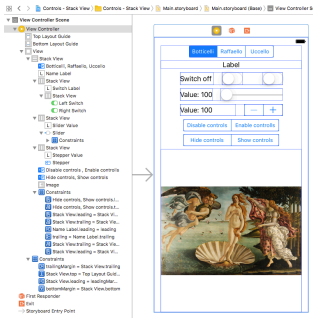


DePaul University 13

13

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 14

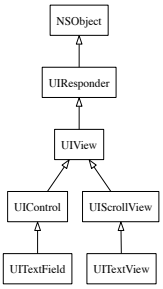
14

Text Input
Text Fields & Text Views

15

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

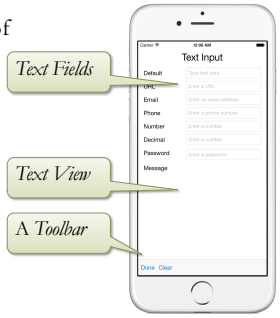


DePaul University 16

16

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

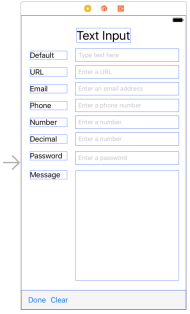


DePaul University 17

17

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

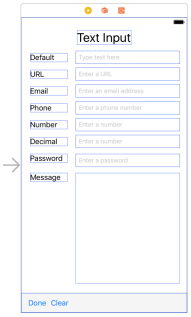


DePaul University 18

18

Auto Layout – Add Constraints

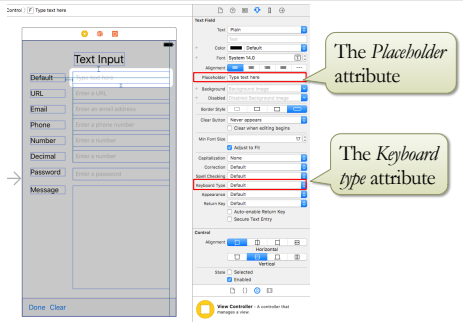
- The "Message" label
  - Pin the top and leading (left) edges
- The "Message" text area
  - Pin the top and bottom edges
  - Align the leading (left) and trailing (right) edges
- The toolbar
  - Ping the bottom, the leading (left), and trailing (right) edges



DEPAUL UNIVERSITY 19

19

The Text Fields – The Keyboard Attributes

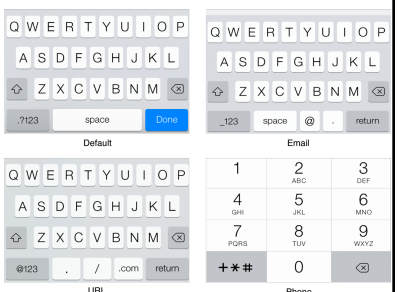


DEPAUL UNIVERSITY 20

20

Text Fields – Soft Keyboard Type

- Types of soft keyboards
- Also support different languages

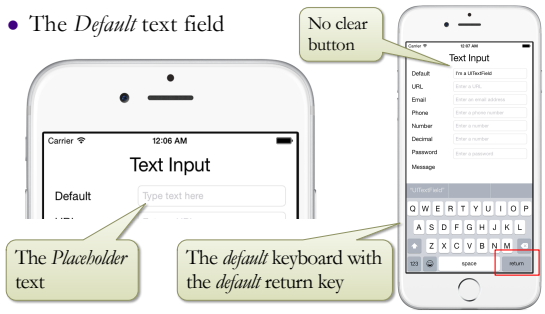


DEPAUL UNIVERSITY 21

21

The Keyboard Attributes

- The *Default* text field



DEPAUL UNIVERSITY 22

22

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 23

23

The Secure Text Input

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

DEPAUL UNIVERSITY 24

24

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

12/13/20 DePaul University 25

25

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 26

26

Dismiss Soft Keyboard

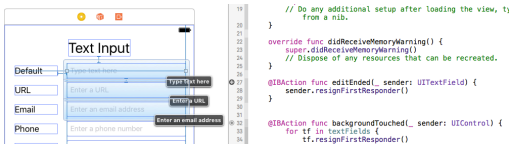
- To dismiss the soft keyboard, stop the text field or text view being the first responder
  - Call the method: `resignFirstResponder`
- When to dismiss the soft keyboard?
  - For text field: when the return key, i.e., the action key, is pressed
- How do we know the return key is pressed?
  - Connect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

DePaul University 27

27

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*"



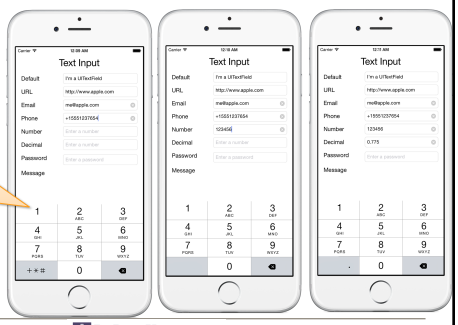

```
@IBAction func editEnded(_ sender: UITextField) {
    sender.resignFirstResponder()
}
```

DePaul University 28

28

The Text Fields – The Number Pads

- *Phone*
- *Number*
- *Decimal*



Where is the return key?

DePaul University 29

30

Dismiss the Number Pads

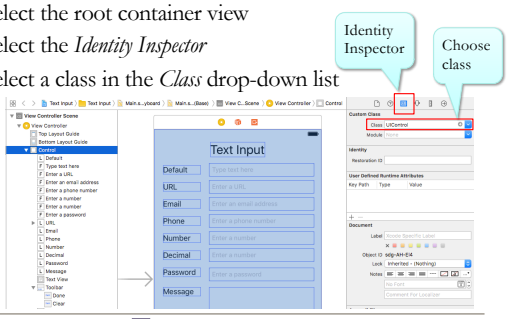
- The number pad has no return or action key
- Solution A:
  - Provide an action key in the app UI
- Solution B:
  - Touch the background, i.e., the container view, to dismiss the keyboard.
  - Change the container view class from *UIView* to *UIControl*
  - Connect an action to the "*Touch Down*" event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

DePaul University 30

31

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 31

32

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

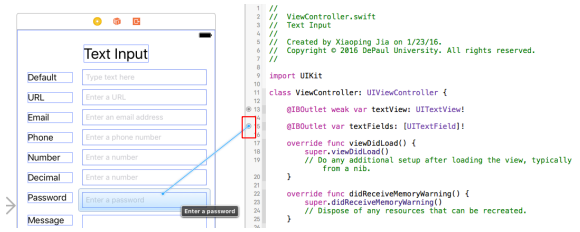
```
@IBOutlet var textFields: [UITextField]!
@IBOutlet weak var textView: UITextView!
```

DePaul University 32

33

Add Views to an Outlet Collection

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



DePaul University 33

34

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.

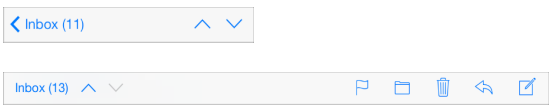
```
@IBAction func backgroundTouched(_ sender: UIControl) {
    for tf in textFields {
        tf.resignFirstResponder()
    }
    textView.resignFirstResponder()
}
```

DePaul University 34

35

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 35

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

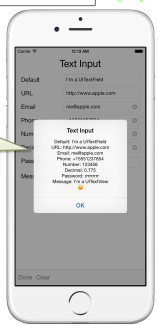

```
@IBAction func clearAction() {
    for tf in textFields {
        tf.text = ""
    }
    textView.text = ""
}
```

DePaul University 36

37

The Popups

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

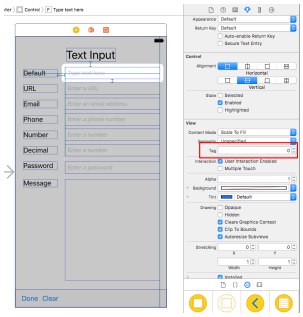


DePaul University 37

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 38

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 39

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 40

41

Action to Display a Popup

```
@IBAction func doneAction() {
    var input : [Int:String] = [:]
    …
    var message = ""
    for (i, label) in labels.enumerated() {
        if let text = input[i] {
            message += "\n\(label): \(text)"
        }
    }
    if let text = textView.text {
        message += "\nMessage: \(text)"
    }
    Display an Alert popup
}
```

DePaul University 41

42

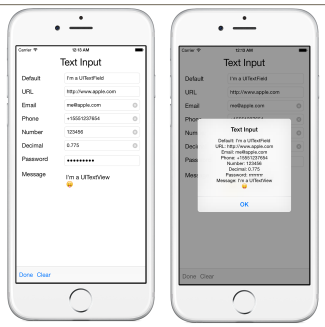
Action to Display a Popup

```
@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, handler: nil)
    alertController.addAction(cancelAction)
    present(alertController, animated: true,
            completion: nil)
}
```

DePaul University 42

43

The Alert Popup



DePaul University 43

44

Sample Code

- Image – Stack – SB.zip
- Controls – Stack – SB.zip
- Text Input – SB.zip

DePaul University 44

45

Next …

- Multiple views
- Segues
- Tabbed views

✧ iOS is a trademark of Apple Inc.

DePaul University 45

46