CSC 471 / 371 Mobile Application Development for iOS

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

1

Outline

- iOS architecture
- App sandbox and permissions
- iOS app lifecycle
- Model-View-Controller architecture
- Target-action pattern
- Outlets and actions
- View controllers

2

The Architecture of iOS

3

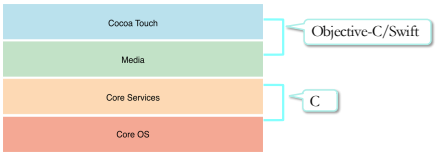
iOS

- A port of Mac macOS for the mobile devices
  - Runs on iPad, iPhone, and iPod touch devices
  - Originally, *iPhone OS*, first released in 2007
  - Related mobile OS: *watchOS*, *tvOS*, both initially released in 2015
- macOS is based on BSD Unix. OS core, *Darwin*, is open source.
- iOS Share many of the frameworks in macOS
  - Darwin, Foundation, Core Graphics, etc.
- Multi-touch UI framework, known as *UI Kit*
  - *Cocoa Touch*, replacing *Cocoa* in macOS
- iOS and macOS apps are not compatible
  - Target different hardware families
  - Share common libraries and components

4

iOS – Layered Architecture

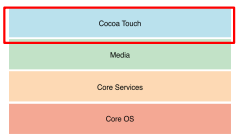
- Four layers
  - Upper layers are object-oriented (Objective-C/Swift)
  - Lower layers, with prefix *Core*, are comprised of functions in C



- All layers are accessible and interoperable with Swift

5

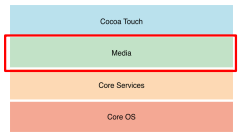
Overview of iOS – Cocoa Touch



- Cocoa Touch (UI)
  - UI widgets
  - Multi-touch events
  - Storyboard
  - Auto layout
  - Notification
  - Camera
  - Accelerometer
  - Map
  - Localization
  - etc.

6

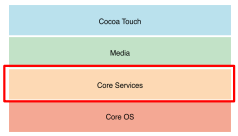
Overview of iOS – Media



- Media
  - Graphics
  - Animation
  - Images
  - Quartz (2D)
  - OpenGL ES
  - Photo library
  - Audio
  - Video
  - etc.

7

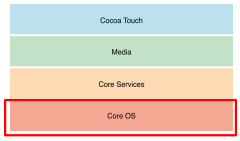
Overview of iOS – Core Services



- Core Services
  - iCould, Pass, Web
  - In-app purchase
  - File access and sharing
  - Multi-threading
  - Networking
  - Core Data and SQLite
  - Foundation
  - Address Book
  - Location, motion
  - etc.

8

Overview of iOS – Core OS



- Core OS
  - macOS Kernel
  - Security
  - Power management
  - Bluetooth
  - Keychain
  - Certificates
  - File system
  - 64-bit support
  - etc.

9

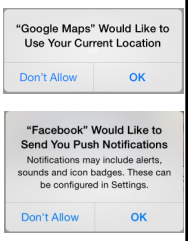
App Sandbox

- iOS apps are all *sandboxed*,
  - To ensure the security of the user data
  - To prevent sharing of data with other applications installed on the same device
  - To minimize the damage of a potential breach
- The sandbox forms a private environment of data and information for each app.
  - Each app has its own directory
  - System directories are not exposed to the apps
  - System manages the locations of the files on behalf of the apps.

10

Permissions

- Shared resources and hardware are available to apps
  - E.g., Contacts, calendars, reminders, photos, geo-location, push notifications, camera, etc.
- Shared resources are accessible through specific APIs
  - No access to raw data or files
- A set of fine-grained permissions control app's access to shared resources and hardware



11

iOS Application Architecture

12

iOS Application

- A program focusing on performing a small set of related, highly cohesive tasks.
  - Small, focused, cohesive
- The smallest unit that can be packaged, installed, and distributed on iOS platforms
  - Presented on the device home screen.
  - Launched by the user
- Packaged into an *Application Bundle*
  - An archive file with suffix: **.app**
  - Include everything needed to run the app

DePaul University 13

13

Characteristics of iOS Apps

- One foreground application at a time
  - Apps remain alive in the background
  - Quick switch between apps, retain resources
- A single window occupied by the foreground app
  - Limited screen size
- Expect quick response time
- Efficient consumption of system resources
  - Memory, battery, bandwidth
  - No *garbage collection*. *Automated Reference Counting* (ARC)

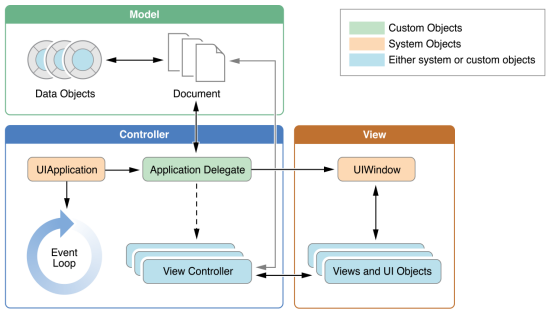
Multi-tasking and split screen in iOS 9

Stay tuned

DePaul University 14

17

The Structure of an iOS App – Model-View-Controller (MVC)



DePaul University 15

18

The Structure of an iOS App

- An app is usually comprised of
  - User Interface (UI) – ***View***
    - Defined in a *storyboard* consists of one or more *scenes*
    - Each scene consists of a hierarchy of *UI widgets*
  - Logic & behavior – ***Controller***
    - Defined in *view controllers*, in Swift (or Objective-C)
    - A view controller typically corresponds to a scene in the storyboard
  - Data – ***Model***
    - Defined in custom classes, in Swift (or Objective-C)

DePaul University 16

19

Model-View-Controller MVC Architecture Pattern

- Principle of *Separation of Concerns*
- Organization of the modules according to their roles and responsibilities:
  - *Models* – responsible for managing data, such as *Product Catalogs, Accounts*
  - *Views* – responsible for visual representations
  - *Controllers* – responsible for interaction and coordination

DePaul University 17

20

Model-View-Controller Separation of Concerns

- Separation of the presentation from the behavior
  - Scenes & widgets: manage the presentation, i.e., the *Views*
  - View controllers: manage the logic &behavior, i.e., the *Controllers*
- Separation of the models from the presentation
  - The *Models* are decoupled from the *Views*
  - The Models can be presented and manipulated in different ways
  - The Models should not hold references to the Views
  - The Models communicate to Views through View Controllers, which act as a mediator,

DePaul University 18

21

The Life of an iOS App

- The life of an iOS app is managed by its hosting environment, i.e., the iOS. **Not the app itself.**
  - Creation, operation, transition between states
  - Termination. An app may be terminated by the system.
- iOS apps are *event-driven*
- iOS apps are notified of and can respond to events through *callback* methods
  - Life cycle events, generated by the system
  - UI events, triggered by the users
  - System events, such as low memory warning

DePaul University 19

22

The Callback Methods

- Callback methods are
  - Implemented in apps, with app specific logic
  - Not invoked within the app that implements the callbacks
  - Invoked externally, typically by the system
- The purposes are
  - Notify the app, i.e., the receiver of the callback, of certain events, e.g., user action, lifecycle events, system events
  - Provide the app an opportunity to react to the events, e.g., to save or restore app state
- Used extensively in iOS apps

DePaul University 20

23

The App Delegate

- Each iOS app has exactly one *app delegate*
  - Generated by the Xcode
    - `AppDelegate.swift`
  - An instance of `UIApplication`
- Represents the customizable portion of the app
  - Allows application specific logic or customizations to be defined here
  - The entry point of the app
  - Handles the initialization of the app
  - Defines the callback methods that respond to application-level life-cycle events

DePaul University 21

24

The View Controllers

- A *scene* is the basic building block in a storyboard.
- A *scene* represents a single screen, or a rectangular region, of UI
- A *scene* consists of a hierarchy of *view* objects
- A *view controller* is associated with each scene
  - A subclass of `UIViewController`
  - Defines the callback methods of the associated scene
  - Manage the presentation of the contents on screen
  - Customizes and/or manages the view objects in the associated scene

DePaul University 22

25

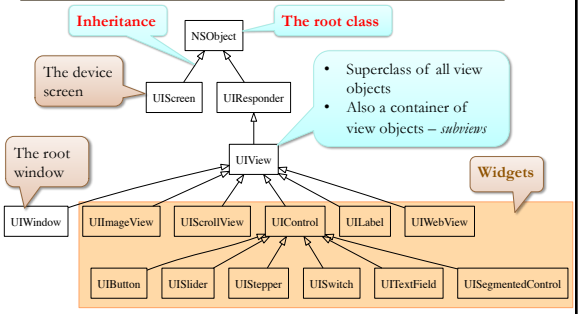
The Views

- The *view* objects, a.k.a. *widgets*, provide the visual appearance of the contents
  - Instances of subclasses of `UIView`
  - Draw contents in rectangle areas
  - Respond to events
- UI Kit provides an extensive set of view objects commonly used by iOS apps
  - Ready to use. **Do not** write your own unless necessary.
  - You may define custom view classes by sub-classing one of the view classes.

DePaul University 23

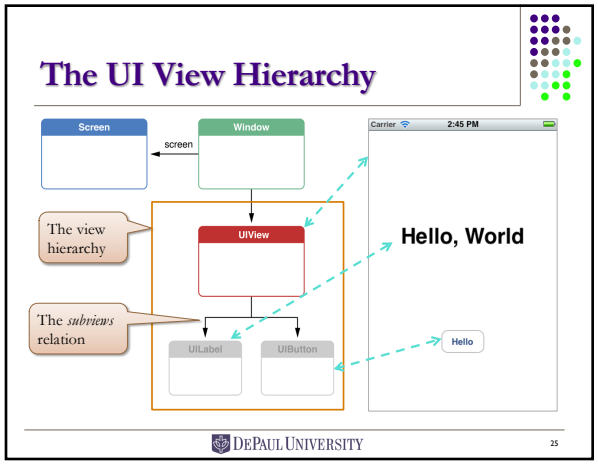
26

Common View Classes in UIKit

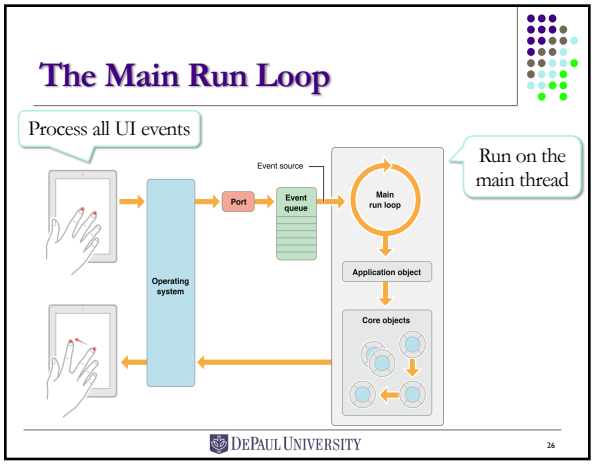


DePaul University 24

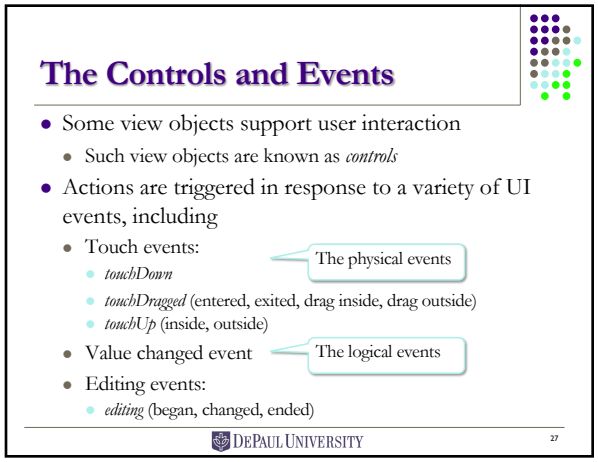
31



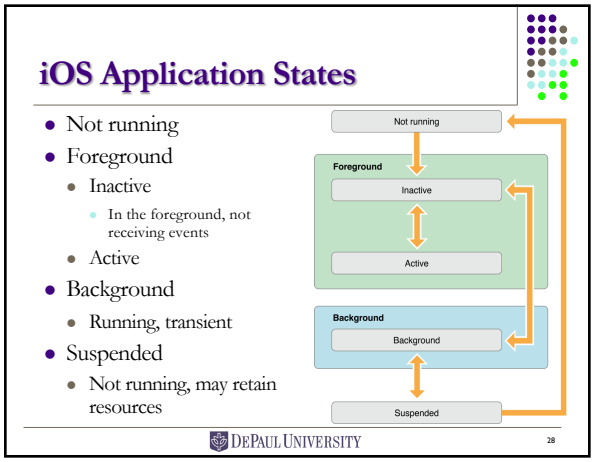
37



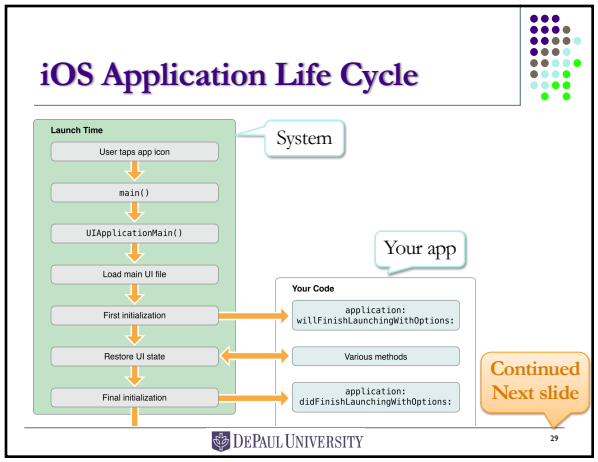
40



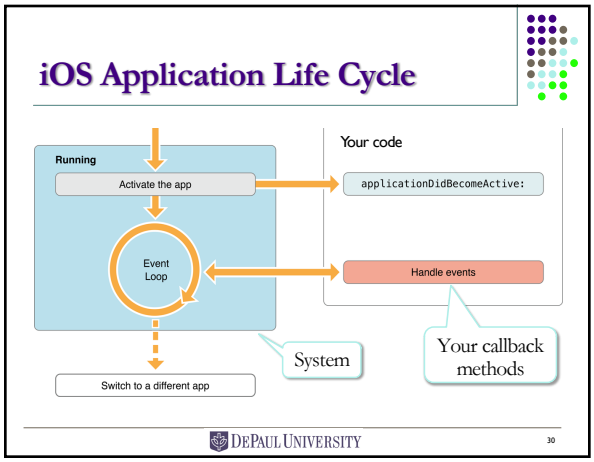
43



44



45



46

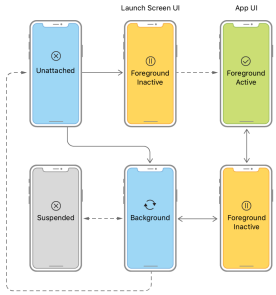
Scenes and Scene Delegate – New in iOS 13

- An app may create multiple copies of the app's UI
  - Each copy of the app's UI is known as a scene
  - User can toggle between scenes using the app switcher
  - On iPad, two scenes of the same app can be displayed side-by-side
- Scene support is optional
  - Apps must explicitly opt in to scenes

DePaul University 31

47

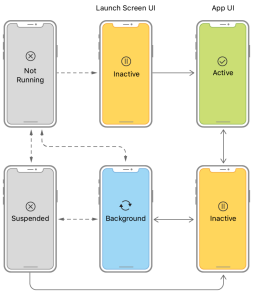
Scene Based Life Cycle – iOS 13



DePaul University 32

48

App Based Life Cycle – iOS 12 or Earlier



DePaul University 33

49

SwiftUI – New in iOS 13

- A new UI framework for iOS, iPadOS, MacOS, WatchOS and tvOS.
  - A Swift DSL for describing UI *declaratively*
- An alternative to UIKit
- Not a replacement of UIKit
- Can integrate with UIKit
- Not supported by iOS 12 or earlier
- Requires Swift 5.1 and Xcode 11

DePaul University 34

50

Building an iOS App

51

Create the User Interface (UI)

- Create *scenes* and *view* hierarchy using the *Interface Builder*
  - Stored in Storyboard or XIB files
- Typically using view classes from the UIKit library
  - Optionally, you may create *custom* view classes by sub-classing existing view or control classes
- View objects in the scenes are initialized by the system
  - Their states can be modified in user code (in *view controllers*)

DePaul University 36

52

Implement the Logic & Behavior

- Implement *view controllers*
  - Each scene in storyboard is associated with a *view controller*
  - Subclass of `UIViewController`
- View controller classes are application specific
  - Written in Swift code (or in Objective-C)
  - Have access to the view objects in the associated scene
  - May modify the states and attributes of the view objects
  - Define callback methods that implement the behaviors and interactions with the view objects

DePaul University 37

53

Connect Scenes and Controllers

- Each scene needs to be connected to its associated controller
  - The *identity* of the view controller
- To access view objects in controller, **you define *outlets*:**
  - A variable declared in the view controller class
  - **Initialized and connected to the view object *by the system***
- To respond to UI events from the view objects, **you define *actions*:**
  - A method defined in the view controller class
  - **Connected to the view object and the event *by the system***
  - **You define the behavior of the action**

In *Interface Builder*

DePaul University 38

56

Define the Outlets

- A special variable declared in the view controller class, using the attribute `@IBOutlet`
  - A reference to a view object in the storyboard/scene
- The *connection* between the outlet and view object **must be explicitly established**
  - The outlet variable will be initialized *by the system*
- Example

```
@IBOutlet weak var label: UILabel!
```

An outlet

A weak reference.

Implicitly unwrapped optional type

In *Interface Builder*

DePaul University 39

62

Swift Attributes

- An attribute begins with `@`

```
@ attribute-name
```

- May appear before any declaration
- Provides additional information about the declaration
- Attributes used by the *Interface Builder* (ignored by the compiler)

```
@IBOutlet
@IBAction
```

Notice the prefix *IB*

DePaul University 40

64

The Target-Action Pattern

- Target-action is a design pattern to deliver a UI *event* and to trigger a response, i.e., an *action*
  - A user triggers an *event* on a view object, known as the *sender*
  - The event is delivered to a designated object, known as the *target*, that handles the event
    - Typically, the target is the view controller associated with the scene containing the view object
  - A method, known as the *action*, of the target is invoked as the response to the event

DePaul University 41

65

Define the Actions

- A special method declared in the view controller class, the *target*, using the attribute `@IBAction`
  - A callback method invoked in response to a UI event
  - The *action* part of the target-action pattern
- The connection among the *sender, event, target,* and *action* **must be explicitly established**
- An action can be connected to multiple senders or events

In *Interface Builder*

DePaul University 42

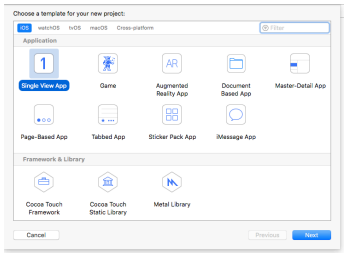
68

Xcode Project Button Demo

69

New Project: Button Demo

- New project
  - iOS
  - Single View App



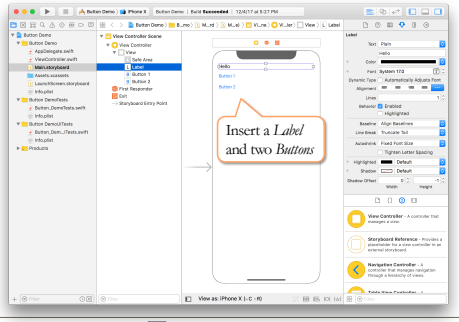
70

Xcode Generated Files

- App & Scene delegate, one for each app
  - `AppDelegate.swift`
  - `SceneDelegate.swift`
  - Leave these alone for simple apps
- Storyboard, user interface, one for each app
  - `Main.storyboard`
- View controllers, one for each screen/scene
  - `*ViewController.swift`
- Single view app template
  - One scene, one view controller

71

Create the UI in Storyboard Interface Builder



73

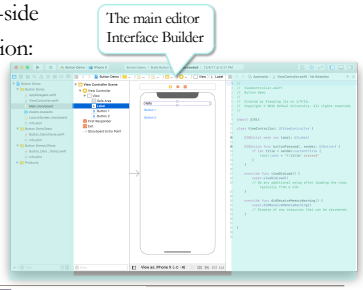
Buttons, Events, Targets, & Actions

- Button – a simple widget that responds to touch events.
  - Simple tap: *Touch Inside Up*
- When a button is touched, it sends an *event*, or *action message*, to the *target object* to trigger an *action*
- The target object is usually the *view controller* that is associated with the view/scene that contains the button, i.e., the widget that receives the touch event
- The action is a *callback method* of the view controller class

74

The Assistant Editor

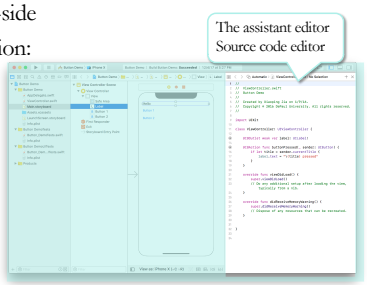
- Xcode allows a main editor and an assistant editor in the editor area, side-by-side
- A useful configuration:
  - Main editor: *the storyboard*



75

The Assistant Editor

- Xcode allows a main editor and an assistant editor in the editor area, side-by-side
- A useful configuration:
  - Main editor: *the storyboard*
  - Assistant editor: *the view controller* associated with the selected scene

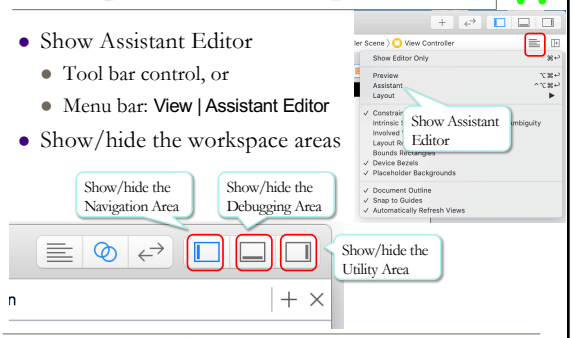


DePaul University 49

76

Manage Xcode Workspace

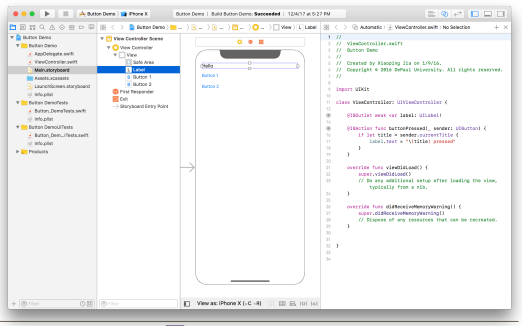
- Show Assistant Editor
  - Tool bar control, or
  - Menu bar: View | Assistant Editor
- Show/hide the workspace areas



DePaul University 50

77

The View & View Controller



DePaul University 51

78

Connect an Outlet

- Ctrl-drag from a view object, the label, to the view controller code

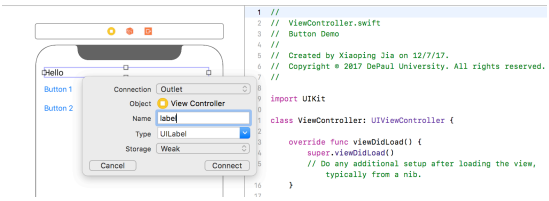
DePaul University 52

79

Outlet Options

- Name: choose a name of the outlet variable
- Type: the declared type of the outlet
- Storage: either *weak* or *strong*

Will discuss later.

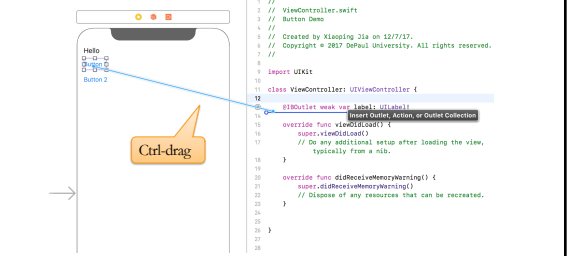


DePaul University 53

80

Connect an Action

- Ctrl-drag from a view object, the first button, to the view controller code

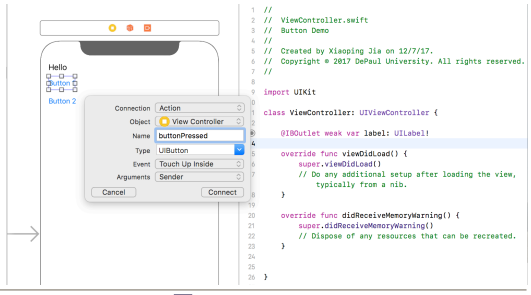


DePaul University 54

81

Action Options

- The initial popup is the same as the one for outlets



DePaul University 55

82

Action Options

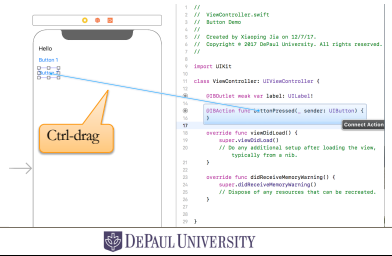
- **Connection type**: *Action,* the default is *Outlet*
- Name: choose a name of the action method
- Type: *Any* or a specific UI type
  - Choose a specific UI type
- Arguments: choose the arguments to the action
  - *None*, *Sender*, *Send* and *Event*
  - Choose *Sender*, the default
- Event
  - Many choices. Choose the default

DePaul University 56

83

Connect Another Action

- You may connect multiple senders to the same action
- Ctrl-drag from the second button to a location over the action method in the view controller



DePaul University 57

84

Options for IBAction Declaration

- An action can be connected to multiple view objects, i.e., senders, or events
- The action method can be declared in one of the following forms:

For one view object and one event

```
@IBAction func action()
```

Most common. For one or more view objects and the same event.

```
@IBAction func action(_ sender: UIButton)

@IBAction func action(_ sender: UIButton,
                      forEvent event: UIEvent)
```

For one or more view objects and one or more different events

DePaul University 58

88

Connection Indicators

- A filled circle indicates a valid connection

Valid connections

```
class ViewController: UIViewController {

    @IBOutlet weak var label: UILabel!

    @IBAction func buttonPressed(_ sender: UIButton) {
    }
```

- An empty circle indicates a broken connection

Broken connections

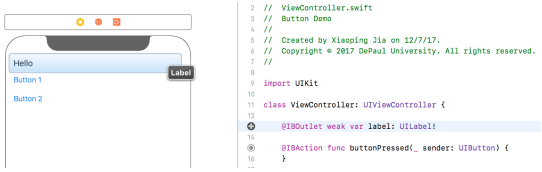
```
class ViewController: UIViewController {

    @IBOutlet weak var label: UILabel!

    @IBAction func buttonPressed(_ sender: UIButton) {
    }
```

DePaul University 59

90

Connection Indicator

- Hover over the connection indicator of the outlet *label*
  - The corresponding view object is highlighted



DePaul University 60

91

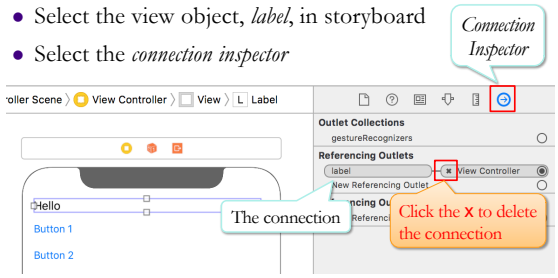
Connection Indicator

- Hover over the connection indicator of the action *buttonPressed*
  - The corresponding view objects are highlighted

92

The Connection Inspector – For an Outlet

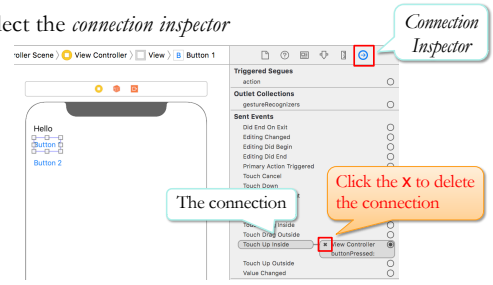
- Select the view object, *label*, in storyboard
- Select the *connection inspector*



95

The Connection Inspector – For an Action

- Select the view object, *Button 1*, in storyboard
- Select the *connection inspector*



98

Broken Connections

- Broken connections of outlets or actions will cause the app to crash *at run time*!
- Fix the broken connection problems
  - The connection must be deleted, if the outlet or action is deleted in code.
  - The connection must be deleted and reestablished if the outlet or action is renamed.
  - Adding new connections to the same object does not automatically remove old broken connections.

99

The View Controller

```
import UIKit
class ViewController: UIViewController {

    @IBOutlet weak var label: UILabel!

    @IBAction func buttonPressed(_ sender: UIButton) {
        if let title = sender.currentTitle {
            label.text = "\(title) pressed"
        }
    }

    override func viewDidLoad() {
        super.viewDidLoad()
    }
}
```

Import framework

Superclass

100

The View Controller

```
import UIKit
class ViewController: UIViewController {

    @IBOutlet weak var label: UILabel!

    @IBAction func buttonPressed(_ sender: UIButton) {
        if let title = sender.currentTitle {
            label.text = "\(title) pressed"
        }
    }

    override func viewDidLoad() {
        super.viewDidLoad()
    }
}
```

An outlet

101

The View Controller

```
import UIKit
class ViewController: UIViewController {

    @IBOutlet weak var label: UILabel!

    @IBAction func buttonPressed(_ sender: UIButton) {
        if let title = sender.currentTitle {
            label.text = "\(title) pressed"
        }
    }

    override func viewDidLoad() {
        super.viewDidLoad()
    }
}
```

An action

Can we simply use `sender.currentTitle` here?

DePaul University 67

103

The View Controller

```
import UIKit
class ViewController: UIViewController {

    @IBOutlet weak var label: UILabel!

    @IBAction func buttonPressed(_ sender: UIButton) {
        if let title = sender.currentTitle {
            label.text = "\(title) pressed"
        }
    }

    override func viewDidLoad() {
        super.viewDidLoad()
    }
}
```

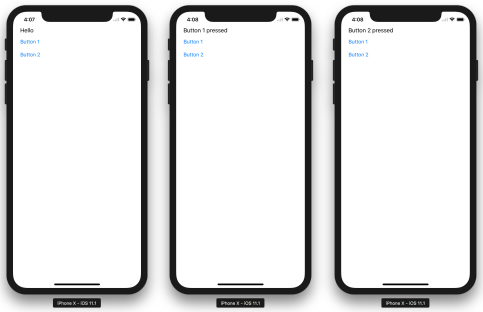
Override superclass method

Lifecycle callback

DePaul University 68

104

Run Button Demo



DePaul University 69

105

Demo

- Only way to understand is to repeat again and again …
- Source code of demo project
  - *Buttons – SB.zip*

DePaul University 70

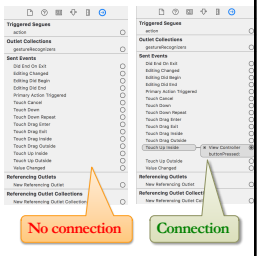
106

Common Issues and Fixes

107

A Widget Does Not Respond

- Cause:
  - The non-responding widget is not connected to an action
- Fix:
  - In the Storyboard, select the widget in question
  - Open the *Connection Inspector*
    - Verify no connection
  - Connect to a new action



DePaul University 72

108

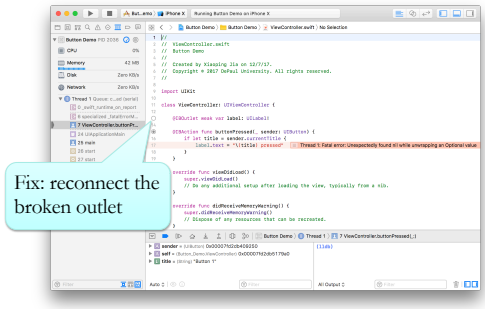
App Crashes When You Touch a Widget

- Check the messages in the console
  - fatal error: unexpectedly found nil while unwrapping an Optional value
    - ➔Broken outlet connection
  - unrecognized selector sent to instance
    - ➔Broken connection to an action,
      - Action method name changed
      - Action method deleted

DEPAUL UNIVERSITY 73

109

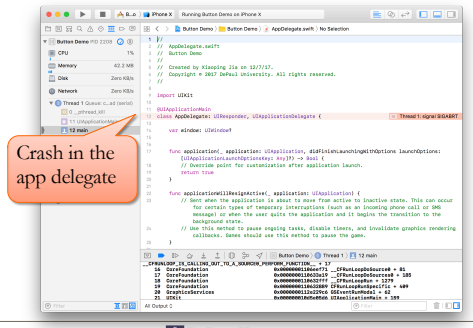
Broken Outlet Connection



DEPAUL UNIVERSITY 74

110

Broken Action

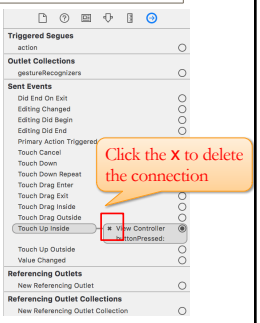


DEPAUL UNIVERSITY 75

111

Broken Action

- Fix:
  - In the Storyboard, select the widget that caused the crash
  - Use the *Connection Inspector* to delete the connection
  - Then reconnect the widget to an action



DEPAUL UNIVERSITY 76

112

Next …

- More UI views and controls
- Images and scroll views
- Switches, sliders, segmented controls, steppers
- Text input
- Auto layout

✧ iOS is a trademark of Apple Inc.

DEPAUL UNIVERSITY 77

113