Sergio Luján Mora
Department of Software and Computing Systems
University of Alicante

WEB APPLICATION PROGRAMMING WITH GOOGLE MAPS
Table of contents

- Introduction
- Static Maps
- Documentation
- Simple Map
- Markers
- Events
- Info Window

Introduction

- Google Maps (GM) API is a geospatial viewing tool
  - GM API allows the programmer to embed GM in web pages with JavaScript
- GM API is used by millions of web sites to provide geolocalized information
- GM API is currently a free product and requires no installation or management
Introduction

• Google reserves the right to put advertising on the map at any point in the future:

  Advertising. The Service currently does not include advertising in the maps images. However, Google reserves the right to include advertising in the maps images provided to you through the Service, but will provide you with ninety (90) days notice prior to the commencement of advertising in the maps images. Such notice may be provided on relevant Google websites, including but not limited to the Google Geo Developers Blog and the Google Maps API Group (or such successor URLs that Google may designate from time to time). During that 90 day period, you may terminate your use of the Service, or provide notice of your refusal to accept advertising in the maps images in accordance with Google’s policies and procedures for providing such notice (which Google may make available from time to time in its sole discretion).

• GM API provides a number of utilities for manipulating maps and adding content to the map

• Success reason:
  – The ability to customize the map display through the addition of application specific data is the true driver of its acceptance
Introduction

• Sign up for an API key
  – Sign up for a Google Account
  – Specify a web site URL \(\rightarrow\) A single API key is valid for a single directory or domain
    • http://localhost/
Web Application Programming with Google Maps

What is the Google Maps API?

The Google Maps API lets you embed Google Maps in your own web pages with JavaScript. The API provides a number of utilities for manipulating maps (just like on the http://maps.google.com web page) and adding content to the map through a variety of services, allowing you to create robust maps applications on your website.

How do I start?

1. Sign up for a Google Maps API key.
2. Read the Maps API Developer’s Guide.
3. Read the Maps API Reference.
Web Application Programming with Google Maps

Google Accounts

Sign in to personalize your Google experience. Google has more to offer when you sign in to your Google Account. You can customize pages, view recommendations, and get more relevant search results.

Sign in on the right in google.com or log in using just an email address and password you choose.

©2009 Google - Google Home - Terms of Service - Privacy Policy - Help

Google Maps API

Google Maps API - API Key Signup

Thank You for Signing Up for a Google Maps API Key!

Your key is:

This key is good for all URLs consisting of this registered domain (and directory, if applicable):

http://localhost/

Note: For more information on the API key system, consult http://code.google.com/apis/maps/documentation/keysystem/

How you use your key depends on what Maps API product or service you use. Your key is valid for use within the entire family of Google Maps API solutions. The following examples show how to use your key within the Maps API product family.

JavaScript Maps API Example

Within the JavaScript Maps API, place the key within the script tag when you load the API:

```html
<!-- Your API key goes here -->
<script src="https://maps.google.com/maps/api/js?sensor=false&key=YOUR_API_KEY" defer="true"></script>
```

Web Application Programming with Google Maps

Introduction

• **Important!**: the web page where Google Maps is used must be freely accessible to end users
  – If the intention is to use in a commercial service, **Google Maps API Premier** must be used
Static Maps

- It provides static maps: the user can’t manipulate (move, zoom, ...) the map
  - It doesn’t require JavaScript
  - The map is created based on query string parameters in the URL
  - The map is sent back to the web browser as an image (GIF, PNG or JPEG)
  - It is useful for light web sites or for mobile devices (PDAs, cell phones, ...)

- In a request, the following information can be defined:
  - The location of the map
  - The size of the map
  - The zoom level
  - The type of map
  - The location of markers with labels
Static Maps

- **Request:**
  
  \[ \text{http://maps.google.com/maps/api/staticmap?parameters} \]

- **Location parameters:**
  - center
  - zoom

- **Configuration parameters:**
  - size
  - format
  - maptype
  - mobile
  - language
Static Maps

- Features parameters:
  - markers
  - path
  - visible

- General parameters:
  - key
  - sensor

Static Maps

- Location (center):
  - Latitudes and longitudes are defined using numerals within a comma-separated text string that have a precision to 6 decimal places
  - Precision beyond the 6 decimal places is ignored
  - Latitudes can take any value between -90 and 90 while longitude values can take any value between -180 and 180
Static Maps

- **Zoom level** (*zoom*):
  - 0 to 21
- **Map type** (*maptype*):
  - roadmaps (default value)
  - satellite
  - terrain
  - hybrid

---

**Usage Limits**

Use of the Google Static Maps API is subject to a query limit of 1000 unique (different) image requests per user per day. Since this restriction is a quota per viewer, most developers should not need to worry about exceeding their quota. However, note that we enforce an additional request rate limit to prevent abuse of the service. Requests of identical images, in general, do not count towards this limit beyond the original request.

If a user exceeds the limit as proscribed above, the following image will be displayed indicating that the quota has been exceeded:

This limit is enforced to prevent abuse and/or repurposing of the Static Maps API, and this limit may be changed in the future without notice. If you exceed the 24-hour limit or otherwise abuse the service, the Static Maps API may stop working for you temporarily. If you continue to exceed this limit, your access to the Static Maps API may be blocked.
Static Maps

- **EXAMPLE**
- Map centered in Lublin
  - Latitude: 51.25
  - Longitude: 22.57
- Zoom level: 14
- Size: 600x400
- Type: road map

```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es" lang="es">
<head><title>Bishkek, Kyrgyzstan</title></head>
<body>
<h1>Bishkek, Kyrgyzstan</h1>
<img src="http://maps.google.com/maps/api/staticmap?center=51.68,39.22&zoom=14&size=600x400&maptype=roadmap&sensor=false&key=API_KEY" width="600" height="400" alt="Map of Bishkek" />
</body>
</html>
```
Colima, México

Static Maps

• **EXERCISE**

  • Modify the previous map:
    – Add two buttons (“+” y “-”) to change the zoom level
    – Add a text label that shows the current zoom level
    – Add a list that allows the user to change the map type
Static Maps

- Markers
  - All the markers defined in the same `markers` parameter share the same visual style
    ```
    markers=markerStyles|markerLocation1|markerLocation2|
    ```
  - If different visual styles are needed, different `markers` parameters must be used
Static Maps

- The set of marker style descriptors is a series of value assignments separated by the pipe (|) character.
- Marker style descriptors:
  - size(tiny, mid, small)
  - color: 24-bit color (0xFF0000) or a predefined color (black, brown, green, purple, yellow, blue, gray, orange, red, white)
  - label (a single uppercase alphanumeric character [A-Z0-9])
    - tiny and small markers are not capable of displaying an alphanumeric character

---

**EXAMPLE**

- Define three markers:
  - Markers 1 y 2:
    - Size: mid
    - Color: red
    - Label: A
    - Locations: (51.248,22.572) and (51.249,22.573)
  - Marker 3:
    - Size: mid
    - Color: blue
    - Label: B
    - Location: (51.2485,22.5725)
Static Maps

markers=size:mid|color:red|label:A|51.248,22.572|51.249,22.573
&markers=size:mid|color:blue|label:B|51.2485,22.5725

EXERCISE

- Map centered in Lublin
- Zoom level: 14
- Size: 600x400
- Type: road map

- Show three markers:
  - A blue marker, with label “C”, on City Center
  - A red marker, with label “U”, on Politecnical University
  - A green marker, with label “H”, on your home
**Colima, México**

- A. Universidad de Colima
- B. Planes Internos

```html
```
Static Maps

• There can be also:
  – Paths
    
    path=color:0xff0000|weight:5|40.737102,-
    73.990318|40.749825,-
    73.987963|40.752946,-
    73.987384|40.755823,-73.986397
  – Polygons
Static Maps

- `path=pathStyles|pathLocation1|pathLocation2|... etc.`

- `weight` (optional) specifies the thickness of the path in pixels. If no weight parameter is set, the path will appear in its default thickness (5 pixels).
- `color` (optional) specifies a color either as a 24-bit (example: `color=0xFFFFCC`) or 32-bit hexadecimal value (example: `color=0xFFFFFFCCFF`), or from the set {black, brown, green, purple, yellow, blue, gray, orange, red, white}. 
Static Maps

- `fillcolor:` (optional) indicates both that the path marks off a polygonal area and specifies the fill color to use as an overlay within that area. The set of locations following need not be a "closed" loop; the Static Map server will automatically join the first and last points.

EXERCISE
- Map centered in Lublin
- Zoom level: 14
- Size: 600x400
- Type: road map
- Draw a mark in the city center and a path from the city center to the University
Documentation

- Google Maps API V2 Developer Guide:
- Google Maps API Concepts:
- Google Maps API Examples:
- Google Maps API Reference:
Simple Map

```html
<script src="http://maps.google.com/maps?file=api&amp;v=2&amp;sensor=false&amp;key=API_KEY" type="text/javascript"></script>
```

- **EXAMPLE**
  - Map centered in Lublin
  - Zoom level: 14
  - Size: 600x400
  - With and without user’s interface
Simple Map

```javascript
function initialize() {
  if (GBrowserIsCompatible()) {
    var map = new GMap2(document.getElementById("map_canvas"));
    // Alicante (Spain)
    map.setCenter(new GLatLng(38.34, -0.48), 13);
    // Bishkek (Kyrgyzstan)
    map.setCenter(new GLatLng(42.9, 74.6), 13);
    // Show user interface
    map.setUIToDefault();
  }
}
</script>
```

- `var map = new GMap2();` ➔ We pass the `<div>` specifying where the map will appear
  - The size of the map will default to the size specified in the `<div>` tag
- `map.setCenter();` ➔ We center the map at a particular latitude, longitude, and zoom level
  - This method must be called first after construction to set the initial state of the map
  - 17 point scale: 0 (entire world) ... 16 street level
- `map.setUIToDefault();` ➔ Adds the default behavior and UI elements
Simple Map

```
<body onload="initialize()" onunload="GUnload()">
<div id="map_canvas" style="width: 500px; height: 400px">
</div>
</body>
```

We have to use the same id in the JavaScript code.
Simple Map

• Full-page map:

```html
html, body, #map_canvas {
  width: 100%;
  height: 100%;
  margin: 0;
  padding: 0;
}
```

Simple Map

• EXERCISE

• Show two maps of different places “side by side”
Simple Map

```java
// map.setUIToDefault();
map.addControl(new GLargeMapControl());
```

**GSmallMapControl()**: Creates a control with buttons to pan in four directions, and zoom in and zoom out.

**GLargeMapControl()**: Creates a control with buttons to pan in four directions, and zoom in and zoom out, and a zoom slider.

**GSmallZoomControl()**: Creates a control with buttons to zoom in and zoom out.

**GLargeMapControl3D()**: Creates a new 3D-style control with buttons to pan in four directions, and zoom in and zoom out, and a zoom slider.

**GSmallZoomControl3D()**: Creates a new 3D-style control with buttons to zoom in and zoom out.
Simple Map

- **EXERCISE**
- In the previous map, show the user interface for a large map
Simple Map

- Control for selecting and switching between map types:
  ```javascript
  // map.setUIToDefault();
  map.addControl(new GLargeMapControl());
  map.addControl(new GMapTypeControl());
  ```

**GMapTypeControl()**: Creates a standard map type control for selecting and switching between supported map types via buttons.

**GMenuMapTypeControl()**: Creates a drop-down map type control for switching between supported map types.

**GHierarchicalMapTypeControl()**: Creates a "nested" map type control for selecting and switching between supported map types via buttons and nested checkboxes.
Simple Map

```
// map.setUIToDefault();
map.addControl(new GLargeMapControl());
map.addControl(new GMapTypeControl());
map.addControl(new GScaleControl());
map.addControl(new GOverviewMapControl());
```

**GScaleControl()**: Creates a control that displays the map scale.

**GOverviewMapControl()**: Creates a collapsible overview mini-map in the corner of the main map for reference location and navigation (through dragging). The GOverviewMapControl creates an overview map with a one-pixel black border. Note: Unlike other controls, you can only place this control in the bottom right corner of the map (G_ANCHOR_BOTTOM_RIGHT).
Simple Map

- By default, the normal view will be displayed
- However, the map type can be changed to several map types with `map.setMapType()`:
  - `G_NORMAL_MAP`: This map type (which is the default) displays a normal street map.
  - `G_SATELLITE_MAP`: This map type displays satellite images.
  - `G_HYBRID_MAP`: This map type displays a transparent layer of major streets on satellite images.
  - `G_PHYSICAL_MAP`: This map type displays maps with physical features such as terrain and vegetation. This map type is not displayed within map type controls by default.

Simple Map

- Special map types:
  - `G_SKY_VISIBLE_MAP`: This map type shows a mosaic of the sky, covering the full celestial sphere.
  - `G_MOON_ELEVATION_MAP`: This map type displays a shaded terrain map of the surface of the Moon, color-coded by altitude. This map type is not displayed within map type controls by default.
  - `G_MOON_VISIBLE_MAP`: This map type displays photographs taken from orbit around the moon. This map type is not displayed within map type controls by default.
Simple Map

• Special map types:
  - G_MARS_ELEVATION_MAP: This map type displays a shaded relief map of the surface of Mars, color-coded by altitude. This map type is not displayed within map type controls by default.
  - G_MARS_VISIBLE_MAP: This map type displays photographs taken from orbit around Mars. This map type is not displayed within map type controls by default.
  - G_MARS_INFRARED_MAP: This map type displays a shaded infrared map of the surface of Mars, where warmer areas appear brighter and colder areas appear darker.

var map = new GMap2(document.getElementById("map_canvas"),
{mapTypes: [G_HYBRID_MAP, G_SATELLITE_MAP]});
Simple Map

- `map.setZoom(zoomLevel)`: Sets zoom to the specified level
- `map.getZoom()`: Retrieves current zoom level
Simple Map

- **EXERCISE**
  - Map centered in Lublin
  - Zoom level: 14
  - Size: 600x400
  - Limit map types to hybrid and satellite
  - Add the following user controls:
    - Large map control
    - Map type control
    - Scale control
    - Overview control

Markers

- Most important and useful feature of Google Maps API:
  - We can add some points (markers) with information to the map → We can show our data → We can create our own applications and services
- The `GMarker()` constructor is used to create icons showing points of interest
Markers

- Add a marker:

```javascript
// Latitude and longitude of the new marker
var point = new GLatLng(lat, lon);
// Create a new marker
var marker = new GMarker(point);
// Add the marker to the map
map.addOverlay(marker);
```

- Delete a marker:

```javascript
var point = new GLatLng(lat, lon);
var marker = new GMarker(point);
map.addOverlay(marker);
// ...
// Delete a specific marker
map.removeOverlay(marker);
// Delete all the markers
map.clearOverlays();
```
Markers

• Example:
  – Add 10 markers to the map at random locations

```javascript
// Returns a GLatLngBounds object
var bounds = map.getBounds();
var southWest = bounds.getSouthWest();
var northEast = bounds.getNorthEast();
var lngSpan = northEast.lng() - southWest.lng();
var latSpan = northEast.lat() - southWest.lat();
for (var i = 0; i < 10; i++) {
  var point = new GLatLng(southWest.lat() + latSpan * Math.random(),
                         southWest.lng() + lngSpan * Math.random());
  map.addOverlay(new GMarker(point));
}
```
Markers

- **EXERCISE**
- Show a map centered in Lublin
- Show a list (<select>) with different places in Lublin (city center, University, etc.)
- When the user selects a place, show a marker in the map
- Show a button to delete all the markers on the map
Markers

- **EXERCISE**
  - Show a map centered in Lublin
  - Show a list `<select>` with different places in Lublin (city center, University, etc.)
  - When the user selects a place, show a marker in the map
  - Show a button to delete the markers on the map

- **Center the map in the marker:**
  
  ```javascript
  var point = new GLatLng(lat, lon);
  var marker = new GMarker(point);
  ...
  map.panTo(point);
  ```

  
Markers

- `GMarkerManager` class is used to efficiently manage hundreds of markers on a map
  - Without the use of this class, the performance of the application can be very poor
  - This class can also be used to reduce the clutter of these markers when viewed at certain map scales
Markers

- `addMarkers(markers:GMarker[], minZoom:Number, maxZoom?:Number):` Adds a batch of markers to this marker manager. The markers are not added to the map, until the `refresh()` method is called.
- `addMarker(marker:GMarker, minZoom:Number, maxZoom?:Number):` Adds a single marker to a collection of markers controlled by this manager. If the marker’s location falls within the map’s current viewport and the map’s zoom level is within the specified zoom level rage, the marker is immediately added to the map.
- `refresh():` Forces the manager to update markers shown on the map. This method must be called if markers were added using the `addMarkers` method.

// Creates a new marker manager
var mkmgr = new GMarkerManager(map);
...

mkmgr.addMarker(new GMarker(point1), 13, 17);
mkmgr.addMarker(new GMarker(point2), 13, 17);
mkmgr.addMarker(new GMarker(point3), 13, 17);
...

mkmgr.refresh();
Markers

- **EXERCISE**
  - Show a map with 50 random markers around Lublin, 50 random markers around Krakow, and 50 random markers around Warsaw
  - When the zoom level is lower than 10, only one marker is shown in each city
  - When the zoom level is 10 or greater, all the markers are shown

- **EXERCISE**
  - Create a web page that shows the different places a person has visited during a trip
  - User interface:
    - Button “Start trip”: the code starts to show a marker for every place every three seconds
    - Button “Stop trip”
  - Use `setInterval()` to execute some code after a specified time-interval
  - Use `clearInterval()` to stop the timer
EXERCISE

Create a web page that allows the user to add markers to a map

User interface:
- Latitude input box
- Longitude input box
- Add marker button
- Remove markers button: remove all markers from the map
Markers

• Show a small map over a marker:

```javascript
marker.showMapBlowup();
```

Markers

• A marker has options, e.g. allow a marker to be moved:

```javascript
var options = {draggable: true};
marker = new GMarker(point, options);
marker.enableDragging();
```
Markers

• How to change the icon of a marker:

```javascript
var myIcon = new GIcon(G_DEFAULT_ICON);
myIcon.image = "my_custom_icon.png";
myIcon.iconSize = new GSize(22, 31);
myIcon.shadow = "my_custom_icon_shadow.png";
myIcon.shadowSize = new GSize(42, 31);
myIcon.iconAnchor = new GPoint(10, 29);
myIcon.infoWindowAnchor = new GPoint(10, 14);
myIcon.printImage = "my_custom_icon_print.gif";
myIcon.mozPrintImage = "my_custom_icon_mozPrint.gif";
myIcon.printShadow = "my_custom_icon_printShadow.gif";
myIcon.transparent = "my_custom_icon_transparent.png";
myIcon.imageMap = [ 10,29, 1,16, 0,5, 5,0, 12,4, 18,2,
21,12, 21,16 ];
```

Events

• Events are “external stimulus” to the map and are usually triggered by the user
  – We can write code that responds to any of the defined events
Events

- There exist different types of events
- Each event gets different parameters
- In order to catch events, an event handler or event listener must be defined for an element (map, marker, ...) and for an event (click, move, ...)

Example:
- Event listener for click event on a map:

```javascript
var map = new GMap2(...);
GEVENT.addListener(map, "click",
function(overlay, latlng) {
  // Action in response to the event
});
```
Events

- **Main events:**
  - `click`
  - `dblclick`
  - `move, movestart, moveend`
  - `drag, dragstart, dragend`
  - `mouseover, mouseout, mousemove`

Info Window

- We can add a pop-up window to display information about the markers
  - But information window can be placed anywhere on a map
Info Window

- Write an info window with HTML in the middle of the map:

```javascript
var html = "A simple <b>text</b>";
map.openInfoWindowHtml(map.getCenter(), html);
```

Info Window

- Write a tabbed info window with HTML in the middle of the map:

```javascript
var info = [
    new GInfoWindowTab("School", "Lublin"),
    new GInfoWindowTab("Hospital", "Warsaw"),
    new GInfoWindowTab("Culture", "Krakow")
];

map.openInfoWindowTabsHtml(map.getCenter(), info);
```
Info Window

- Show information in a marker:
  ```javascript
  var point = new GLatLng(lat, lon);
  map.addOverlay(new GMarker(point));
  marker.openInfoWindowHtml("Something we want to show");
  
  // We can also use:
  // marker.openInfoWindowTabsHtml(tabs);
  ```

Info Window

- **EXERCISE**
  - Show an information window when the user clicks on a map
Info Window

• **SOLUTION:**
  - An event listener for event `click` must be defined in the map:

```javascript
GEvent.addListener(map, "click", function(overlay, latlng) {
  map.openInfoWindowHtml(latlng, "Some text we want to show");
});
```

Info Window

• **EXERCISE**
  • A draggable marker shows an info window
Info Window

• SOLUTION:

```javascript
var marker = new GMarker(center, {draggable: true});
GEvent.addListener(marker, "dragstart", function() {
  map.closeInfoWindow();
});
GEvent.addListener(marker, "dragend", function() {
  marker.openInfoWindowHtml("Just bouncing along...");
});
map.addOverlay(marker);
```

Info Window

• EXERCISE

• Show an information window when the user clicks on a map
• In the information window, show the coordinates (latitude and longitude) of the clicked point
Info Window

• SOLUTION
  – An event listener for event click must be defined in the map
  – The second parameter of the event listener are the coordinates

```javascript
GEvent.addListener(map, "click", function(overlay, latlng) {
  map.openInfoWindowHtml(latlng, "Latitude: " + latlng.lat() + "<br />Longitude: " + latlng.lng());
});
```

• EXERCISE
• The user clicks two points in a map
• Use function
  `latlng2.distanceFrom(latlng1)` to calculate the distance in meters between two points
Info Window

• **EXERCISE**
  • We want to show an info window over a marker when we click the marker

• **SOLUTION**
  • We have to add an event handler for a “click” to the marker:

```javascript
GEvent.addListener(marker, "click", function() {
  marker.openInfoWindowHtml("Something we want to show");
});
```
• **EXERCISE**
  • Create a web page that allows the user to add markers to a map
  • User interface:
    - **Latitude** input box
    - **Longitude** input box
    - **Add marker** button
    - **Remove markers** button: remove all markers from the map
  • **Add Content** input box to the previous exercise: user can insert text to show in an info window
Info Window

• EXERCISE
• We want to add a new marker with content to a map when we click the map
  – Define `addListener()` for the map to handle the “click event”
  – Create a new marker
    • Ask for the content of the info window of the marker, user JavaScript function `prompt()`
    • Define `addListener()` for the marker to handle the “click event” to show the info window
  – Add the new marker to the map
EXERCISE

Create a web page that allows the user to find interesting places (museums, restaurants, universities, ...) in different cities.

When the user selects a city, the map must be centered in the city.
- Send a request to the server to get the coordinates of the city center.

When the user selects a type of place, markers are shown with the names and descriptions of the places.
- Send a request to the server to get the names, descriptions, and coordinates of the different places.
Web Application Programming with Google Maps

Info Window

```
// ...
<?php
// ...
</p>
```

Client

Server

JSON

?city=
?city=&place=

??>
var cities = ["Alicante", 38.3, -0.5], ["Lublin", 51.2, 22.6], ["Bishkek", 42.9, 74.6]];

var places = ["Alicante", "University", 38.3, -0.5], ["Cinema", 38.31, -0.51], ["Theatre", 38.32, -0.52]], ["Lublin", ["University", 51.2, 22.6]], ...];