Getting Started

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. There are many JavaScript tutorials available on the Web.

This conceptual documentation is designed to let you quickly start exploring and developing applications with the Google Maps API. We also publish the Google Maps API Reference.

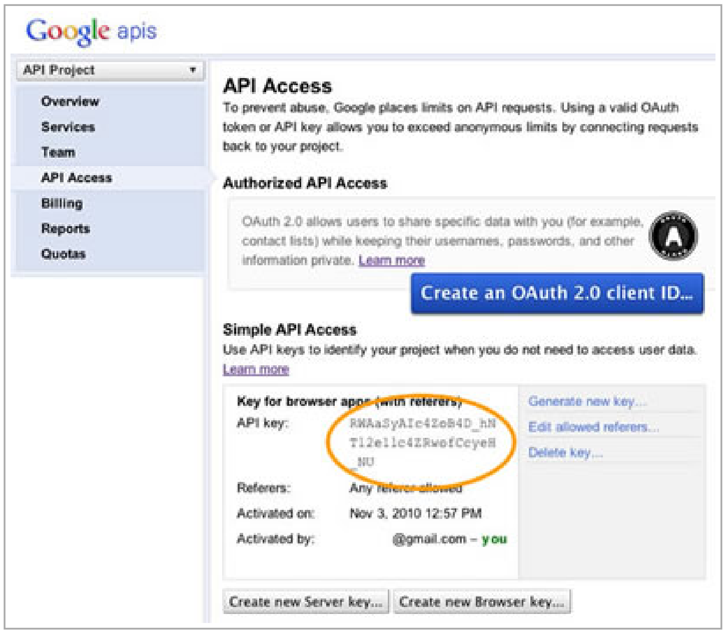
Obtaining an API Key

All Maps API applications* should load the Maps API using an API key. Using an API key enables you to monitor your application's Maps API usage, and ensures that Google can contact you about your application if necessary. If your application's Maps API usage exceeds the Usage Limits, you must load the Maps API using an API key in order to purchase additional quota.

> * **Google Maps API for Work** developers must *not* include a key in their requests. Please refer to Loading the Google Maps JavaScript API for instructions.

To create your API key:

1. Visit the Google Developers Console and log in with your Google Account.
2. Click the **Services** link from the left-hand menu.
3. Activate the **Google Maps JavaScript API v3** service.
4. Click the **API Access** link from the left-hand menu. Your API key is available from the **API Access** page, in the **Simple API Access** section. Maps API applications use the **Key for browser apps**.



API Access

To prevent abuse, Google places limits on API requests. Using a valid OAuth token or API key allows you to exceed anonymous limits by connecting requests back to your project.

Authorized API Access

OAuth 2.0 allows users to share specific data with you (for example, contact lists) while keeping their usernames, passwords, and other information private. Learn more

Simple API Access

Use API keys to identify your project when you do not need to access user data. Learn more

Key for browser apps (with referers)
API key: RWAaSyAIc4ZoB4D_hN Tl2e1lc4ZRwofCcyeH _NU
Referers: Any referer allowed
Activated on: Nov 3, 2010 12:57 PM
Activated by: @gmail.com – you

By default, a key can be used on any site. We strongly recommend that you restrict the use of your key to domains that you administer, to prevent use on unauthorized sites. You can specify which domains are allowed to use your API key by clicking the **Edit allowed referrers...** link for your key.

Hello, World

The easiest way to start learning about the Google Maps API is to see a simple example. The following web page displays a map centered on Sydney, New South Wales, Australia:

```
<!DOCTYPE html>
<html>
  <head>
    <style type="text/css">
      html, body, #map-canvas { height: 100%; margin: 0; padding: 0;}
    </style>
    <script type="text/javascript"
      src="https://maps.googleapis.com/maps/api/js?key=API_KEY">
    </script>
    <script type="text/javascript">
      function initialize() {
        var mapOptions = {
          center: { lat: -34.397, lng: 150.644},
          zoom: 8
        };
        var map = new google.maps.Map(document.getElementById('map-canvas'),
            mapOptions);
      }
      google.maps.event.addDomListener(window, 'load', initialize);
    </script>
  </head>
  <body>
<div id="map-canvas"></div>
  </body>
</html>
```

View example (map-simple.html)

Even in this simple example, there are a few things to note:

1. We declare the application as HTML5 using the `<!DOCTYPE html>` declaration.
2. We include the Maps API JavaScript using a `script` tag.
3. We create a `div` element named "map-canvas" to hold the Map.
4. We create a JavaScript object literal to hold a number of map properties.
5. We create a JavaScript "map" object, passing it the `div` element and the map properties.
6. We use an event listener to load the map after the page has loaded.

These steps are explained below.

Declaring Your Application as HTML5

We recommend that you declare a true `DOCTYPE` within your web application. Within the examples here, we've declared our applications as HTML5 using the simple HTML5 `DOCTYPE` as shown below:

```
<!DOCTYPE html>
```

Most current browsers will render content that is declared with this `DOCTYPE` in "standards mode" which means that your application should be more cross-browser compliant. The `DOCTYPE` is also designed to degrade gracefully; browsers that don't understand it will ignore it, and use "quirks mode" to display their content.

Note that some CSS that works within quirks mode is not valid in standards mode. In specific, all percentage-based sizes must inherit from parent block elements, and if any of those ancestors fail to specify a size, they are assumed to be sized at 0 x 0 pixels. For that reason, we include the following `<style>` declaration:

```
<style type="text/css">
  html { height: 100% }
  body { height: 100%; margin: 0; padding: 0 }
  #map-canvas { height: 100% }
</style>
```

This CSS declaration indicates that the map container `<div>` (named `map-canvas`) should take up 100% of the height of the HTML body. Note that we must specifically declare those percentages for `<body>` and `<html>` as well.

Loading the Google Maps API

```
<html>
  <head>
    <script type="text/javascript"
      src="https://maps.googleapis.com/maps/api/js?key=API_KEY">
    </script>
```

The URL contained in the `script` tag is the location of a JavaScript file that loads all of the symbols and definitions you need for using the Google Maps API. This `script` tag is required.

The `key` parameter contains your application's API key. See Obtaining an API Key for more information.

Google Maps API for Work users should refer to Loading the Google Maps JavaScript API in the Maps API for Work documentation for important information about loading the Maps API in their applications.

HTTPS or HTTP

We think security on the web is pretty important, and recommend using HTTPS whenever possible. As part of our efforts to make the web more secure, we've made all of the Google Maps APIs available over HTTPS. Using HTTPS encryption makes your site more secure, and more resistant to snooping or tampering.

```
<script src="https://maps.googleapis.com/maps/api/js?key=API_KEY"
  type="text/javascript"></script>
```

If required, you can load the Google Maps JavaScript API over HTTP by requesting `http://maps.googleapis.com/`, or `http://maps.google.cn` for users in China.

Libraries

When loading the JavaScript Maps API via the URL you may optionally load additional *libraries* through use of the `libraries` URL parameter. Libraries are modules of code that provide additional functionality to the main JavaScript API but are not loaded unless you specifically request them. For more information, see Libraries in the V3 Maps API.

Asynchronously Loading the API

You may wish to load the Maps API JavaScript code after your page has finished loading, or on demand. To do so, you can inject your own `<script>` tag in response to a `window.onload` event or a function call, but you need to additionally instruct the Maps JavaScript API bootstrap to delay execution of your application code until the Maps JavaScript API code is fully loaded. You may do so using the `callback` parameter, which takes as an argument the function to execute upon completing loading the API.

The following code instructs the application to load the Maps API after the page has fully loaded (using `window.onload`) and write the Maps JavaScript API into a `<script>` tag within the page. Additionally, we instruct the API to only execute the `initialize()` function after the API has fully loaded by passing `callback=initialize` to the Maps API bootstrap:

```
function initialize() {
  var mapOptions = {
    zoom: 8,
    center: new google.maps.LatLng(-34.397, 150.644)
  };

  var map = new google.maps.Map(document.getElementById('map-canvas'),
      mapOptions);
}

function loadScript() {
  var script = document.createElement('script');
  script.type = 'text/javascript';
  script.src = 'https://maps.googleapis.com/maps/api/js?v=3.exp' +
      '&signed_in=true&callback=initialize';
  document.body.appendChild(script);
}

window.onload = loadScript;
```

View example (map-simple-async.html)

Map DOM Elements

```
<div id="map-canvas" style="width: 100%; height: 100%"></div>
```

For the map to display on a web page, we must reserve a spot for it. Commonly, we do this by creating a named `div` element and obtaining a reference to this element in the browser's document object model (DOM).

In the example above, we define a `<div>` named "map-canvas" and set its size using style attributes. Note that this size is set to "100%" which will expand to fit the size on mobile devices. You may need to adjust these values based on the browser's screensize and padding. Note that the map will always take its size from that of its containing element, so you must always set a size on that `<div>` explicitly.

Map Options

```
var mapOptions = {
  center: new google.maps.LatLng(-34.397, 150.644),
  zoom: 8
};
```

To initialize a Map, we first create a *`Map options`* object to contain map initialization variables. This object is not constructed; instead it is created as an object literal. There are two required options for every map: `center` and `zoom`.

```
var mapOptions = {};
```

Latitudes and Longitudes

Because we want to `center` the map on a specific point, we create a `LatLng` object to hold this location by passing the location's coordinates in the order: latitude, longitude:

```
center: new google.maps.LatLng(-34.397, 150.644)
```

LatLng object literals

Instead of creating a new `google.maps.LatLng` object each time you'd like to add a geographic coordinate, you can use a LatLng object literal. LatLng object literals are supported from version 3.16 and later. They provide a convenient way to add a coordinate, and can be used interchangably with a LatLng object in most places in the API. When you create an object, or call a method, using a LatLng object literal, the Google Maps JavaScript API will replace it with a new `google.maps.LatLng` behind the scenes.

Looking back at our earlier examples, we could use any of the following lines of code to center the map object.

```
center: new google.maps.LatLng(-34.397, 150.644)
center: {lat: -34.397, lng: 150.644}
center: {lng: 150.644, lat: -34.397}
```

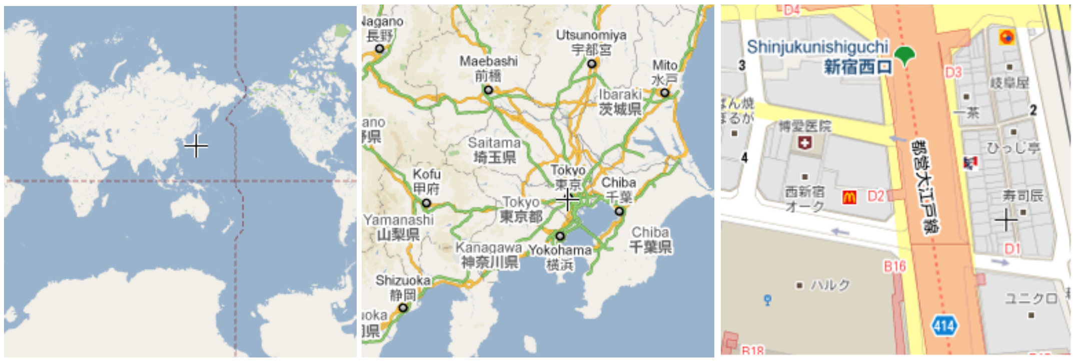
Zoom Levels

The initial resolution at which to display the map is set by the `zoom` property, where zoom `0` corresponds to a map of the Earth fully zoomed out, and higher zoom levels zoom in at a higher resolution.

```
zoom: 8
```

Offering a map of the entire Earth as a single image would either require an immense map, or a small map with very low resolution. As a result, map images within Google Maps and the Maps API are broken up into map "tiles" and "zoom levels." At low zoom levels, a small set of map tiles covers a wide area; at higher zoom levels, the tiles are of higher resolution and cover a smaller area.

The following three images reflect the same location of Tokyo at zoom levels 0,7 and 18.



For information on how the Maps API loads tiles based on the current zoom level, see Tile Coordinates in the Map Types documentation.

The Map Object

```
var map = new google.maps.Map(document.getElementById("map-canvas"),
    mapOptions);
```

The JavaScript class that represents a map is the `Map` class. Objects of this class define a single map on a page. (You may create more than one instance of this class - each object will define a separate map on the page.) We create a new instance of this class using the JavaScript `new` operator.

When you create a new map instance, you specify a `<div>` HTML element in the page as a container for the map. HTML nodes are children of the JavaScript `document` object, and we obtain a reference to this element via the `document.getElementById()` method.

This code defines a variable (named `map`) and assigns that variable to a new `Map` object, also passing in options defined within the `mapOptions` object literal. These options will be used to initialize the map's properties. The function `Map()` is known as a *constructor* and its definition is shown below:

Constructor	Description
`Map(mapDiv:Node, opts?:MapOptions )`	Creates a new map inside of the given HTML container — which is typically a DIV element — using any (optional) parameters that are passed.

Loading the Map

While an HTML page renders, the document object model (DOM) is built out, and any external images and scripts are received and incorporated into the `document` object. To ensure that our map is placed on the page after the page has fully loaded, we only execute the function which constructs the `Map` object once the `<body>` element of the HTML page receives an `onload` event. Doing so avoids unpredictable behavior and gives us more control on how and when the map is drawn.

The Google Maps JavaScript API provides a set of events that you can handle to determine state changes. For more information, see Map Events.

```
google.maps.event.addDomListener(window, 'load', initialize);
```

Alternatively, you can use the `body` tag's `onload` attribute.

```
<body onload="initialize()">
```

Troubleshooting

> **Note:** The `sensor` parameter is no longer required.

To help you get your maps code up and running, Brendan Kenny and Mano Marks point out some common mistakes and how to fix them in this video.

If your code isn't working:

- Look for typos. Remember that JavaScript is a case-sensitive language.
- Check the basics - some of the most common problems occur with the initial map creation. Such as:
  - Confirm that you've specified the `zoom` and `center` properties in your map options.
  - Ensure that you have declared a div element in which the map will appear on the screen.
  - Ensure that the div element for the map has a height. By default, div elements are created with a height of 0, and are therefore invisible.

  Refer to our examples for a reference implementation.
- Use a JavaScript debugger to help identify problems, like the one available in the Chrome Developer Tools. Start by looking in the JavaScript console for errors.
- Post questions to Stack Overflow. Guidelines on how to post great questions are available on the Support page.

Last updated April 15, 2015.