9. Google Maps

Creating a new Android application that uses the Google Maps Android API v2 requires several steps. Many of the steps outlined in this section will only have to be performed once, but some of the information will be a handy reference for future applications. The overall process of adding a map to an Android application is as follows:

- Download and configure the Google Play services SDK, which includes the Google Maps Android API. If you use the Google Maps Mobile SDK for Work you must download and configure the Google Maps Mobile SDK for Work static library.
- Obtain an API key. To do this, you will need to register a project in the Google APIs Console, and get a signing certificate for your app.
- Add the required settings in your application's manifest.
- Add a map to your application.
- Publish your application.

9.1 Setting up Google Play Services

To make the Google Play services APIs available to your app:

- Copy the library project at <android-sdk>/extras/google/google_play_services/libproject/google-play-services_lib/ to the location where you maintain your Android app projects.
- Import the library project into your Eclipse workspace. Click **File > Import**, select **Android > Existing Android Code into Workspace**, and browse to the copy of the library project to import it.
- In your app project, reference Google Play services library project. To add a reference to a library project, follow these steps:
    - Make sure that both the project library and the application project that depends on it are in your workspace. If one of the projects is missing, import it into your workspace.
    - In the **Package Explorer**, right-click the dependent project and select **Properties**.
    - In the **Properties** window, select the "**Android**" properties group at left and locate the **Library** properties at right.
    - Click **Add** to open the **Project Selection** dialog.
    - From the list of available library projects, select a project and click **OK**.
    - When the dialog closes, click **Apply** in the **Properties** window.
    - Click **OK** to close the **Properties** window.

Note: You should be referencing a copy of the library that you copied to your development workspace—you should not reference the library directly from the Android SDK directory.

- After you've added the Google Play services library as a dependency for your app project, open your app's manifest file and add the following tag as a child of the <application> element:

```
<meta-data
            android:name="com.google.android.gms.version"
            android:value="@integer/google_play_services_version" />
```

9.2 Displaying the debug certificate fingerprint

Firstly we need the *fingerprint* of our debug keystore, which is needed for getting the key for using Google Maps. To do this we have to execute the following command in a console:

```
keytool -list -v -keystore
"C:\Users\your_user_name\.android\debug.keystore" -alias
androiddebugkey -storepass android -keypass android
```

Please note that the user directory has to be changed in the command. Keytool is a command included in Java JDK, so the "*path*" environment variable has to be setup correctly.

We have to copy the string in format SHA1, which is something like this: BB:0D:AC:74:D3:21:E1:43:07:71:9B:62:90:AF:A1:66:6E:44:5D:75. It is important to highlight that each computer will have different numbers. The SHA1 can also be obtained from Eclipse in **Window -> Preferences -> Android -> Build.**

9.3 Obtaining an API Key

Obtain Key from here :

```
https://code.google.com/apis/console/?noredirect
```

If your application is registered with the Google Maps Android API v2 service, then you can request an API key. It's possible to register more than one key per project.

10 Navigate to your project in the Google APIs Console.
11 In the **Services** page, verify that the "Google Maps Android API v2" is enabled.
12 In the left navigation bar, click **API Access**.
13 In the resulting page, click **Create New Android Key**....
14 In the resulting dialog, enter the SHA-1 fingerprint, then a semicolon, then your application's package name. For example:

```
BB:0D:AC:74:D3:21:E1:43:67:71:9B:62:91:AF:A1:66:6E:44:5D:75;com.exam
ple.android.mapexample
```

15 The Google APIs Console responds by displaying **Key for Android apps (with certificates)** followed by a forty-character API key, for example:

```
AIzaSyBdVl-cTICSwYKrZ95SuvNw7dbMuDt1KG0
```

9.4 Add API Key to your application

Follow the steps below to include the API key in your application's manifest, contained in the file **AndroidManifest.xml**. From there, the Maps API reads the key value and passes it to the Google Maps server, which then confirms that you have access to Google Maps data.

In AndroidManifest.xml, add the following element as a child of the <application> element, by inserting it just before the closing tag </application>:

```
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="API_KEY"/>
```

Substitute your API key for **API_KEY** in the value attribute. This element sets the key com.google.android.maps.v2.API_KEY to the value of your API key, and makes the API key visible to any MapFragment in your application.

Save AndroidManifest.xml and re-build your application.

9.5 Setting up Application and Manifest

An Android application that uses the Google Maps Android API should specify the following settings in its manifest file, AndroidManifest.xml:

- A reference to the Google Play services version. If you have followed the steps on this page up to this point, you have already added the required declaration to your application manifest.
- The Maps API key for the application. The key confirms that you've registered with the Google Maps service via the Google APIs Console. If you have followed the steps on this page up to this point, you have already added the API key to your application manifest.
- Permissions that give the application access to Android system features and to the Google Maps servers. See below for instructions on adding this setting.
- (Recommended) Notification that the application requires OpenGL ES version 2. External services can detect this notification and act accordingly. For example, Google Play Store won't display the application on devices that don't have OpenGL ES version 2. See below for instructions on adding this setting.

Now we will add permissions and notification that application requires OpenGL ES Version 2 in AndroidManifest.xml as a child of manifest tag file.

```
<permission
android:name="com.example.mapdemo.permission.MAPS_RECEIVE"
android:protectionLevel="signature"/>
<uses-permission
android:name="com.example.mapdemo.permission.MAPS_RECEIVE"/>
```

And we also need to add these other permissions:

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERV
ICES
"/>
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

Finally we add

```
<uses-feature
android:glEsVersion="0x00020000"
android:required="true"/>
```

9.6 Adding the Map to your Application

The easiest way to test that your application is configured correctly is to add a simple map. You will have to make changes in two files: the XML file that defines the app's layout, and the main activity Java file.
Add the following fragment in the app's layout XML file. If you created a 'hello world' app using the Android Developer Tools (ADT) package in Eclipse, the file is at res/layout/activity-main.xml. Replace the entire contents of that file with the following code.

```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
          android:id="@+id/map"
          android:layout_width="match_parent"
          android:layout_height="match_parent"

android:name="com.google.android.gms.maps.SupportMapFragment"/>
```

Compile and run the code.

9.7 Using Google Map Class

Once the map is working you can try following code in your onCreate method of MainActivity to set the Map View to specific coordinates, add markers, zoom and etc.

```
double latitude = 60.221058;
            double longitude = 24.805060;

            String name = "Intro to Android Course Location";
            GoogleMap map = ((SupportMapFragment)
getSupportFragmentManager()
                        .findFragmentById(R.id.map)).getMap();
            map.addMarker(new MarkerOptions().position(
                        new LatLng(latitude,
longitude)).title(name));
            LatLng latLng = new LatLng(latitude, longitude);
            map.moveCamera(CameraUpdateFactory.newLatLng(latLng));
            map.animateCamera(CameraUpdateFactory.zoomTo(15), 2000,
null);
```

10. Geocoding

We will send a URL to SearchTask which will return us JSON and parse that JSON differently, in order to get the co ordinates for the typed address.

```
API Call : http://maps.googleapis.com/maps/api/geocode/json?address=
```

JSON Parser can look like this

```
if (status.equals("OK"))
{
JSONArray array = jsonObject.getJSONArray("results");
JSONObject item = array.getJSONObject(0);

JSONObject point =
item.getJSONObject("geometry").getJSONObject("location");

coordinates[0] = point.getDouble("lat");
coordinates[1] = point.getDouble("lng");


System.out.println("Latitude: "+ coordinates[0] + " - Longitude: " +
coordinates[1]);
}
```

11. Location Services (Optional)

For accessing the location systems we need to use *LocationManager*, which is obtained with *LOCATION_SERVICE*.
Firstly we have to create two attributes as follows:

```
LocationManager mylocManager;
LocationListener mylocListener;
```

LocationManager will allow to access the location service and LocationListener will be the class that will manage the different location events.
Then we have to write the following line of code in the "*onResume"* method.

```
mylocManager = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
```

Now we instantiate the MyLocationListener class, which will implement *LocationListener*, which will handle the different location events.

```
mylocListener = new MyLocationListener();
```

The following line of code will register the listener receiving the coordinates from the device.

```
mylocManager.requestLocationUpdates(LocationManager.NETWORK_PROVID
ER,
0, 0, mylocListener);
```

The last line of code receives coordinates by using the network of the phone (GPRS, 3G, etc.). If we want to use the GPS, we have to use the following code:

```
mylocManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
0, 0,
mylocListener);
```

We could change between systems using the different parameters:
LocationManager.GPS_PROVIDER and
LocationManager.NETWORK_PROVIDER.
The "*onResume*" method should be as follows:

```
@Override
public void onResume()
{
super.onResume();
mylocManager = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
mylocListener = new MyLocationListener();
mylocManager.requestLocationUpdates(LocationManager.NETWORK_PROVI
DER, 0, 0, mylocListener);
}
```

On the other hand, in the "*onPause*" method we have to remove the listener because if the application is closed or the activity is in paused state, the listener continues working. So if we do not do this, the battery of the device will run out quickly.

```
@Override
public void onPause()
{
super.onPause();
mylocManager.removeUpdates(mylocListener);
}
```

Finally, we have to create the class that implements the *LocationListener.* It will have the methods for receiving the coordinates.

```
public class MyLocationListener implements LocationListener {
@Override
public void onLocationChanged(Location loc) {
String coordinates = "My coordinates are: " + "Latitude = "
+ loc.getLatitude() + "- Longitude = " + loc.getLongitude();
Toast.makeText(getApplicationContext(), coordinates,
Toast.LENGTH_LONG).show();
}
@Override
public void onProviderDisabled(String provider) {
Toast.makeText( getApplicationContext(),"Gps
Disabled",Toast.LENGTH_SHORT ).show();
}
@Override
public void onProviderEnabled(String arg0) {
Toast.makeText( getApplicationContext(),"Gps
Enabled",Toast.LENGTH_SHORT ).show();
}
@Override
public void onStatusChanged(String provider, int status, Bundle
extras) {
// TODO Auto-generated method stub
}
}
```

The "*onLocationChanged*" method allows receiving the location coordinates. It is important to highlight that the system uses the coordinates system called "geodesic decimal degrees", but many other information systems use other different location systems.
The "*getLatitude()*" and "*getLongitude()*" methods return the latitude and longitude, respectively. The first one is the distance from the Equator until the position of the coordinate. If the place is in the north this value will be positive, if instead it is in the south it will be a negative value. The longitude is the distance from the Greenwich meridian: if the place is to the left, it will be a negative value; and positive if the position is to the right of the meridian.

```
loc.getLatitude();
loc.getLongitude();
```

The "*onProviderEnabled*" *and* "*onProviderDisabled*" methods detect if the location system is enabled or disabled.
The "*onStatusChanged*" method allows knowing the state of the GPS, according to the following values:

```
public static final int OUT_OF_SERVICE = 0;
public static final int TEMPORARILY_UNAVAILABLE = 1;
public static final int AVAILABLE = 2;
```

Finally, we have to add this line to the manifest file. It is the permission for accessing the location systems of the device. We will add this line in the "*AndroidManifest.xml*" file, before the *"application"* node.

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