Map Objects

Maps are represented in the API by the `GoogleMap` and `MapFragment` classes.

The Map Object

The Google Maps Android API allows you to display a Google map in your Android application. These maps have the same appearance as the maps you see in the Google Maps for Mobile (GMM) app, and the API exposes many of the same features. Two notable differences between the GMM Application and the Maps displayed by the Google Maps Android API are:

- Map tiles displayed by the API don't contain any personalized content, such as personalized smart icons.
- Not all icons on the map are clickable. For example, transit station icons can't be clicked. However, markers that you add to the map are clickable, and the API has a listener callback interface for various marker interactions.

In addition to mapping functionality, the API also supports a full range of interactions that are consistent with the Android UI model. For example, you can set up interactions with a map by defining listeners that respond to user gestures.

The key class when working with a Map object is the `GoogleMap` class. `GoogleMap` models the map object within your application. Within your UI, a map will be represented by either a `MapFragment` or `MapView` object.

`GoogleMap` handles the following operations automatically:

- Connecting to the Google Maps service.
- Downloading map tiles.
- Displaying tiles on the device screen.
- Displaying various controls such as pan and zoom.
- Responding to pan and zoom gestures by moving the map and zooming in or out.

In addition to these automatic operations, you can control the behavior of maps with objects and methods of the API. For example, `GoogleMap` has callback methods that respond to keystrokes and touch gestures on the map. You can also set marker icons on your map and add overlays to it, using objects you provide to `GoogleMap`.

MapFragment

`MapFragment`, a subclass of the Android `Fragment` class, allows you to place a map in an Android Fragment. `MapFragment` objects act as containers for the map, and provide access to the `GoogleMap` object.

Unlike a `View`, a `Fragment` represents a behavior or a portion of user interface in an Activity. You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities. Refer to the Android documentation on Fragments to learn more.

The Google Maps Android API requires API level 12 or higher for the support of MapFragment objects. If you are targeting an application earlier than API level 12, you can access the same functionality through the SupportMapFragment class. You will also have to include the Android Support Library.

MapView

MapView, a subclass of the Android View class, allows you to place a map in an Android View. A View represents a rectangular region of the screen, and is a fundamental building block for Android applications and widgets. Much like a MapFragment, the MapView acts as a container for the Map, exposing core map functionality through the GoogleMap object.

Users of this class must forward all the Activity life cycle methods - such as onCreate(), onDestroy(), onResume(), and onPause() - to the corresponding methods in the MapView class.

Add a Map to an Android application

The basic steps for adding a map are:

1. **(Once)** Follow the steps in [Getting Started][start] to get the API, obtain a key and add the required attributes to your Android Manifest.
2. Add a Fragment object to the Activity that you want to handle the map. The easiest way to do this is to add a <fragment> element to the layout file for the Activity
3. In the Activity object's onCreate() method, get a handle to the GoogleMap object in the MapFragment. The GoogleMap object is the internal representation of the map itself; to set view options for a map, you modify its GoogleMap object.
4. The last step is to add permissions and other settings to your application's manifest, AndroidManifest.xml.

Once you've followed these steps, you can set the initial options for the GoogleMap object. The MapFragment automatically displays the map at the completion of the onCreate() method.

Add a Fragment

To define a Fragment object in an Activity's layout file, add a <fragment> element. In this element, set the android:name attribute to "com.google.android.gms.maps.MapFragment". This automatically attaches a MapFragment to the Activity.

For example, the following layout file contains a <fragment> element:

```
<?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" />
```

You can also add a MapFragment to an Activity in code. To do this, create a new MapFragment instance, and then call FragmentTransaction.add() to add the Fragment to the current Activity

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

Adding Map Code

To work with the map inside your code, start by setting the layout file as the content view for the Activity. For example, if the layout file has the name main.xml, use this code:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
```

Next, get a handle to the map by calling FragmentManager.findFragmentById(), passing it the resource ID of the <fragment> element you added in the previous step. Cast the return value to a GoogleMap. For example, the following line puts a handle to the map into the variable mMap:

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

Notice that the resource ID `R.id.map` is added automatically to the Android project when you build the layout file.

With the handle to the `GoogleMap` object for the `MapFragment`, you can set initial options for the map.

Verify Map availability

Before you can interact with a `GoogleMap` object, you will need to confirm that an object can be instantiated, and that the Google Play services components are correctly installed on the target device. You can verify that the `GoogleMap` is available by calling the `MapFragment.getMap()` or `MapView.getMap()` methods and checking that the returned object is not null.

An example of a test to confirm the availability of a `GoogleMap` is shown below. This method can be called from both the `onCreate()` and `onResume()` stages to ensure that the map is always available.

```
private void setUpMapIfNeeded() {
    // Do a null check to confirm that we have not already instantiated the map.
    if (mMap == null) {
        mMap = ((MapFragment) getFragmentManager().findFragmentById(R.id.map))
                            .getMap();
        // Check if we were successful in obtaining the map.
        if (mMap != null) {
            // The Map is verified. It is now safe to manipulate the map.

        }
    }
}
```

Map Types

There are many types of maps available within the Google Maps Android API. A map's type governs the overall representation of the map. For example, an atlas usually contains **political** maps that focus on showing boundaries, and **road** maps that show all of the roads for a city or region.

The Google Maps Android API offers four types of maps, as well as an option to have no map at all:

Normal
: Typical road map. Roads, some man-made features, and important natural features such as rivers are shown. Road and feature labels are also visible.

Hybrid
: Satellite photograph data with road maps added. Road and feature labels are also visible.

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. Some roads and labels are also visible.

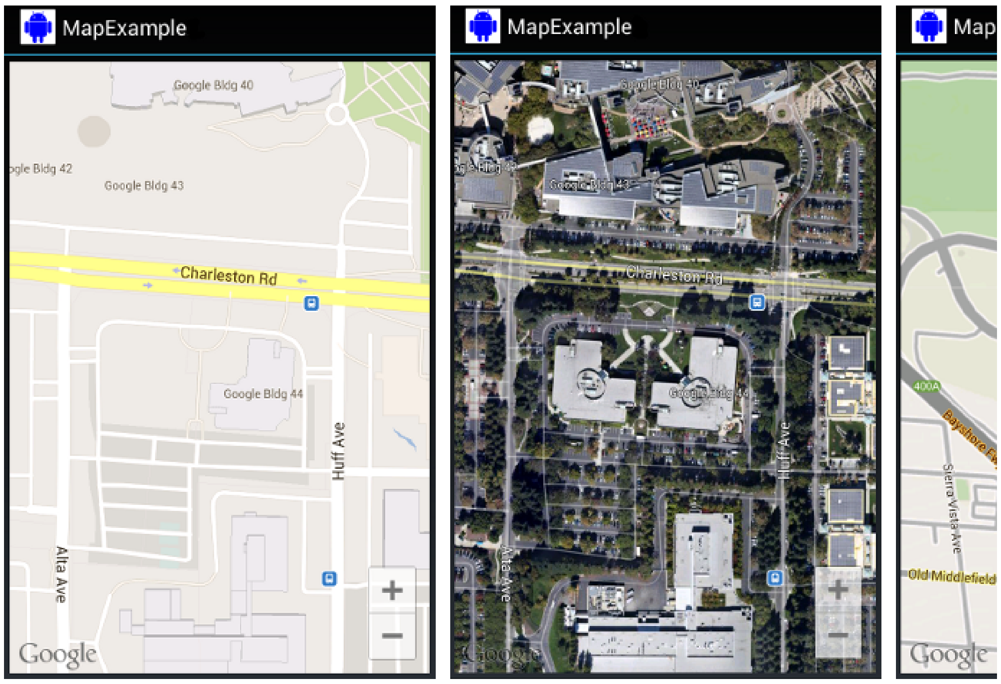
None
: No tiles. The map will be rendered as an empty grid with no tiles loaded.

Change the Map Type

To set the type of a map, call the `GoogleMap` object's `setMapType()` method, passing one of the type constants defined in `GoogleMap`. For example, to display a satellite map:

```
GoogleMap map;
...
// Sets the map type to be "hybrid"
map.setMapType(GoogleMap.MAP_TYPE_HYBRID);
```

The image below shows a comparison of normal, hybrid and terrain maps for the same location:



Configuring Initial State

The Maps API allows you to configure the initial state of the map to suit your application's needs. You can specify the following:

- The camera position, including: location, zoom, bearing and tilt. See Changing the Map View for more details on camera positioning.
- The map type.
- Whether the zoom buttons and/or compass appear on screen.
- Which gestures a user can use to manipulate the camera.

You can configure the initial state of the map either via XML, if you have added the map to your Activity's layout file, or programmatically, if you have added the map that way.

Using XML Attributes

This section describes how to set the initial state of the map if you have added a map to your application using an XML layout file.

The Maps API defines a set of custom XML attributes for a `MapFragment` or a `MapView` that you can use to configure the initial state of the map directly from the layout file. The following attributes are currently defined:

- `mapType`. This allows you to specify the type of map to display. Valid values include: `none`, `normal`, `hybrid`, `satellite` and `terrain`.
- `cameraTargetLat`, `cameraTargetLng`, `cameraZoom`, `cameraBearing`, `cameraTilt`. These allow you to specify the initial camera position. See here for more details on Camera Position and its properties.
- `uiZoomControls`, `uiCompass`. These allow you to specify whether you want the zoom controls and compass to appear on the map. See `UiSettings` for more details.
- `uiZoomGestures`, `uiScrollGestures`, `uiRotateGestures`, `uiTiltGestures`. These allow you to specify which gestures are enabled/disabled for interaction with the map. See `UiSettings` for more details.
- `zOrderOnTop`. Control whether the map view's surface is placed on top of its window. See SurfaceView.setZOrderOnTop(boolean) for more details. Note that this will cover all other views that could appear on the map (e.g., the zoom controls, the my location button).
- `useViewLifecycle`. Only valid with a `MapFragment`. This attribute specifies whether the lifecycle of the map should be tied to the fragment's view or the fragment itself. See here for more details.

In order to use these custom attributes within your XML layout file, you must first add the following namespace declaration (you can choose any namespace, it doesn't have to be `map`):

```
xmlns:map="http://schemas.android.com/apk/res-auto"
```

You can then add the attributes with a `map:` prefix into your layout components, as you would with standard

Android attributes.

The following XML code snippet shows how to configure a `MapFragment` with some custom options (the same attributes can be applied to a `MapView` as well):

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:map="http://schemas.android.com/apk/res-auto"
  android:id="@+id/map"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  class="com.google.android.gms.maps.SupportMapFragment"
  map:cameraBearing="112.5"
  map:cameraTargetLat="-33.796923"
  map:cameraTargetLng="150.922433"
  map:cameraTilt="30"
  map:cameraZoom="13"
  map:mapType="normal"
  map:uiCompass="false"
  map:uiRotateGestures="true"
  map:uiScrollGestures="false"
  map:uiTiltGestures="true"
  map:uiZoomControls="false"
  map:uiZoomGestures="true"/>
```

Programmatically

This section describes how to set the initial state of the map if you have added a map to your application programmatically.

If you have added a `MapFragment` (or `MapView`) programmatically, then you can configure its initial state by passing in a `GoogleMapOptions` object with your options specified. The options available to you are exactly the same as those available via XML. You can create a `GoogleMapOptions` object like this:

```
GoogleMapOptions options = new GoogleMapOptions();
```

And then configure it as follows:

```
options.mapType(GoogleMap.MAP_TYPE_SATELLITE)
    .compassEnabled(false)
    .rotateGesturesEnabled(false)
    .tiltGesturesEnabled(false);
```

To apply these options when you are creating a map, do one of the following:

- If you are using a `MapFragment`, use the `MapFragment.newInstance(GoogleMapOptions options)` static factory method to construct the fragment and pass in your custom configured options.
- If you are using a `MapView`, use the `MapView(Context, GoogleMapOptions)` constructor and pass in your custom configured options.

Last updated June 6, 2013.