Location Manager & Map

Sisoft Technologies Pvt Ltd
SRC E7, Shipra Riviera Bazar, Gyan Khand-3, Indirapuram, Ghaziabad
Website: www.sisoft.in Email:info@sisoft.in
Phone: +91-9999-283-283

Location API

1. Location API
2. LocationManager
3. LocationProvider
4. Checking the Status of Location Provider
5. Prompt the user to Enabled Location Providers(GPS)
6. Selecting LocationProvider via Criteria
7. Getting last know location and location updates
8 Forward and reverse Geocoding
9. Proximity Alert
10. Testing on Emulator
    - 8.1Activating GPS on the emulator
    - 8.2 Setting the geoposition

Location API

- Most Android devices allow to determine the current geolocation. This can be done via a GPS (Global Positioning System) module, via cell tower triangulation or via wifi networks.

- Android contains the android.location package which provides the API to determine the current geo position.

LocationManager

- The LocationManager class provides access to the Android location service. This services allows to access location providers, to register location update listeners and proximity alerts and more
- As with other system services, you do not instantiate a LocationManager directly. Rather, you request an instance from the system by calling getSystemService(Context.LOCATION_SERVICE). The method returns a handle to a new LocationManager instance.

Permissions

- If you want to access the GPS sensor, you need the ACCESS_FINE_LOCATION permission. Otherwise you need the ACCESS_COARSE_LOCATION permission.

Location Provider

- The LocationProvider class is the superclass of the different location providers which deliver the information about the current location. This information is stored in the Location class.
- The Android device might have several LocationProvider available and you can select which one you want to use. In most cases you have the followng LocationProvider available.

Table 1. LocationProvider

LocationProvider	Description
network	Uses the mobile network or WI-Fi to determine the best location. Might have a higher precision in closed rooms then GPS.
gps	Use the GPS receiver in the Android device to determine the best location via satellites. Usually better precision than network.
passive	Allows to participate in location of updates of other components to save energy

Location Provider

- **gps** –> (GPS, AGPS)
  Name of the GPS location provider. This provider determines location using satellites. Depending on conditions, this provider may take a while to return a location fix. Requires the permission android.permission.ACCESS_FINE_LOCATION.
- **network** –> (AGPS, CellID, WiFi MACID)
  Name of the network location provider. This provider determines location based on availability of cell tower and WiFi access points. Results are retrieved by means of a network lookup. Requires either of the permissions android.permission.ACCESS_COARSE_LOCATION or android.permission.ACCESS_FINE_LOCATION.
- **passive** –> (CellID, WiFi MACID)
  A special location provider for receiving locations without actually initiating a location fix. This provider can be used to passively receive location updates when other applications or services request them without actually requesting the locations yourself. This provider will return locations generated by other providers. Requires the permission android.permission.ACCESS_FINE_LOCATION, although if the GPS is not enabled this provider might only return coarse fixes.

Checking the status of Location Provider

- For checking the status, if a LocationProvider is enabled via the isProviderEnabled() method
- If its not enabled, User may be directed to the settings via an Intent with the Settings.ACTION_LOCATION_SOURCE_SETTINGS action for the android.provider.Settings class.

Prompt the user to Enabled GPS

```
LocationManager service = (LocationManager) getSystemService(LOCATION_SERVICE);
boolean  enabled = service .isProviderEnabled(LocationManager.GPS_PROVIDER);

// Check if enabled and if not send user to the GPS settings // Better solution would be to
display a dialog and suggesting to // go to the settings

 if (!enabled)
{
   Intent intent = new Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
  startActivity(intent);
}
```

Selecting Location Provider via Criteria

- A class indicating the application criteria for selecting a location provider. Providers maybe ordered according to accuracy, power usage, ability to report altitude, speed, and bearing, and monetary cost.

  Criteria criteria = **new Criteria();**

  - **public void setAccuracy (int accuracy) :**Indicates the desired accuracy for latitude and longitude. Accuracy may be ACCURACY_FINE if desired location is fine, else it can be ACCURACY_COARSE. More accurate location may consume more power and may take longer.
  - **public void setCostAllowed (boolean costAllowed):** Indicates whether the provider is allowed to incur monetary cost.
  - **public void setPowerRequirement (int level) :** Indicates the desired maximum power level. The level parameter must be one of NO_REQUIREMENT, POWER_LOW, POWER_MEDIUM, or POWER_HIGH.

- Selecting the location provider on above defined criteria

  String provider = locMgr.getBestProvider(criteria, enabledOnly(true or false)**);**

Getting Last Known Location

- On getting the Location Provider, you request for last known location by the provider

```
Location loc=locMgr.getLastKnownLocation(provider);
```

- A location can consist of a latitude, longitude, timestamp, and other information such as bearing, altitude and velocity
- All locations generated by the LocationManager are guaranteed to have a valid latitude, longitude, and timestamp (both UTC time and elapsed real-time since boot), all other parameters are optional

```
if (location != null)
{
        latitude = location.getLatitude();
        longitude = location.getLongitude();
}
```

Receive periodic location updates

- You can register a LocationListener object with the LocationManager class to receive periodic updates about the geoposition.

  requestLocationUpdates(String provider, long minTime, float minDistance, LocationListener listener)

- LocationListener is interface used for receiving notifications, its method

  public void onLocationChanged(Location loc)

  should be implemented to handle updates

Geocoding: android.location.Geocoder

- The Geocoder class allows to determine the geo-coordinates (longitude, laditude) for a given address and possible addresses for given geo-coordinates.
- Geocoding is the process of transforming a street address or other description of a location into a (latitude, longitude) coordinate.
- Reverse geocoding is the process of transforming a (latitude, longitude) coordinate into a (partial) address.
- Method **getFromLocationName(String locationName, int maxResults)** returns are array of Addresses that describe the named location. These addresses contains the latitude and longitude of the location.
- Method **getFromLocation(double latitude, double longitude, int maxResults)** returns an array of addresses that are known to describe the area immediately surrounding the given latitude and longitude. The returned addresses will be localized for the locale provided to this class's constructor

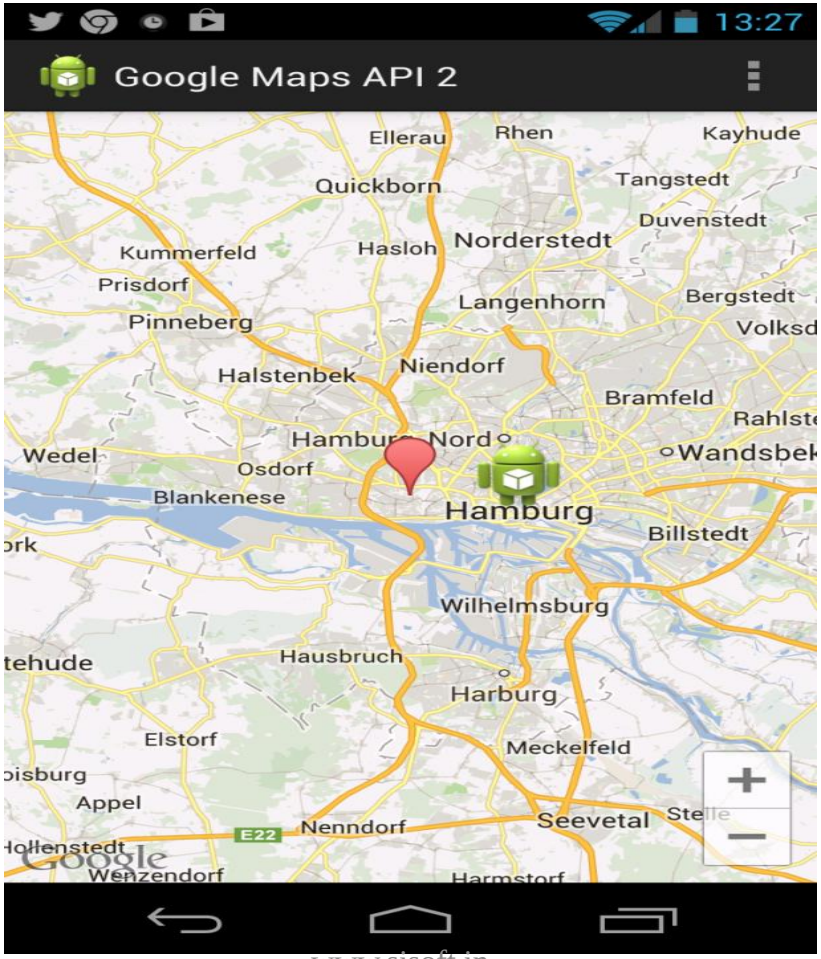
Proximity Alert

- Alerts that get generated when the user is physically located near a specific Point Of Interest (POI)
- This is done using method **addProximityAlert(double latitude, double longitude, float radius, long expiration, PendingIntent intent)**
  - The latitude and longitude define the co-ordinate of the location
  - Radius is the range for which alert should work.
  - You can also register an Intent which allows to define a proximity alert, this alert will be triggered if the device enters a area given by a longitude, latitude and radius (proximity alert)
  - The fired Intent will have a boolean extra added with key KEY_PROXIMITY_ENTERING. If the value is true, the device is entering the proximity region; if false, it is exiting.
  - After the number of milliseconds given by the expiration parameter, the location manager will delete this proximity alert and no longer monitor it. A value of -1 indicates that there should be no expiration time.

Testing on Emulator

- Goto DDMS Perspective
- Goto Emulator Control
- There is Location Controls in lower part of the window. GPS of emulator can be mocked from this place

Android Google Map

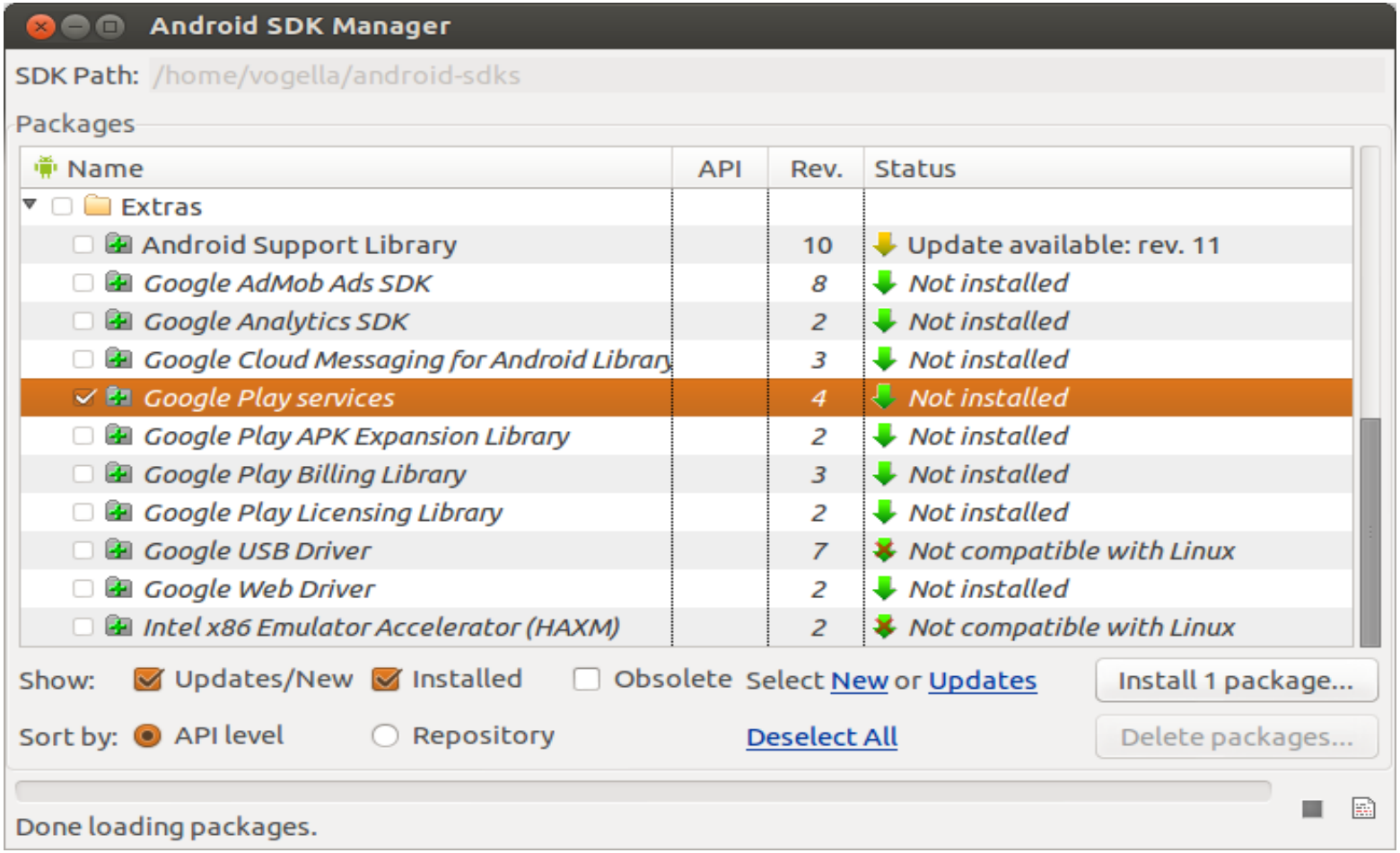


Google Maps

- Google Play Services
  - Install Google Play Services
  - Register with the Google APIs Console
  - Getting the Google Map key
  - Creating the SHA-1 for your signature key
  - Create key for your application
- Google Maps
  - MapsView
  - MapFragment
  - Markers
  - Changing the GoogleView

Install Google Play services

- Open the Android SDK Manager and install Extras → Google Play services



Setup Google Play Services SDK

- The Google Play services SDK is saved in your Android SDK environment at <android-sdk>/ extras/google/google_play_services/
- Import the library project(google_play_services_lib) into your workspace Import -> Project -> Android -> Existing Android Code into Workspace
- Modify build path (Right Click App-> Select Properties -> Java Build Path) to include the recently added jar (google-play-services.jar) [located under the lib dir of the google-play-service-lib project]
- Modify Project properties->Android-> Add in Library google play services lib
- 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" />
```

4.Getting the Google Map key

- **Create an API project in the Google Developer Console**
- **Obtain a Google Maps API key**
- **Add the API key to your application**

Register with the Google Developer Console

- https://console.developers.google.com/project
- Create new project
- Select the Google MAP API and Enable the API.

Register with the Google Developer Console

- https://console.developers.google.com/project
- You have to register in the *Google APIs Consolе* that you want to use Google Maps for Android. You can reach this console via the following link: Google APIs Console . Select here the Services entry.

Enable Google Map APIs Console

- API & auth -> APIs
- Activate the Google Maps Android API v2.

- API & auth -> Credentials -> Create new Key

SHA-1 fingerprint for your signature key

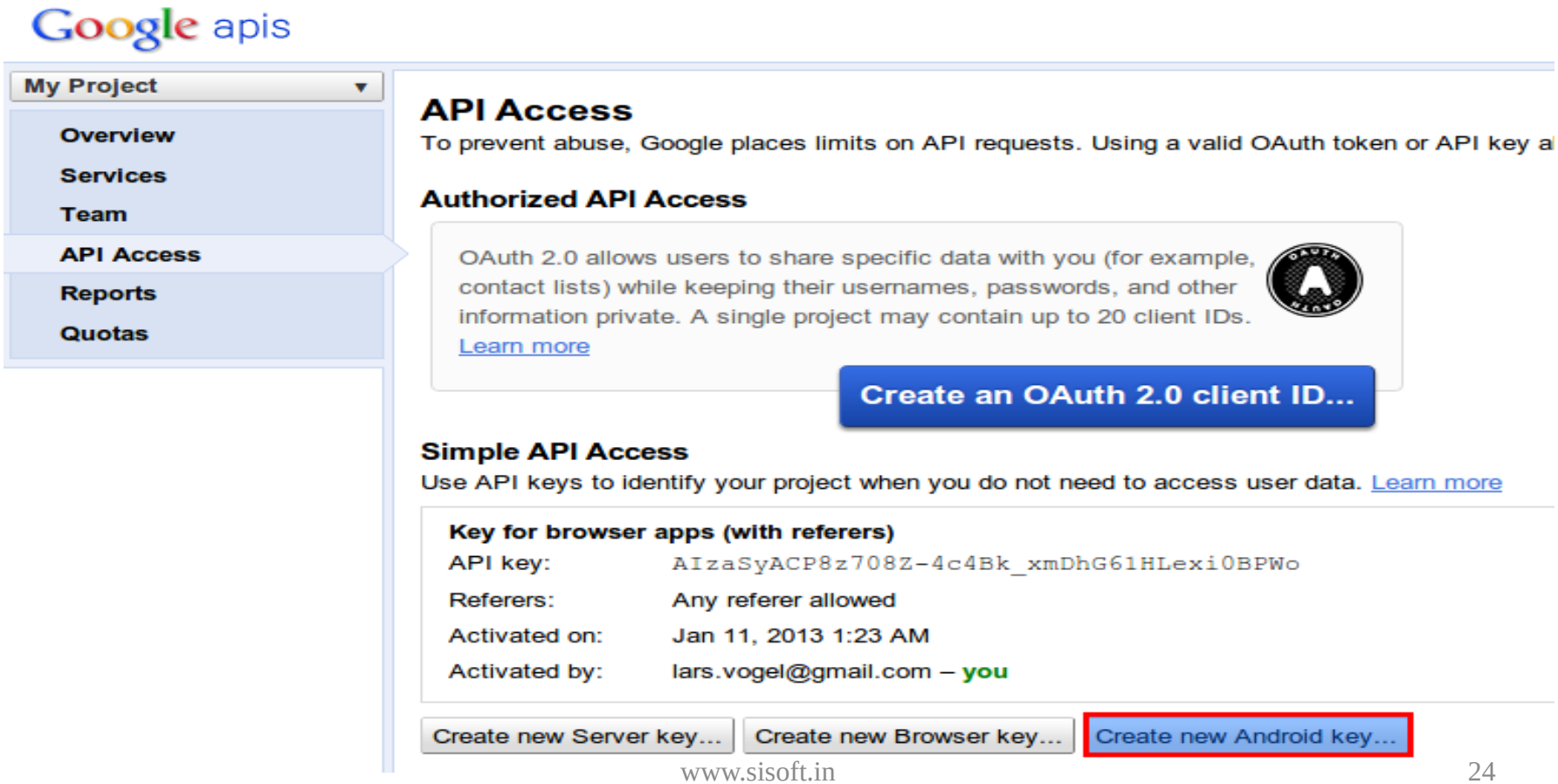
- The Eclipse debug key for signing your application can be found in the userhome/.android/debug.keystore file.
- To create the SHA-1 for your debug keystore you use the keytool command from your JDK installation pointing to the debug.keystore file.

```
keytool -list -v -alias androiddebugkey  -keystore
<path_to_debug_keystore>debug.keystore  -storepass android -keypass
android
```

- Copy the SHA-1 output, as you need this later.
- Or Windows->Preferences->Android->Build (Take SHA1 fingerprint)

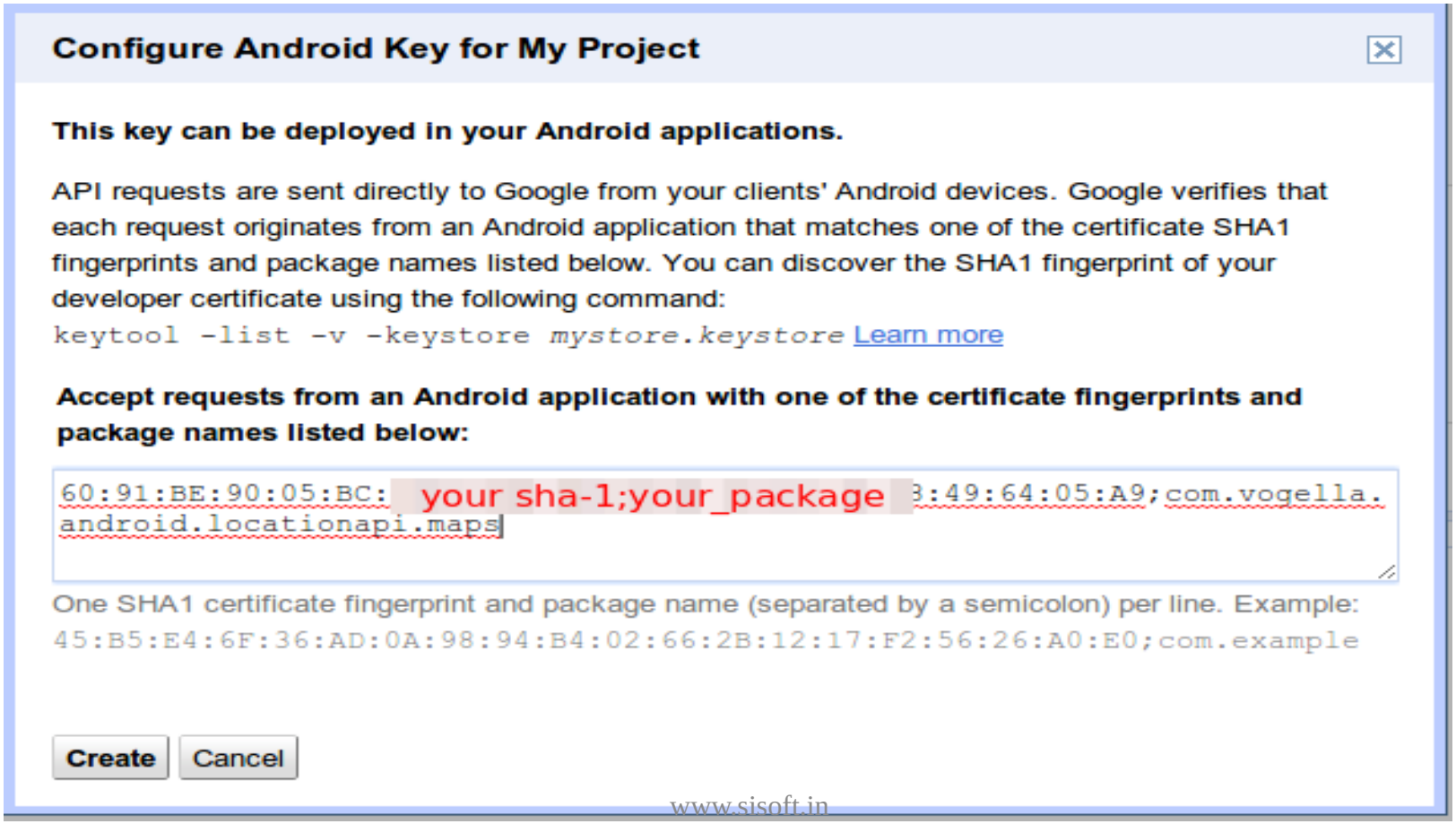
4.4. Create key for your application

You need later to register your application via its package in this console together with the SHA-1 fingerprint of your signature key. For this you select the entry and click on the API Access entry. Afterwards click on the Create new Android key... entry.



Google Maps Demo

- Enter your SHA-1 fingerprint and the package of your application separated by a semicolon. For example you can use the com.vogella.android.locationapi.maps package.



Add the API key to your application

- 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.

Add the API key to your application

- 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.

Specify requirement for OpenGL ES version 2

- The Google Maps Android API uses OpenGL ES version 2 to render the map. If OpenGL ES version 2 is not installed, your map will not appear.
- <uses-feature> element should be added as a child of the <manifest> element in AndroidManifest.xml:

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

- This notifies external services of the requirement. In particular, it has the effect of preventing Google Play Store from displaying your app on devices that don't support OpenGL ES version 2.

Google Maps : MapsView

- Google provides via Google play a library for using Google Maps in your application. The following description is based on the Google Maps Android API v2 which provides significant improvements to the older API version.
- The library provides the com.google.android.gms.maps.MapFragment class and the MapView class for displaying the map component.

MapFragment

- The MapFragment class extends the Fragment class and provides the life-cycle management and the services for displaying a GoogleMap widget. GoogleMap is the class which shows the map. The MapFragment has the getMap() method to access this class.
- the LatLng class can be used to interact with the GoogleView class

Markers

- Marker is an icon placed at a particular point on the map's surface. It can be created on map using *Marker* Class
- The LatLng class can be used to interact with the map

```
// get a handle to the Map Fragment
GoogleMap map = ((MapFragment) getFragmentManager()
                .findFragmentById(R.id.map)).getMap();
// Add a marker at Sisoft.
Marker marker = map.addMarker(new MarkerOptions()
    .position(new LatLng(28.6487628, 77.3578666))
    .title(“Sisoft")
    .snippet(“Mobility Training Center"));
```

Google maps : permissions

- ***ACCESS_NETWORK_STATE*** – To check network state whether data can be downloaded or not
- ***INTERNET*** – Used by the API to download map tiles from Google Maps servers
- ***WRITE_EXTERNAL_STORAGE*** – To write to external storage as google maps store map data in external storage
- **com.google.android.providers.gsf.permission.READ_GSERVICES**: Allows the API to access Google web-based services.
- ***ACCESS_COARSE_LOCATION*** – To determine user's location using WiFi and mobile cell data
- ***ACCESS_FINE_LOCATION*** – To determine user's location using GPS

Google Maps Demo: Manifest File


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

   <uses-sdk  android:minSdkVersion="8“   android:targetSdkVersion="18" />

   <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"/>

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

Google Maps Demo: Manifest File


```
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name="in.sisoft.demo_map.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.gms.version"
     android:value="@integer/google_play_services_version" />

<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="AIzaSyB6gE3SsbZ_X1aeFwnhSE-thzq5hfGWuBw"/>
  </application>  </manifest>
```

.XML

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

Activity.java

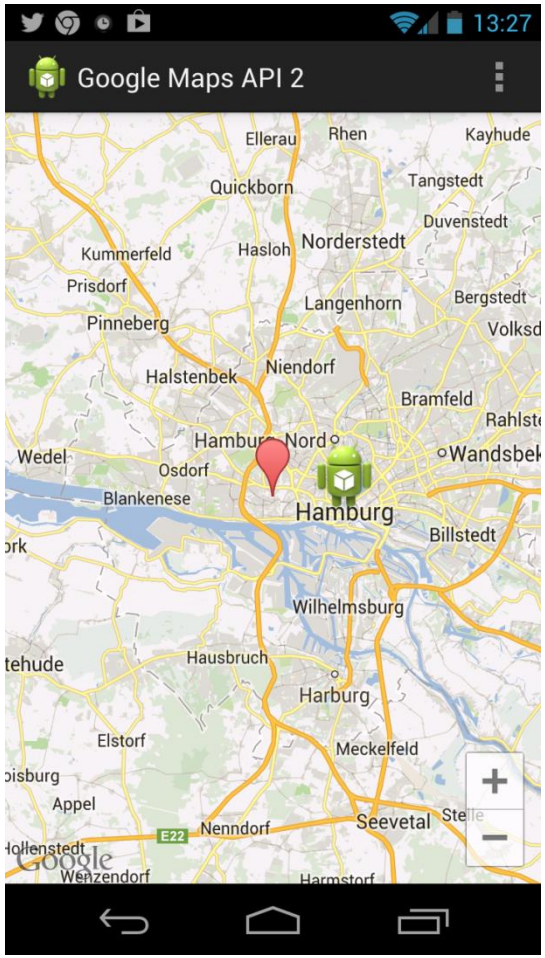
```
package in.sisoft.demo_map;

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

public class MainActivity extends Activity {

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

RUN


13:27
Google Maps API 2
Ellerau
Rhen
Kayhude
Quickborn
Tangstedt
Duvenstedt
Kummerfeld
Hasloh
Norderstedt
Prisdorf
Langenhorn
Bergstedt
Pinneberg
Halstenbek
Niendorf
Bramfeld
Wedel
Osdorf
Nord
Wandsbek
Blankenese
Hamburg
Billstedt
Wilhelmsburg
Hausbruch
Harburg
Elstorf
Meckelfeld
Appel
Nenndorf
Seevetal
E22
Google