Markers

Markers indicate single locations on the map. You can customize your markers by changing the default color, or replacing the marker icon with a custom image.

Introduction

Markers identify locations on the map. Markers use a standard icon, common to the Google Maps look and feel. It's possible to change the icon's color, image or anchor point via the API. Markers are objects of type `Marker`, and are added to the map with the `GoogleMap.addMarker(markerOptions)` method.

Marker icons are drawn oriented against the device's screen rather than the map's surface, so rotating, tilting or zooming the map will not necessarily change the orientation of the marker.

Markers are designed to be interactive. By default, they receive `click` events, for example, and are often used within event listeners to bring up info windows. Setting a marker's `draggable` property to `true` allows the user to change the position of the marker. Use a long press to activate the ability to move the marker.

Add a marker

The below example demonstrates how to add a marker to a map. The marker is created at coordinates `0,0`, and displays the string "Hello world" in an infowindow when clicked.

```
private GoogleMap mMap;
mMap = ((MapFragment) getFragmentManager().findFragmentById(R.id.map)).getMap();
mMap.addMarker(new MarkerOptions()
        .position(new LatLng(0, 0))
        .title("Hello world"));
```

Customize a marker

Markers may define an icon to show in place of the default icon. Defining an icon involves setting a number of properties that affect the visual behavior of the marker.

Markers support customization through the following properties:

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

Icon

A bitmap that's displayed in place of the default marker image. You can't change the icon once you've created the marker.

The below snippet creates a simple marker, with the default icon.

```
static final LatLng MELBOURNE = new LatLng(-37.81319, 144.96298);
Marker melbourne = mMap.addMarker(new MarkerOptions()
                          .position(MELBOURNE));
```

With the `title()` and `snippet()` methods, you can add additional content to your marker that will appear in an infowindow when the marker is clicked.

```
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"));
```

Change the default marker

It's possible to customize the color of the default marker image by passing a `BitmapDescriptor` object to the icon() method. You can use a set of predefined colors in the `BitmapDescriptorFactory` object, or set a custom marker color with the `BitmapDescriptorFactory.defaultMarker(float hue)` method. The hue is a value between 0 and 360, representing points on a color wheel.

```
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")
                          .icon(BitmapDescriptorFactory.defaultMarker(BitmapDesc
```

If you'd like to change more than just the color of the marker, you can set a custom marker image, often called an icon. Custom icons are always set as a `BitmapDescriptor`, and defined using one of four methods in the `BitmapDescriptorFactory` class.

`fromAsset(String assetName)`

Creates a custom marker using an image in the assets directory.

`fromBitmap (Bitmap image)`

Creates a custom marker from a Bitmap image.

`fromFile (String path)`

Creates a custom icon from a file at the specified path.

`fromResource (int resourceId)`

Creates a custom marker using an existing resource.

The below snippet creates a marker with a custom icon.

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

Info windows

An info window allows you to display information to the user when they tap on a marker on a map. By default, an info window is displayed when a user taps on a marker if the marker has a title set. Only one info window is displayed at a time. If a user clicks on another marker, the current window will be hidden and the new info window will be displayed. You can show an info window programmatically by calling `showInfoWindow()` on the target marker. An info window can be hidden by calling `hideInfoWindow()`.

An info window is drawn oriented against the device's screen, centered above its associated marker. The default info window contains the title in bold, with the (optional) snippet text below the title.

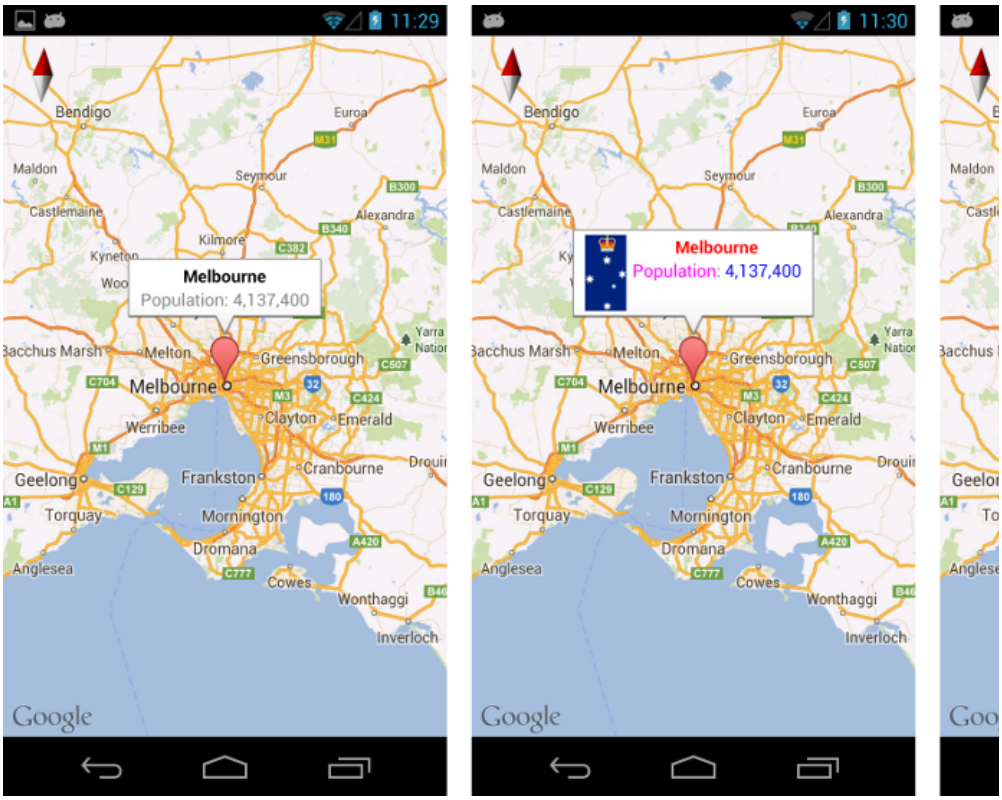
You are also able to customize the contents and design of info windows. To do this, you must create a

concrete implementation of the `InfoWindowAdapter` interface and then call `GoogleMap.setInfoWindowAdapter()` with your implementation. The interface contains two methods for you to implement: `getInfoWindow(Marker)` and `getInfoContents(Marker)`. The API will first call `getInfoWindow(Marker)` and if `null` is returned, it will then call `getInfoContents(Marker)`. If this also returns `null`, then the default info window will be used.

The first of these (`getInfoWindow()`) allows you to provide a view that will be used for the entire info window. The second of these (`getInfoContents()`) allows you to just customize the contents of the window but still keep the default info window frame and background.

> **Note:** The info window that is drawn is *not* a live view. The view is rendered as an image (using `View.draw(Canvas)`) at the time it is returned. This means that any subsequent changes to the view will not be reflected by the info window on the map. To update the info window later (e.g., after an image has loaded), call `showInfoWindow()`. Furthermore, the info window will not respect any of the interactivity typical for a normal view such as touch or gesture events. However you can listen to a generic click event on the whole info window as described in the section below.

The image below shows a marker with a default info window, custom info contents and a custom info window.



Event handling

The Maps API allows you to listen and respond to marker events. To listen to these events, you must set the corresponding listener on the `GoogleMap` object to which the markers belong. When the event occurs on one of the markers on the map, the listener's callback will be invoked with the corresponding `Marker` object passed through as a parameter. To compare this `Marker` object with your own reference to a `Marker` object, you must use `equals()` and not `==`.

The following events can be listened to:

- Marker click events
- Marker drag events
- Info window click events

Marker click events

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. This method returns a boolean that indicates whether you have consumed the event (i.e., you want to suppress the default behavior). If it returns `false`, then the default behavior will occur in addition to your custom behavior. The default behavior for a marker click event is to show its info window (if available) and move the

camera such that the marker is centered on the map.

Marker drag events

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 the user takes their finger off the screen, the marker will stay in that position. 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()`.

> **Note:** By default, a marker is not draggable. A marker must explicitly be set to be draggable before it can be dragged by a user. This can be done either with `MarkerOptions.draggable(boolean)` prior to adding it to the map, or `Marker.setDraggable(boolean)` once it has been added to the map.

Info window click events

You can use an `OnInfoWindowClickListener` to listen to click events on an info window. To set this listener on the map, call `GoogleMap.setOnInfoWindowClickListener(OnInfoWindowClickListener)`. When a user clicks on an info window, `onInfoWindowClick(Marker)` will be called and the info window will be highlighted in the default highlight color (Holo Blue for devices running Ice Cream Sandwich and newer, orange for earlier versions of Android).

As mentioned in the previous section on info windows, an info window is not a live `View`, rather the view is rendered as an image onto the map. As a result, any listeners you set on the view are disregarded and you cannot distinguish between click events on various parts of the view. You are advised not to place interactive components — such as buttons, checkboxes, or text inputs — within your custom info window.

Last updated June 6, 2013.