iOS Dev Accelerator
Week 5 Day 2

- Notification Center
- Map Overlays
- NSDate and NSDateFormatter

Notification Center

Broadcast Model

- Normally one object fires a method on another object. This is how information gets passed in your app.
- But this requires that one object knows who the receiver of the method is, and what methods that receiver implements.
- This effectively couples the two objects together that otherwise may come from independent areas of your app.
- To avoid this scenario, a broadcast model can be introduced using the notification pattern.
- An object posts a notification, which is dispatched to the appropriate observers through NSNotificationCenter.

Notification Pattern

- The notification pattern is used to pass around information related to the occurrence of events.
- A notification encapsulates information about an event, such as a network connection closing or a button being pressed.
- Objects that need to know about an event register with the notification center that it wants to be notified when that event takes place. This object is considered an observer.

Notification Pattern vs Delegation

- Notification pattern is similar to delegation, except for:
    - Any number of objects can receive a notification, where delegation is is a one-to-one relationship.
    - An object may receive any message from the notification center, not just the predefined delegate methods.
    - The object posting the notification does not have to know the observer exists.

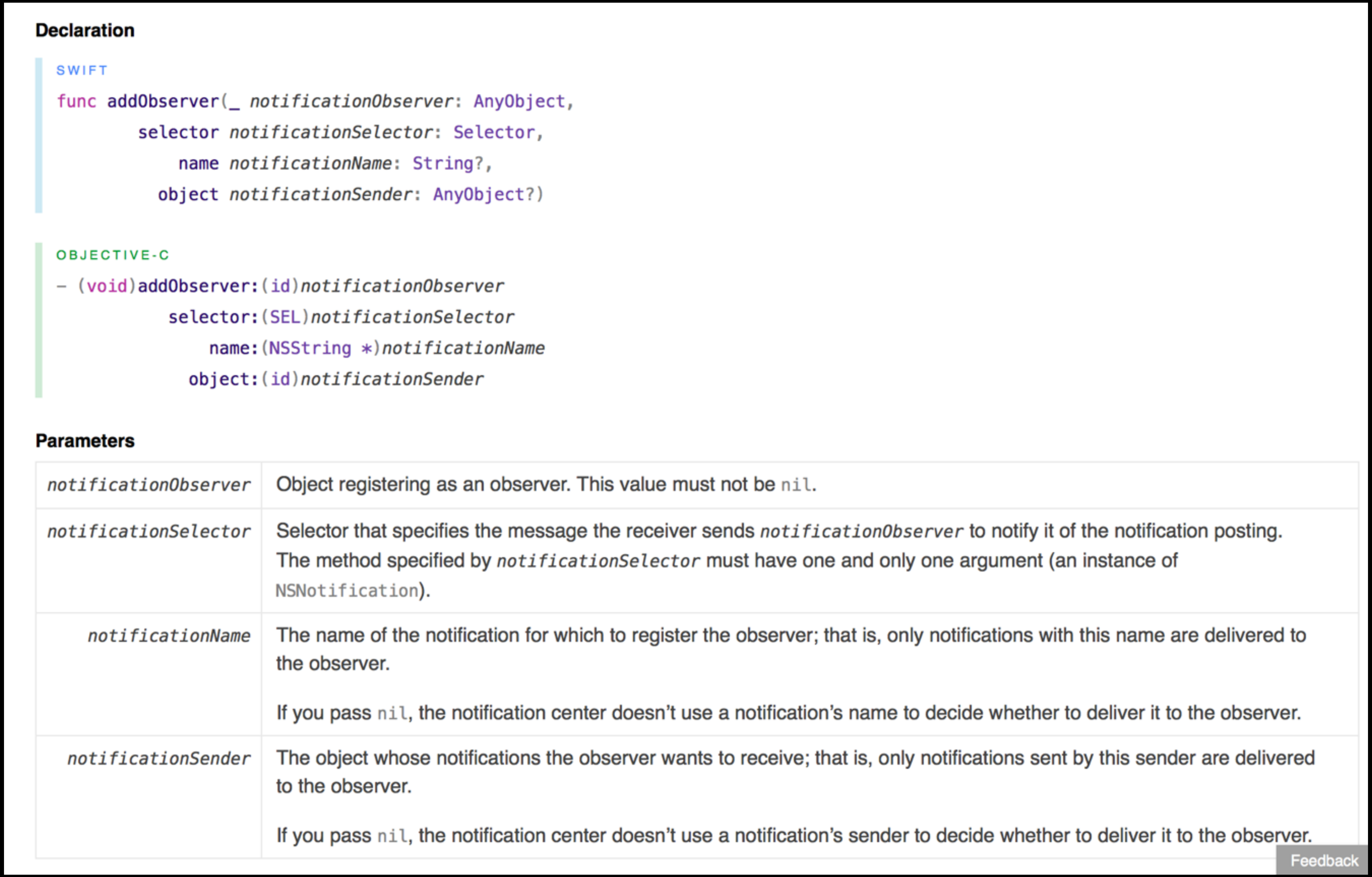
Notification Center

- A notification center manages the sending and receiving of notifications.
- Each program, or app, has a default notification center you can access with NSotificationCenter.defaultCenter(). Almost always you will be posting notifications to this center.
- A notification center delivers notifications synchronously. To send them asynchronously you will have to use a notification queue.

Registering for Notifications

- You register an object to receive a notification by invoking the notification center method addObserver:selector:name:object
- Name and object are both optional.
- If you only specify an object, the observer will get all notifications containing that object aka were sent by this object
- If you only specify a name, the observer will receive that notification every time its posted, regardless of the object associated with it.

Registering for Notifications



Declaration

SWIFT

```
func addObserver(_ notificationObserver: AnyObject,
        selector notificationSelector: Selector,
            name notificationName: String?,
          object notificationSender: AnyObject?)
```

OBJECTIVE-C

```
- (void)addObserver:(id)notificationObserver
           selector:(SEL)notificationSelector
               name:(NSString *)notificationName
             object:(id)notificationSender
```

Parameters

notificationObserver	Object registering as an observer. This value must not be `nil`.
notificationSelector	Selector that specifies the message the receiver sends *notificationObserver* to notify it of the notification posting. The method specified by *notificationSelector* must have one and only one argument (an instance of `NSNotification`).
notificationName	The name of the notification for which to register the observer; that is, only notifications with this name are delivered to the observer. If you pass `nil`, the notification center doesn't use a notification's name to decide whether to deliver it to the observer.
notificationSender	The object whose notifications the observer wants to receive; that is, only notifications sent by this sender are delivered to the observer. If you pass `nil`, the notification center doesn't use a notification's sender to decide whether to deliver it to the observer.

Unregistering for Notifications

- Before an object that is observing is deallocated, **it must tell the notification center to stop sending it notifications.**
- If you don't unregister, the next notification gets sent to a nonexistent object and your app will crash :(
- The notification center keeps weak references to all of the observers, so it doesn’t ‘own’ the observers. So it doesn't care if those observers are still alive or not. It just keeps firing notifications until they unregister.

Unregistering for Notifications

- 2 methods for unregistering:
  - removeObserver: - removes the passed in object from all notifications
  - removeObserver:name:object: - removes only the matching entries from the notification center’s dispatch table.

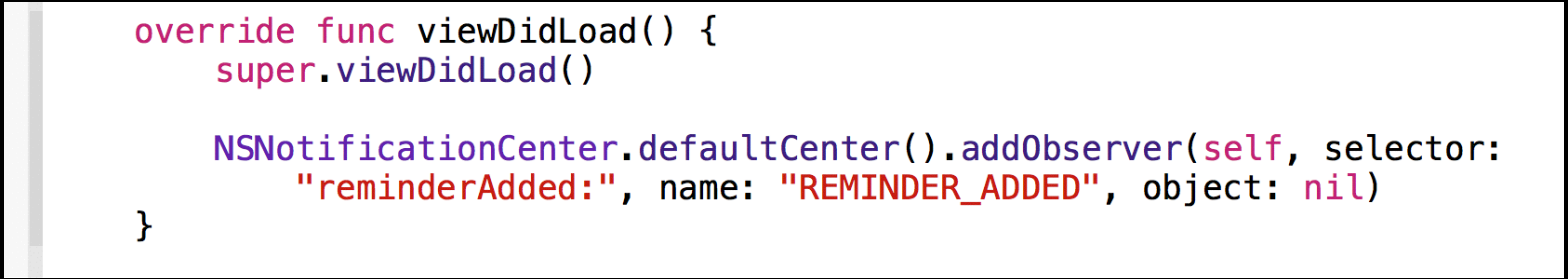
Posting Notifications

- Posting notifications is as simple as calling postNotificationName:Object: or postNotificationName:object:userInfo: on the default notification center.
- Notifications are immutable, so once they are created they cannot be modified.
- **If the observer will need some object related to the event, you can pass that object in the userInfo dictionary.**

Receiving the notifications

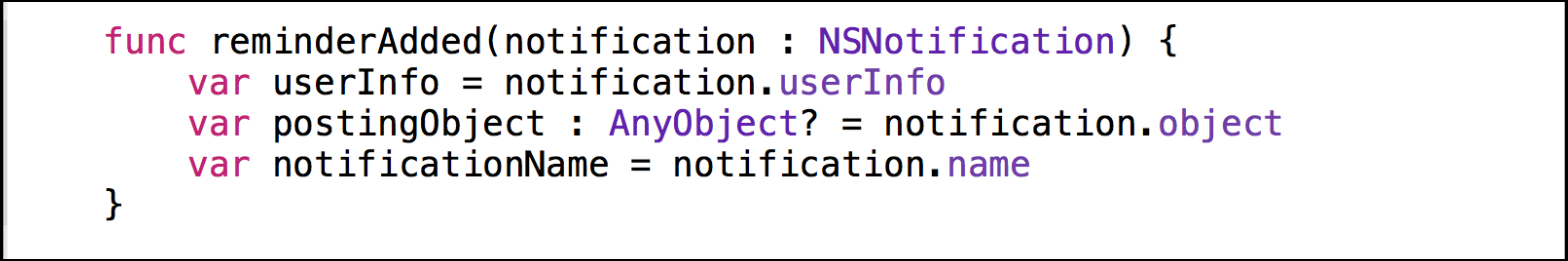
- When you register for a notification you pass in a selector, aka a method, that you want to fire when that notification is received.
- This selector can only have at most one parameter, which is the notification object itself that was posted.
- This notification has properties: to the object that originally fired the notification, the name of the notification, and the userInfo object.

Receiving the notifications



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

    NSNotificationCenter.defaultCenter().addObserver(self, selector:
        "reminderAdded:", name: "REMINDER_ADDED", object: nil)
}
```



```swift
func reminderAdded(notification : NSNotification) {
    var userInfo = notification.userInfo
    var postingObject : AnyObject? = notification.object
    var notificationName = notification.name
}
```

Demo

Map Overlays

Map Overlays

- “Overlays offer a way to layer content over an arbitrary region of the map”
- Overlays are usually defined by multiple coordinates, which is different from the single point of an annotation.
- Overlays can be contiguous or noncontiguous lines, rectangles, circles, and other shapes.
- Those shapes can then be filled and stroked with color.

Overlay overview

- Getting an overlay onscreen involves two different objects working together:
    - An overlay object, which is an instance of a class that conforms to the MKOverlay protocol. This object manages the data points of the overlay.
    - An overlay renderer, which is a subclass of MKOverlayRenderer that does the actual drawing of the overlay onto the map.

Overlay workflow

1. Create an overlay object and add it to the map view
2. return an overlay renderer in the delegate method mapView:rendererForOverlay:

Overlay objects

- Overlay objects are typically small data objects that simply store the points that define what the overlay should represent and other attributes.
- MapKit defines several concrete objects you can use for your overlay objects if you want to display a standard shaped overlay.
- Otherwise, you can use any class as an overlay object as long as it conforms to the MKOverlay protocol.
- The map view keeps a reference to all overlay objects and uses the data contained in those objects to figure out when it should be displaying those overlays.

Overlay objects

- Concrete objects: MKCircle, MKPolygon, MKPolyline
- MKTileOverlay: use if your overlay can be represented by a series of bitmap tiles
- Subclass MKShape or MKMultiPoint for custom shapes
- Use your own objects with the MKOverlay protocol

`OverlayRenderer`

- MapKit provides many standard overlay renderers for standard shapes.
- You don't add the renderer directly to the map, instead the delegate object provides an overlay renderer when the map view asks for one.

OverlayRenderer Objects

- Standard shapes: MKCircleRenderer, MKPolygonRenderer, MKPolylineRenderer.
- Tiled: MKTileOverlayRenderer
- Custom shapes : MKOverlayPathRenderer
- For the most customization, subclass MKOverlayRenderer and implement your custom drawing code.

Demo

NSDate & NSDateFormatter

Dates & Time

- There are 3 primary classes you will use when representing time in your apps:
    - NSDate - a representation of an exact point in time
    - NSCalendar - a representation of a particular calendar type (Buddhist, Chinese, Hebrew, etc)
    - NSDateComponents - a representation of a particular part of a date, like an hour, minute, day, year, etc.

Dates

- NSDate represents a single point in time.
- Date objects are immutable
- The standard unit of time for date objects is floating point value typed as NSTimeInterval expressed in seconds.
- NSDate computes times as seconds relative to an absolute reference time: the first instant of January 1, 2001, GMT.
- Dates before that time are stored as negative values. Dates after stored as positive.

Creating Date Objects

- To get a date that represents the current time:
  1. just init a NSDate instance OR
  2. use the class method .date on NSDate
- To create a specific date:
  1. Use one of NSDate's initWithTimeInterval init methods
  2. Use an NSCalendar and date components

Demo

Basic Date Calculations

- There are many ways to compare dates:
  - isEqualToDate:
  - compare:
  - laterDate:
  - earlierDate:
- and you can also use the method timeIntervalSinceDate: on NSDate to get the time interval from 2 dates

Demo

Calendars

- You use calendar objects to convert between times and date components (years, days, minutes, etc)
- NSCalendar provides an implementation for many different types of calendars.
- Calendar types are specified by constants in NSLocale.
- the method currentCalendar returns the user’s preferred locale.

Date Components

- You can represent a component of a date using the NSDateComponent class.
- Use methods setDay:, setMonth:, and setYear: to set those individual components.

Putting it all together: Using Dates, Date Components, and Calendars.

- You can grab the components of a date using the method components:fromDate: on the class NSCalendar:
    - The components parameter actually takes in a bit mask composed of Calendar Unit constants.
- You can create a date from components by creating a component, a calendar, and then running dateFromComponents: on the calendar.

Demo

Date Calculations

- Use the dateByAddingComponents:toDate:options: on a calendar to add components (hours, minutes, days,years) to an existing date. They can be negative values to take time away as well.

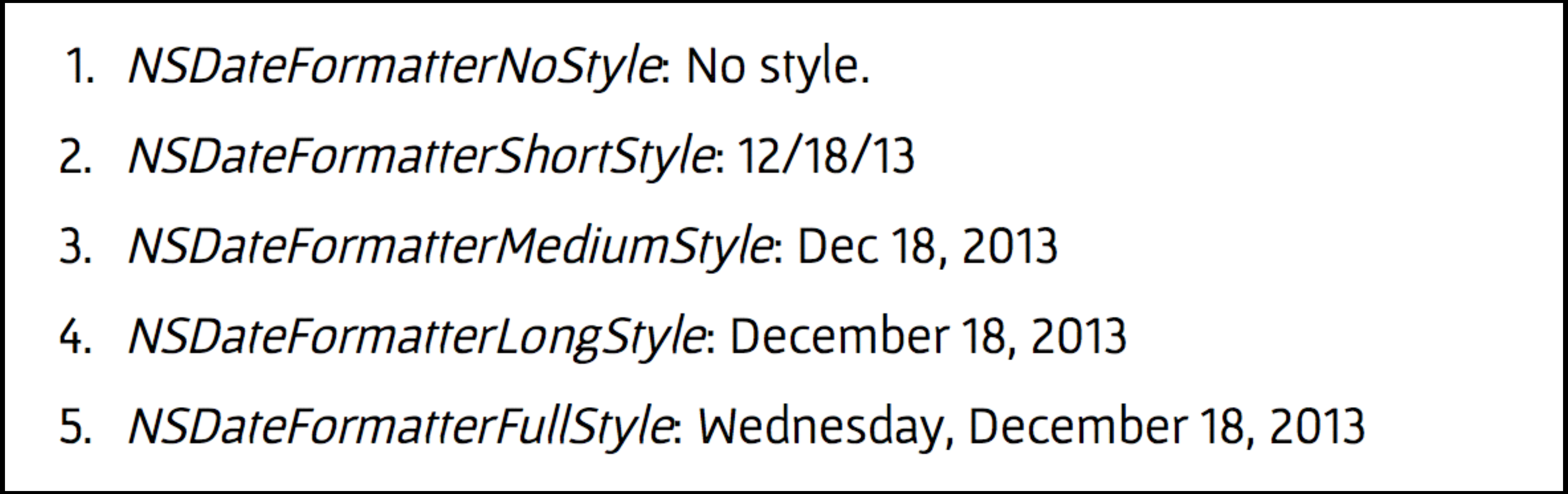
Demo

NSDateFormatter

- NSDateFormatter is a subclass of NSFormatter, which has a wide range of specialized subclasses:
    - NSNumberFormatter
    - NSDateFormatter
    - NSByteCountFormatter
    - NSDateFormatter
    - NSDateComponentsFormatter
    - NSDateIntervalFormatter
    - NSEnergyFormatter
    - NSMassFormatter
    - NSLengthFormatter
    - MKDistanceFormatter

NSDateFormatter

- You use an NSDateFormatter to get text representations of both dates and times
- You can set the NSDateFormatter's style to set the desired style of date:

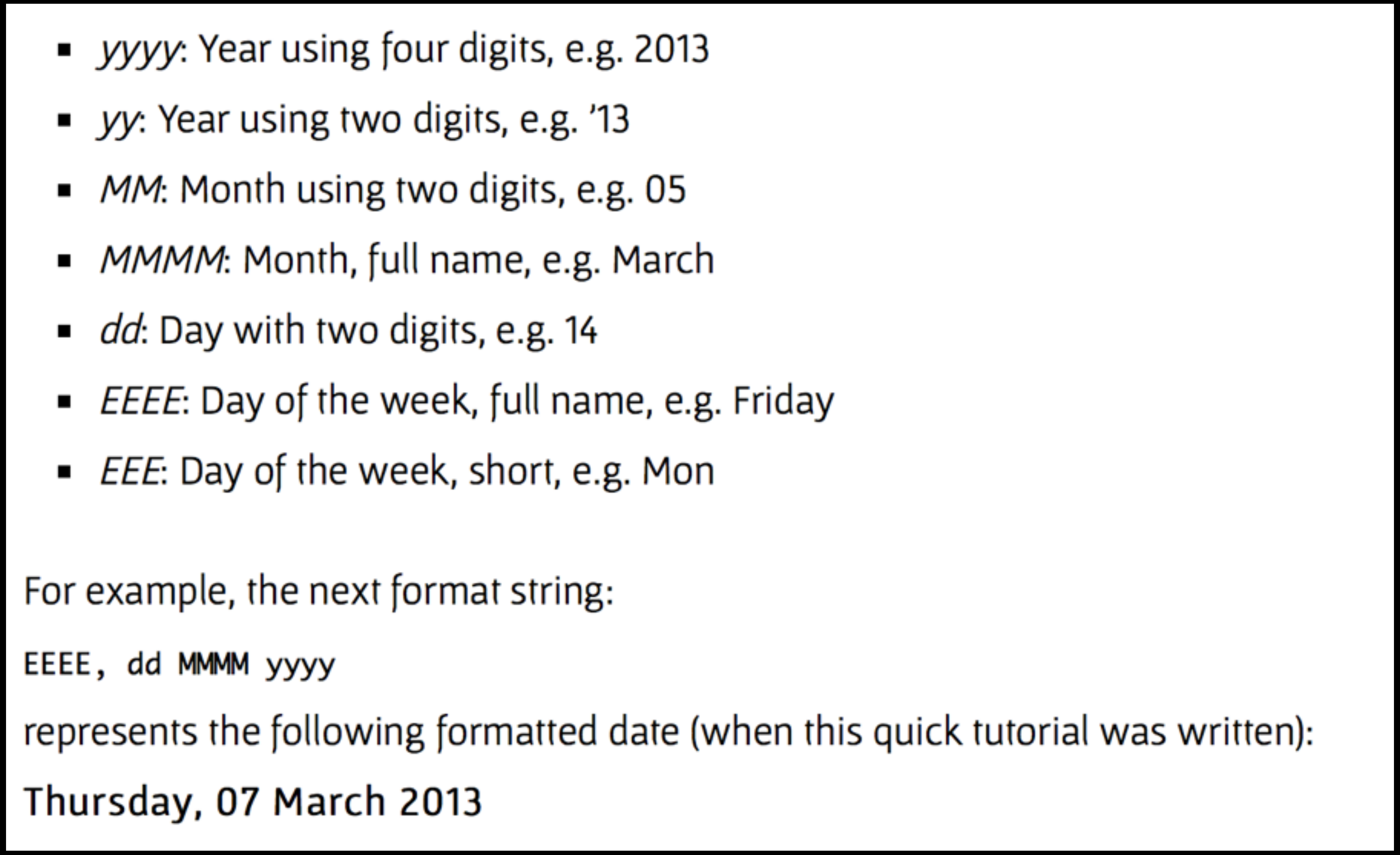


1. *NSDateFormatterNoStyle*: No style.
2. *NSDateFormatterShortStyle*: 12/18/13
3. *NSDateFormatterMediumStyle*: Dec 18, 2013
4. *NSDateFormatterLongStyle*: December 18, 2013
5. *NSDateFormatterFullStyle*: Wednesday, December 18, 2013

*sourced from http://gtiapps.com/?p=1086

Custom style

- If none of the pre-baked styles fit your needs, you can create a custom date format using the setDateFormat: method.
- This takes in a format string that follows a specific pattern:



- *yyyy*: Year using four digits, e.g. 2013
- *yy*: Year using two digits, e.g. ’13
- *MM*: Month using two digits, e.g. 05
- *MMMM*: Month, full name, e.g. March
- *dd*: Day with two digits, e.g. 14
- *EEEE*: Day of the week, full name, e.g. Friday
- *EEE*: Day of the week, short, e.g. Mon

For example, the next format string:

```
EEEE, dd MMMM yyyy
```

represents the following formatted date (when this quick tutorial was written):

Thursday, 07 March 2013

*sourced from http://gtiapps.com/?p=1086

Demo