The following is a detailed description of the set up steps required to enable an Android project to implement a map. The steps are:

Step (1) Get Google Play Services

Step (2) Get your Google API Key

Step (3) Set permissions in the manifest file

Step (4) Enable your app use to use OPENGL

Step (5) Set up an activity and layout to display the map

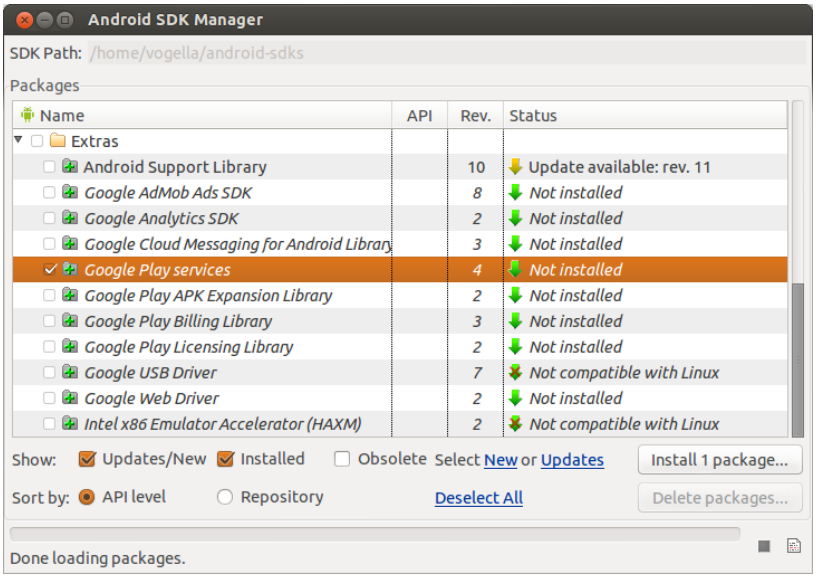
Note: Currently the Android emulator does not support the display of maps. There are workarounds but best practice is to use a real Android device.

Step (1) Get Google Play Services

This Step (1) is also explained online at

https://developers.google.com/android/guides/setup

1a Install Google Play Services– In Android Studio: Tools/ Android SKD/ Extras



1b Add Google Play Services to your workspace

In Android Studio

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

Open the **build.gradle** file inside your application module directory.

Note: Android Studio projects contain a top-level build.gradle file and a build.gradle file for each module. Be sure to edit the file for your application module. See Building Your Project with Gradle for more information about Gradle.

Add a new build rule under dependencies for the latest version of play-services. For example:

```
apply plugin: 'com.android.application'
...

dependencies {
    implementation 'com.google...'
}
```

Be sure you update this version number each time Google Play services is updated.

Be sure you update this version number each time Google Play services is updated. The dependency for Google Maps as at November 2018 **Google Maps:**
com.google.android.gms:play-services-maps:16.0.0

Note: If the number of method references in your app exceeds the 65K limit, your app may fail to compile. You may be able to mitigate this problem when compiling your app by specifying only the specific Google Play services APIs your app uses, instead of all of them. For information on how to do this, see Selectively compiling APIs into your executable.

Save the changes and click Sync Project with Gradle Files in the toolbar.

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"

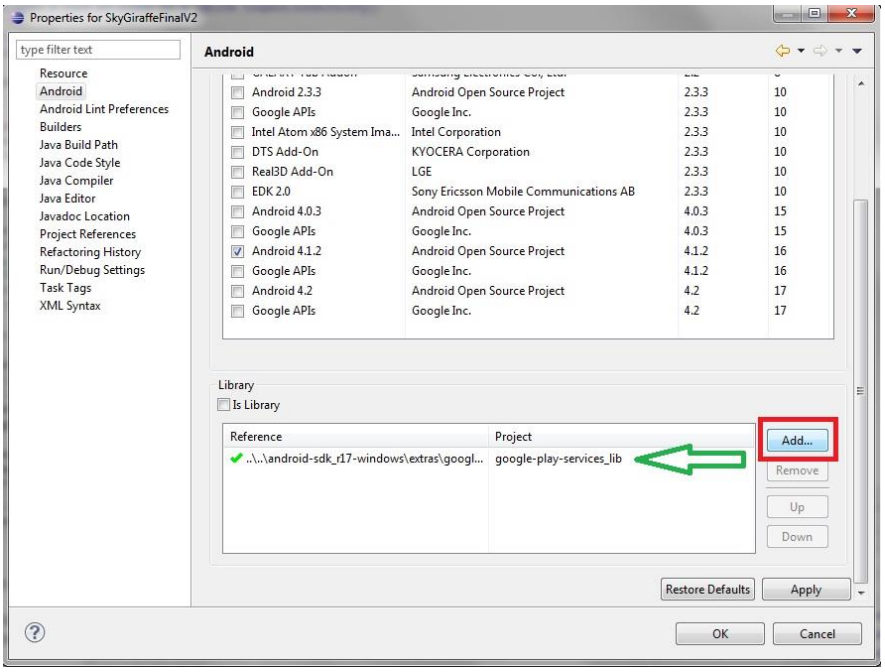
android:value="@integer/google_play_services_version" />

1c Link *your* Android project to Google Play Services Lib.

If using Android studio, you shouldn't need to do this after Step B. But it is included here for IDEs such as Eclipse:

If Using Eclipse When Google Play Services has been successfully added to your work space, create a reference from your project to this library:

Right-Click your project and choose “Properties” ; go to the Android section, in the lower part press the “Add…” button and add a reference to that library – to appear like:

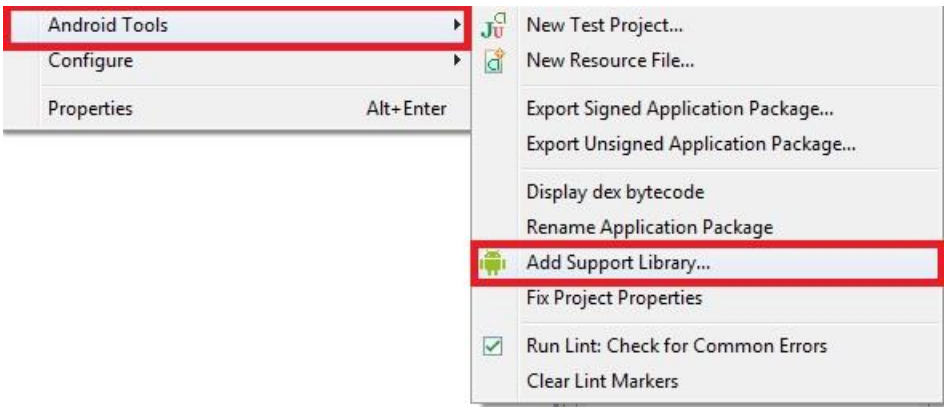


Note 1: Note: If you try to reference google-play-service library and you receive a red X next to this reference, you should move the .jar file to a location where its path will be shorter and then reference it again.

Note 2 IF you want your app to work on systems prior to API11 is to import the support library, you need to do the following:

using Eclipse: Right-Click you project and choose “Android Tools” and then choose

“Add Support Library…” as shown below



END OF Step (1) Google Play Services

Step (2) Get your Google API Key

2a: Retrieve your SHA1 key for your Android environment

For debugging purposes, developers just use the debug.keystore file to get the SHA1 key. This requires the use of the tool 'Keytool' that come with Java installation.

Open you command prompt, head to the following location:

C:\Program Files\Java\<your JDK or JRE instllation>\bin>

Run the next command (make sure to put in the correct path for your set up):

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

The output will look something like:

r

```
C:\Program Files\Java\jdk1.6.0_26\bin>keytool -list -v -keystore C:\Users\Emil\.
android\debug.keystore -storepass android -keypass android

Keystore type: JKS
Keystore provider: SUN

Your keystore contains 1 entry

Alias name: androiddebugkey
Creation date: Jan 22, 2013
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Android Debug, O=Android, C=US
Issuer: CN=Android Debug, O=Android, C=US
Serial number: 51368b77
Valid from: Tue Jan 22 22:49:54 IST 2013 until: Thu Jan 15 22:49:54 IST 2043
Certificate fingerprints:
         MD5:  your MD5 key will be here
         SHA1: <---------  your SHA1 key will be here  ---------->
         Signature algorithm name: SHA256withRSA
         Version: 3
```

Note down/copy the **exact** SHA1 key.

NOTE: Supposedly can now see these in Android environment within Eclipse. Go to Window -> Preferences -> Android -> Build. You'll see the SHA1 and MD5 keys in this window.

There is no way that I am aware of to see the SHA1 key within Android Studio.

2b: Generate an API key for your project using the Google API Console

Go to the Google API Console at:
https://developers.google.com/maps/documentation/android-sdk/signup

Click on the **Get Started** option under Step 1 on the page. Pick "Maps" and follow the steps.

Google now require (as of July 2018) an account with a credit card to get an API key. This is really cumbersome for students. However, you won't get charged for the level you'll use it for a CA project as you get free credit to your "acccount2.

The following video takes you through the steps:
https://elfsight.com/blog/2018/06/how-to-get-google-maps-api-key-guide/

You will need to your SHA1 key.

2c: Add the generated Google API key into the manifest file

Put this in just before the closing application tag in the manifest file: </application> - remembering to put in your actual key from step 2B as the value!

```
<meta-data

android:name="com.google.android.geo.API_KEY"

android:value="Your Google Maps API Key" />
```

END OF Step (2) Get your Google API key.

Step (3) Set permissions in the manifest file

Add the following permissions to the manifest file for your project. Replace packagename will your package name e.g `com.test.map`

```
<permission android:name="packagename.permission.MAPS_RECEIVE"
             android:protectionLevel="signature"/>

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

<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" />
```

Step (4) Enable your app use to use OPENGL

Google Maps uses OpenGL to render maps. A dd OpenGL support to our application by adding this to the Manifest file:

```
<uses-feature

android:glEsVersion="0x00020000"

android:required="true"/>
```

Step (5) Set up an activity and a layout, as per notes. Test with a real device


Map
Satellite
Hybrid
Terrain
Dunshaughlin
Ashbourne
Donabate
Applewood
Swords
Malahide
Portmarnock
Clonee
Northwest
Business Park
Baldoyle
Ballymun
Blanchardstown
Howth
Raheny
Maynooth
Drumcondra
Lucan
Chapelizod
Dublin
Celbridge
Ballsbridge
Clondalkin
Straffan
Terenure
Newcastle
Blackrock
Clane
Rathcoole
Tallaght
Sandyford
Glenageary
Sallins
Kill
Carrickmines
Shankill
Naas
Aylesbury
Bray
Enniskerry