Module: Mobile Application development (Android)

Session 49: Location Based Services

**Location Based Services**

* Location Manager
* Location Providers
* Geodecoders

## Location manager

The Android location manager gives location in terms of longitude and latitude for the location of the phone. Depending on the location provider selected (could be based on GPS, WiFi or Cellular) the accuracy of the location will vary.

The key Android plumbing for location is:

* Location Manager, which provides the coordinates
* Location Providers, which can make a number of trade offs to offer the user the capability they want

A number of services can be built using these simple components:

* get the user's current location
* periodically get the user location as the move around -- provides a trail of bread crumbs
* use proximity alerts when you move in and out of a predefined area (e.g., Time Square)

As part of the onCreate() code of the WhereAmI app we see the location manager being set up. The user gets a location manager by specifying the LOCATION\_SERVICE as input to the getSystemService() of the activity. A location manager is returned.

**public** **class** WhereAmI **extends** Activity {

@Override

**public** void onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.main);

LocationManager locationManager;

String svcName = Context.LOCATION\_SERVICE;

locationManager = (LocationManager)getSystemService(svcName);

In the Manifest you will see that it is necessary to get the user's permission to track their location or get a location reading:

<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />

There are fine and coarse permissions that the developer can use. The defaults is: if you ask for ACCESS\_FINE\_LOCATION then by default you get coarse.

FIne is typically associated with GPS and coarse location with Cellular or network.

## Location Provider

Mobile phones can provide location from a set of providers that make a number of trade offs. For example, GPS has good accuracy outdoors but is costly in terms of energy tax for using the GPS chips on the phone. In contrast, Cellular is cheap in terms of energy consumption but could provide very coarse location information (say in the upper valley) because of the lack of cell tower density but could be great in the city. There are a number of trade offs that the user might want to make when selecting a location provider. Basically depending on what location device the user selects there are a number of different trade offs in:

* power consumption
* longitude/latitude accuracy
* altitude accuracy
* speed
* direction information

The user can specify the location provider explicitly in the code using a number of constants:

* LocationManager.GPS\_PROVIDER
* LocationManager.NETWORK\_PROVIDER
* LocationManager.PASSIVE\_PROVIDER

It would be poor programming to rigidly specify the provider -- for example, the user might turn off GPS -- then what. It is better to let the Android systems match the user's needs to what providers are on offer. To do this we use Criteria as shown below. The code states that the user requires:

* coarse accuracy
* low power consumption
* no altitude, bearing or speed

The code snippet is taken again from onCreate(). The user specifies the level of location information and then asks the system for the best provider given what is currently available and the user's needs.

Criteria criteria = **new** Criteria();

criteria.setAccuracy(Criteria.ACCURACY\_FINE);

criteria.setPowerRequirement(Criteria.POWER\_LOW);

criteria.setAltitudeRequired(**false**);

criteria.setBearingRequired(**false**);

criteria.setSpeedRequired(**false**);

criteria.setCostAllowed(**true**);

String provider = locationManager.getBestProvider(criteria, **true**);

## Finding Where Am I

Getting the location of the device (and user) is a cool service offered by phones. There are a number of best practices when apps use location. First, you need to respect the privacy of the user; only updating the location when necessary; letting the user know if you are tracking them and where the information is being stored, communicated, etc; and allowing the user to disable tracking.

To find the last location of the device you need to use getLastKnownLocation(), as shown below in the onCreate() code snippet. getLastKnownLocation returns a location indicating the data from the last known location fix obtained from the given provider. This can be done without starting the provider. Note that this location could be out-of-date, for example if the device was turned off and moved to another location. If the provider is currently disabled, null is returned.

Location l = locationManager.getLastKnownLocation(provider);

updateWithNewLocation(l);

locationManager.requestLocationUpdates(provider, 2000, 10,

locationListener);

}

Once the user gets the last location it class the updateWithNewLocation() function with the Location. updateWithNewLocation() prints out the longitude and latitude of the location and then gets a human readable address from the coordinates using Geodecoders. We will discuss Geodecoders in a moment.

## Tracking your location

But the last line of code in the onCreate() refreshes the current location is the key to **tracking location of a device**. The locationListener() method sets up a call back locationListener() which is called periodically. As shown above, onCreate() calls requestLocationUpdates() passing a [LocationListener](http://developer.android.com/reference/android/location/LocationListener.html) object called locationListener. The other parameters are:

* minTime: minimum time interval between location updates, in milliseconds. It is important to the right value of minTime to meet the apps needs and conserve battery life. Each location update requires power from GPS, WIFI, Cell and other radios.
* minDistance: minimum distance between location updates, in meters. The minDistance parameter is used to control the frequency of location updates. If it is greater than 0 then the location provider will only send your application an update when the location has changed by at least minDistance meters, AND at least minTime milliseconds have passed.

When a location change has been detected then locationListener callback is fired -- that is, the LocationListener is used for receiving notifications from the LocationManager when the location has changed. Assuming that the LocationListener has been registered with the location manager uses the requestLocationUpdates(String, long, float, LocationListener) method as the call back for handling the new [Location](http://developer.android.com/reference/android/location/Location.html) object. There are four callbacks:

* onLocationChanged(), which is called when a new location is available.
* onProviderDisabled(), when the provider is disabled (e.g., the user turn the GPS off).
* onProviderEnabled(), when the provider is enabled (e.g., the user turns the WiFI on).
* onStatusChanged(), when the provider status changes. This method is called when a provider is unable to fetch a location or if the provider has recently become available after a period of unavailability.

In our code snippet we simply call updateWithNewLocation() if the onLocationChanged() is called.

**private** final LocationListener locationListener = **new** LocationListener() {

**public** void onLocationChanged(Location location) {

updateWithNewLocation(location);

}

**public** void onProviderDisabled(String provider) {}

**public** void onProviderEnabled(String provider) {}

**public** void onStatusChanged(String provider, int status,

Bundle extras) {}

};

## Geocoding

In the updateWithNewLocation() method the Location object is transformed into two types of output:

* longitude and latitude
* a readable address (e.g., 360 Georgie Best Street, Manchester, England

[Geocoder](http://developer.android.com/reference/android/location/Geocoder.html) do the translation from long/lat to address. Geocoding is the process of transforming a street address or other description of a location into a (latitude, longitude) coordinate. Geocoder supports two services:

* forward geocoding: from address to long/lat
* reverse geocoding: from long/lat to address Reverse geocoding is the process of transforming a (latitude, longitude) coordinate into a (partial) address. The amount of detail in a reverse geocoded location description may vary, for example one might contain the full street address of the closest building, while another might contain only a city name and postal code. The Geocoder class requires a backend service that is not included in the core android framework.

The Geocoder class comes with the Google Maps library (see next section). To use the library you have to import it into the application. In addition, the Geocoder class uses a server to translate over the Internet so you need to add the following permission to the Manifest:

<uses-permission android:name="android.permission.INTERNET" />

The first part of the updateWithNewLocation() method simply prints out the longitude and latitude to the UI. The interesting stuff is the next section which does the reverse geocoding.

First the Geocoder is created by setting the Locale.getDefault() locale (used to define your usual location and language -- Hanover and English, sorry, I mean American). The getFromLocation(latitude, longitude, 1) returns an [Address](http://developer.android.com/reference/android/location/Address.html) object. The list can contain a number of possible results; that is, getFromLocation returns an array of Addresses that are known to describe the area immediately surrounding the given latitude and longitude. The parameters are:

* latitude: the latitude a point for the search
* longitude: the longitude a point for the search
* maxResults: max number of addresses to return. Smaller numbers (1 to 5) are recommended

In the code below maxResults = 1. We only want the servers best shot. Once the Address is returned the method uses a StringBuilder to get and format the address for display to the UI, as shown in the snippet below.

**private** void updateWithNewLocation(Location location) {

TextView myLocationText;

myLocationText = (TextView)findViewById(R.id.myLocationText);

String latLongString = "No location found";

String addressString = "No address found";

**if** (location != **null**) {

double lat = location.getLatitude();

double lng = location.getLongitude();

latLongString = "Lat:" + lat + "\nLong:" + lng;

double latitude = location.getLatitude();

double longitude = location.getLongitude();

Geocoder gc = **new** Geocoder(**this**, Locale.getDefault());

**try** {

List<Address> addresses =gc.getFromLocation(latitude, longitude, 1);

StringBuilder sb = **new** StringBuilder();

**if** (addresses.size() > 0) {

Address address = addresses.get(0);

**for** (int i = 0; i < address.getMaxAddressLineIndex(); i++)

sb.append(address.getAddressLine(i)).append("\n");

sb.append(address.getLocality()).append("\n");

sb.append(address.getPostalCode()).append("\n");

sb.append(address.getCountryName());

}

addressString = sb.toString();

} **catch** (IOException e) {}

}

myLocationText.setText("Your Current Position is:\n" +

latLongString + "\n\n" + addressString);

}