



Programming with Android: The Google Maps Library

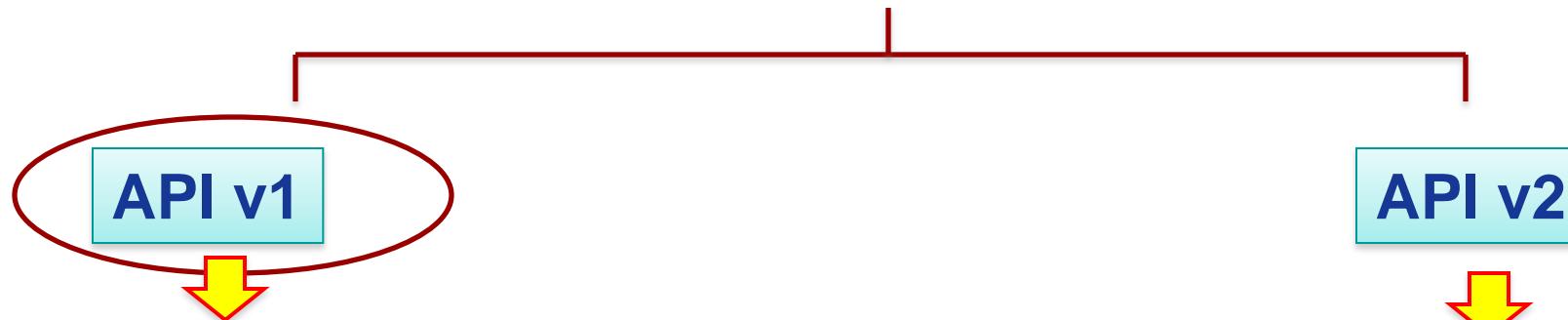
Slides taken from

Luca Bedogni

Marco Di Felice

Android: Deploying Map-based Apps

Two versions of Android Google Maps API



- Deprecated, not supported anymore since 18th March 2013.
- Still used for Android device with versions < 3.0 (unless API set is extended with support packages)

- Different installation procedures.
- Novel methods to insert a Map inside an Android app.
- Improved caching and visualization capabilities.

Android: Deploying Map-based Apps

Two versions of Android Google Maps API

API v1



- Deprecated, not supported anymore since 18th March 2013.
- Still used for Android device with versions < 3.0 (unless API set is extended with support packages)

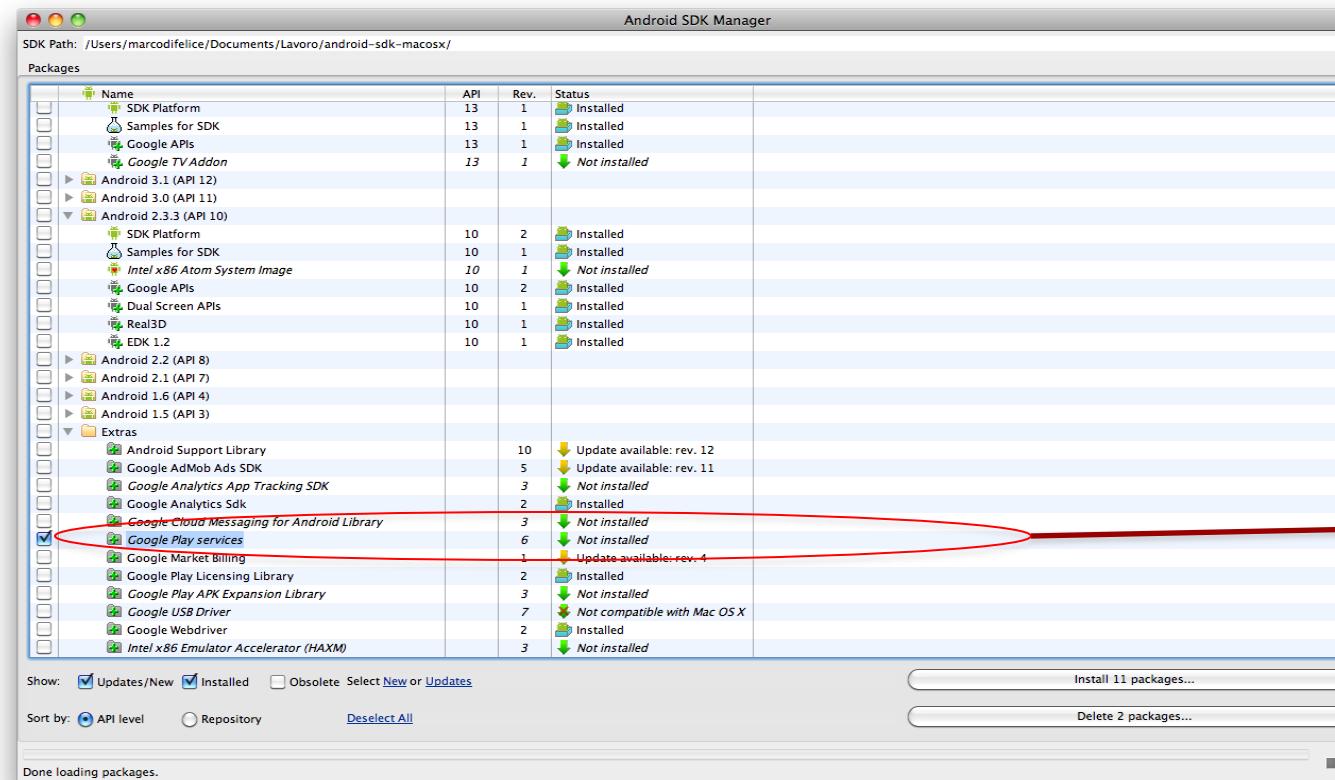
API v2



- Different installation procedures.
- Novel methods to insert a Map inside an Android app.
- Improved caching and visualization capabilities.

Android: Installing Google APIs

STEP -1: Install and Setup Google Play Service SDK



Window → Android SDK Manager
→ Installed packages

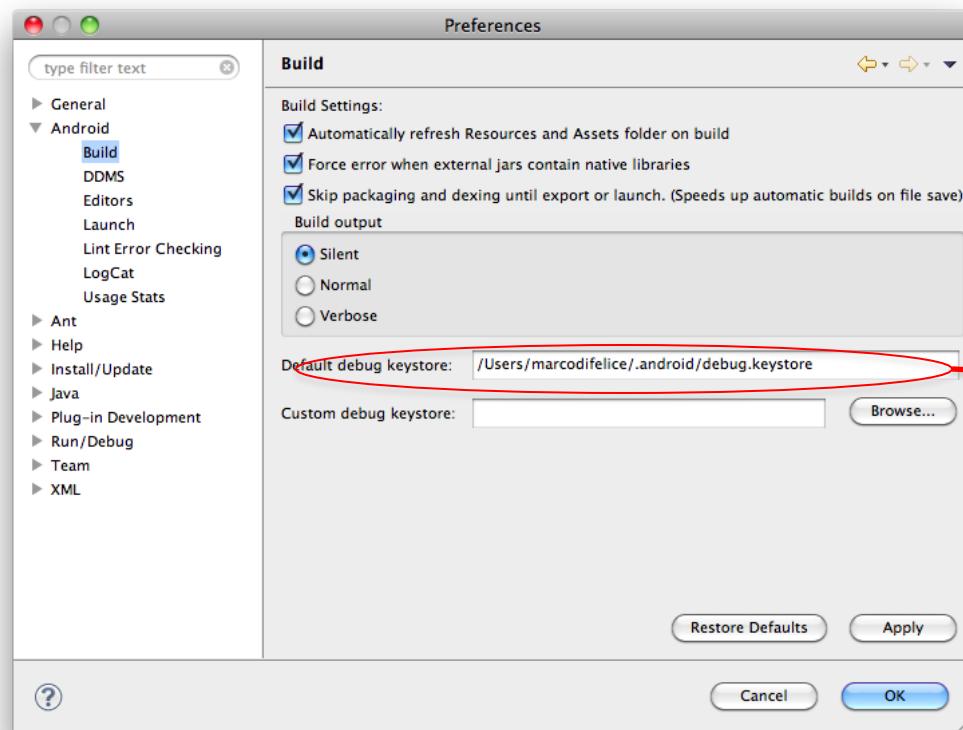
Check Google Play is **installed**,
or **install** it otherwise

<http://developer.android.com/google/play-services/setup.html>

Android: Getting a Google Maps API Key

STEP 0: Get a valid Maps API Key to utilize the Google Maps library.

0.1: Retrieve the fingerprint MD5 of the certificate used to sign the apps.



Window → Preferences → Android
→ Build

Get the debug **keystore** path

Android: Getting a Google Play API Key

STEP 0: Get a valid Google Play API Key to utilize the Google Maps library.

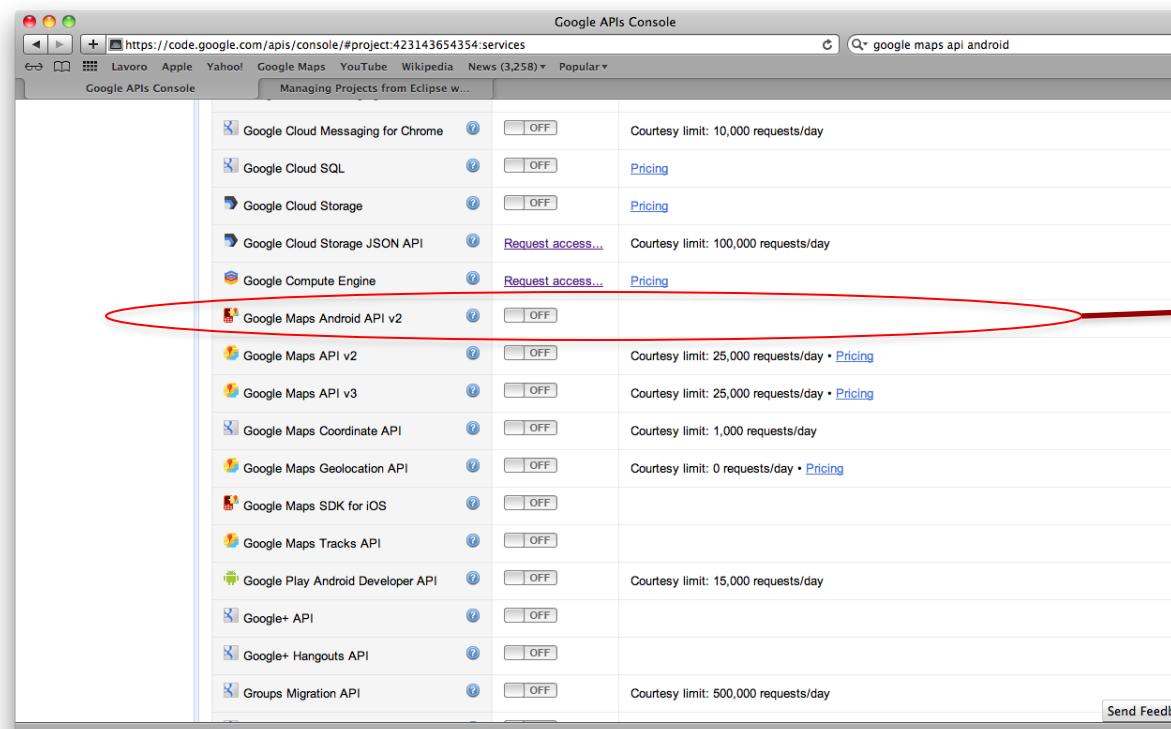
0.1: Retrieve the fingerprint SHA1 of the certificate used to sign the apps.

```
mylaptop:~ marco$ keytool -list -keystore  
/Users/marcodifelice/.android/debug.keystore -  
storepass android -keypass android  
...  
androiddebugkey, Feb 1, 2011, PrivateKeyEntry,  
Certificate fingerprint (SHA1):  
A2:34:B1:A3:A5:BB:11:21:21:B3:20:56:92:12:AB:DB
```

Android: Getting a Google Play API Key

STEP 1: Navigate with a browser to <https://code.google.com/apis/console/>

1.1: Select the Google service you intend to use for your apps.



Enable Google Maps
Android v2 API

Android: Getting a Google Play API Key

STEP 1: Navigate with a browser to <https://accounts.google.com/>

1.2: Get an Google Play API Activation Key

- Select the API Acess
- Insert the SHA1 Key, followed by the package's name:

BB:0D:AC:74:D3:21:E1:43:67:71:9B:62:91:AF:A1:66:6E:44:5D:75;
com.example.android.mapexample

- Generate and save the obtained Activation Key
- For each application/package → get a new Activation Key.**

Android: Google MAPs library overview

What can I do with Google MAPs v2 library in Android?

1. Integrate a Google Map into an Android application

1. Manage the camera

1. Add information layers to the Map

1. Manage user events

Android: Inserting a Map inside the App

Permissions should be added to the `AndroidManifest.xml`, and the Activation Key must be specified in the meta-data.

- Internet Access
- Localization capabilities
- Access to Google Web services
- OpenGL ES version 2 libraries
- Access to network state

Android: Inserting a Map inside the App

Permissions should be added to the AndroidManifest.xml, and the Activation Key must be specified in the meta-data.

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

```
<permission  
    android:name="com.example.mapdemo.permission.MAPS_RECEIVE"  
    android:protectionLevel="signature"/>  
<uses-permission  
    android:name="com.example.mapdemo.permission.MAPS_RECEIVE"/>  
<uses-feature  
    android:glEsVersion="0x00020000"  
    android:required="true"/>
```

Android: Inserting a Map inside the App

Permissions should be added to the AndroidManifest.xml, and the Activation Key must be specified in the meta-data.

```
<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"/>
```

Android: Inserting a Map inside the App

In order to insert a Google Map into a mobile Application:

- Add a **MapFragment** to the current Activity:

```
<?xml version="1.0" encoding="utf-8"?>
<fragment
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.MapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Android: Fragments

Fragment → A portion of the user interface in an Activity.

Introduced from **Android 3.0 (API Level 11)**

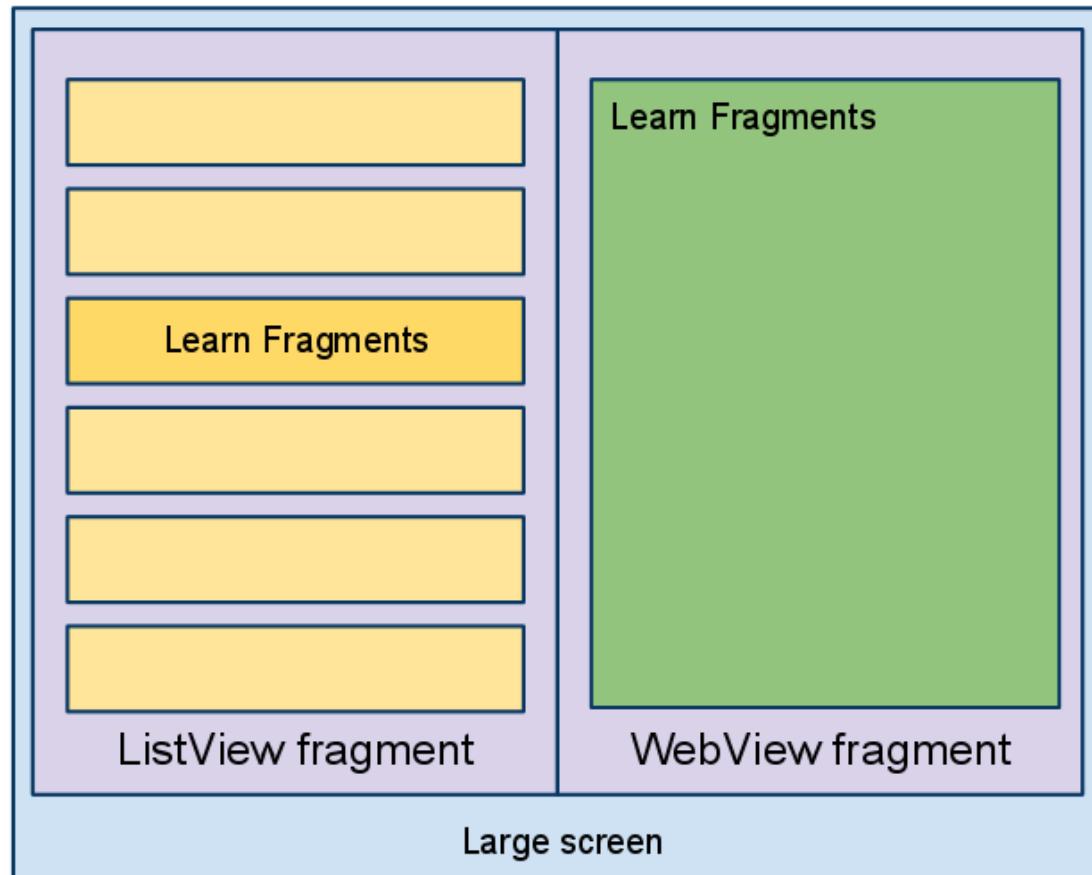
Practically, a **Fragment** is a modular section of an Activity.

DESIGN PHILOSOPHY

- **Structure** an Activity as a collection of Fragments.
- **Reuse** a Fragment on different Activities ...

Android: Fragments Design Philosophy

EXAMPLE: Structuring an Application using 1 Activity and 2 Fragments.



Android: Inserting a Map inside the App

A MapFragment is a container of the **GoogleMap** object, which is a View containing the map and managing the events.

```
private GoogleMap mMap;  
...  
mMap = ((MapFragment)  
getFragmentManager().findFragmentById(R.id.map)).getMap();
```

Differences with **Android Maps v1 libs**:

- No need to use a MapActivity, use a regular Activity instead.
- Improved caching and drawing functionalities.

Android: **Customize** the Map

How to customize the Google Map?

- Define the **Map type**, governing the overall representation of the map

```
nMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
```

Normal → Typical road map.

Hybrid → Satellite photograph data with road maps added.

Satellite → Satellite photograph data. Road and feature labels are not visible.

Terrain → Topographic data. The map includes colors, contour lines and labels, and perspective shading.

None → no tiles, empty grid.

Android: Customize the Map

The **LatLng** class allows to define a point on the map, expressed through the latitude/longitude coordinates.

```
private static final LatLng BOLOGNA_POINT = new  
LatLng(44.496781,11.356387);
```

```
private static final LatLng FLORENCE_POINT = new  
LatLng(43.771373,11.248069);
```

LatLng class (API v2) → **Geopoint** class (API v1)

Android: **Customize** the Map

Developers can handle the **events** on the Google Map.

Events are managed through the **listener mechanism** seen so far ...

CLICK events → Implement the `OnMapClickListener` interface and the `OnMapLongClickListener` method.

CAMERA events → Implement the `OnCameraChangeListener` interface and the `onCameraChange(CameraPosition)` method.

Android: Customize the Map

Developers can handle the **events** on the Google Map.

```
public class MainActivity extends Activity
    implements OnMapClickListener {
private GoogleMap mMap;

protected void onCreate(Bundle savedInstanceState) {
...
mMap.setOnMapClickListener(this);
...

public void onMapClick(LatLng position) {
    // Handle the click events here ...
}
```

Android: **Customize the Map**

How to customize the Google Map?

- Define the **properties of the Camera** applied to the Map.

Location → expressed in forms of latitude/longitude coordinates.

Zoom → defines the scale levels of the map.

Bearing → defines the map orientation, i.e. the direction in which a vertical line on the map points, measured in degrees clockwise from north.

Tilt → viewing angle, measured as degrees from the nadir.

Android: **Customize the Map**

How to customize the Google Map?

- Define the **properties of the Camera** applied to the Map.

Location → expressed in forms of latitude/longitude coordinates.

Zoom → defines the scale levels of the map.

Bearing → defines the map orientation, i.e. the direction in which a vertical line on the map points, measured in degrees clockwise from north.

Tilt → viewing angle, measured as degrees from the nadir.

Android: Customize the Map

Camera properties can be set individually, or collectively through the **CameraPosition** object.

```
private static final LatLng BOLOGNA_POINT = new  
LatLng(44.496781,11.356387);
```

```
CameraPosition cameraPosition = new CameraPosition.  
Builder()  
.target(BOLOGNA_POINT)  
.zoom(17)  
.bearing(90)  
.tilt(30)  
.build();
```

Android: Customize the Map

Two methods to modify the position of the camera:

```
mMap.moveCamera(cameraPosition);
```

- Update the camera properties immediately.

```
mMap.animateCamera(cameraPosition);
```

```
mMap.animateCamera(cameraPosition, duration, call);
```

- Update the camera properties through an animation, eventually adding a delay and a callback to be invoked when the animation stops.

Android: **Customize** the Map

Markers can be used to identify locations on the GoogleMap.

Markers can be customized in terms of:

- **Icon** to be displayed
- **Position** of the marker on the map
- **Title** and text to be displayed
- **Events** to be managed

Android: Customize the Map

Markers can be used to identify locations on the GoogleMap.

```
private static final LatLng BOLOGNA_POINT = new  
LatLng(44.496781,11.356387);  
  
Marker bologna =  
myMap.addMarker(newMarkerOptions().position(BOLOGNA_  
POINT));  
  
Marker bologna= mMap.addMarker(new MarkerOptions()  
    .position(Bologna)  
    .title("Bologna downtown")  
    .snippet("visit the city centre"));
```

Android: **Customize** the Map

Markers can be used to identify locations on the GoogleMap.

EVENTS associated to a Marker:

ClickEvents → implement the `OnMarkerClickListener` interface, and the `onMarkerClick(Marker)`method.

DragEvents → implement the `OnMarkerDragListener` interface, and the `onMarkerDragEnd(Marker)`method.

InfoWindow Click Events → implement the `onInfowindowClickListener` interface, and the `onInfowindowClick(Marker)`method.

Android: **Customize the Map**

Shapes can be used to identify sections of the GoogleMap.

Polylines → define a set of LatLong objects, and connect them through a set of lines. Possible to define the stroke and colors of the lines.

Polygons → define a set of LatLong objects, and connect them through a closed polygon. Possible to define the stroke and colors of the lines.

Circles → define a LatLong object and a radius, and draw a circle centered at the point. Define pen color/stroke as above.

Android: Customize the Map

Shapes can be used to identify sections of the GoogleMap.

```
PolygonOptions rectOptions = new PolygonOptions()
    .add(BOLOGNA_P1)
    .add(BOLOGNA_P2)
    .add(BOLOGNA_P3);
Polygon polyline = mMap.addPolygon(rectOptions);
```

```
CircleOptions circleOptions = new CircleOptions()
    .center(BOLOGNA_P1)
    .radius(1000)
    .strokeColor(Color.RED);
```

```
Circle circle = mMap.addCircle(circleOptions);
```