Android Working with Google Maps

The Google Maps API for Android provides developers with the means to create apps with localization functionality. Version 2 of the Maps API was released at the end of 2012 and it introduced a range of new features including 3D, improved caching, and vector tiles.

Step 1

Google Maps Android API V2 is part of Google's Play services SDK, which you must download and configure to work with your existing Android SDK installation in Eclipse to use the mapping functions. In Eclipse, choose *Window > Android SDK Manager*. In the list of packages that appears scroll down to the *Extras*folder and expand it. Select the **Google Play services** checkbox and install the package.

Step 2

Once Eclipse downloads and installs the Google Play services package, you can import it into your workspace. Select *File > Import > Android > Existing Android Code into Workspace* then browse to the location of the downloaded Google Play services package on your computer. It should be inside your downloaded Android SDK directory, at the following location:**extras/google/google_play_services/libproject/google-play-services_lib**.

Get a Google Maps API Key

Step 1

To access the Google Maps tools, you need an API key, which you can obtain through a Google account. The key is based on your Android app debug or release certificate. If you have released apps before, you might have used the keytool resource to sign them. In that case, you can use the keystore you generated at that time to get your API key. Otherwise you can use the debug keystore, which Eclipse uses automatically when you generate your apps during development.

The API key will be based on the SHA-1 fingerprint from your debug or release certificate. To retrieve this you will need to run some code on the Command Line or in a Terminal. There are detailed instructions in the Google Developers Maps API guide for different operating systems and options. The following is an overview.

To use the debug certificate stored in the default location, run the following in a Terminal if you're on Linux:

```
keytool -list -v -keystore ~/.android/debug.keystore -alias androiddebugkey -storepass android -keypass android.
```

You will need to alter the path to the debug keystore if it's in a different location. On the Windows Command Line use the following for the default location, amending it to suit your C-Drive user folder:

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

If you want to use a release certificate, you first need to locate your release keystore. Use the following to display the keystore details, replacing "your_keystore_name" with the path and name of the keystore you are using:

```
keytool -list -keystore your_keystore_name
```

You will be prompted to enter your password, then you should see the aliases associated with the keystore. Enter the following, amending it to reflect your keystore and alias names:

```
keytool -list -v -keystore your_keystore_name -alias your_alias_name
```

Whether you're using a debug or release certificate, you will be presented with several lines of output. Look for the line beginning "SHA1" in the "Certificate fingerprints" section; it should comprise 20 HEX numbers with colons between them. Copy this line and append your app's intended package name to the end of it, storing it for future reference. For example:

Step 2

Now you can use your certificate SHA-1 fingerprint to get an API key for Google Maps. Sign into your Google account and navigate to the APIs Console. If you haven't used the console before it will prompt you to create a project.
Go ahead and create one, then rename it if you wish by selecting the drop-down list in the top left corner, which may have the default name "API Project" displayed on it.

Step 3

The APIs console manages access to many Google services, but you need to switch on each one you want to use individually. Select *Services* from the list on the left of the APIs console. You should see a list of Google services, scroll down to **Google Maps Android API V2** and click to turn it on for your account.
Follow the instructions to agree to the API terms.

Step 4

Now we can get a key for the app. Select *API Access* on the left-hand-side of the API console. You may already see a key for browser apps, but we need one specifically for Android apps. Near the bottom of the page, select **Create new Android key**.

In the pop-up window that appears, enter the SHA-1 fingerprint you copied from your certificate (with your package name appended) and click **Create**.

The API Access page should update with your new key. In the new *Key for Android apps* section, copy the key listed next to *API key* and stores it securely.

Add a Map to the App

- Now we can get the app ready to use mapping functions by adding the appropriate information to the Project Manifest file. Open it and switch to the XML tab to edit the code. First let's add the Google Maps API key inside the application element, using the following syntax:

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

Include your own API key as the value attribute.

Now we need to add some permissions to the Manifest. Inside the Manifest element but outside the application element, add the following, amending it to include your own package name:

```
<permission

    android:name="your.package.name.permission.MAPS_RECEIVE"
    android:protectionLevel="signature" />
<uses-permission android:name="your.package.name.permission.MAPS_RECEIVE"/>
```

- The Maps API is associated with lots of permissions, the need for which of course depends on your own app. The Developer Guide recommends adding the following to any app using Maps:

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

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

Finally, we need to add a feature element for OpenGL ES version 2:

```
<uses-feature

    android:glEsVersion="0x00020000"
    android:required="true"/>
```

- At last we are ready to include a map in the app. Open your app's main layout file. Replace the contents with the following:

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:map="http://schemas.android.com/apk/res-auto"
    android:id="@+id/the_map"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:name="com.google.android.gms.maps.MapFragment"
    map:cameraTilt="45"
    map:cameraZoom="14"
```

```
/>
```

- This is a Map Fragment. We give it an ID so that we can identify it in Java and provide (optional) initial values for camera tilt and zoom. You can specify lots of settings here, including latitude/longitude, bearing, and map type, as well as toggling the presence of features such as compass, rotation, scrolling, tilting, and zooming. These options can also be set from Java.
- At this point you can run your app. You will need to run it on an actual device, as the the emulator does not support Google Play services. Provided that the necessary resources are installed on your test device, you should see a map appear when the app runs. By default, the user can interact with the map in more or less the same way they would with the Google Maps app (rotating, etc.) using multi-touch interaction.

Displaying Zooming Controls

We can add zooming controls in order to zoom in or zoom out maps. Google maps had inbuilt zooming controls, so all we need to do is call couple of lines:

```
// Displaying Zooming controls
        MapView mapView = (MapView) findViewById(R.id.mapView);
        mapView.setBuiltInZoomControls(true);
```

Changing Map Display Type

You can also change map type like satellite, street view etc.,

```
mapView.setSatellite(true); // Satellite View

mapView.setStreetView(true); // Street View

mapView.setTraffic(true); // Traffic View
```

Showing Location by passing Latitude and Longitude

Below code will show a location on the map by passing latitude and longitude of that location.

```
MapController mc = mapView.getController();
double lat = Double.parseDouble("48.85827758964043"); // latitude
double lon = Double.parseDouble("2.294543981552124"); // longitude
GeoPoint geoPoint = new GeoPoint((int)(lat * 1E6), (int)(lon * 1E6));
mc.animateTo(geoPoint);
mc.setZoom(15);
mapView.invalidate();
```

Getting Latitude and Longitude of location that was touched

You can also get the latitude and longitude of location which was touched. Open your**AddItemizedOverlay.java** and add following method:

```
AddItemizedOverlay.java
package com.androidhive.googlemaps;

import java.util.ArrayList;

import android.content.Context;
import android.graphics.drawable.Drawable;
import android.util.Log;
import android.view.MotionEvent;
import android.widget.Toast;

import com.google.android.maps.GeoPoint;
import com.google.android.maps.ItemizedOverlay;
import com.google.android.maps.MapView;
import com.google.android.maps.OverlayItem;

public class AddItemizedOverlay extends ItemizedOverlay<OverlayItem> {

       /*................. Add this method ........*/
       @Override
       public boolean onTouchEvent(MotionEvent event, MapView mapView)
       {
             if (event.getAction() == 1) {

               GeoPoint geopoint = mapView.getProjection().fromPixels(
                   (int) event.getX(),
                   (int) event.getY());
               // latitude
               double lat = geopoint.getLatitudeE6() / 1E6;
               // longitude
               double lon = geopoint.getLongitudeE6() / 1E6;
               Toast.makeText(context, "Lat: " + lat + ", Lon: "+lon,
Toast.LENGTH_SHORT).show();
           }
           return false;
       }

    }
```

Adding Marker on the map

For displaying marker on the map you need to create new class which extends**ItemizedOverlay**. Create new class and name it as **AddItemizedOverlay.java** and write following code.

```
AddItemizedOverlay.java
package com.androidhive.googlemaps;
```

```
import java.util.ArrayList;

import android.content.Context;
import android.graphics.drawable.Drawable;
import android.util.Log;

import com.google.android.maps.ItemizedOverlay;
import com.google.android.maps.OverlayItem;

public class AddItemizedOverlay extends ItemizedOverlay<OverlayItem> {

       private ArrayList<OverlayItem> mapOverlays = new
ArrayList<OverlayItem>();

       private Context context;

       public AddItemizedOverlay(Drawable defaultMarker) {
            super(boundCenterBottom(defaultMarker));
       }

       public AddItemizedOverlay(Drawable defaultMarker, Context context) {
            this(defaultMarker);
            this.context = context;
       }

       @Override
       protected OverlayItem createItem(int i) {
          return mapOverlays.get(i);
       }

       @Override
       public int size() {
          return mapOverlays.size();
       }

       @Override
       protected boolean onTap(int index) {
          Log.e("Tap", "Tap Performed");
          return true;
       }

       public void addOverlay(OverlayItem overlay) {
          mapOverlays.add(overlay);
           this.populate();
       }

    }
```

References:
https://code.google.com/apis/console
http://mobile.tutsplus.com/tutorials/android/android-sdk-working-with-google-maps-application-setup/
http://www.androidhive.info/2012/01/android-working-with-google-maps/