Developing Applications for iOS

Lecture 9:
Device Capabilities

Prof. Dr. Radu Ionescu
raducu.ionescu@gmail.com
Faculty of Mathematics and Computer Science
University of Bucharest
Content

- Core Location: GPS + Compass
- Accelerometer
- Map Kit
Core Location

Framework for managing location and heading

- No user-interface.

Basic object is CLLocation

- It has many properties: coordinate, altitude, floor, speed, horizontal/verticalAccuracy, timestamp, course.

- Where (approximately) is this location?

```swift
var coordinate: CLLocationCoordinate2D { get }
struct CLLocationCoordinate2D {
    var latitude: CLLocationDegrees // a double
    var longitude: CLLocationDegrees // a double
}
var altitude: CLLocationDistance { get }
```

A negative value means “below sea level”.
Core Location

- How close to that latitude/longitude is the actual location?

```swift
var horizontalAccuracy: CLLocationAccuracy { get }
var verticalAccuracy: CLLocationAccuracy { get }
```

- Both are measured in meters. A negative value means the coordinate or altitude (respectively) is invalid.

- The accuracy depends on the hardware. You can specify the desired accuracy of the device location:

  ```swift
  kCLLocationAccuracyBestForNavigation
  kCLLocationAccuracyBest
  kCLLocationAccuracyNearestTenMeters
  kCLLocationAccuracyHundredMeters
  kCLLocationAccuracyKilometer
  kCLLocationAccuracyThreeKilometers
  ```

- The phone should be plugged in to power source when the desired accuracy is `kCLLocationAccuracyBestForNavigation`.

- The more accuracy you request, the more battery will be used.
Core Location

The iDevice does its best given a specified accuracy request

- GPS (very accurate, lots of power).
- Wi-Fi node database lookup (more accurate, more power).
- Cellular tower triangulation (not very accurate, but low power).

Speed

```swift
var speed: CLLocationSpeed { get }
```

- Measured in meters/second.
- Note that the speed is instantaneous (not average speed).
- Generally it’s useful as “advisory information” when you are in a vehicle.
- A negative value means “speed is invalid”.

Core Location

Course

var course: CLLocationDirection { get }

- Values are measured in degrees starting at due north and continuing clockwise around the compass. Thus, North is 0 degrees, East is 90 degrees, and so on.
- Not all devices can deliver this information. A negative value means “direction is invalid”.

Time Stamp

var timestamp: Date { get }

- Pay attention to these since locations will be delivered on an inconsistent time basis.

Distance (in meters) between CLLocations

func distance(from location: CLLocation) -> CLLocationDistance
Core Location

How do you get a CLLocation?

- Always from a CLLocationManager (sent to you via its delegate) when you are interested in the device location.
- Can also use initializer when you are interested in a different location:

  ```swift
  init(latitude: CLLocationDegrees, longitude: CLLocationDegrees)
  ```
Core Location

- The device location can be tested in the iOS Simulator from Xcode.
**CLLocationManager**

- General approach to using it:

1. Always request authorization to use location services and check to see whether the desired services are available as described in Requesting Permission to Use Location Services.

2. Create an instance of the CLLocationManager class and store a strong reference to it somewhere in your app.

3. Keeping a strong reference to the location manager object is required until all tasks involving that object are complete.

4. Assign a custom object to the delegate property. This object must conform to the CLLocationManagerDelegate protocol.

5. Configure any additional properties relevant to the desired service.

6. Call the appropriate start method to begin the delivery of events.
The use of location services requires user authorization:

1. When your application first tries to use location monitoring, user will be asked if it’s okay to do so. Setup one of these two keys in your project’s Info.plist:

   - `NSLocationWhenInUseUsageDescription`
   - `NSLocationAlwaysUsageDescription`

2. Call the `authorizationStatus()` class method to get the current authorization status for your app.

   If the authorization status is `restricted` or `denied`, your app is not permitted to use location services and you should abort your attempt to use them.

3. If the authorization status was `notDetermined`, call one of the following methods to request the appropriate type of authorization from the user:

   - `func requestWhenInUseAuthorization()`
   - `func requestAlwaysAuthorization()`
Kinds of location monitoring

- Accuracy-based continuous updates.
- Updates only when significant changes in location occur.
- Region-based updates.
- Heading monitoring.
- Proximity to beacons updates.
CLLocationManager

Checking to see what your hardware can do

- Has the user enabled location services in Settings?
  
  class func locationServicesEnabled() -> Bool

- Can this hardware provide heading info (compass)?
  
  class func headingAvailable() -> Bool

- Can get events for significant location changes (requires a cellular radio)?
  
  class func significantLocationChangeMonitoringAvailable() -> Bool

- Is region monitoring available?
  
  class func isMonitoringAvailable(for regionClass: AnyClass) -> Bool

- Does the device support ranging of Bluetooth beacons?
  
  class func isRangingAvailable() -> Bool
CLLocationManager

Getting the information from the CLLocationManager

- You can just ask the CLLocationManager for the location or heading, but usually we don’t.
- Instead, we let it update us when the location changes (enough) via its delegate.
CLLocationManager

Accuracy-based continuous location monitoring

- Always set the desired accuracy as low as possible:
  
  ```swift
  var desiredAccuracy: CLLocationAccuracy { get set }
  ```

- Only changes in location of at least this distance (in meters) will fire a location update to you:
  
  ```swift
  var distanceFilter: CLLocationDistance { get set }
  ```

Use the value `kCLDistanceFilterNone` to be notified of all movements. This is also the default value.

Starting and stopping the monitoring

```swift
func startUpdatingLocation()
func stopUpdatingLocation()
```

- Be sure to turn updating off when your application is not going to consume the changes!
Get notified via the CLLocationManager’s delegate

- The CLLocationManagerDelegate methods that give location updates are:

  ```swift
  func locationManager(_ manager: CLLocationManager,
                       didUpdateTo newLocation: CLLocation,
                       from oldLocation: CLLocation)
  ```

  ```swift
  func locationManager(_ manager: CLLocationManager,
                      didUpdateLocations locations: [CLLocation])
  ```

- Because it can take several seconds to return an initial location, the location manager typically delivers the previously cached location data immediately.

- It delivers more up-to-date location data as it becomes available.

- Therefore it is always a good idea to check the timestamp of any location object before taking any actions.
Heading monitoring

• Only changes in heading of at least this many degrees will fire a location update to you:

```swift
var headingFilter: CLLocationDegrees { get set }
```

• Heading of “zero degrees” is the heading of the “top” of the device.

• With the next property, you can change that “top” (for example, `.landscapeLeft`, `.faceDown`):

```swift
var headingOrientation: CLDeviceOrientation { get set }
```

Start the monitoring

```swift
func startUpdatingHeading()

func stopUpdatingHeading()
```

• Be sure to turn updating off when your application is not going to consume the changes!
Get notified via the CLLocationManager's delegate

```swift
func locationManager(_ manager: CLLocationManager, 
    didUpdateHeading newHeading: CLHeading)
```

Error reporting to the delegate

```swift
func locationManager(_ manager: CLLocationManager, 
    didFailWithError error: Error)
```

- Not always a fatal thing, but pay attention to this delegate method.
- The CLError.locationUnknown error is likely temporary, keep waiting (for a while at least).
- If the user denies your application’s use of the location service, this method reports a CLError.denied error. Upon receiving such an error, you should stop the location service.
- If the heading could not be determined because of strong interference from nearby magnetic fields, this method will return a CLError.headingFailure. Keep waiting then.
Heading

CLHeading

- There are two types of heading (because the Earth's North Pole is not exactly the magnetic north):

  var magneticHeading: CLLocationDirection { get }  
  var trueHeading: CLLocationDirection { get }  

- Negative values mean “this heading is unreliable” (i.e. don’t use it).
- You won’t get trueHeading if location services are turned off (e.g. by the user).

  var headingAccuracy: CLLocationDirection { get }  

- Basically how far off the magnetic heading might be from actual magnetic north (in degrees).
- A negative value means “this heading is not valid”.

  var timestamp: Date { get }

Heading calibration user-interface

- Automatically put on screen by iOS, but can be prevented by the CLLocationManager’s delegate:

```swift
func locationManagerShouldDisplayHeadingCalibration(_ manager: CLLocationManager) -> Bool
```

- Or dismissed (maybe after a timer or something) using CLLocationManager instance method:

```swift
func dismissHeadingCalibrationDisplay()
```
Significant Location Changes

Significant location change monitoring in CLLocationManager

- “Significant” is not strictly defined. Think vehicles, not walking. Likely uses cell towers.

```swift
func startMonitoringSignificantLocationChanges() -> CLLocationManagerDelegate
func stopMonitoringSignificantLocationChanges()
```

- Be sure to turn updating off when your application is not going to consume the changes!

- You get notified via the CLLocationManager’s delegate. Same as for accuracy-based updating if your application is running.
Significant Location Changes

This service works even if your application is not running

- Or is in the background (we haven’t talked about multitasking yet).
- You will get launched and your application delegate will receive the message `application(didFinishLaunchingWithOptions:)` with an options dictionary that will contain this key (it indicates that the application was launched in response to an incoming location event):
  ```swift
  UIApplicationLaunchOptionsKey.location
  ```
- You should use this as a signal to create and configure a new `CLLocationManager`. Get the latest location via:
  ```swift
  var location: CLLocation? { get }
  ```
- Or start location services again. Upon doing so, your delegate receives the corresponding location data.
- If you are running in the background, don’t take too long (a few seconds)!
Region-based Monitoring

Region-based location monitoring in CLLocationManager

func startMonitoring(for region: CLRegion)
func stopMonitoring(for region: CLRegion)

Get notified via the CLLocationManager’s delegate

func locationManager(_ manager: CLLocationManager, didEnterRegion region: CLRegion)
func locationManager(_ manager: CLLocationManager, didExitRegion region: CLRegion)
func locationManager(_ manager: CLLocationManager, monitoringDidFailFor region: CLRegion?, withError error: Error)
Region-based Monitoring

Works even if your application is not running!

- In exactly the same way as “significant location change” monitoring.
- The regions in this property are shared by all instances of the `CLLocationManager` class in your application:
  ```swift
  var monitoredRegions: Set<CLRegion> { get }
  ```
- The set of monitored regions persists across application termination/launch.
- You cannot add regions to this property directly.
- Instead, you must register regions by calling:
  ```swift
  func startMonitoring(for region: CLRegion)
  ```
Region-based Monitoring

CLRegion

- CLRegions are tracked by name (identifier) because they survive application termination/relaunch.

- The CLRegion class defines an abstract area that can be tracked. In iOS, you do not create instances of this class directly; instead, you instantiate subclasses that define specific types of regions.

- You can configure which notifications should be generated:

  ```swift
  var notifyOnEntry/Exit: Bool { get set }
  ```

Regions (currently) require large location changes to fire

- Probably based on same technology as “significant location change” monitoring.

- Likely both of these “fire” when a new cell tower is detected.

- Definitely they would not use GPS (that would be very expensive power-wise).
Region-based Monitoring

Region monitoring size limit

- This property defines the largest boundary distance allowed from a region’s center point:

  ```swift
  var maximumRegionMonitoringDistance: CLLocationDistance { get }
  ```

- Attempting to monitor a region larger than this (radius in meters) will generate a `CLError.regionMonitoringFailure` error (which will be sent via the delegate method mentioned on a previous slide).

- If this property returns a negative value, then region monitoring is not working.
CMMotionManager

- The CMMotionManager class is the gateway to the motion services provided by iOS. These services provide an app with accelerometer data, rotation-rate data, magnetometer data, and other device-motion data.

- As a device moves, its hardware reports linear acceleration changes along the primary x, y, z axes in three-dimensional space.

- The device accelerometer reports values for each axis in units of G-force.

- You can use this data to detect both the current orientation of the device (relative to the ground) and any instantaneous changes to that orientation.
Accelerometer

How to get accelerometer data

- You create a `CMMotionManager` object:

```swift
var motionManager = CMMotionManager()
```

- Specify the interval at which you want to receive events:

```swift
var accelerometerUpdateInterval: TimeInterval {
    get
    set
}
```

This property is measured in seconds. You may also change this property while the manager gives updates.

- To start/stop accelerometer updates use the following methods:

```swift
func startAccelerometerUpdates()
func stopAccelerometerUpdates()
```

- This time, there is NO delegate. To get data from the accelerometer use the following property:

```swift
var accelerometerData: CMAccelerometerData? {
    get
}
```
The following code will also handle accelerometer updates. This is more elegant, but it requires advanced Swift knowledge (we already discussed about closures):

```swift
self.motionManager?.startAccelerometerUpdates(to: OperationQueue(),
withHandler: {
    [unowned self] (accelerometerData, error) in

    if error == nil {
        self.rollX = (accelerometerData?.acceleration.x)! * kFilterFactor + self.rollX * (1.0 - kFilterFactor)
        self.rollY = (accelerometerData?.acceleration.y)! * kFilterFactor + self.rollY * (1.0 - kFilterFactor)
    }
})
```

- `kFilterFactor` is a constant between 0 and 1 defined in your code somewhere. Actually a value near 0.1 is a good option.
- And `rollX`, `rollY` are properties of the `self` object:

```swift
var rollX: UIAccelerationValue = 0
```
Map Kit

MKMapView displays a map
MKMapView displays a map

The map can have annotations on it

- Each annotation is simply a coordinate, a title and a subtitle. They are displayed using an MKAnnotationView (MKPinAnnotationView shown here).
Map Kit

MKMapView displays a map

The map can have annotations on it

- Each annotation is simply a coordinate, a title and a subtitle. They are displayed using an MKAnnotationView (MKPinAnnotationView shown here).

Annotations can have a callout

- It appears when the annotation view is tapped. By default just shows the title and subtitle. But you can add left and right accessory views.

- In this example, left is a UIImageView, right is a detail disclosure UIButton (UIButtonTypeDetailDisclosure).
MKMapView

- Create with `init` or drag from Object Library in Interface Builder.
- Displays an array of objects which implement `MKAnnotation`:
  ```swift
  var annotations: [MKAnnotation] { get }
  ```

- `MKAnnotation` protocol:
  ```swift
  protocol MKAnnotation: NSObjectProtocol {
      var coordinate: CLLocationCoordinate2D { get }
      optional var title: String? { get }
      optional var subtitle: String? { get }
  }
  ```
MKAnnotation

Note that the annotations property is readonly

```swift
var annotations: [MKAnnotation] { get }
```

- Must add/remove annotations explicitly:
  ```swift
  func addAnnotation(_ annotation: MKAnnotation)
  func addAnnotations(_ annotations: [MKAnnotation])
  func removeAnnotation(_ annotation: MKAnnotation)
  func removeAnnotations(_ annotations: [MKAnnotation])
  ```

  Generally a good idea to add all your annotations up-front

- Allows the MKMapView to be efficient about how it displays them.
- Annotations are light-weight, but annotation views are not.
- MKMapView reuses annotation views similar to how UITableView reuses cells. Usually, we end up using only a few annotation views.
What do annotations look like on the map?

- By default they look like a pin.
- Annotations are drawn using an `MKAnnotationView` subclass.
- The default one is `MKPinAnnotationView` (which is why they look like pins).
- You can create your own or set properties on existing `MKAnnotationViews` to modify the look.
What do annotations look like on the map?

- By default they look like a pin.
- Annotations are drawn using an MKAnnotationView subclass.
- The default one is MKPinAnnotationView (which is why they look like pins).
- You can create your own or set properties on existing MKAnnotationViews to modify the look.

What happens when you touch on an annotation (e.g. the pin)?

- Depends on the MKAnnotationView that is associated with the annotation (more on this later).
- By default, nothing happens, but if canShowCallout is true in the MKAnnotationView, then a little box will appear showing the annotation’s title and subtitle. And this little box (the callout) can be enhanced with left/rightCalloutAccessoryViews.
MKAnnotation

- The following delegate method is also called when you touch on an annotation:

```swift
func mapView(_ mapView: MKMapView, didSelect view: MKAnnotationView)
```

- This is a great place to set up the MKAnnotationView's callout accessory views lazily.

- For example, you might want to wait until this method is called to download an image to show.
MKAnnotationView

How are MKAnnotationViews created and associated with annotations?

- Very similar to UITableViewCell cells in a UITableView. Implement the following MKMapViewDelegate method (if not implemented, returns a pin view):

```swift
func mapView(_ mapView: MKMapView, viewFor annotation: MKAnnotation) -> MKAnnotationView?
{
    var p = mapView.dequeueReusableAnnotationView(withIdentifier: "P")
    if p == nil
    {
        p = MKPinAnnotationView(annotation: annotation, reuseIdentifier: "P")
    }
    p?.canShowCallout = true
    // build pinView’s callout accessory views here
    p?.annotation = annotation // this can happen twice
    // Maybe load up accessory views here (if not too expensive)?
    // Or reset them and wait until
    // mapView:didSelectAnnotationView: to load actual data. */
    return p
}
```
MKAnnotationView

Interesting properties

- The annotation should be treated as if it is readonly:
  
  ```swift
  var annotation: MKAnnotation? { get set }
  ```

- The pin itself can be replaced with another image:
  
  ```swift
  var image: UIImage? { get set }
  ```

- Left and right callout accessory views:
  
  ```swift
  var leftCalloutAccessoryView: UIView? { get set }
  // maybe a UIImageView
  ```

  ```swift
  var rightCalloutAccessoryView: UIView? { get set }
  // maybe a detail disclosure UIButton
  ```

- Set this to `false` to ignore touch events (no delegate method, no callout):
  
  ```swift
  var isEnabled: Bool { get set }
  ```
MKAnnotationView

Interesting properties

- Where the image (pin) should be relative to the coordinate point of the associated annotation:

  ```swift
  var centerOffset: CGPoint { get set }
  ```

- Where the callout view should be relative to the top-center point of the annotation view:

  ```swift
  var calloutOffset: CGPoint { get set }
  ```

  When this property is set to (0, 0), the anchor point of the callout bubble is placed on the top-center point of the annotation view’s frame.

- Users can drag annotations. Only works if the associated annotation implements `setCoordinate` and this property is set to true:

  ```swift
  var isDraggable: Bool { get set }
  ```
MKAnnotationView

- If you set one of the callout accessory views to a UIControl, for example:

```swift
pinView?.rightCalloutAccessoryView = UIButton(type: .detailDisclosure)
```

- Then the following MKMapViewDelegate method will get called when the accessory view is touched:

```swift
func mapView(_ mapView: MKMapView, annotationView view: MKAnnotationView, calloutAccessoryControlTapped control: UIControl)
```
MKAnnotationView

Using `mapView(didSelect:)` to load up callout accessories

- Example: Using a downloaded thumbnail image for `leftCalloutAccessoryView`.
- Create a `UIImageView`. Assign it to `leftCalloutAccessoryView` in `mapView(viewFor:)`.
- Reset the `UIImageView`'s image to `nil` there as well.
- Then load the image on demand like this:

```swift
func mapView(_ mapView: MKMapView, didSelect view: MKAnnotationView)
{
    if view.leftCalloutAccessoryView is UIImageView
    {
        let imageView = view.leftCalloutAccessoryView as! UIImageView
        imageView.image = UIImage(named: "thumb")
    }
}
```
MKMapView

- Configuring the map view’s display type:

```swift
var mapType: MKMapType {
    get set
} = .standard, .satellite, .hybrid
```
MKMapView

• Showing the user’s current location:
  `var showsUserLocation: Bool { get set }`
  `var isUserLocationVisible: Bool { get }`
  `var userLocation: MKUserLocation { get }`

  MKUserLocation is an object which conforms to MKAnnotation which holds the user’s location.

• Restricting the user’s interaction with the map:
  `var isZoomEnabled: Bool { get set }`
  `var isScrollEnabled: Bool { get set }`
MKMapView

• Controlling the region the map is displaying:

```swift
var region: MKCoordinateRegion { get set }
struct MKCoordinateRegion {
    var center: CLLocationCoordinate2D
    var span: MKCoordinateSpan
}
struct MKCoordinateSpan {
    var latitudeDelta: CLLocationDegrees
    var longitudeDelta: CLLocationDegrees
}
func setRegion(_ region: MKCoordinateRegion, animated: Bool)
```

• Can also set the center point only:

```swift
var centerCoordinate: CLLocationCoordinate2D { get set }
func setCenter(_ coordinate: CLLocationCoordinate2D, animated: Bool)
```
MKMapView

Map loading notifications

- Remember that the maps are downloaded from the Internet.
- These methods are called whenever a new group of map tiles need to be downloaded from the server (whenever you expose portions of the map by panning or zooming the content):

  ```swift
  func mapViewWillStartLoadingMap(_ mapView: MKMapView)
  func mapViewDidFinishLoadingMap(_ mapView: MKMapView)
  func mapViewDidFailLoadingMap(_ mapView: MKMapView,
                                 withError error: Error)
  ```

  Lots of functions to convert points, regions, rects, etc.

- Take a look over the documentation.
- Examples:

  MKMapRectContainsPoint, MKMapPointForCoordinate, etc.
Overlays

**Overlays**

- Mechanism is similar to annotations (uses MKOverlayView instead of MKAnnotationView).

  ```swift
  func add(_ overlay: MKOverlay, 
            level: MKOverlayLevel)
  func addOverlays(_ overlays: [MKOverlay])
  func remove(_ overlay: MKOverlay)
  func removeOverlays(_ overlays: [MKOverlay])
  ```

**MKOverlay protocol**

- Protocol which includes MKAnnotation plus these:

  ```swift
  var boundingMapRect: MKMapRect { get }
  func intersects(_ mapRect: MKMapRect) -> Bool
  // optional method, uses boundingMapRect otherwise
  ```

- Overlays are associated with MKOverlayViews via delegate (just like annotations are associated with MKAnnotationViews):

  ```swift
  func mapView(_ mapView: MKMapView, 
               rendererFor overlay: MKOverlay)
  -> MKOverlayRenderer
  ```
MKOverlayView

- MKOverlayRenderer subclasses must be able to draw the overlay:
  ```swift
  func draw(_ mapRect: MKMapRect,
            zoomScale: MKZoomScale,
            in context: CGContext)
  ```

- This is not quite like `drawRect()` (because you’ll notice that you are provided the context).

- But you will still use CoreGraphics to draw (this method must be thread-safe, by the way).

- Also notice that the rectangle to draw is in map coordinates, not view coordinates.

- Converting to/from map points/rects from/to view coordinates:
  ```swift
  func point(for mapPoint: MKMapPoint) -> CGPoint
  func mapPoint(for point: CGPoint) -> MKMapPoint
  func rect(for mapRect: MKMapRect) -> CGRect
  func mapRect(for rect: CGRect) -> MKMapRect
  ```
Next Time

Managing and Storing Data:

- Property Lists
- Archiving Objects
- Filesystem Storing
- SQLite
- Closures (recap)
- Grand Central Dispatch
- URL Requests