

Application Note

Google Maps JavaScript API

Dan Pittsley

April 3, 2015

Executive summary

In this document, we give a brief outline and demonstration of the Google Maps JavaScript API. We begin with an introduction to JavaScript, which is a prototype-based scripting language with dynamic typing and first-class functions. The document subsequently explains some key features of the Google Maps API – including markers and info windows, along with general syntax and layout.

Keywords – Google Maps, API, JavaScript, markers, info windows

Table of Contents

<u>Introduction</u>	3
<u>JavaScript</u>	3
Dynamic	3
Functional	3
Prototype-Based	4
Simple Examples	4
<u>Google Maps JavaScript API</u>	4
General Example	5
Markers	6
Info Windows	8
<u>Conclusion</u>	9
<u>References</u>	9

Introduction

This document will introduce the basic topics concerned with the Google Maps JavaScript API. Google Maps is a desktop and mobile web mapping service application. It allows users to view satellite images, street maps, and Street View perspectives, as well as offering functions such as route planner. Google Maps supports maps embedded on third-party websites via the Google Maps API. The Google Maps API allows developers to integrate Google Maps into their websites at no cost. The API was initially only JavaScript, and was only temporarily expanded to include an API for Adobe Flash applications. Thus, this document will also provide a brief outline of JavaScript.

JavaScript

JavaScript is a dynamic computer programming language commonly used as part of web browsers. Its implementation allows for dynamic interaction with users using client-side scripts. JavaScript is also used in server-side network programming with runtime environments such as Node.js. It is classified as a prototype-based scripting language with dynamic typing and first-class functions, making it a multi-paradigm language - supporting object-oriented, imperative, and functional programming. JavaScript supports much of the syntax from C - supporting syntax such as *if*, *for*, and *while* statements. JavaScript is dynamic, functional, prototype-based. More information on JavaScript can be found at www.w3schools.com. A great resource for learning JavaScript is www.codecademy.com.

Dynamic

Types in JavaScript are associated with values, not variables. JavaScript is mainly object-based – objects are associative arrays, augmented with prototypes. Properties and their values can be changed or deleted during run-time.

Functional

JavaScript treats functions as first-class-citizens, so the language supports passing functions as arguments to their functions. Thus, functions are like objects. JavaScript also supports functions that are passed as arguments to high-order functions, or are used to construct the result of a higher-order function.

Prototype-Based

While many other object-oriented languages use classes for inheritance, JavaScript uses prototypes. Many class-based features can be reproduced using prototypes. Functions can be used as object constructors. Placing *new* before a function call will create an instance of a prototype, inheriting properties and methods from the constructor. Another distinction between JavaScript and other object-oriented languages is that JavaScript does not differentiate between a function and method definition - the distinction occurs during the calling.

Simple Examples

The following code examples provide a very basic demonstration of simple JavaScript code. For a more detailed explanation, see the websites mentioned above.

Hello World

```
console.log("Hello World!");
```

Variables

```
var name = value;
```

```
var x; // defines the variable x with no default value
var y = 2; // defines the variable y with a value of 2
```

Recursive Function

```
function factorial(n) {
    if (n === 0) {
        return 1;
    }
    return n * factorial(n - 1);
}
```

Anonymous Function

```
var displayClosure = function() {
    var count = 0;
    return function () {
        return ++count;
    };
}
var inc = displayClosure();
inc(); // returns 1
inc(); // returns 2
inc(); // returns 3
```

Google Maps JavaScript API

The remainder of this document is designed for people familiar with JavaScript or object-oriented programming concepts.

General Example

The best way to begin this section is with some example code; we will subsequently explain the key components. The following web page displays a map centered on Sydney, Australia. The bolded numbers preceding every fifth line enumerate the line numbers. To view this on a web page, see references section.

```
1 <!DOCTYPE html>
  <html>
    <head>
      <style type="text/css">
5      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>
10     <script type="text/javascript">
        function initialize() {
          var mapOptions = {
            center: { lat: -34.397, lng: 150.644},
            zoom: 8
15          };
          var map = new google.maps.Map(document.getElementById('map-canvas'),
            mapOptions);
          }
        google.maps.event.addDomListener(window, 'load', initialize);
20     </script>
    </head>
    <body>
      <div id="map-canvas"></div>
    </body>
25 </html>
```

This basic example has several important things to observe:

- **Line 1:** <!DOCTYPE html> declares the application HTML5. Google recommends that a true DOCTYPE be declared within the web application. This will allow the application to be compatible with more browsers, and degrade “gracefully”.
- **Lines 7-9:** The <script> tag includes the Maps API JavaScript. The URL in the script tag is the location of a JavaScript file that loads all of the symbols and definitions that are needed to use the Google Maps API. API_KEY is the applications specific API key, which is obtained from Google.

- **Line 23:** For the map to display, there must be a spot for it. In the example above, the entire map is simply held in a <div> tag named “map-canvas”. The map will always be the same size as the element that contains it.
- **Lines 12-15:** A JavaScript object literal is created to hold some map properties - namely, location (latitude, longitude) and zoom (resolution at which to display the map). These are the two required options.
- **Line 16:** A JavaScript “map” object is created. This object is passed the div element along with the map properties. Objects of the JavaScript *Map* class define a single map on the page. A new instance of this class is created using the JavaScript new operator. This object is not constructed, but it is created as an object literal. When a new map instance is created, you specify a <div> HTML element in the page as a container for the map. See the function Map() constructor below:

```
Map(mapDiv:Node, opts?:MapOptions )
```

This creates a new map inside of the given HTML container – which is typically a DIV element – using any (optional) parameters that are passed.

- **Line 19:** An event listener is used to load the map after the page is loaded. This feature can be changed to allow the Maps API JavaScript code on demand. While an HTML page renders, the document object model (DOM) is built, and any external images and scripts are received and incorporated into the document object. To make sure the map is displayed after fully loading the page, we only execute the function that constructs the Map object once the <body> element of the HTML page receives an “onload” event.

Markers

A marker identifies a location on a map. They are depicted as a red drop pin by default, but they can also display a custom image. You can set a custom marker within the marker’s constructor, or by calling setIcon() on the marker. Markers are a type of *overlay* and are designed to be interactive. As such, they receive ‘click’ events, and can be designed to be draggable.

Adding a Marker

To add a marker, use the `google.maps.Marker` constructor. This takes a single *Marker* options object literal, specifying the initial properties of the marker. Two important fields for construction of a marker are *position* and *map*:

- **position** is required when construction a marker. It specifies the *LatLng*, and identifies the initial location of the marker.
- **map** is an optional field, which specifies the *Map* on which to place the marker. Without using the *map* option, the marker is created, but not displayed on the map. You can still add the marker later by calling `setMap()`.

In the example below, the marker is placed on the map at construction of the marker using the *map* property in the marker options.

```
var myLatLng = new google.maps.LatLng(-25.363882,131.044922);
var mapOptions = {
  zoom: 4,
  center: myLatLng
}
var map = new google.maps.Map(document.getElementById("map-canvas"),
mapOptions);

// To add the marker to the map, use the 'map' property
var marker = new google.maps.Marker({
  position: myLatLng,
  map: map,
  title:"Hello World!"
});
```

Removing a Marker

To remove a marker, simply call `setMap()` with a null argument. This does not delete the marker, but removes it from the map. If you want to delete the marker, you must set the marker itself to null. Below is an example for removing a marker.

```
marker.setMap(null);
```

Animate a Marker

It is also possible to animate a marker using the Google Maps API. This will allow the markers to have dynamic movement during different situations. Using the marker's *animation* property specifies the animation for that marker. Animation property is

of type `google.maps.Animation`. Two Animation values that are supported are DROP and BOUNCE.

- **DROP** indicates that the marker should drop from the top of the map to its final location when it is first placed. After this action, the animation will automatically revert to null.
- **BOUNCE** indicates that the marker should bounce in place. This bouncing animation continues to occur unless its animation property is explicitly set to null.

Info Windows

Info windows display content in a popup window above the map, at a given location. The info window has a content area and tapered stem, which is attached to a specific location. These are usually attached to markers (red pins).

The *InfoWindow* constructor takes an *InfoWindowOptions* object literal, which specifies the initial parameters for displaying the info window. The *InfoWindowOptions* contains the following fields:

- ***content*** contains either string of text or DOM node to display in the info window.
- ***pixelOffset*** contains an offset from the tip of the info window to the location on which the info window is anchored.
- ***position*** contains the LatLng at which the info window is anchored.
- ***maxWidth*** specifies the maximum width of the info window in pixels.

Info windows are not displayed automatically on the map. To make the info window visible, one must call `open()` on the *InfoWindow*, passing it the Map on which to open. One should also provide the Marker with which to anchor the info window. Info windows can be closed by calling `close()`, or it will close when the user clicks the close control. You can change the location of an info window by calling `setPosition()` on the info window.

Conclusion

In this document, we have given a brief outline and demonstration of the Google Maps JavaScript API. The document has presented the most useful and basic concepts that are necessary in order to develop using Google Maps. The reader should now be able to create basic applications using this document for reference. Additional resources about JavaScript and Google Maps API can be found at www.tutorialspoint.com/javascript/ and <https://developers.google.com/maps/>, respectively.

References

- <https://developers.google.com/maps/documentation/javascript/examples/map-simple>
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/New_in_JavaScript/1.8.5
- <https://www.apps.ietf.org/rfc/rfc4329.html#sec-7.1>
- <https://www.crockford.com/javascript/little.html>
- <https://developers.google.com/maps/documentation/javascript/>
- <https://developer.apple.com/library/mac/documentation/Miscellaneous/Reference/UTIRef/Articles/System-DeclaredUniformTypeIdentifiers.html>