

Google Maps

Using Google Maps

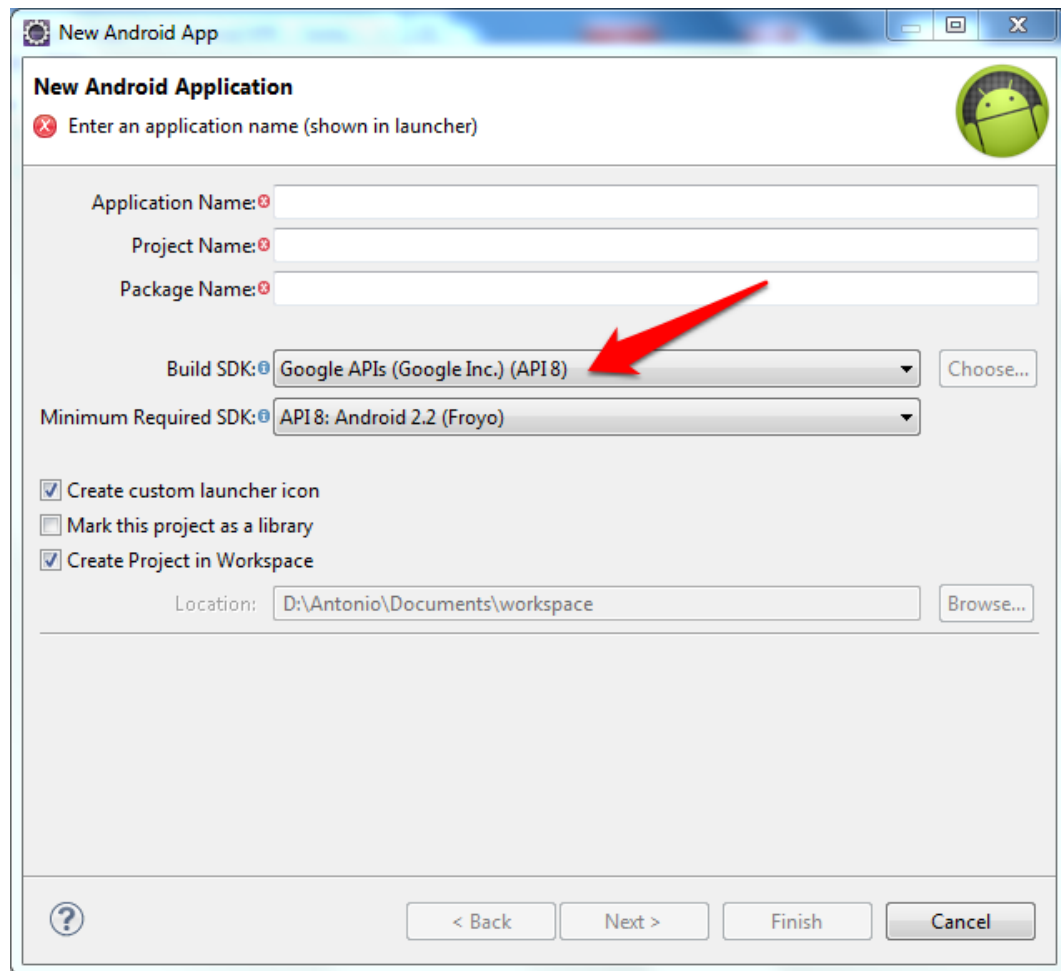
Contents

<u>GOOGLE MAPS API V2.....</u>	<u>3</u>
GETTING THE KEY	3
ADDING THE KEY AND PERMISSIONS TO THE PROJECT	5
ADDING A MAP	6

Google Maps API v2

Create a new Project called "GMaps". If we use Google Maps (integrated in our app, i.e., maps in the app) we have to select the Android API with "Google APIs", and in this case we could select, for example, API 17. By using Google Maps API v2, we will integrate maps in our Android apps, so this example will show a place on a map in our app.

It is important to highlight that it is recommended to use the last version of the Android API because there are some elements only available from version 3.0.



Getting the key

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.

Once we have copied these numbers, we go to the website Google API Console: <https://code.google.com/apis/console/>

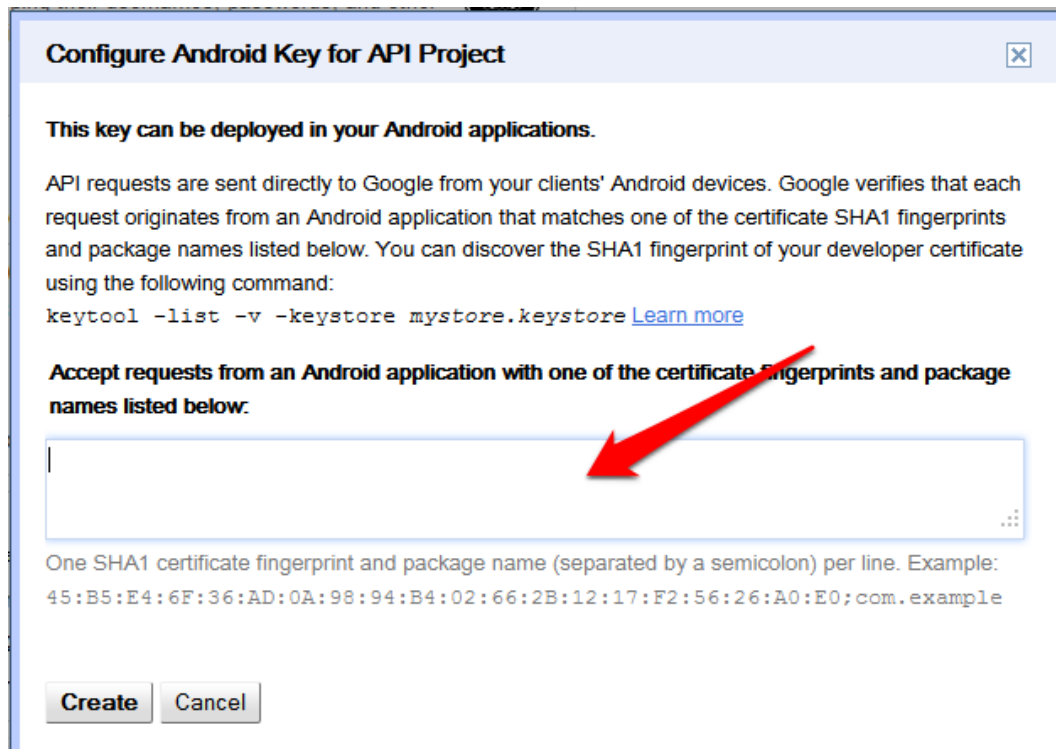
If it is the first time that we visit this webpage, we have to click on the “Create Project” option. Click on “Enable an API” and a list of different APIs will be shown.

In the list we have to activate (i.e., move the switcher to *ON*) the Google Maps Android API v2 service.

Later we go to the “Credentials” option in the left menu, click on the “Create new key” option and then on “Android key”.

In the window we have to insert the SHA1 numbers followed by a semi colon “;” and the package name of our Android project. For example:

BB:0D:AC:74:D3:21:E1:43:07:71:9B:62:90:AF:A1:66:6E:44:5D:75;com.example.sesion2g
maps



Configure Android Key for API Project

This key can be deployed in your Android applications.

API requests are sent directly to Google from your clients' Android devices. Google verifies that each request originates from an Android application that matches one of the certificate SHA1 fingerprints and package names listed below. You can discover the SHA1 fingerprint of your developer certificate using the following command:

```
keytool -list -v -keystore mystore.keystore Learn more
```

Accept requests from an Android application with one of the certificate fingerprints and package names listed below:

One SHA1 certificate fingerprint and package name (separated by a semicolon) per line. Example:
45:B5:E4:6F:36:AD:0A:98:94:B4:02:66:2B:12:17:F2:56:26:A0:E0;com.example

Finally we have to click on the “Create” option.

If this process has been successfully performed, a new key for Android will be shown. The part important for this case is the string called “API Key”.

Adding the key and permissions to the project

Now we have to add this key to the project. To do this, we add the following lines to the *AndroidManifest.xml* file, as a child of the *application* node.

```
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="your_api_key"/>
<meta-data
    android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version" />
```

You have to replace “*your_api_key*” by the key obtained in the website.

Then we have to add the permissions to the project in the *AndroidManifest.xml* file. We have to replace “*com.example.mapdemo*” by the package of our project. These lines have to be included as a child of the *manifest* node.

```
<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_GSERVICES"
    />
<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"/>
```

For more information about these permissions, see the following website:

https://developers.google.com/maps/documentation/android/start#installing_the_google_maps_android_v2_api

Finally we have to add the OpenGL v2 requirement. This is required to correctly execute Google Maps v2.

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

Adding a map

To do this, we have to install the “Google Play services” project by using the SDK Manager in the “extras” tab. If it is not installed we have to install it before performing the next steps.

Once it is installed, we have to import the Google Play services project into Eclipse in this way:

Click on *File -> Import...*, select *Android -> Existing Android Code into Workspace*. We have to find the project, which should be installed in the directory:

```
<android-sdk-folder>/extras/google/google_play_services/libproject/google-play-services_lib
```

Now we have to add the project as a library to our project. Click with the right mouse button on the project and select *Properties -> Android -> Library, Add -> google-play-services_lib*

Once this is done, we will create a new layout and a new activity.

In the layout we have to write the following lines:

```
<?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"
    class="com.google.android.gms.maps.SupportMapFragment"/>
```

This fragment allows showing the map. A fragment in Android (introduced in version 3.0 and higher) is like a subset of an activity.

In the activity we have to write the following lines:

```
package com.example.sesion2gmaps;

import android.os.Bundle;
import android.support.v4.app.FragmentActivity;

public class MapActivity extends FragmentActivity {

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

At this moment we could test the app, and a map will be displayed but without any marker. If the map is not displayed, we have to check the previous steps.

Now we have to create a marker for showing the position of a place. In the “onCreate” method write the following code:

```
//Coordinates of the Real Madrid stadium
double latitude= 40.451519;
double longitude=-3.688960;
String name="Santiago Bernabeu, Real Madrid Stadium";
```

Finally, we create a “*marker*” for showing the place with the latitude and longitude on the map:

```
GoogleMap map = ((SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map)).getMap();
map.addMarker(new MarkerOptions().position(new LatLng(latitude,
longitude)).title(name));
```

The first line allows getting the map defined in the layout file. With this, we will be able to manage the map, i.e., to add markers, to zoom, etc.

The second line adds a new marker in the position indicated by the longitude and latitude. The name will be displayed when the marker is clicked.

Practical exercise 1

Integrate the code shown in this lesson with the practical exercise 1 of the “Connectivity Using Web Services” guide, i.e., show a map with one marker for each result obtained by the web service from the data entered by the user in the EditText.