Android Location & Maps Tutorial

Tiago Guerreiro
tjvg@di.fc.ul.pt

This tutorial provides a tutored first approach to the Android Location Services and the usage of the Google Maps API v2 for Android. It is a hands-on tutorial to Location & Maps. Basic examples are given and explained when needed along with brief overviews of the frameworks and apis involved.

The tutorial covers:

- **Location services**
- **Getting the current location**
- **Location updates**
- **Accuracy of location**
- **Showing a map with Google Maps API v2**
- **Creating an overlay over a map**

This tutorial was prepared for the following setup:

1. Android 2.3+

Location on Android

The Location Services API enables the developer to create location-aware Android applications without having to know details about the underlying technologies (e.g., GPS, Wi-FI, GSM). The Location Services provide a layer that enables the developer to receive the current location as well as its updates. The accuracy of the location depends on the active location sensors and the location permissions requested in the Manifest. As an example, one can get location information from the GPS sensor if the permission is requested to the finer location and if GPS is able to get location.

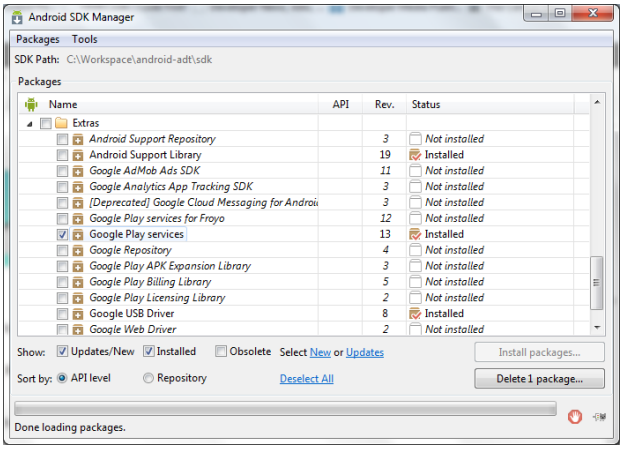
In the following subsections, the steps and requirements to create a location-aware application are presented. Basic Android programming skills are assumed and where simple procedures are required, explanations are short in detail.

Create a new Android Project

Create a new Android Application Project.

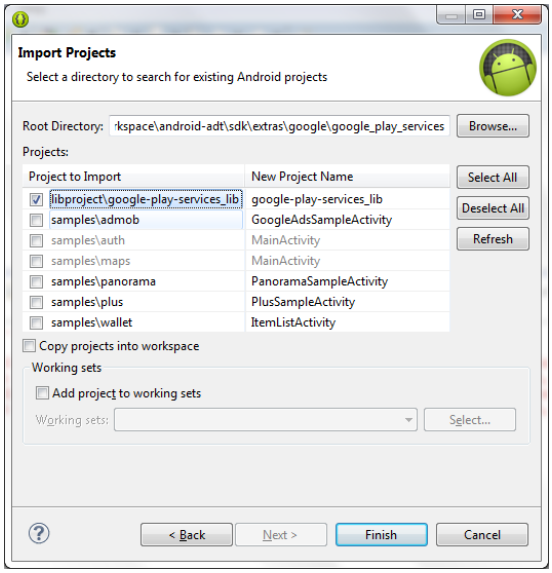
Setting up Google Play Services

Access to some APIs is performed through the usage of the Google Play Services API instead of the basic Android SDK Platform version. If you haven't done so you need to add the Google Play Services SDK to your current installation. To do so, you need to select and install them in the SDK Manager (Install Package upon selection).

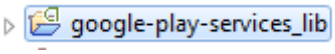


Now you need to import Google Play Services library into your application current workspace.

File >> Import, select **Android > Existing Android Code into Workspace**, and browse to the copy of the library project to import it. The Google Play services SDK is saved in your Android SDK environment at *<android-sdk>/extras/google/google_play_services/.*

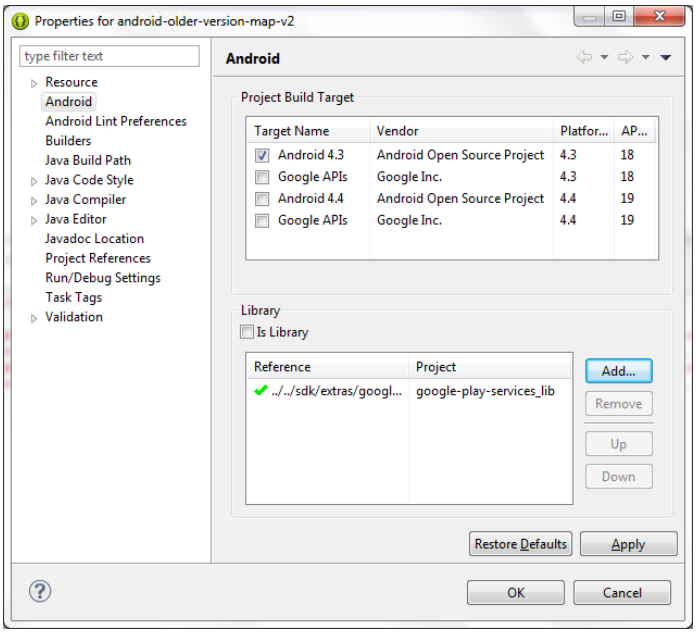


Now, you should be able to see the **google-play-servies-lib** project imported into your workspace.



And now, you need to add Google Play Services to your App.

- Right-click your App project
- Select **Properties**
- In the Properties window select **Android**
- Under the Library section click **Add**
- Select **google-play-services-lib**
- Click **OK**



Create the layout for the test application

Edit your xml (layout) file to include two textviews, one to show the latitude and another to show the longitude, as shown below.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/txtLong"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView1"
        android:text="Long"
        android:textSize="24dp" />

    <TextView
        android:id="@+id/txtLat"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/txtLong"
        android:text="Lat"
        android:textSize="24dp" />
</RelativeLayout>
```

Create the code for the test application

Besides extending the Activity class, you will need to **implement three interfaces,** two of them from the Google Play Services SDK.

```
public class MainActivity extends Activity implements
    GooglePlayServicesClient.ConnectionCallbacks,
GooglePlayServicesClient.OnConnectionFailedListener,
    LocationListener { ... }
```

Their role is as follows:

- **GooglePlayServicesClient.ConnectionCallbacks** : defines what you want to do on the connection to the location services & on the dis-connection . You need to implement *onConnected*() & *onDisconnected*()
- **GooglePlayServicesClient.OnConnectionFailedListener**: defines what you want to do if the connection failed. You need to implementon *ConnectionFailed*()
- **LocationListener**: Called when the location has changed. You need to implement *onLocationChanged*()

Let's start by defining our class variables.

```
    LocationClient mLocationClient;
    Location mCurrentLocation;
    LocationRequest mLocationRequest;

    TextView txtLong,txtLat;
```

And upon that, we just need to implement the callbacks required. The first one, is the *OnCreate* callback where we need to initialize our text views and our location objects.:

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        txtLong = (TextView) findViewById(R.id.txtLong);
        txtLat = (TextView) findViewById(R.id.txtLat);

        mLocationClient = new LocationClient(this, this, this);

        mLocationRequest = LocationRequest.create();
        mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
        // Set the update interval to 5 seconds
        mLocationRequest.setInterval(1000 * 5);
        // Set the fastest update interval to 1 second
        mLocationRequest.setFastestInterval(1000 * 1);
    }
```

Next, we need to override the methods that deal with connecting and disconnecting of the location services and with the starting and stopping of the app itself.

```
    @Override
    protected void onStart() {
        super.onStart();
        mLocationClient.connect();
    }

    @Override
    protected void onStop() {
        super.onStop();
        mLocationClient.disconnect();
    }

    @Override
    public void onConnectionFailed(ConnectionResult connectionResult) {
        Toast.makeText(this, "Connection Failed", Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onDisconnected() {
        Toast.makeText(this, "Disconnected.", Toast.LENGTH_SHORT).show();

    }
```

And finally, the code that deals with getting and updating the location of the device:

```
@Override
public void onConnected(Bundle arg0) {

  if(mLocationClient != null)
       mLocationClient.requestLocationUpdates(mLocationRequest,  this);
       Toast.makeText(this, "Connected", Toast.LENGTH_SHORT).show();

       if(mLocationClient != null){
          mCurrentLocation = mLocationClient.getLastLocation();
          try{
            txtLat.setText(mCurrentLocation.getLatitude()+"");
            txtLong.setText(mCurrentLocation.getLongitude()+"");
          }catch(NullPointerException npe){
            Toast.makeText(this, "Failed to Connect",
Toast.LENGTH_SHORT).show();
            Intent intent = new
Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
            startActivity(intent);
         }
```

```
     }
}

@Override
public void onLocationChanged(Location location) {
   Toast.makeText(this, "Location changed.", Toast.LENGTH_SHORT).show();
   mCurrentLocation = mLocationClient.getLastLocation();
    txtLat.setText(mCurrentLocation.getLatitude()+"");
    txtLong.setText(mCurrentLocation.getLongitude()+"");
}
```

Final tweaks: If they are not added already, add the following import expressions to your application code:

```
import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.GooglePlayServicesClient;
import com.google.android.gms.location.LocationClient;
import com.google.android.gms.location.LocationListener;
import com.google.android.gms.location.LocationRequest;

import android.app.Activity;
import android.content.Intent;

import android.location.Location;
import android.os.Bundle;
import android.provider.Settings;
import android.widget.TextView;
import android.widget.Toast;
```

Location Accuracy and final tweaks

Android has two location permissions: ACCESS_COARSE_LOCATION and ACCESS_FINE_LOCATION. The permission you choose controls the accuracy of the current location. If you request only coarse location permission, Location Services obfuscates the returned location to an accuracy that's roughly equivalent to a city block.

Edit your manifest to include the following:

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

And to include the reference to the Google Play Services:

```
<application...

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

</application>
```

Run the app.

Maps on Android

Google Maps Android API, can be used to add maps to your application. The API provides functions to handle the access to Google Maps servers, data downloading, map display, and response to map gestures. "Google Maps Android API v2" allows you to add maps into an activity as a fragment with a simple XML snippet. This tutorial shows how to do so.

Requirements

- Android Developer Tools (ADT) (or Eclipse + ADT plugin)
- AVD Android 4.4 – API Level 19 "emulator" or actual device with Android 3.0+
- Google Play Services revision 13
- Android Support Library revision 19

Create Android Application project

Create an application with a target and min_sdk versions adapted to what is mentioned in the requirements.

Add Google Play Services to the App

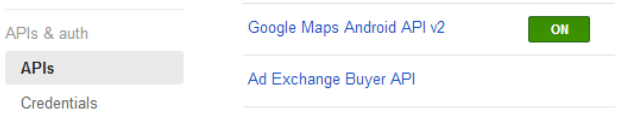
Follow the steps used in the "Location" example.

Obtain Google Maps v2 API key

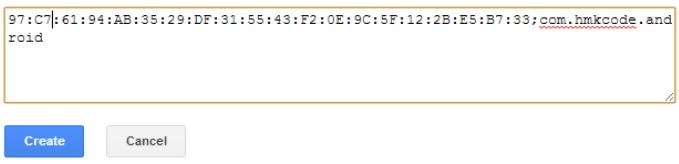
a) Get SHA1 Fingerprint
   a. On Eclispe, Open Window>> Preferences>>Android>>Build>SHA1 fingerprint
   b. Copy your SHA1 Fingerprint
   c. Another option is to use the command line to get the key:

```
keytool -list -v -keystore C:\Users\<your user
name>\.android\debug.keystore -storepass android -keypass
android
```

   d. Go to Google Cloud Console (https://cloud.google.com/console)
   e. Click on "Create Project"
   f. Enter Project Name and Project ID "you may change the default Project ID". Create.
   g. Once created, within the Project page, go to API & Auth >> APIs >> **Google Maps Android API v2** switch ON



   h. Go to API & Auth >> Credentials
   i. Under the Public API Access >> Create New Key
   j. In the "Create a new key" pop-up window select Android key
   k. Enter **SHA1 Fingerprint;package name**



   l. View the Key:

i. API & Auth >> Credentials
ii. Under the Public API Access
iii. Look for **Key for Android Application**

Use the Key

To use the key go to AndroidManifest.xml and add the <meta-data> under <application> as following:

```
    <application.......>
        ..............
        <meta-data
            android:name="com.google.android.maps.v2.API_KEY"
            android:value="AIzaSyD0bnkRinqfKgRcdHAJTovrm9GtY.......... />
    </application>
```

Create a Layout with a Map

One can embed maps into an activity as a fragment just by resorting to the layout XML file. Paste the following code into your layout xml file:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <fragment
          android:id="@+id/map"
          android:layout_width="match_parent"
          android:layout_height="match_parent"
          android:name="com.google.android.gms.maps.MapFragment"/>
</RelativeLayout>
```

Create the code for the application

Nothing needs to be done, other than associating the layout with the activity, for the map to be shown:

```
package xxxx;

import android.os.Bundle;
import android.app.Activity;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Tweaking the manifest

Your manifest should be adapted to offer the required permissions for the Google Maps API to function properly. Particularly, we need to add:

- <use-feature> for OpenGL 2.0
- <uses-permission> set of permissions
- <meta-data> for API Key
- <meta-data> for Google Play services version

Adapt the example below to your own package names:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.test.mapas"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="19" />
    <uses-feature
        android:glEsVersion="0x00020000"
        android:required="true"/>

    <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" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.test.mapas.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <meta-data
            android:name="com.google.android.maps.v2.API_KEY"
            android:value="AIzaSyCyzIXBgH0WPQGq9s9zG9gfWvvC...." />

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

</manifest>
```

Run the App

It should be straightforward on a 3.0+ device.

To run it on the emulator your first need to configure the emulator to do so (follow the instructions in http://hmkcode.com/run-google-map-v2-on-android-emulator/).

Homework

1. Create an app that integrates Maps and Location. In this app, the user should be able to start and stop recording his movement traces. It should then at least: show a map, have a way to start and stop recording a trace, show the traces. All traces should be kept onscreen within a session. The remaining components of the application depend on your creativity.