# Google Maps API Documentation

Google Maps API
Sign up for an API key
**API Documentation**
API Terms of Use
API Blog
API Discussion Group



The Google Maps JavaScript API lets you embed Google Maps in your own web pages. To use the API, you need to sign up for an API key and then follow the instructions below.

The API is new, so there may be bugs and slightly less than perfect documentation. Bear with us as we fill in the holes, and join the Maps API discussion group to give feedback and discuss the API.

## Table of Contents



## Introduction

### Audience

This documentation is designed for people familiar with JavaScript programming and object-oriented programming concepts. You should also be familiar with Google Maps from a user's point of view.

## The "Hello, World" of Google Maps

The easiest way to start learning about this API is to see a simple example. The following web page displays a 300x300 map centered on Palo Alto:

```
<!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">
  <head>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8"/>

        <title>Google Maps JavaScript API Example: simple</title>
        <script src="http://maps.google.com/maps?file=api&v=1&key=ABQIAAAAEfCuQGsNiSWxRgf_vfNWaRQjskl1-
YgiA_BGX2yRrf7htVrbmBTEB0IH-F489GrwP8-dHLib7cKKIQ"
                        type="text/javascript">
        </script>

        <script type="text/javascript">
        //<![CDATA[

        function onLoad() {
          // The basics.
          //
          // Creates a map and centers it on Palo Alto.

          if (GBrowserIsCompatible()) {
                var map = new GMap(document.getElementById("map"));
                map.centerAndZoom(new GPoint(-122.1419, 37.4419), 4);
          }
        }

        //]]>
        </script>
  </head>
  <body onload="onLoad()">

        <div id="map" style="width: 500px; height: 300px"></div>
  </body>
</html>
```

You can download this example to edit and play around with it, but you'll have to replace the key in that file with your own Maps API key. (If you register a key for a particular directory, it works for all subdirectories as well.)

The URL in the example above (`http://maps.google.com/maps?file=api&v=1`) is the location of a JavaScript file that includes all of the symbols you need for placing Google Maps on your pages. Your page must contain a `script` tag pointing to that URL, using the key you got when you signed up for the API. If your Maps API key were "abcdefg", then your `script` tag might look like this:

```
<script src="http://maps.google.com/maps?file=api&v=1&key=abcdefg"

                type="text/javascript">
</script>
```

The main class exported by the Google Maps API is `GMap`, which represents a single map on the page. You can create as many instances of this class as you want (one for each map on the page). When you create a new map instance, you specify a named element in the page (usually a `div` element) to contain the map. Unless you specify a size explicitly, the map uses the size of the container to determine its size.

The methods for manipulating and adding overlays to map instances are detailed below.

## Browser Compatibility

The Google Maps API supports the same browsers as the Google Local site. See the list of supported browsers on Google Local. Since different applications require different behaviors for users with incompatible browsers, the Maps API provides a global method (`GBrowserIsCompatible()`) to check compatibility, but it does not perform any automatic behavior when it detects an incompatible browser. The script `http://maps.google.com/maps?file=api&v=1` will parse in almost every browser without errors, so you can safely include that script before checking for compatibility.

None of the examples in this document check for compatibility (other than the first example, above), nor do they display an error message for older browsers. Clearly real applications should do something more friendly with incompatible browsers, but we have omitted such checks to make the examples more readable.

Note that certain browsers may have particular quirks regarding JavaScript. For example, IE doesn't allow JavaScript inside tables.

## XHTML and VML

We recommend that you use standards-compliant XHTML on pages that contain maps. When browsers see the XHTML `DOCTYPE` at the top of the page, they execute the page in "standards compliant mode," which makes layout and behaviors much more predictable across browsers.

If you want to show polylines on your map (like the lines used by Google Maps to show driving directions), you need to include the VML namespace and some CSS code in your XHTML document, to make everything work properly in IE. Your XHTML document should begin like this:

```
<!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" xmlns:v="urn:schemas-microsoft-com:vml">
  <head>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8"/>

        <title>Your Page Title Here</title>
        <style type="text/css">
        v\:* {
          behavior:url(#default#VML);
        }
```

```
      </style>
      <script src="http://maps.google.com/maps?file=api&v=1&key=abcdefg"

                      type="text/javascript">
      </script>
  </head>
```

See Microsoft's VML workshop for more information.

### API Updates

The `v=1` part of the URL `http://maps.google.com/maps?file=api&v=1` refers to "version 1" of the API. When we do a significant update to the API in the future, we will change the version number and post a notice on Google Code and the Maps API discussion group.

After a new version is released, we will try to run the old and new versions concurrently for about a month. After a month, the old version will be turned off, and code that uses the old version will no longer work.

The Maps team will transparently update the API with the most recent bug fixes and performance enhancements. These bug fixes should only improve performance and fix bugs, but we may inadvertently break some API clients. Please use the Maps API discussion group to report such issues.

### Geocoding, Routing, etc.

The Google Maps API does not include geocoding or routing services at this time; in particular, you can't specify a location using its street address.

However, there are a number of free geocoders on the web; if you need the latitude and longitude of a particular address, you can use one of those.

## Examples

Each of the following examples shows only the relevant JavaScript code, not the full HTML file. You can plug the JS code into the skeleton HTML file shown earlier, or you can download the full HTML file for each example by clicking the link after the example.

### The Basics

The following example (the same as the code from the example shown earlier) creates a map and centers it on Palo Alto. Note that you can drag the map around just as you can drag maps on the Google Maps site.

```
var map = new GMap(document.getElementById("map"));
map.centerAndZoom(new GPoint(-122.1419, 37.4419), 4);
```

View example (simple.html)

## Map Movement and Animation

The `recenterOrPanToLatLng` method recenters the map around a given point. If the specified point is in the current map viewport, the map pans smoothly to the point; if not, the map jumps to the point.

The following example displays a map, then waits two seconds, then pans to a new center.

```
var map = new GMap(document.getElementById("map"));
map.centerAndZoom(new GPoint(-122.1419, 37.4419), 4);
window.setTimeout(function() {
  map.recenterOrPanToLatLng(new GPoint(-122.1569, 37.4569));
}, 2000);
```

View example (animate.html)

## Adding Controls to the Map

You can add controls to the map with the `addControl` method. In the following example code, we add a `GSmallMapControl` (a small version of the control that lets visitors pan and zoom the map) and a `GMapTypeControl` (which lets visitors switch between Map mode and Satellite mode).

```
var map = new GMap(document.getElementById("map"));
map.addControl(new GSmallMapControl());
map.addControl(new GMapTypeControl());
map.centerAndZoom(new GPoint(-122.1419, 37.4419), 4);
```

View example (controls.html)

## Event Listeners

To register an event listener, call the `GEvent.addListener` method. Pass it a map, an event to listen for, and a function to call when the specified event occurs. In the following example code, we display the latitude and longitude of the center of the map after the visitor drags the map.

Note that this example relies on the presence of a `div` tag named "message" in the HTML body, where the latitude and longitude are displayed. See the example file for more information.

```
var map = new GMap(document.getElementById("map"));
GEvent.addListener(map, 'moveend', function() {
  var center = map.getCenterLatLng();
  var latLngStr = '(' + center.y + ', ' + center.x + ')';
  document.getElementById("message").innerHTML = latLngStr;
```

```
});
map.centerAndZoom(new GPoint(-122.1419, 37.4419), 4);
```

View example (event.html)

## Opening an Info Window

To create an info window, call the `openInfoWindow` method, passing it a location and a DOM element to display. The following example code displays an info window that points to the center of the map and displays a simple "Hello, world" message.

You would typically place an info window above a marker, but you can place an info window anywhere on the map.

```
var map = new GMap(document.getElementById("map"));
map.centerAndZoom(new GPoint(-122.1419, 37.4419), 4);
map.openInfoWindow(map.getCenterLatLng(),
                                    document.createTextNode("Hello, world"));
```

View example (infowindow.html)

## Map Overlays

To add a marker to a map, first create a GPoint; then give that point as the location of a new GMarker; then pass the marker to `addOverlay`.

To add a polyline to a map, first create an array of points; then use those points to create a new GPolyline; then pass the polyline to `addOverlay`.

The following example code creates ten random markers and a random polyline, to illustrate the use of map overlays.

Remember to add the VML namespace and the necessary CSS code; see XHTML and VML for more information.

```
// Center the map on Palo Alto.
var map = new GMap(document.getElementById("map"));
map.addControl(new GSmallMapControl());
map.addControl(new GMapTypeControl());
map.centerAndZoom(new GPoint(-122.1419, 37.4419), 4);

// Add 10 random markers in the map viewport using the default icon.
var bounds = map.getBoundsLatLng();
var width = bounds.maxX - bounds.minX;
var height = bounds.maxY - bounds.minY;
for (var i = 0; i < 10; i++) {
  var point = new GPoint(bounds.minX + width * Math.random(),
                                          bounds.minY + height * Math.random());
  var marker = new GMarker(point);
```

```
  map.addOverlay(marker);
}

// Add a polyline with 4 random points. Sort the points by
// longitude so that the line does not intersect itself.
var points = [];
for (var i = 0; i < 5; i++) {
  points.push(new GPoint(bounds.minX + width * Math.random(),
                                                  bounds.minY + height * Math.random()));
}
points.sort(function(p1, p2) { return p1.x - p2.x; });
map.addOverlay(new GPolyline(points));
```

View example (overlay.html)

## Click Handling

To handle clicks, call `GEvent.addListener` and pass the 'click' event.

In the following code example, when the visitor clicks anywhere in the map, we create a new marker at that point. When the visitor clicks a marker, we remove it from the map.

```
var map = new GMap(document.getElementById("map"));
map.addControl(new GSmallMapControl());
map.addControl(new GMapTypeControl());
map.centerAndZoom(new GPoint(-122.1419, 37.4419), 4);

GEvent.addListener(map, 'click', function(overlay, point) {
  if (overlay) {
        map.removeOverlay(overlay);
  } else if (point) {
        map.addOverlay(new GMarker(point));
  }
});
```

View example (click.html)

## Display Info Window Above Markers

In the following example, we show a custom info window above each marker by listening to the click event for each marker. We take advantage of function closures to customize the info window content for each marker.

```

// Center the map on Palo Alto.
var map = new GMap(document.getElementById("map"));
map.addControl(new GSmallMapControl());
```

```
map.addControl(new GMapTypeControl());
map.centerAndZoom(new GPoint(-122.1419, 37.4419), 4);

// Create a marker whose info window displays the given number.
function createMarker(point, number) {
  var marker = new GMarker(point);

  // Show this marker's index in the info window when it is clicked.
  var html = "Marker #<b>" + number + "</b>";
  GEvent.addListener(marker, 'click', function() {
        marker.openInfoWindowHtml(html);
  });

  return marker;
}

// Add 10 random markers in the map viewport using the default icon.
var bounds = map.getBoundsLatLng();
var width = bounds.maxX - bounds.minX;
var height = bounds.maxY - bounds.minY;
for (var i = 0; i < 10; i++) {
  var point = new GPoint(bounds.minX + width * Math.random(),
                                                bounds.minY + height * Math.random());
  var marker = createMarker(point, i + 1);
  map.addOverlay(marker);
}
```

View example (markerinfowindow.html)

## Creating Icons

The following example creates a new type of marker, using the Google Ride Finder "mini" markers as an example. We have to specify the foreground image, the shadow image, and the points at which we anchor the icon to the map and anchor the info window to the icon.

```

// Create our "tiny" marker icon
var icon = new GIcon();
icon.image = "http://labs.google.com/ridefinder/images/mm_20_red.png";
icon.shadow = "http://labs.google.com/ridefinder/images/mm_20_shadow.png";
icon.iconSize = new GSize(12, 20);
icon.shadowSize = new GSize(22, 20);
icon.iconAnchor = new GPoint(6, 20);
icon.infoWindowAnchor = new GPoint(5, 1);

// Center the map on Palo Alto.
var map = new GMap(document.getElementById("map"));
map.addControl(new GSmallMapControl());
map.addControl(new GMapTypeControl());
map.centerAndZoom(new GPoint(-122.1419, 37.4419), 4);
```

```
// Create one of our tiny markers at the given point.
function createMarker(point) {
  var marker = new GMarker(point, icon);
  GEvent.addListener(marker, 'click', function() {
        marker.openInfoWindowHtml("You clicked me!");
  });
  return marker;
}

// Add 10 random markers in the map viewport.
var bounds = map.getBoundsLatLng();
var width = bounds.maxX - bounds.minX;
var height = bounds.maxY - bounds.minY;
for (var i = 0; i < 10; i++) {
  var point = new GPoint(bounds.minX + width * Math.random(),
                                                    bounds.minY + height * Math.random());
  var marker = createMarker(point);
  map.addOverlay(marker);
}
```

View example (icon.html)

## Using Icon Classes

In many cases, your icons may have different foregrounds, but the same shape and shadow. The easiest way to achieve this behavior is to use the copy constructor for the `GIcon` class, which copies all the properties over to a new icon which you can then customize. The following example demonstrates this icon-copying technique.

Note: this example isn't very localizable or scalable; it relies on the existence of a series of letter images corresponding to letters in the Latin-1 codeset.

```
// Create a base icon for all of our markers that specifies the
// shadow, icon dimensions, etc.
var baseIcon = new GIcon();
baseIcon.shadow = "http://www.google.com/mapfiles/shadow50.png";
baseIcon.iconSize = new GSize(20, 34);
baseIcon.shadowSize = new GSize(37, 34);
baseIcon.iconAnchor = new GPoint(9, 34);
baseIcon.infoWindowAnchor = new GPoint(9, 2);
baseIcon.infoShadowAnchor = new GPoint(18, 25);

// Center the map on Palo Alto.
var map = new GMap(document.getElementById("map"));
map.addControl(new GSmallMapControl());
map.addControl(new GMapTypeControl());
map.centerAndZoom(new GPoint(-122.1419, 37.4419), 4);
```

```
// Create a marker whose info window displays the letter corresponding
// to the given index.
function createMarker(point, index) {
  // Create a lettered icon for this point using our icon class from above
  var letter = String.fromCharCode("A".charCodeAt(0) + index);
  var icon = new GIcon(baseIcon);
  icon.image = "http://www.google.com/mapfiles/marker" + letter + ".png";
  var marker = new GMarker(point, icon);

  // Show this marker's index in the info window when it is clicked.
  var html = "Marker <b>" + letter + "</b>";
  GEvent.addListener(marker, 'click', function() {
        marker.openInfoWindowHtml(html);
  });

  return marker;
}

// Add 10 random markers in the map viewport.
var bounds = map.getBoundsLatLng();
var width = bounds.maxX - bounds.minX;
var height = bounds.maxY - bounds.minY;
for (var i = 0; i < 10; i++) {
  var point = new GPoint(bounds.minX + width * Math.random(),
                                       bounds.minY + height * Math.random());
  var marker = createMarker(point, i);
  map.addOverlay(marker);
}
```

View example (iconclass.html)

## Using XML and Asynchronous RPC ("AJAX") with Maps

In the following example, we download a static file ("data.xml") that contains a list of latitude and longitude coordinates in XML. When the download completes, we parse the XML and create a marker at each of the specified locations.

Note: If the XML file includes HTML code, be sure to encode the HTML appropriately in the XML file. For example, change all occurrences of "<" in the HTML tags in the XML file to "<", then convert them back in your JavaScript code after dlownloading the XML file. (The XML file used by the following example doesn't include any HTML, so it doesn't need any encoding or decoding.)

```
// Center the map on Palo Alto.
var map = new GMap(document.getElementById("map"));
map.addControl(new GSmallMapControl());
map.addControl(new GMapTypeControl());
map.centerAndZoom(new GPoint(-122.1419, 37.4419), 4);

// Download the data in data.xml and load it on the map. The format we
// expect is:
```

```
// <markers>

//   <marker lat="37.441" lng="-122.141"/>
//   <marker lat="37.322" lng="-121.213"/>
// </markers>

var request = GXmlHttp.create();
request.open('GET', 'data.xml', true);
request.onreadystatechange = function() {
  if (request.readyState == 4) {
        var xmlDoc = request.responseXML;
        var markers = xmlDoc.documentElement.getElementsByTagName("marker");
        for (var i = 0; i < markers.length; i++) {
        var point = new GPoint(parseFloat(markers[i].getAttribute("lng")),
                                                parseFloat(markers[i].getAttribute("lat")));
        var marker = new GMarker(point);
        map.addOverlay(marker);
        }
  }
}
request.send(null);
```

View example (async.html). Requires external XML data file named data.xml (or you can create your own).

## Troubleshooting

If your code doesn't seem to be working, here are some approaches that might help you track down the problem:

- Make sure your API key is valid for the directory where your file is.
- View your page in a different browser.
- Look at the JavaScript console in your browser, if any.
- Look for typos. Remember that JavaScript is a case-sensitive language.
- Post to the Maps API discussion group (after looking to see whether anyone else has asked a similar question recently). Include a link to the page that's having the problem.
- See the Reference section for other forums.

## Other resources

Here are some additional resources. Note that these sites are not owned or supported by Google.

- Mapki is a wiki with information about the Maps API, including an FAQ page.

## API Overview

## The GMap class

An instance of GMap represents a single map on the page. You can create as many instances of this class as you want (one for each map on the page). When you create a new map instance, you specify a named element in the page (usually a div element) to contain the map. Unless you specify a size explicitly, the map uses the size of the container to determine its size.

The GMap class exports methods to manipulate the map's center and zoom level and to add and remove overlays (such as GMarker and GPolyline instances). It also exports a method to open an "info window" to display information; see The Info Window for details.

For more information about GMap, see the GMap class reference.

## Events

You can add dynamic elements to your application by using event listeners. An object exports a number of named events, and your application can "listen" to those events using the static methods GEvent.addListener or GEvent.bind. For example, the following code snippet shows an alert every time the visitor clicks anywhere in the map:

```
var map = new GMap(document.getElementById("map"));
map.centerAndZoom(new GPoint(-122.1419, 37.4419), 4);
GEvent.addListener(map, 'click', function() {
  alert("You clicked the map.");
});
```

GEvent.addListener takes a function as the third argument, which promotes the use of function closures for event handlers. If you want to bind an event to a class method, you can use GEvent.bind. For example:

```
function onLoad() {

        function MyApplication() {
          this.counter = 0;
          this.map = new GMap(document.getElementById("map"));
          GEvent.bind(this.map, 'click', this, this.onMapClick);
        }

        MyApplication.prototype.onMapClick = function() {
          this.counter++;
          alert("You have clicked the map " + this.counter +
                                (this.counter == 1 ?" time.":" times."));
        }

        var application = new MyApplication();
        application.map.centerAndZoom(new GPoint(-122.1419, 37.4419), 4);

}
```

View example (bind.html)

## The Info Window

Each map has a single "info window," which displays HTML content in a floating window above the map. The info window looks a little like a comic-book word balloon; it has a content area and a tapered stem, where the tip of the stem is at a specified point on the map. You may have seen the info window appear when you click a marker in Google Maps or Google Local.

You can't show more than one info window at once for a given map, but you can move the info window and change its contents as needed.

The basic info window method is `openInfoWindow`, which takes a point and an HTML DOM element as arguments. The info window appears with its tip at the given point on the map, and it displays the DOM element in its content area.

The `openInfoWindowHtml` method is similar, but it takes an HTML string as its second argument rather than a DOM element.

Similarly, `openInfoWindowXslt` takes a point, an XML DOM element, and the URL of an XSLT document, and it applies the XSLT to the XML to produce the info window contents. This method downloads the XSLT asynchronously if it has not already been downloaded by the user's browser.

To display an info window above an overlay like a marker, you can pass an optional third argument: a pixel offset between the specified point and the tip of the info window. So, if your marker is 10 pixels tall, you might pass the pixel offset `GSize(0, -10)`. (For more information, see the GSize class reference.)

The `GMarker` class exports `openInfoWindow` methods that handle the pixel offset issues for you based on the size and shape of the icon, so you generally don't have to worry about calculating icon offsets in your application.

## Overlays

Overlays are objects on the map that are tied to latitude/longitude coordinates, so they move when you drag or zoom the map and when you switch projections (such as when you switch from Map to Satellite mode).

The Maps API exports two types of overlays: markers, which are icons on the map, and polylines, which are lines made up of a series of points.

### Markers and Icons

The `GMarker` constructor takes an icon and a point as arguments and exports a small set of events, such as "click". See the overlay example above for a simple example of creating markers.

The most difficult part of creating a marker is specifying the icon, which is complex because of the number of different images that make up a single icon in the Maps API. However, if you just want a generic icon, you can create a GMarker without specifying an icon.

Icons are usually in the form of stylized pushpin images, with a tip that appears at the location specified in the `GMarker` constructor.

Every icon has (at least) a foreground image and a shadow image. The shadow should be created at a 45 degree angle (upward and to the right) from the foreground image, and the bottom left corner of the shadow image should align with the bottom-left corner of the icon foreground image. The shadow should be a 24-bit PNG image with alpha transparency so that the edges of the shadow look correct on top of the map.

The `GIcon` class requires you specify the size of these images when you initialize the icon so the Maps API can create image elements of the appropriate size. This is the minimum amount of code required to specify an icon (in this case, the icon used on Google Maps):

```
var icon = new GIcon();
icon.image = "http://www.google.com/mapfiles/marker.png";
icon.shadow = "http://www.google.com/mapfiles/shadow50.png";
icon.iconSize = new GSize(20, 34);
icon.shadowSize = new GSize(37, 34);
```

The `GIcon` class also exports seven other properties that you should set to get maximum browser compatibility and functionality from your icons. For example, the imageMap property specifies the shape of the non-transparent parts of the icon image. If you do not set this property in your icon, the entire icon image (including the transparent parts) will be clickable in Firefox/Mozilla. See the GIcon class reference for more information.

### Polylines

The `GPolyline` constructor takes an array of points as an argument, and creates a series of line segments that connect those points in the given sequence. You can also specify the color, weight, and opacity of the line. The color should be in the hexadecimal numeric HTML style; for example, use "#ff0000" for red. `GPolyline` does not understand named colors.

The following code snippet creates a 10-pixel-wide red polyline between two points:

```
var polyline = new GPolyline([new GPoint(-122.1419, 37.4419),
                                                    new GPoint(-122.1519, 37.4519)],
                                                    "#ff0000", 10);
map.addOverlay(polyline);
```

In Internet Explorer, Google Maps uses VML to draw polylines, so be sure to include the VML namespace and the relevant CSS code. (See XHTML and VML.) In all other browsers, we request an image of the line from Google servers and overlay that image on the map, refreshing the image as necessary as the map is zoomed and dragged around.

## Controls

To add pan, zoom, or map-type controls to your map, use the `addControl` method. The Maps API comes with a handful of built-in controls you can use in your maps:

`GLargeMapControl`
: a large pan/zoom control used on Google Maps

`GSmallMapControl`
: a smaller pan/zoom control used on Google Local

`GSmallZoomControl`
: a small zoom control (no panning controls) used in the small map blowup windows used to display driving directions steps on Google Maps

`GMapTypeControl`
: lets the visitor toggle among map types (such as Map and Satellite)

For example, to add the panning/zooming control you see on Google Maps to your map, you would include the following line in your map initialization:

```
map.addControl(new GLargeMapControl());
```

The control will be attached to the top left corner of the map just as it is on Google Maps.

### XML and RPC

The Google Maps API exports a factory method for creating `XmlHttpRequest` objects that work in recent versions of IE, Firefox, and Safari. For example:

```
var request = GXmlHttp.create();
request.open('GET', 'myfile.txt', true);
request.onreadystatechange = function() {
  if (request.readyState == 4) {
        alert(request.responseText);
  }
}
request.send(null);
```

You can parse an XML document with the static method `GXml.parse`, which takes a string of XML as its only argument. This method is compatible with all browsers, falling back on a JavaScript XML parser if the browser has no native XML parsing facilities. We make no guarantees as to the performance or correctness of this "fallback" parser.

Note that the Google Maps API does not require the use of XML or `XmlHttpRequest` to function, as it is a pure JavaScript/DHTML API.

## Class Reference

### `GMap`

An instance of `GMap` represents a single map on the page. See the discussion above for more information.

#### Constructor

| Constructor | Description |
|---|---|

| | |
|---|---|
| `GMap(container, mapTypes?, width?, height?)` | Creates a new map inside of the given HTML container, which is typically a `div` element. We use the default set of map types (`[G_MAP_TYPE, G_HYBRID_TYPE, G_SATELLITE_TYPE]`) unless a different set is specified. Likewise, we use the size of the container to determine the map width and height unless a width and height are explicitly specified. |

### Methods

#### Configuration

| Method | Description |
|---|---|
| `enableDragging()` | Enables dynamic dragging (enabled by default). |
| `disableDragging()` | Disables dynamic dragging. |
| `draggingEnabled()` | Returns true if dynamic dragging is enabled. |
| `enableInfoWindow()` | Enables the info window on this map (enabled by default). |
| `disableInfoWindow()` | Disables the info window on this map. |
| `infoWindowEnabled()` | Returns true if the info window is enabled on this map. |

#### Controls

| | |
|---|---|
| `addControl(control)` | Adds the given map control to this map. |
| `removeControl(control)` | Removes the given map control from this map. |

#### State

| Method | Description |
|---|---|
| `getCenterLatLng()` | Returns the center point of the map viewport in latitude/longitude coordinates. |
| `getBoundsLatLng()` | Returns the latitude/longitude bounds of the map viewport. |
| `getSpanLatLng()` | Returns the width and height of the map viewport in latitude/longitude ticks. |
| `getZoomLevel()` | Returns the integer zoom level of the map. |

| | |
|---|---|
| `centerAtLatLng(latLng)` | Centers the map at the given point. |
| `recenterOrPanToLatLng(latLng)` | Centers the map at the given point, doing a fluid pan to the point if it is within the current map viewport. |
| `zoomTo(zoomLevel)` | Zooms to the given integer zoom level, ignoring the request if the given zoom level is outside the bounds of the current map type. |
| `centerAndZoom(latLng, zoomLevel)` | Atomically centers and zooms the map. Useful to initialize the map with an initial center and zoom level, as in the examples above. |
| `getMapTypes()` | Returns an array of map types supported by this map (currently `G_MAP_TYPE`, `G_HYBRID_TYPE`, and `G_SATELLITE_TYPE`). |
| `getCurrentMapType()` | Returns the map type currently in use (`G_MAP_TYPE`, `G_HYBRID_TYPE`, or `G_SATELLITE_TYPE`). |
| `setMapType(mapType)` | Switches this map to the given map type (`G_MAP_TYPE`, `G_HYBRID_TYPE`, or `G_SATELLITE_TYPE`). |

**Overlays**

| Method | Description |
|---|---|
| `addOverlay(overlay)` | Adds the given overlay object (GMarker or GPolyline) to the map. |
| `removeOverlay(overlay)` | Removes the given overlay object from the map. |
| `clearOverlays()` | Removes all of the overlays from the map. |

**Info Window**

| Method | Description |
|---|---|
| `openInfoWindow(latLng, htmlElem, pixelOffset?, onOpenFn?, onCloseFn?)` | Displays the info window with the given HTML content at the given point. `htmlElem` should be an HTML DOM element. If `pixelOffset` (GSize) is given, we offset the info window by that number of pixels, which lets users place info windows above markers and other overlays. If `onOpenFn` is given, we call that function when the window is displayed. If `onCloseFn` is given, we call that function when the window is closed. |
| `openInfoWindowHtml(marker, htmlStr, pixelOffset?, onOpenFn?, onCloseFn?)` | Like `openInfoWindow`, but takes an HTML string rather than an HTML DOM element. |

| | |
|---|---|
| `openInfoWindowXslt(marker, xmlElem, xsltUri, pixelOffset?, onOpenFn?, onCloseFn?)` | Like `openInfoWindow`, but takes an XML element and the URI of an XSLT document to produce the content of the info window. The first time a URI is given, the file at that URI is downloaded with GXmlHttp and subsequently cached. |
| `showMapBlowup(point, zoomLevel?, mapType?, pixelOffset?, onOpenFn?, onCloseFn?))` | Shows a blowup of the map at the given `GPoint`. If the `zoomLevel` and `mapType` parameters are not given, we default to a zoom level of 1 and the current map type. |
| `closeInfoWindow()` | Closes the info window if it is open. |

### Events

| Event | Arguments | Description |
|---|---|---|
| `click` | `overlay, point` | Triggered when the user clicks the map or an overlay on the map. If the click was on an overlay, we pass the overlay as an argument to the event handler. Otherwise, we pass the latitude/longitude point that was clicked on the map. |
| `move` | *none* | Triggered when the map is moving. This event is triggered continuously as the map is dragged. |
| `movestart` | *none* | Triggered at the beginning of a continuous pan/drag movement. This event is *not* triggered when the map moves discretely. |
| `moveend` | *none* | Triggered at the end of a discrete or continuous map movement. This event is triggered once at the end of a continuous pan. |
| `zoom` | `oldZoomLevel`, `newZoomLevel` | Triggered after the map zoom level changes. |
| `maptypechanged` | *none* | Triggered after the map type (Map, Hybrid, or Satellite) changes. |
| `infowindowopen` | *none* | Triggered after the info window is displayed. |
| `infowindowclose` | *none* | Triggered after the info window is closed. |
| `addoverlay` | `overlay` | Triggered after an overlay is added to the map. |
| `removeoverlay` | `overlay` | Triggered after an overlay is removed from the map. Not triggered if `clearOverlays` is called—see the `clearoverlays` event below. |

| | | |
|---|---|---|
| `clearoverlays` | *none* | Triggered after all overlays are cleared from the map. |

## `GMarker`

`GMarker` is a type of map overlay that shows an icon at a single point on the map. The constructor takes an instance of GIcon, which can be shared among many markers, and the point at which it should be displayed. `GMarker` also includes some convenience methods to open info windows over the marker, which is common for Google Maps hacks.

### Constructor

| Constructor | Description |
|---|---|
| `GMarker(point, icon?)` | Creates a marker with the given icon at the given point. If no icon is given, we use the default Google Maps icon. |

### Methods

| Method | Description |
|---|---|
| `openInfoWindow(htmlElem)` | Opens an info window with the given HTML content over this marker. `htmlElem` should be an HTML DOM element. |
| `openInfoWindowHtml(htmlStr)` | Like `openInfoWindow`, but takes an HTML string rather than an HTML DOM element. |
| `openInfoWindowXslt(xmlElem, xsltUri)` | Like `openInfoWindow`, but takes an XML element and the URI of an XSLT document to produce the content of the info window. The first time a URI is given, the file at that URI is downloaded with GXmlHttp and subsequently cached. |
| `showMapBlowup(zoomLevel?, mapType?)` | Shows a blowup of the map over this marker. We use a default zoom level of 1 and the current map type if the `zoomLevel` and `mapType` parameters are not given. |

### Events

| Event | Arguments | Description |
|---|---|---|
| `click` | *none* | Triggered when the user clicks on this marker. |
| `infowindowopen` | *none* | Triggered when the info window is opened above this marker with one of the methods above. |
| `infowindowclose` | *none* | Triggered when the info window above this marker is closed. |

## GPolyline

A polyline represents a vector polyline overlay on the map. A polyline is drawn with the vector drawing facilities of the browser if they are available or an image overlay from Google servers otherwise.

### Constructor

| Constructor | Description |
|---|---|
| `GPolyline(points, color?, weight?, opacity?)` | Constructs a polyline from the given array of latitude/longitude points. `color` is a hex HTML color (such as "#0000ff"), `weight` is an integer representing the width of the line in pixels, and `opacity` is a float between 0 and 1. |

## GIcon

An icon specifies the images used to display a marker on the map. For browser compatibility reasons, specifying an icon is actually quite complex. See the discussion above for more information. Note that you can use the default Maps icon if you don't want to specify your own.

Before you can display an icon, you must specify (at a minimum) the `image`, `shadowImage`, `iconSize`, `shadowSize`, and `iconAnchor` properties. If you use an info window, you must also specify the `infoWindowAnchor` property of the icon.

### Constructor

| Constructor | Description |
|---|---|
| `GIcon(copy?)` | Creates a new icon, copying the properties of the given icon if given. |

### Properties

| Property | Description |
|---|---|
| `image` | The foreground image URL of the icon. |
| `shadow` | The shadow image URL of the icon. |
| `iconSize` | The pixel size of the foreground image of the icon. |
| `shadowSize` | The pixel size of the shadow image. |
| `iconAnchor` | The pixel coordinate relative to the top left corner of the icon image at which we should anchor this icon to the map. |

| | |
|---|---|
| `infoWindowAnchor` | The pixel coordinate relative to the top left corner of the icon image at which we should anchor the info window to this icon. |
| `printImage` | The URL of the foreground icon image we should use for printed maps. It should be the same size as the main icon image. |
| `mozPrintImage` | The URL of the foreground icon image we should use for printed maps in Firefox/Mozilla. It should be the same size as the main icon image. |
| `printShadow` | The URL of the shadow image we should use for printed maps. It should be a GIF image since most browsers cannot print PNG images. |
| `transparent` | The URL of a virtually transparent version of the foreground icon image used to capture IE click events. This image should be a 24-bit PNG version of the main icon image with 1% opacity, but the same shape and size as the main icon. |
| `imageMap` | An array of integers representing the x/y coordinates of the image map we should use to specify the clickable part of the icon image in non-IE browsers. |

## `GEvent`

All event registration and triggering is handled by the `GEvent` class. All methods in the `GEvent` class are *static* methods; you should call them using the form `GEvent.bind(...)` rather than `(new Event()).bind(...)`.

### Static Methods

| Method | Description |
|---|---|
| `addListener(source, eventName, listenerFn)` | Calls the given `listenerFn` function when the given event is triggered on the given source instance. We return an opaque listener token that can be used with `removeListener`. |
| `removeListener(listener)` | Removes the given listener, which should be a listener token returned by `addListener`. |
| `clearListeners(source, eventName)` | Removes all listeners for the given event on the given source. |
| `trigger(source, eventName, args...)` | Triggers the given event on the given source with the given list of arguments. |

| | |
|---|---|
| `bind(source, eventName, object, method)` | Binds the given method of the given object to the given source event. When the given event is triggered, the given method is called with `object` as the `this`. For example: `GEvent.bind(map, 'move', this, this.onMapMove)`. |

### `GXmlHttp`

The `GXmlHttp` class exports a factory method to create cross-browser-compatible XmlHttpRequest instances.

#### Static Methods

| Method | Description |
|---|---|
| `create()` | Factory method to construct a new `XmlHttpRequest` instance. |

### `GXml`

The `GXml` class provides a static method to parse a string of XML. The parser should work in any browser, though it falls back on a JavaScript XML parser by default if there is no native parser in the web browser. That default JavaScript XML parser may be quite slow, depending on the browser's JavaScript implementation.

#### Static Methods

| Method | Description |
|---|---|
| `parse(xmlStr)` | Parses the given string of XML, returning the XML DOM. |
| `value(xmlNode)` | Returns the textual content in the given XML element or node. Useful to pull out the text nodes from inside of an XML element. |

### `GXslt`

The `GXslt` class provides factory methods to apply XSLT to XML in a brower-independent way. The class should work on any browser, though it falls back on a JavaScript XSLT implementation by default if there is no native XSLT processor in the web browser. That default JavaScript XSLT processor may be quite slow, depending on the browser's JavaScript implementation.

#### Static Methods

| Method | Description |
|---|---|
| `create(xsltXmlDoc)` | Returns a `GXslt` instance from the given XML DOM, which should be an XSLT document. |

### Methods

| Method | Description |
|---|---|
| `transformToHtml(xmlDoc, htmlContainer)` | Transforms the given XML document using this XSLT, placing the resulting HTML in the given HTML DOM element container. |

## `GPoint`

A `GPoint` represents a single, 2-dimensional coordinate. If a `GPoint` represents a latitude/longitude, then `x` is the longitude and `y` is the latitude, in decimal notation.

### Constructor

| Constructor | Description |
|---|---|
| `GPoint(x, y)` | Creates a new point with the given coordinate values. |

### Properties

| Property | Description |
|---|---|
| `x` | The x (or horizontal) coordinate of the point. |
| `y` | The y (or vertical) coordinate of the point. |

## `GSize`

`GSize` represents a 2-dimensional size measurement. If a `GSize` represents a latitude/longitude span, `width` is the number of longitude degrees, and `height` is the number of latitude degrees.

### Constructor

| Constructor | Description |
|---|---|
| `GSize(width, height)` | Creates a new size with the given measurement values. |

### Properties

| Property | Description |
|---|---|
| `width` | The width measurement. |

| | |
|---|---|
| `height` | The height measurement. |

### GBounds

`GBounds` represents a 2-dimensional bounding box. If the `GBounds` is in the latitude/longitude coordinate system, the x coordinates represent longitude and the y coordinates represent latitude. If the latitude/longitude bounds crosses the International Date Line, the "minimum" coordinates refer to the top left coordinates rather than the mathematical minimum of the two coordinates.

#### Constructor

| Constructor | Description |
|---|---|
| `GBounds(minX, minY, maxX, maxY)` | Creates a new bounds with the given coordinates. |

#### Properties

| Property | Description |
|---|---|
| `minX` | The x coordinate of the top left corner of the bounds. |
| `minY` | The y coordinate of the top left corner of the bounds. |
| `maxX` | The x coordinate of the bottom right corner of the bounds. |
| `maxY` | The y coordinate of the bottom right corner of the bounds. |