ATLS 4120/5120: Mobile Application Development
Week 7: Core location and Map Kit

Location Services consist of two pieces:
- Core Location which gets information about the user's location
- Maps which provides the displaying and annotation of maps

Core Location
The Core Location framework enables iOS devices to determine their location using 3 methods:
- Cell tower triangulation
  - fairly accurate in areas with high cell tower density but less accurate in areas where there is a greater distance between towers
  - Low power usage
  - Requires cell phone connectivity
- Wi-Fi Positioning Service(WPS)
  - uses the mac addresses from nearby wi-fi access points to make a guess at your location
  - More accurate
  - More power
- Global Positioning System(GPS)
  - Most accurate
  - Uses a lot of power
  - Not available on wifi only iPads

You can't decide which one should be used, iOS decides.

The **CLLocationManager** class handles location related activites
- Location data is stored in the location property as a **CLLocation** object
  - Coordinates stored as a **CLLocationCoordinate2D** struct that contains latitude and longitude
    - **location.coordinate.latitude**
    - **location.coordinate.longitude**
- Horizontal accuracy **location.horizontalAccuracy**
  - Radius of uncertainty around the location's position
  - the transparent blue circle around the tracking dot you see on the map
- Altitude **location.altitude**
- Vertical accuracy **location.verticalAccuracy**
- Timestamp (**NSDate** object) **location.timestamp**
  - representing the time at which the location was determined
- Distances measured in meters
- The **distanceFilter** property lets you set a minimum distance, in meters, a device must move before you are notified
  - The default is **kCLDistanceFilterNone** which reports all movements
- Location services is one of the biggest battery draining activities in iOS so you should be thoughtful in the frequency of requesting the location.
- The **desiredAccuracy** property is of type CLLocationAccuracy which lets you determine the accuracy of the data
  - **kCLLocationAccuracyBestForNavigation**
  - **kCLLocationAccuracyBest**
    - this is the most accurate
    - default

    - **kCLLocationAccuracyNearestTenMeters**
    - **kCLLocationAccuracyHundredMeters**
    - **kCLLocationAccuracyKilometer**
    - **kCLLocationAccuracyThreeKilometers**
- Don't specify a degree of accuracy any greater than you need.

Data reported by Core Location can be inaccurate

- Location can be nil
- **horizontalAccuracy** <0 means the location is invalid
- Locations can be reported out of order
- Locations initialized before your app was initialized can be reported
- Validating your data before you use it is a good idea.

Kinds of location monitoring:

- Accuracy based continual location updates
- Updates only when "significant" changes in location occur
- Region-based updates
- Heading monitoring from the compass

Not all devices support different types of location updating, so it's a good idea to check first

Permission authorization

- Before using location services you must request permission
    - New in iOS8
    - If you try to use location services without permission it will fail silently
    - In iOS7 and prior, the location manager used to automatically ask for permission when it was asked to start (if it needed it) but now you must explicitly ask for that permission at a time of choosing
- **requestWhenInUseAuthorization()** allows the app to get location updates only when the app is in the foreground
- **requestAlwaysAuthorization()** allows the app to receive location updates both when the app is in the foreground and in the background
    - Will also give you WhenInUse authorization
- Many tutorials show both but you usually only need one
- Apple recommends that, in a real application, you should delay making the request until you actually need to use location services. The reason for this is that the user is more likely to agree if it's obvious why you need access to the device's location, based on operation that has been requested, than if an app requests permission as soon as it launches.
- Requires associated key in Info.plist
    - **NSLocationWhenInUseUsageDescription**
    - **NSLocationAlwaysUsageDescription**
- **authorizationStatus()** returns the apps authorization status
- Authorization statuses:
    - **kCLAuthorizationStatusNotDetermined**
        - User hasn't yet been asked to authorize location updates, you need to request authorization
    - **kCLAuthorizationStatusRestricted**
        - User has location services turned off in Settings (Parental Restrictions)
        - your app is not permitted to use location services
    - **kCLAuthorizationStatusDenied**

        - User has been asked for authorization and tapped "No" (or turned off location in Settings)
        - your app is not permitted to use location services
    - **kCLAuthorizationStatusAuthorized**
        - User has been asked for authorization and tapped "Yes" on iOS 7 and lower.
    - **kCLAuthorizationStatusAuthorizedAlways**
        - User authorized background use
    - **kCLAuthorizationStatusAuthorizedWhenInUse**
        - User has authorized use only when the app is in the foreground
- If the authorization status is **kCLAuthorizationStatusRestricted** or **kCLAuthorizationStatusDenied**, your app is not permitted to use location services
- If the authorization status is **kCLAuthorizationStatusAuthorizedAlways** or **kCLAuthorizationStatusAuthorizedWhenInUse** you are permitted to use location services and MapKit can show the user's location
- The user can change the authorization so it's important to make sure your app can detect and respond properly to changes in its authorization status.

The **CLLocationManagerDelegate** is notified of all location related updates.

- All methods are optional
- **locationManager(_:, didUpdateLocations)**
    - tells the delegate there's a new location value
    - array of locations with the most recent last
    - CLLocationManager object with current location
- **locationManager(_:, didFailwithError)**
    - tells the delegate that the location manager was unable to retrieve a location value.
- **locationManager(_:, didChangeAuthorizationStatus)** gets called when the authorization status changes

Core Location steps:

- Check that you have authorization to use location services
    - Request permission if you haven't already
- Initialize the location manager
- Assign yourself as the delegate
- Configure the manager for the types of updates you want
- Call **startUpdatingLocation()**to start receiving location updates
    - If you call startUpdatingLocation() without requesting permission it will fail silently
- If you ever stop needing location data you should call **stopUpdatingLocation()** to preserve battery life

MapKit

The MapKit framework provides an interface to embed maps

- iOS 5.1 and earlier this interfaced with Google maps.
- iOS 6 and later with Apple Maps

The **MKMapView** class lets you display and manipulate maps.

- Supports standard, satellite, and hybrid maps.
- Interacts nicely with Core Location
- Can change the position and the zoom level of the map
- Map views support flick and pinch gestures

- Set the **mapType** property for the type of map
  - **MKMapType.Standard**
    - default
  - **MKMapType.Satellite**
  - **MKMapType.Hybrid**
  - **MKMapTypeSatelliteFlyover** (new in iOS9)
    - Displays a satellite image of the area with flyover data where available.
  - **MKMapTypeHybridFlyover** (new in iOS9)
    - Displays a hybrid satellite image with flyover data where available.
  -
- **showsUserLocation** is a Boolean that indicates whether the map should try to display the user's location. Default is false.
- **setRegion(_:, animated)** sets the region to display in the map view
  - Region is a **MKCoordinateRegion** (struct)
    - center is latitude and longitude point on which the map is centered
      - **CLLocationCoordinate2D**
    - span defines how much of the map should be visible
      - **MKCoordinateSpan**

The **MKMapViewDelegate** protocol receives map-related updates about changes in map status and to coordinate displaying annotations.

- **mapView(_:, regionWillChangeAnimated)** tells the delegate that the region displayed by the map view is about to change
- **mapViewWillStartLoadingMap(_:)** tells the delegate that map view is about to retrieve some map data.
- **mapViewDidFailLoadingMap(_:, withError)** tells the delegate that the view was unable to load the map data.

Annotations offer a way to highlight coordinates on the map and provide additional information
The **MKPointAnnotation** class associates an annotation with a coordinate location on the map
The **MKAnnotation** protocol provides annotation information to a map view

- coordinate (required)
- title (optional)
- subtitle (optional)

Use **MKMapView .addAnnotation()** to add the annotation to your map view

- The "red pin" is default

Use the **MKAnnotationView** class to create custom annotation views

<u>Basic Map</u>
(location basic)

ViewController.swift

```
    override func viewDidLoad() {
        let location = CLLocationCoordinate2D(
            latitude: 40.74836,
            longitude: -73.984607
        )
        let span = MKCoordinateSpanMake(0.05, 0.05) //defines the area
spanned by a map region
        let region = MKCoordinateRegion (center: location, span: span)
```

```
//region of the map to be displayed
        mapView.setRegion(region, animated: true) //animates changing the
currently visible region
        let annotation = MKPointAnnotation() //create an annotation
        annotation.coordinate=location //sets the coordinates of the
annotation
        annotation.title="Empire State Building" //sets the title of the
annotation
        annotation.subtitle="New York" //sets the subtitle of the annotation
        mapView.addAnnotation(annotation) //adds the annotation to the map
view
        super.viewDidLoad()
    }
```

You will see the map (default mapType) and the annotation when you run it but it's set to only that location.

Location
Create a new Single view app called Location for the iPhone.
Click on the Target and go into the Build Phases tab.
Open Link Binary with Libraries and click the + to add the CoreLocation and MapKit frameworks under iOS 9.3.

To use these frameworks you must import them into ViewController.swift

```
import MapKit
import CoreLocation
```

Now go into MainStoryboard and add a Map View to take up the whole view. Pin the leading(-16), trailing(-16), top(0) and bottom(0) so it fills up the view.

Create an outlet connection for it called mapView.

In Supporting Files click on the Info.plist file. Click on the + next to "Information Property List" and manually enter this key (it is not in the list)
NSLocationWhenInUseUsageDescription with the string "This application requires location services"

ViewController.swift
Adopt `CLLocationManagerDelegate`

```
class ViewController: UIViewController, CLLocationManagerDelegate
```

Define a CLLocationManager instance.

```
var locationManager = CLLocationManager()
```

Add CLLocationManager delegate methods

```
//called when a new location value is available
    func locationManager(manager: CLLocationManager!, didUpdateLocations
locations: [CLLocation]) {
        let span = MKCoordinateSpanMake(0.05, 0.05) //defines the area
spanned by a map region
        let region = MKCoordinateRegionMake(manager.location!.coordinate,
span) //region of the map to be displayed
```

```
        mapView.setRegion(region, animated: true) //animates changing the
currently visible region
    }

    //called when the authorization status for the application changed.
    func locationManager(manager: CLLocationManager!,
didChangeAuthorizationStatus status: CLAuthorizationStatus) {
        println("didchangeauth")
        if status==CLAuthorizationStatus.AuthorizedWhenInUse {
            locationManager.startUpdatingLocation() //starts the location
manager
        }
    }

    //called when a location cannot be determined
    func locationManager(manager: CLLocationManager!, didFailWithError
error: NSError!) {
        var errorType=String()
        if let clError=CLError(rawValue: error.code) {
            if clError == .Denied {
                errorType="access denied"
                let alert=UIAlertController(title: "Error", message:
errorType, preferredStyle: UIAlertControllerStyle.Alert)
                let okAction:UIAlertAction=UIAlertAction(title: "OK",
style:UIAlertActionStyle.Default, handler: nil)
                alert.addAction(okAction)
                presentViewController(alert, animated: true, completion:
nil)
            }
        }
    }
```

Add to viewDidLoad

```
     mapView.mapType=MKMapType.Hybrid //hybrid with map and satellite
        var status = CLLocationManager.authorizationStatus()
        if status==CLAuthorizationStatus.NotDetermined{
            locationManager.requestWhenInUseAuthorization()
        }
        locationManager.delegate=self
        locationManager.desiredAccuracy=kCLLocationAccuracyBest //specify
the desired accuracy
        locationManager.distanceFilter=kCLDistanceFilterNone //specify the
distance a device must move laterally(in meters) to generate an update. We
specify to be notified of all movements
        mapView.showsUserLocation=true
```

When you run it in the simulator will not know your actual location. Either run it on a device or use Debug | Location | Custom to enter in your own longitude and latitude.
Address lookup web site http://stevemorse.org/jcal/latlon.php
Once you've set location services in the simulator you can't be prompted again. To be prompted again you must "Reset Content and Settings".
Or go into Settings in the simulator. In Privacy | Location you can turn location services off for the

whole device. Or scroll down to your app and change Location to Never. Then go back into your app and you'll see the alert.

Annotation

Now let's use annotations to add markers to the map to represent our location.

```
    var annotation = MKPointAnnotation()
```

update locationManager(_:, didUpdateLocations) to add the annotation

```
        annotation.coordinate=manager.location!.coordinate
        annotation.title="You are here"
        annotation.subtitle="Latitude:
\(manager.location!.coordinate.latitude), Longitude:
\(manager.location!.coordinate.longitude)"
        mapView.addAnnotation(annotation)
```

A tap on the pushpin will display the title and subtitle.

On a device as you move around the pushpin will move with you.

Changing Location Service Permissions

Users can grant or revoke location permission via the Settings app. You can test this on the simulator. Launch the app and grant yourself permission to use Core Location (if you've previously denied permission, you'll need to remove and reinstall the app first). You should see your location on the map. Now go to the Settings app and choose Privacy | Location. Turn the location services switch to OFF and go back to your application. You'll see that the map no longer shows your position. That's because the location manager called the locationManager(_:, didChangeAuthorizationStatus) method with authorization code CLAuthorizationStatus.Denied so the application stops receiving position updates and tells Map Kit to stop tracking the user's position.

Now go back to the Settings app, turn location services back on and run your app again and you'll find that it's tracking your position again.

You can also switch Location Services off per app. Back in Settings Privacy | Location below the location services switch you'll see a list of all the apps that are using it, including your app. Clicking the application name takes you to another page where you can allow or deny access to your application. At the moment, the application can use location services while the user is using the app. If you click Never, that permission is revoked.

Note: You might need to go into your app's target and in the Capabilities tab turn Maps on.