Mobile Application Development
Aileen Pierce

CORE LOCATION
MAP KIT

Location Services

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

Core Location

- The Core Location framework enables iOS devices to determine their location using 3 methods:
- Cell tower triangulation
  - Not always very accurate
  - Low power usage
- Wi-Fi Positioning Service(WPS)
  - More accurate
  - More power
- Global Positioning System(GPS)
  - Most accurate
  - Uses a lot of power

Core Location

- The **`CLLocationManager`** class handles location related activites
- The **`CLLocationManagerDelegate`** is notified of all location related updates.
- **`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.

Core Location

- 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
  - Altitude **`location.altitude`**
  - Vertical accuracy **`location.verticalAccuracy`**
  - Timestamp (**`NSDate`** object) **`location.timestamp`**
    - representing the time at which the location was determined

Core Location

- 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

Accuracy

- The **`desiredAccuracy`** property lets you determine the accuracy of the data
  - **`kCLLocationAccuracyBestForNavigation`**
  - **`kCLLocationAccuracyBest`**
  - **`kCLLocationAccuracyNearestTenMeters`**
  - **`kCLLocationAccuracyHundredMeters`**
  - **`kCLLocationAccuracyKilometer`**
  - **`kCLLocationAccuracyThreeKilometers`**
- Don’t specify a degree of accuracy any greater than you need.

Distance Filter

- 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.

Core Location

- 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.

Core Location

- Before using location services you must request permission
- **`CLLocationManager.authorizationStatus()`** returns the apps authorization status
- If the authorization status is **`kCLAuthorizationStatusNotDetermined`** you need to request authorization

Core Location

- **`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
- Requires associated key in Info.plist
  - **`NSLocationWhenInUseUsageDescription`**
  - **`NSLocationAlwaysUsageDescription`**

Core Location

- The delegate method **`locationManager(_, didChangeAuthorizationStatus)`** gets called when the authorization status changes
- Authorization statuses:
  - **`kCLAuthorizationStatusNotDetermined`**
  - **`kCLAuthorizationStatusRestricted`**
  - **`kCLAuthorizationStatusDenied`**
  - **`kCLAuthorizationStatusAuthorized`**
  - **`kCLAuthorizationStatusAuthorizedAlways`**
  - **`kCLAuthorizationStatusAuthorizedWhenInUse`**

Core Location

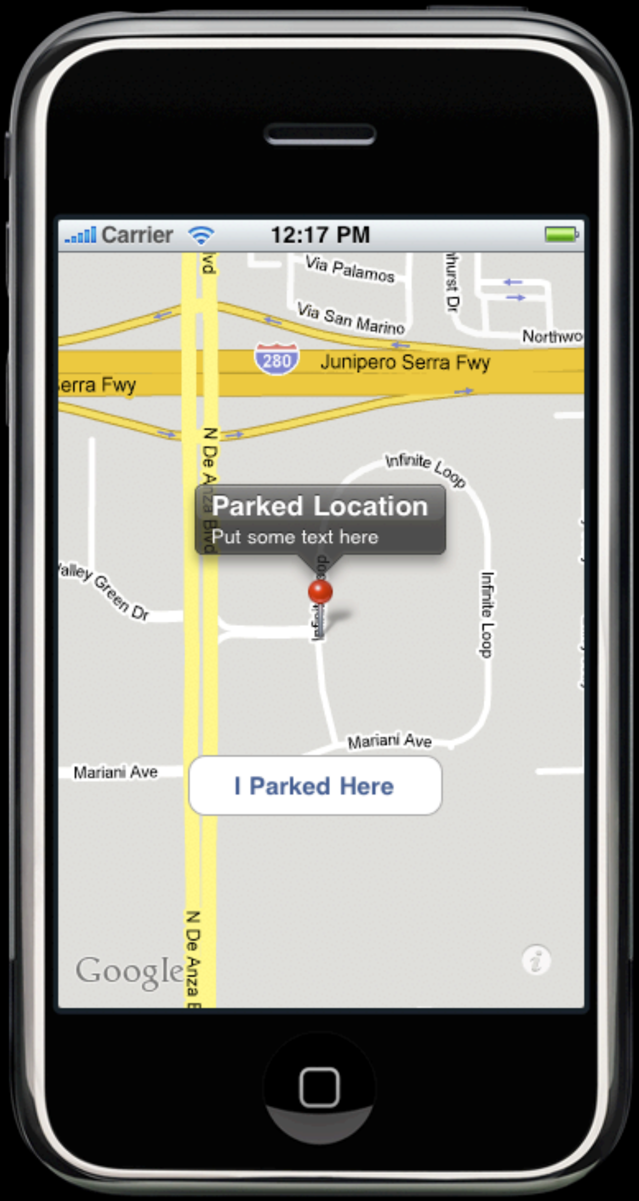
- If the authorization status is **`kCLAuthorizationStatusRestricted`** or **`kCLAuthorizationStatusDenied`**, your app is not permitted to use location services
- If the authorization status is **`kCLAuthorizationStatusAuthorizedAlways`** or **`kCLAuthorizationStatusAuthorizedWhenInUse`** you are permitted to use location services and MapKit can show the user's location

Core Location

- 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 ever stop needing location data you should call **`stopUpdatingLocation()`** to preserve battery life

Map Kit

- MapKit is a framework that provides an interface to embed maps
- Supports standard, satellite, and hybrid maps.
- Interacts nicely with Core Location



Map Kit

- The **`MKMapView`** class lets you display and manipulate maps.
- 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.Satellite`**
  - **`MKMapType.Standard`**
  - **`MKMapType.Hybrid`**
- **`showsUserLocation`** is a Boolean that indicates whether the map should try to display the user's location. Default is false.

Map Kit

- **`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`**

Map Kit

- The **`MKMapViewDelegate`** protocol methods are notified 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

- Annotations offer a way to highlight coordinates on the map and provide additional information
- MapKit provides standard annotation views through the **`MKPointAnnotation`** class
- The **`MKAnnotation`** protocol provides annotation information to a map view
  - coordinate
  - title
  - subtitle

Annotations

- Use **`addAnnotation()`** in the **`MKMapView`** class to add the annotation to your map view
- The “red pin” is default
- Use the **`MKAnnotationView`** class to create custom annotation views