iPhone Programming
Core Location

Tim Gegg-Harrison

Core Location

- GPS
- Coordinates:
  - Latitude
  - Longitude

Core Location…

- In Info.plist, add:
  - NSLocationWhenInUseUsageDescription
  - NSLocationAlwaysUsageDescription
- Add frameworks:
  - CoreLocation
  - MapKit

Core Location…

- Steps to set up core location (CLLocationManager)
  - request authorization
    ```
    manager.requestWhenInUseAuthorization()
    manager.requestAlwaysAuthorization()
    ```
  - set up distance filter
    ```
    manager.distanceFilter = kCLDistanceFilterNone
    ```
  - set up accuracy of location
    ```
    manager.desiredAccuracy = kCLLocationAccuracyNearestTenMeters
    ```
  - identify delegate
    ```
    manager.delegate = self
    ```
  - start getting location updates
    ```
    manager.startUpdatingLocation()
    ```
  - stop getting location updates
    ```
    manager.stopUpdatingLocation()
    ```

Core Location…

- Location updates (CLLocationManagerDelegate)

```
func locationManager(_ manager: CLLocationManager,
didUpdateLocations locations: [CLLocation]) {
     let location = locations[locations.count-1]
      currentLatitude = location.coordinate.latitude
      currentLongitude = location.coordinate.longitude
}
```

Core Location…

- Reverse geocoding (CLGeocoder)

```
geocoder.reverseGeocodeLocation(location, completionHandler:
  { (placemarks: [CLPlacemark]?, error: Error?) -> Void in
      NSLog("Geocoder obtained ...")
      if error == nil  {
         if placemarks != nil && placemarks!.count > 0 {
            let placemark: CLPlacemark = placemarks![0]
            self.currentAddressLabel.text =
             "\(placemark.subThoroughfare!) \
             (placemark.thoroughfare!)\n\(placemark.locality!),
             \(placemark.administrativeArea!) \
             (placemark.postalCode!)\n\(placemark.country!)"
         }
         else {
            NSLog("No placemark data...")
         }
      }
      else {
         NSLog("Geocoder error: \(error)")
      }
  })
```

Mapkit

- An `MKMapView` object provides an embeddable map interface, similar to the one provided by the Maps application.
- You use this class as-is to display map information and to manipulate the map contents from your application.
- You can center the map on a given coordinate, specify the size of the area you want to display, and annotate the map with custom information.

Mapkit…

- When you initialize a map view, you should specify the initial region for that map to display. You do this by setting the region property of the map.
- A **region** is defined by a center point and a horizontal and vertical distance, referred to as the span.
- The **span** defines how much of the map at the given point should be visible and is also how you set the zoom level.
- Specifying a large span results in the user seeing a wide geographical area and corresponds to a low zoom level.

Mapkit…

- In addition to setting the span programmatically, the `MKMapView` class supports many standard interactions for changing the position and zoom level of the map.
- In particular, map views support flick and pinch gestures for scrolling around the map and zooming in and out.
- Support for these gestures is enabled by default but can also be disabled using the `scrollEnabled` and `zoomEnabled` properties.
- Although you should not subclass the `MKMapView` class itself, you can get information about the map view's behavior by providing a delegate object.
- The map view calls the methods of your custom delegate to let it know about changes in the map status and to coordinate the display of custom annotations.

MKCoordinateRegion

- `center`
  - The center point of the region.
- `span`
  - The horizontal and vertical span representing the amount of map to display. The span also defines the current zoom level used by the map view object.
- The center point of the region is defined with `CLLocationCoordinate2D`
  - `latitude`
    - The latitude in degrees. Positive values indicate latitudes north of the equator. Negative values indicate latitudes south of the equator.
  - `longitude`
    - The longitude in degrees. Measurements are relative to the zero meridian, with positive values extending east of the meridian and negative values extending west of the meridian.

Annotating the map

- The `MKMapView` class supports the ability to annotate the map with custom information.
- Because a map may have potentially large numbers of annotations, map views differentiate between the annotation objects used to manage the annotation data and the view objects for presenting that data on the map.
- An **annotation object** is any object that conforms to the `MKAnnotation` protocol.
- Annotation objects are typically implemented using existing classes in your application's data model.
- This allows you to manipulate the annotation data directly but still make it available to the map view.
- Each annotation object contains information about the annotation's location on the map along with descriptive information that can be displayed in a callout.