Notifications

Dr. Sarah Abraham

University of Texas at Austin
CS329e
Spring 2020

Notifications

- Provide information to user based on time or location
- Sent internally within app (local) or externally (remote)
- App determines schedule, system handles delivery

Types of Notifications

- Types of notifications in iOS apps:
    - KVO (Observer pattern)
    - Basic notifications
    - Remote notifications
    - Scheduled local notifications
    - Active notifications

KVO

- Key Value Observing
- Allows objects to be notified of changes to specific properties of other objects
    1. Make property dynamic
    2. Add observer for any property to be monitored
    3. Implement `observeValueforKeyPath` method
    4. Remove observer in `deinit`
- Key-value observing works for any class that inherits from NSObject

Notifications and KVO

- Notifications are an implementation of the *observer* design pattern
  - Same general idea as event-driven and MVC patterns
- Object maintains list of observers and notifies them when event they're registered to receive occurs
- Used to implement distributed event-handling

Property Setup for Monitoring

```
class ObjectToObserve: NSObject {
    dynamic var myValue = “Initial value”
    func updateProperty(String newValue) {
        myValue = newValue
    }
}
```

Observer Setup

```
private var myContext = 0

class MyObserver: NSObject {

    var objectToObserve = ObjectToObserve()

   override init() {

        super.init()

        objectToObserve.addObserver(self,
        forKeyPath: "myValue", options: .new,
        context: &myContext)

   }

   deinit {

        objectToObserve.removeObserver(self,
        forKeyPath: "myValue", context:
        &myContext)

   }
```

```
override func
observeValueForKeyPath(keyPath: String?,
ofObject: AnyObject?, change: [String :
AnyObject]?, context:
UnsafeMutablePointer<Void>) {

    if context == &myContext {

        /* Handle changes in value here */

    } else {

      //Pass along other changes in value

      super.observeValueForKeyPath(keyPath,
      ofObject: object, change: change,
      context: context)

    }

}
```

```
}
```

Basic Notifications

- Use `NSNotificationCenter` framework
  - Singleton like `NSUserDefaults`
  - Communication tool internal to app
- Notify other parts of application that something has occurred
- Notification-handling is synchronous
  - All observers receive and process their notifications before `postNotification` returns

Creating Basic Notifications

1. Add observer
2. Implement notification handling
3. Issue post notification

Add Observer

- Register observer for notification of event
- Usually called during view setup (`viewDidLoad`) in class that needs event notification
- Notification key is constant that is broadcasted to listeners

```
NSNotificationCenter.defaultCenter().addObserver(self,
selector: #selector(eventNotificationHandler), name:
eventHappenedNotificationKey, object: nil)
```

Implement Handler

- Method called when notification is posted
- Must be registered with the observer object

```
func eventNotificationHandler(notification:
NSNotification) { /* process event here */ }
```

Issue Post Notification

- Broadcasts to all observers that listen for that key
- Necessary data passed through userInfo argument

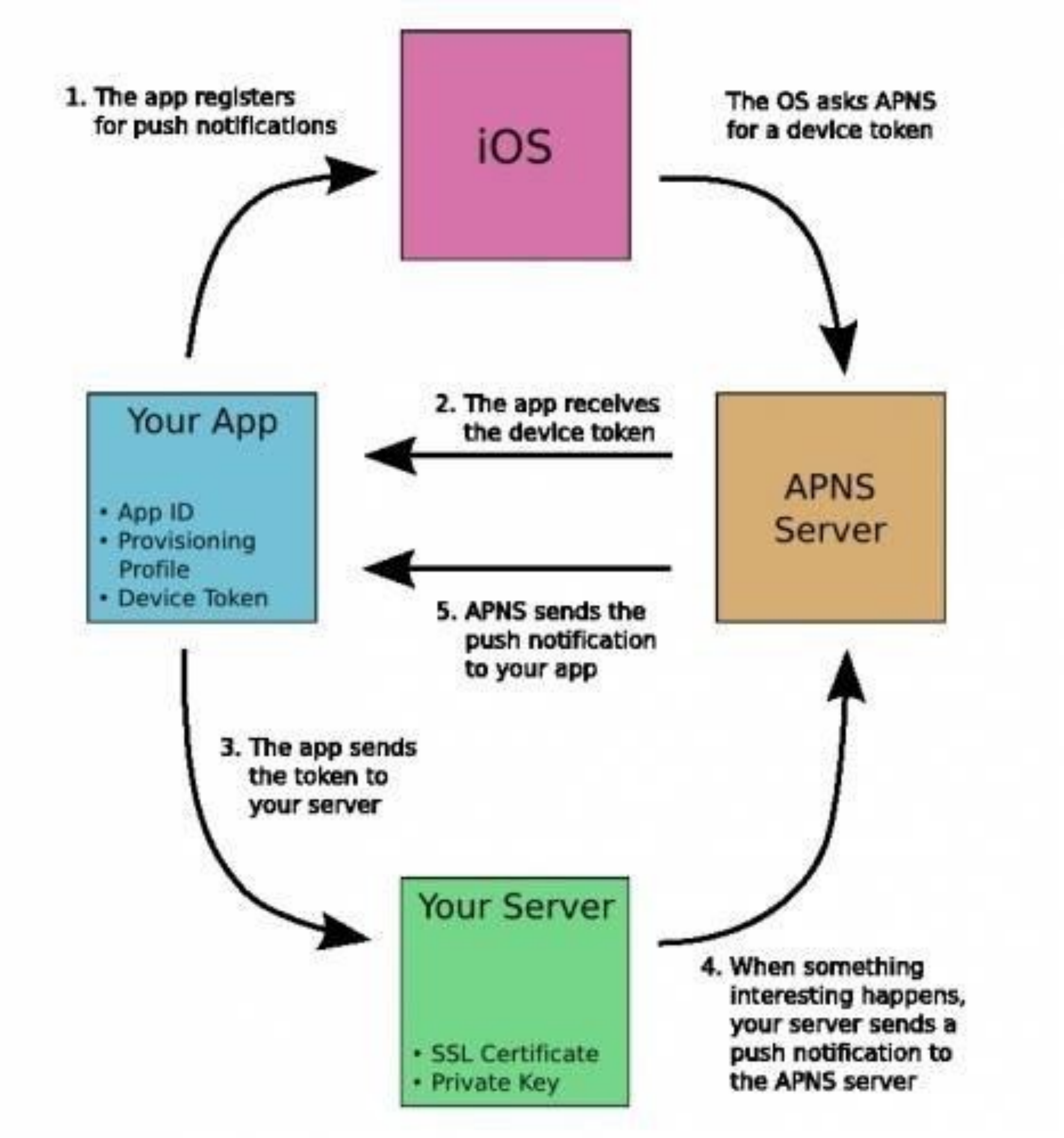
```
NSNotificationCenter.defaultCenter().postNotificationName
(eventHappenedNotificationKey, object: nil, userInfo:nil)
```

Remote Notifications

- Generated outside of application
- Sent through APNS (Apple Push Notification Server)
- Remote notifications displayed within pop-down view
- Touching a remote notification launches associated app

Push Notification Flow

- Requires app's server to connect to APNS server to generate notification


1. The app registers for push notifications
iOS
The OS asks APNS for a device token
Your App
• App ID
• Provisioning Profile
• Device Token
2. The app receives the device token
APNS Server
5. APNS sends the push notification to your app
3. The app sends the token to your server
Your Server
• SSL Certificate
• Private Key
4. When something interesting happens, your server sends a push notification to the APNS server

Scheduled Local Notifications

- Notifications sent to app at specific time
- Scheduled in the operating system
- App does not have to be running to receive scheduled notifications

Scheduled Local Notification Flow

1. `registerUserNotificationsSettings` registers to receive notifications (called in `didFinishLaunchingWithOptions`)

2. `scheduleLocalNotification` defines one or more notifications and schedules them for delivery

3. Implement `didReceiveLocalNotification` in app delegate to process deliveries

Remote Versus Local?

- What situations make sense to use remote notifications?
- What situations make sense to use scheduled local notifications?

Active Notifications

- Allows for more interactivity with notifications
    - Respond to notifications directly from the banner
- Lessens disruption to current application
- Delivery mechanism is the same as other notifications

Instapoll Question: Notifications

- Which of these notifications requires an external server?
  - Key Value Observers
  - Basic Notifications
  - Push Notifications
  - Scheduled Local Notifications
  - Active Notifications

Selectors

- Name that identifies a method
- Used to select and execute this method at runtime
    - Dynamic function pointer
- Can choose appropriate method at runtime based on class
    - Subclass implementations might be different but same call can be issued
- Compiler ensures selector names are unique

Using Selectors

- Where have we seen selectors?
  - Notifications, Timers, Bar Button Items etc

```
NSTimer.scheduledTimerWithTimeInterval(1.,
target: self, selector:
#selector(timedMethod), userInfo: nil,
repeats: true)
```

When to Use Selectors

- Allows custom creation of callback functionality
- Some functionality explicitly requires selectors
    - NSNotification
    - NSTimer
- Implicitly happening every time you create a widget in Interface Builder!

```
button.addTarget(self, action: #selector(buttonAction),
forControlEvents: .TouchUpInside)
```

Remember Blocks/Closures?

- Small, self-contained pieces of code
- Encapsulate units of work to execute concurrently
    - Utilizes multiple cores in the device
- Code written at point of invocation, but executed later in context of method implementation
- All local variables available (unlike callbacks)
- Code that executes is connected to code that calls it (unlike callbacks)

Using Closures

- Closure defined within bracket {} syntax

```
let divide = { (dividend: Int, divisor:
Int) -> Int in

    return dividend/divisor

}

let quotient = divide(10, 2)
```