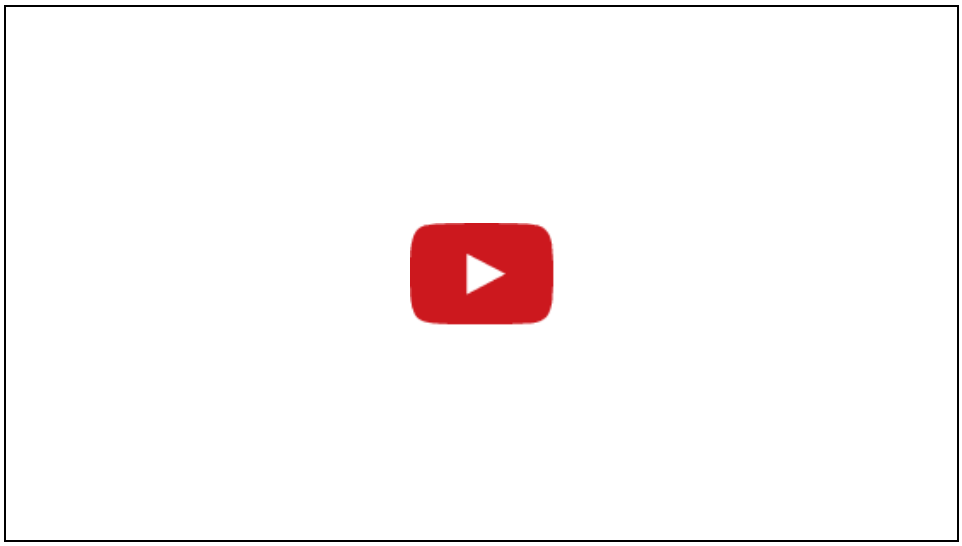
Developing with the JavaScript Client Library

Contents

Introduction

The JavaScript client library makes it easier for you to write JavaScript that works with Google services (Calendar, Analytics, etc.) through their APIs. This page shows you how to use the JavaScript client library by example.

This video from Google I/O 2012 is a good overview of what Google APIs are and how to consume them using the JavaScript client library:



Supported environments

The JavaScript client library supports the following browser environments:

- Chrome 8+
- Firefox 3.5+
- MSIE 8+
- Safari 4+

How to access a Google service using the JavaScript client library

To interact with a Google service using the JavaScript client library, your code must do the following:

- Load the JavaScript client library.
- Set the access credentials.
- Load the API for the service you want to work with.
- Initialize an object that encapsulates the request you want to make.
- Execute the request object.

The following sections go through each of these tasks.

Loading the client library and the API

To load the JavaScript client library, your code can use a script tag like the following:

```
<script src="https://apis.google.com/js/client.js?onload=OnLoadCallback"></script>
```

Setting credentials

To get access to Google services using the service APIs, your application must get credentials recognized by Google. Google defines two API access levels:

Level	Description	Requires:
Simple	API calls do not access any private user data	API key
Authorized	API calls can read and write private user data, or the application's own data	OAuth 2.0 credential

To get Google credentials, follow the instructions on the Getting Started page.

For example, your code can set the API key using the `gapi.client.setApiKey` method, as follows:

```
gapi.client.setApiKey(YOUR API KEY);
```

For details and other code samples that show you how to present credentials to the Google Auth server using the JavaScript client library, see the the Authentication page.

Building and executing the request

The JavaScript client library supports three alternatives for assembling an API request and executing it:

- Load the API interface first, then initialize the service object to include the parameters of the request, then call this object's `then` method, in separate commands.
- Assemble a REST request that specifies the API and the request parameters, then pass it to the client library's `request` method.
- Manually assemble and execute the request using CORS.

Option 1: Load the service API, then assemble the request

Once the JavaScript client library is loaded, your code can load a specific API with a command like the following:

```
gapi.client.load(API_NAME, API_VERSION, CALLBACK);
```

Where

- *API_NAME* is the name of the API,
- *API_VERSION* is the version of the API,
- *CALLBACK* is an optional callback function to be executed once the API has been loaded. If a callback is not provided, a promise is returned.

For example, this code loads version 1 of the Google+ API:

```
gapi.client.load('plus', 'v1').then(function() { console.log('loaded.'); });
```

> **Note:** A call to `gapi.client.load` results in a network request to the Discovery API. It is only necessary to call this once per API on a given page.

To review a list of supported APIs and versions, go to the APIs Explorer.

The code that assembles the API request follows this pattern:

```
// Returns a request object which can be executed (as below) or batched
gapi.client.METHOD_NAME(PARAMETERS_OBJECT).
  then(onFulfilled, onRejected);
```

For example, you can send a search request to the Google+ API using the following method:

```
gapi.client.plus.activities.search({'query': 'Google+', 'orderBy': 'best'}).then(function(res
p) {
  console.log(resp.result);
}, function(reason) {
  console.log('Error: ' + reason.result.error.message);
});
```

To test API methods on the fly, or see an individual API's documentation for detailed method descriptions, try the APIs Explorer.

> **Note:** The syntax for this example and others on this page is the same for authorized and unauthorized requests. For information about how to prompt the user for authorization using the JavaScript client library, see the Authentication page.

When you use this pattern to assemble and then execute a request, you have two alternatives for the syntax of the *PARAMETERS_OBJECT*: REST or JSON-RPC.

The documentation for each Google service API specifies the REST syntax for its methods. (For links to the documentation for one of these APIs, follow the link for that API on the APIs Explorer page.)

Option 2: Use `gapi.client.request`

A more general way to make requests is to use `gapi.client.request`. Your code does not have to load the API first, as in the first option, but it must still set the API key. While you need to manually fill in REST parameters with this option, it saves one network request for loading the API.

To execute the same Google+ search as in the first example, using `gapi.client.request`:

```
var restRequest = gapi.client.request({
  'path': '/plus/v1/activities',
  'params': {'query': 'Google+', 'orderBy': 'best'}
});
restRequest.then(function(resp) {
  console.log(resp.result);
}, function(reason) {
  console.log('Error: ' + reason.result.error.message);
});
```

For more on making REST requests, see the API docs for each supported API.

Option 3: Use CORS

Google APIs support CORS. For more information on using CORS to make requests, see the CORS Support page.

Last updated October 2, 2014.