Google Maps

- Features
- Prerrequisites
- Adding a Map
- Starting the Map
- Markers
- Shapes
- Camera updates
- Events
- More...

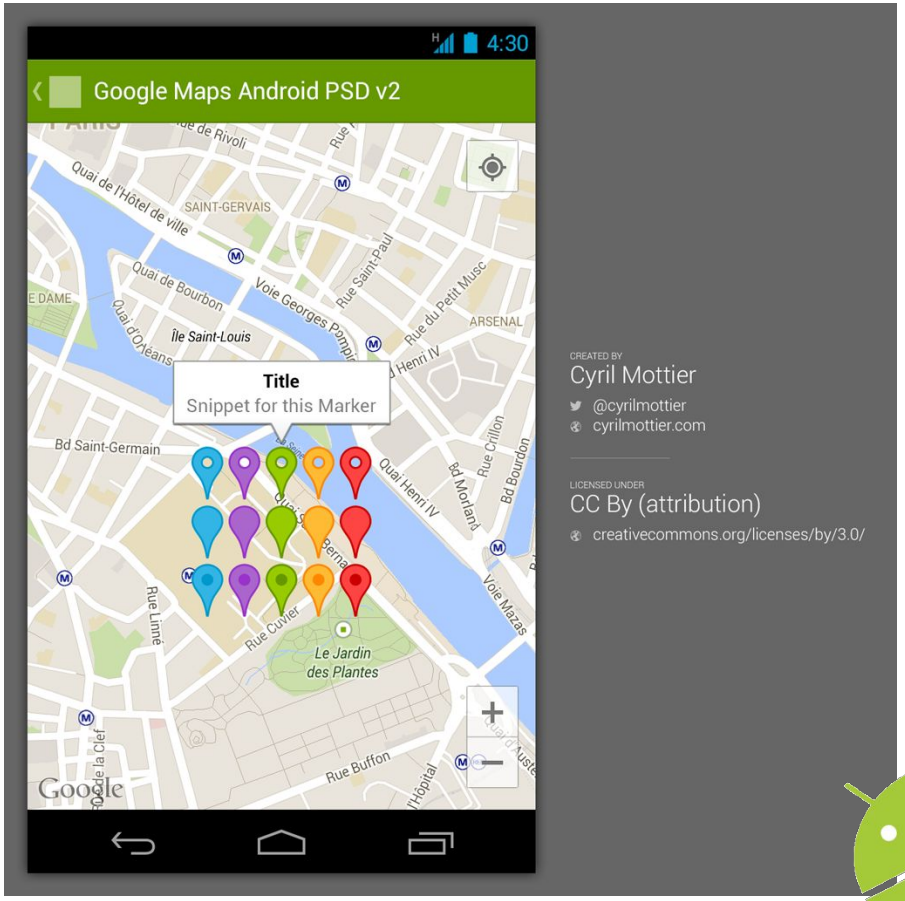
Google Maps Features

Adds maps views and interaction to your app

Automatically connects to Gmaps Servers

Has an API to:

- Add markers
- Add shapes (lines, polygons...)
- Add Overlays
- Change the Camera view
- Animate the camera view
- Manage the events on the map
  - Clicks on map
  - Clicks in markers
  - Gestures



Prerrequisites

For developing:

- The project must have the Google Play Services dependency
- You must get a GoogleMaps Api Key → specific per app

For the emulator:

- Have an image with Google Play Services

For the final device:

- Must have the GooglePlay Services app installed → if not, the user is asked to install instead of viewing the Map

Adding a Map

Add a Map Fragment

- In the layout XML

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

- By java code

```
mMapFragment = MapFragment.newInstance();
 FragmentTransaction fragmentTransaction =
       getFragmentManager().beginTransaction();
 fragmentTransaction.add(R.id.my_container, mMapFragment);
 fragmentTransaction.commit();
```

Adding a Map

Add a Map View

- Similar to the Fragment, MapView, a subclass of the Android View class, allows you to place a map in an Android View

```
<com.google.android.gms.maps.MapView android:id="@+id/mapview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" />
```


Starting the Map

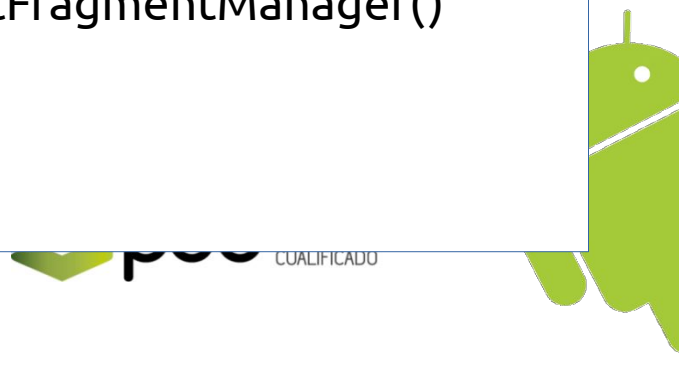
Add map code

- Implement the **OnMapReadyCallback** interface in the activity

```
public class MainActivity extends FragmentActivity
   implements OnMapReadyCallback {
...
}
```

- In your Activity's onCreate() method, set the layout file as the content view get the MapFragment (or View) and set up the **getMapAsync()**

```
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main);
   MapFragment mapFragment = (MapFragment) getFragmentManager()
   .findFragmentById(R.id.map);
      mapFragment.getMapAsync(this);
}
```



Starting the Map

- Use the onMapReady(GoogleMap) callback method to get a handle to the GoogleMap object. The callback is triggered when the map is ready to be used.

```
@Override
public void onMapReady(GoogleMap map) {
   map.addMarker(new MarkerOptions()
      .position(new LatLng(0, 0))
      .title("Marker"));
}
```


Markers

- You can add markers

```
map.addMarker(new MarkerOptions()
      .position(new LatLng(10, 10))
      .title("Hello world"));
```

- And make it draggable!

```
static final LatLng PERTH = new LatLng(-31.90, 115.86);
Marker perth = mMap.addMarker(new MarkerOptions()
                  .position(PERTH)
                  .draggable(true));
```


Markers

Position (Required)

The LatLng value for the marker's position on the map. This is the only required property for a Marker object.

Anchor

The point on the image that will be placed at the LatLng position of the marker. This defaults to the middle of the bottom of the image

Alpha

Sets the opacity of the marker. Defaults to 1.0.

Title

A string that's displayed in the info window when the user taps the marker.

Snippet

Additional text that's displayed below the title.

Icon

A bitmap that's displayed in place of the default marker image.

Markers

Draggable

Set to true if you want to allow the user to move the marker. Defaults to false.

Visible

Set to false to make the marker invisible. Defaults to true.

Flat or Billboard orientation

By default, markers are oriented against the screen, and will not rotate or tilt with the camera. Flat markers are oriented against the surface of the earth, and will rotate and tilt with the camera. Both types of markers do not change size based on zoom. Use GroundOverlays if you desire this effect.

Rotation

The orientation of the marker, specified in degrees clockwise. The default position changes if the marker is flat. The default position for a flat marker is north aligned. When the marker is not flat, the default position is pointing up and the rotation is such that the marker is always facing the camera.

Shapes

Three types of shape

- **Polyline**

The Polyline class defines a set of connected line segments on the map. A Polyline object consists of a set of LatLng locations, and creates a series of line segments that connect those locations in an ordered sequence.

```
// Instantiates a new Polyline object and adds points to define a rectangle
PolylineOptions rectOptions = new PolylineOptions()
    .add(new LatLng(37.35, -122.0))
    .add(new LatLng(37.45, -122.0))
        // North of the previous point, but at the same longitude
    .add(new LatLng(37.45, -122.2))
        // Same latitude, and 30km to the west
    .add(new LatLng(37.35, -122.2))
        // Same longitude, and 16km to the south
    .add(new LatLng(37.35, -122.0));
        // Closes the polyline.

// Get back the mutable Polyline
Polyline polyline = myMap.addPolyline(rectOptions);
```

Shapes

- **Polygon**

Polygon objects are similar to Polyline objects in that they consist of a series of coordinates in an ordered sequence. However, instead of being open-ended, polygons are designed to define regions within a closed loop with the interior filled in.

```
Polygon polygon = map.addPolygon(new PolygonOptions()
        .add(new LatLng(0, 0), new LatLng(0, 5),
                new LatLng(3, 5), new LatLng(0, 0))
        .strokeColor(Color.RED)
        .fillColor(Color.BLUE));
```

Shapes

Three types of shape

- **Circle**

A circle is defined to be the set of all points on the Earth's surface which are radius meters away from the given center

```
// Instantiates a new CircleOptions
//object and defines the center and radius
CircleOptions circleOptions = new CircleOptions()
    .center(new LatLng(37.4, -122.1))
    .radius(1000)); // In meters

// Get back the mutable Circle
Circle circle = myMap.addCircle(circleOptions);
```


Camera Updates

To change the the position of the camera, you must specify where you want to move the camera, using a **CameraUpdate**.

The Maps API allows you to create many different types of CameraUpdate using **CameraUpdateFactory**. The following options are available:

- **Zoom**

CameraUpdateFactory.zoomIn(),
CameraUpdateFactory.zoomOut(),
CameraUpdateFactory.zoomTo(float),
CameraUpdateFactory.zoomBy(float) and
CameraUpdateFactory.zoomBy(float, Point)

- **Changing position**

CameraUpdateFactory.newCameraPosition(CameraPosition)

- **Others: Scrolling, setting boundaries...**

Camera Updates

For applying a CameraUpdate to the camera, use:

- map.**moveCamera**(CameraUpdate)
- map.**animateCamera**(CameraUpdate [millis, callback])

You can also change the **position of the Camera**, by using CameraPosition

```
// Construct a CameraPosition focusing on //
Mountain View and animate the camera to that position.
CameraPosition cameraPosition = new CameraPosition.Builder()
   .target(MOUNTAIN_VIEW)     // Sets the center of the map to Mountain View
   .zoom(17)                  // Sets the zoom
   .bearing(90)               // Sets the orientation of the camera to east
   .tilt(30)                  // Sets the tilt of the camera to 30 degrees
   .build();                  // Creates a CameraPosition from the builder
map.animateCamera(CameraUpdateFactory.newCameraPosition(cameraPosition));
```

Events

- **Events when clicking on a map**

If you want to respond to a user tapping on a point on the map, you can use an **OnMapClickListener** which you can set on the map by calling **GoogleMap**.**setOnMapClickListener**(**OnMapClickListener**)

You can also listen for long click events with an **OnMapLongClickListener** which you can set on the map by calling **GoogleMap.setOnMapLongClickListener(OnMapLongClickListener)**.

- **Marker clicks**

You can use an **OnMarkerClickListener** to listen for click events on the marker. To set this listener on the map, call **GoogleMap**.**setOnMarkerClickListener**(**OnMarkerClickListener**)

When a user clicks on a marker, **onMarkerClick**(**Marker**) will be called and the marker will be passed through as an argument

Events

- **Marker dragging**

You can use an **OnMarkerDragListener** to listen for drag events on a marker. To set this listener on the map, call **GoogleMap**.**setOnMarkerDragListener**. To drag a marker, a user must long press on the marker.

When a marker is dragged, onMarkerDragStart(Marker) is called initially. While the marker is being dragged, onMarkerDrag(Marker) is called constantly.
At the end of the drag onMarkerDragEnd(Marker) is called.
You can get the position of the marker at any time by calling Marker.getPosition()

Events

- **Polyline and Polygon click**

Use an OnPolygonClickListener to listen to click events on a clickable polygon. To set this listener on the map, call GoogleMap.setOnPolygonClickListener(OnPolygonClickListener).

When a user clicks on a polygon, you will receive an onPolygonClick(Polygon) callback.

Use an OnPolylineClickListener to listen to click events on a clickable polyline. To set this listener on the map, call GoogleMap.setOnPolylineClickListener(OnPolylineClickListener).

When a user clicks on a polyline, you will receive an onPolylineClick(Polyline) callback.

And more..

- **Overlays**
    - **Ground Overlays**
    - **TileOverlays**
- **InfoWindows**

```
static final LatLng MELBOURNE = new LatLng(-37.81319, 144.96298);
Marker melbourne = mMap.addMarker(new MarkerOptions()
                    .position(MELBOURNE)
                    .title("Melbourne")
                    .snippet("Population: 4,137,400"));
...
melbourne.showInfoWindow();
```