

In this session, you will learn how to use the features of the Google API client for JavaScript to build rich web applications. Some of the features we will demonstrate include authentication and CORS



About Brendan O'Brien

[View full profile](#)

Brendan is an engineer working on the Google+ team in Mountain View. He has eight years experience coding in JavaScript and four at Google where he has worked on both UI and JS infrastructure.



Building Web Applications that use Google APIs and the JavaScript Client for Google APIs

Brendan O'Brien
Software Engineer, Google+

Google APIs

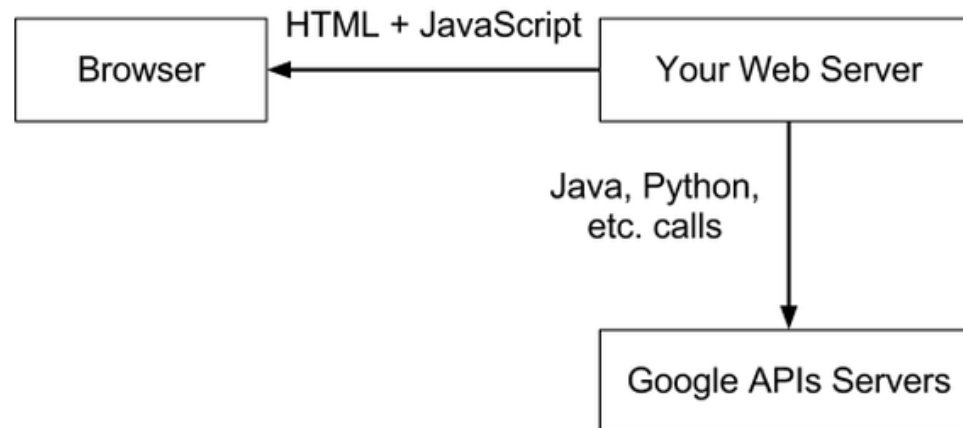
- Google provides RESTful APIs for many of its products and services
- Google+, Calendar, Analytics, and more than 35 other Discovery-based APIs
- Requests made to <https://www.googleapis.com>, Google's APIs frontend

GET <https://www.googleapis.com/urlshortener/v1/url?shortUrl=http://goo.gl/fbsS>

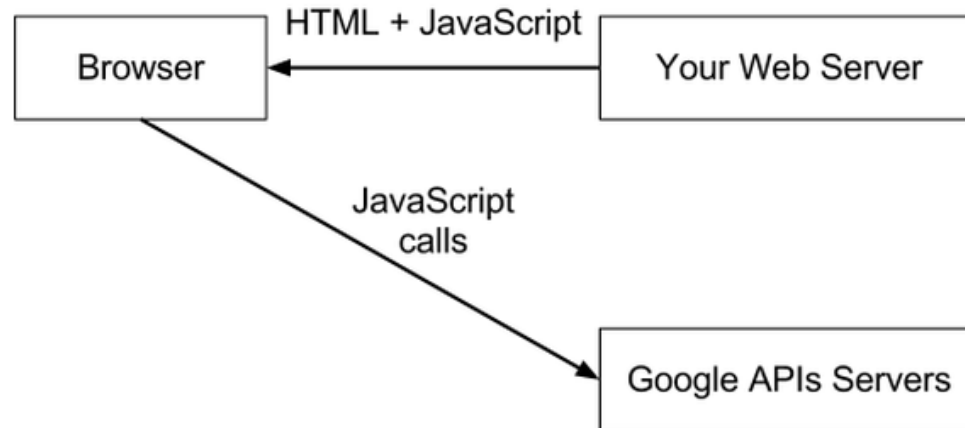
API clients

- For ease of access, Google provides API clients in many languages
- Google maintains nine clients including JavaScript, Java, Objective-C, and Python
- [Google's API Clients](#)

Previously: Google APIs from a web server



Google APIs from JavaScript and HTML



Demo

The code

JAVASCRIPT

```
function load() {  
  gapi.client.setApiKey('AIQaZyAdjHPT5tb7Nu56WJ_nlrMGOAgUgtKjiPM');  
  gapi.client.load('urlshortener', 'v1', makeRequest);  
}  
  
function makeRequest() {  
  function handleResponse(resp) { console.log(resp); }  
  
  var request = gapi.client.urlshortener.url.get({  
    'shortUrl': 'http://goo.gl/fbsS'  
  });  
  request.execute(handleResponse);  
}
```

Demo

In action

Enter a short URL and click "Expand URL" to get the full URL.

Short URL

Results



Google APIs JavaScript Client

About the client

Compatibility

- Supported browsers
 - Chrome 8+
 - Firefox 3.5+
 - MSIE 8+
 - Safari 4+
- Supported authorization protocols
 - OAuth 2.0
- Supported APIs
 - [Google APIs Explorer](#)

What does the client provide?

RESTful requests

APIs are based on the REST protocol, also the basis for HTTP. URLs identify resources, and HTTP verbs act on those resources.

```
var restRequest = gapi.client.request({  
  'path': '/urlshortener/v1/url',  
  'params' : {'shortUrl' : 'http://goo.gl/fbsS'}  
});  
restRequest.execute(function(jsonResponse, rawResponse) {  
  // Handle request result  
});
```

JAVASCRIPT

What does the client provide?

JSON-RPC

JSON-RPC is another request format for making API calls. The request is encapsulated as JSON in a POST body, and the response is JSON.

```
var rpcRequest = gapi.client.urlshortener.url.get({  
  'shortUrl': 'http://goo.gl/fbsS'  
});  
rpcRequest.execute(function(jsonResponse, rawResponse) {  
  // Handle request result  
});
```

JAVASCRIPT

What does the client provide?

OAuth 2.0 authentication and authorization

- Interface to authenticate user and authorize app
- Provide Client ID to identify app, and set of scopes to request. Get Access Token back
- Access token automatically sent with all requests

```
var clientId = '837050751313';  
var scopes = 'https://www.googleapis.com/auth/calendar';  
gapi.auth.authorize({client_id: clientId, scope: scopes, immediate: false}, handleAuthResult);
```

JAVASCRIPT



Using the Google APIs JavaScript Client

Before you begin

APIs Console

- The APIs Console is where developers
 - Create an API key and Client ID to identify their app
 - Activate APIs the app needs to access
 - Manage their project: Quota, billing, team management

Before you begin

APIs Explorer and documentation

- Starting point for documentation
- Learn about individual APIs
- Try API calls directly from the explorer
- [APIs Explorer](#)
- Demo short URL: <http://goo.gl/fbsS>

Loading the JS Client

The URL for the JS Client is `https://apis.google.com/js/client.js`

The JS Client loads asynchronously in two phases. Use the `?onload` parameter to use the client once it has finished loading.

```
<html>
  <head>
    <script>
      function init() {
        // Use the JS Client
      }
    </script>
    <script src="https://apis.google.com/js/client.js?onload=init">
    </script>
  </head>
</html>
```

HTML

Making a REST request

Method description

REST requests allow the manipulation of HTTP method, headers, and body.

```
gapi.client.request({  
  'path': pathToMakeRequest,    // String, Required  
  'params': urlParams,          // Key-value object, Optional.  
  'method': httpMethod,        // String, Optional. Defaults to GET  
  'headers': httpHeaders,      // Key-value object, Optional.  
  'body': httpBody,            // String, Optional.  
  'callback': callbackFunction // Function, Optional. If provided, request is executed immediately.  
});
```

JAVASCRIPT

Making a REST request

Example

JAVASCRIPT

```
gapi.client.request({
  'path': '/calendar/v3/users/me/calendarList',
  'method': 'POST',
  'headers': {
    'Content-Type': 'application/json' // This is the default, included for illustration
  },
  'body': JSON.stringify({
    'id': 'obrien.io.demo@gmail.com',
    'selected': true
  }),
  'callback': function(jsonResponse, rawResponse) {
    // Response is inserted Calendar
  }
});
```

Making a REST request

Callback

REST callback function take two parameters

- `jsonResponse` is the response parsed as JSON. It will be `false` if the response is not valid JSON.
- `rawResponse` is the raw HTTP response as a string. It contains not just the response in the body but also the headers, status, and statusText.

Making a JSON-RPC request

Directly creating an RpcRequest

The RpcRequest class can be created directly via the `gapi.client.rpcRequest` method.

Format:

```
gapi.client.rpcRequest(method, apiVersion, rpcParams);
```

JAVASCRIPT

Example:

```
var request = gapi.client.rpcRequest('plus.people.search', 'v1', {'query': 'Lewis'});  
request.execute(requestCallback);
```

JS

Making a JSON-RPC request

Registered methods

```
gapi.client.register('plus.people.search', {'apiVersion': 'v1'});
```

JAVASCRIPT

The above call registers the `plus.people.search` method:

```
gapi.client.plus.people.search(rpcParams)
```

Executing a registered JSON-RPC method returns an `RpcRequest` class.

```
var request = gapi.client.plus.people.search({'query': 'Lewis'});  
function requestCallback(jsonResponse, rawResponse) {  
  // Do something with jsonResponse (Object) or rawResponse (string)  
}  
request.execute(requestCallback);
```

JAVASCRIPT

Loading an API

Registering an API surface

An API can be loaded and pre-registered before use. This is not required, but provides some convenience.

```
gapi.client.load('plus', 'v1', function onReady() {  
  // Executes when the requested API is ready to use  
});
```

JAVASCRIPT

This registers all the methods which the Google+ API supports in the `gapi.client` namespace.

Making a JSON-RPC request

Handling JSON-RPC responses

JAVASCRIPT

```
var request = gapi.client.plus.people.search({'query': 'Lewis'});  
function requestCallback(jsonResponse, rawResponse) {  
  // Do something with jsonResponse (Object) or rawResponse (string)  
}  
request.execute(requestCallback);
```

- **jsonResponse:** Extracts the method response value from HTTP Request and parses as valid JSON. Will be **false** if the response is not valid JSON.
- **rawResponse:** The entire RPC response as a string. This is always present.

Making a JSON-RPC request

Batch operations

RpcRequests can be batched and executed in parallel. Callbacks can be supplied for individual requests, as well as a callback for the entire operation.

JAVASCRIPT

```
var batch = gapi.client.newRpcBatch();
var requestLarry = gapi.client.rpcRequest('plus.people.search', 'v1', {'query': 'Larry'});
var requestSergey = gapi.client.plus.people.search({'query': 'Sergey'});
function larryCallback(jsonResponse, batchResponse) {};
batch.add(requestLarry, {
  'id': 'larry', // Optional, used to identify responses.
  'callback': larryCallback // Optional
});
batch.add(requestSergey);
function batchCallback(jsonResponse, batchResponse) {};
batch.execute(batchCallback);
```

Making a JSON-RPC request

Batch operations callbacks

Individual batch callbacks are executed first, then the overall batch callback. All batch callbacks get two parameters, similar to individual RpcRequests.

- **jsonResponse:** Response parsed as JSON
 - For individual callbacks: This is the response for the individual operation
 - For batch callbacks: This is all responses as a map of batch ID to response
- **batchResponse:** The full batch response as a string. Always contains all batch responses

```
batchResponse = {  
  'larry': { /* People search results for 'larry' */ },  
  'random1234': { /* People search results for 'sergey' */ }  
};
```

JAVASCRIPT

Using the CORS transport

For browsers which support CORS, developers can make direct XHR calls to <https://www.googleapis.com>.

CORS requests are constructed similar to REST requests, by constructing the path, params, headers, and body.

```
var xhr = new XMLHttpRequest();
xhr.open('POST', 'https://www.googleapis.com/urlshortener/v1/url?key=AIzaDyAdjuPT5Pb4Nu56WJ_nlrMGOAgUAtKjiPM');
xhr.setRequestHeader('Content-Type', 'application/json');
xhr.onload = function() {
    alert(xhr.responseText);
};
xhr.send(JSON.stringify({
    'longUrl': 'http://www.google.com'
}));
```

JAVASCRIPT

Why CORS?

- Native transport obviates need for custom cross-domain handling
- Also allows for faster requests
- Standards-compliant, non-proprietary, native to major modern browsers.

Encouraged to try it, with caveats that browser support is currently partial and JS Client updates will be missed.

Can I use CORS? <http://caniuse.com/#search=cors>

Request format summary

	JSON-RPC	REST
Set & read HTTP method, headers, and body	No	Yes
Batching	Built-in	By hand
Auto-loading APIs	Yes	No

Transport summary

	JS Client (XD3)	CORS
Request Auto-Formatting	Yes	No
Cross-Browser	Yes	Partial
Browser native	No	Yes
Built-in auth	Yes	No



Google APIs JavaScript Client

Authentication and authorization with OAuth 2.0

OAuth 2.0 overview

"OAuth is a security protocol that enables users to grant third-party access to their web resources without sharing their password"^[1]

OAuth consists of two pieces: Authentication and authorization

Links

- OAuth 2.0 Official Site and Working Draft
<http://oauth.net/2/>
- Google Developers OAuth 2.0 Documentation
<https://developers.google.com/accounts/docs/OAuth2>
- Brief intro from Hueniverse.com
<http://hueniverse.com/2010/05/introducing-oauth-2-0/>

¹<http://hueniverse.com/2010/05/introducing-oauth-2-0/>

OAuth 2.0

Authorization and authentication

Authorize

Retrieves your profile name using the Google Plus API.

OAuth 2.0

API Key and Client ID

To identify your app and access unprotected resources, use an API Key.

For protected resources, a Client ID is exchanged for an access token, which provides access to a user's protected resources.

<https://code.google.com/apis/console/>

Auth in the JS Client

JAVASCRIPT

```
function init() {
  gapi.client.setApiKey(apiKey);
  window.setTimeout(checkAuth, 1);
}

function checkAuth() {
  gapi.auth.authorize({client_id: clientId, scope: scopes, immediate: true}, handleAuthResult);
}

function handleAuthResult(authResult) {
  var authorizeButton = document.getElementById('authorize-button');
  if (authResult && !authResult.error) {
    authorizeButton.style.visibility = 'hidden';
    makeApiCall();
  } else {
    authorizeButton.style.visibility = '';
    authorizeButton.onclick = handleAuthClick;
  }
}

function handleAuthClick(event) {
  gapi.auth.authorize({client_id: clientId, scope: scopes, immediate: false}, handleAuthResult);
}
```

Auth in the JS Client

JAVASCRIPT

```
function handleClientLoad() {
  gapi.client.setApiKey(apiKey);
  window.setTimeout(checkAuth, 1);
}

function checkAuth() {
  gapi.auth.authorize({client_id: clientId, scope: scopes, immediate: true}, handleAuthResult);
}

function handleAuthResult(authResult) {
  var authorizeButton = document.getElementById('authorize-button');
  if (authResult && !authResult.error) {
    authorizeButton.style.visibility = 'hidden';
    makeApiCall();
  } else {
    authorizeButton.style.visibility = '';
    authorizeButton.onclick = handleAuthClick;
  }
}

function handleAuthClick(event) {
  gapi.auth.authorize({client_id: clientId, scope: scopes, immediate: false}, handleAuthResult);
}
```

Auth in the JS Client

JAVASCRIPT

```
function handleClientLoad() {
  gapi.client.setApiKey(apiKey);
  window.setTimeout(checkAuth, 1);
}

function checkAuth() {
  gapi.auth.authorize({client_id: clientId, scope: scopes, immediate: true}, handleAuthResult);
}

function handleAuthResult(authResult) {
  var authorizeButton = document.getElementById('authorize-button');
  if (authResult && !authResult.error) {
    authorizeButton.style.visibility = 'hidden';
    makeApiCall();
  } else {
    authorizeButton.style.visibility = '';
    authorizeButton.onclick = handleAuthClick;
  }
}

function handleAuthClick(event) {
  gapi.auth.authorize({client_id: clientId, scope: scopes, immediate: false}, handleAuthResult);
}
```

Refreshing the access token

When authorization succeeds the returned auth token contains an expiration field: `expires_in`. Use this value to schedule an access token refresh, which typically expires in one hour.

```
function handleAuthResult(authResult) {  
  if (authResult && !authResult.error) {  
    // Subtract five minutes from expires_in to ensure timely refresh  
    var authTimeout = (authResult.expires_in - 5 * 60) * 1000;  
    setTimeout(checkAuth, authTimeout);  
  }  
}
```

JAVASCRIPT

Note: This field does not auto-update, the value should only be trusted just after initial or refresh auth succeeds

Making authorized calls

To make OAuth 2.0 calls, add an Authorization header field with the value "Bearer <access_token>"

The JS Client handles this for you. Once authorization is complete, the header is automatically added to all API calls.

Authorization: Bearer ya29.AHES6ZQnT_mCNMi1Z49FG3-os0oAjGMu5vFpQwN1c1K3X7o

JS Client Links

- Home - <http://code.google.com/p/google-api-javascript-client/>
- Discussion group - <https://groups.google.com/forum/?fromgroups#!forum/google-api-javascript-client>
- Announcements blog - [Google API JavaScript Client Discussion Group](#)

I/O Sessions and Codelabs

- [Building Web applications in JavaScript that use Google APIs \(Codelab\)](#)
- [Optimizing Your Code Using Features of Google APIs \(Session\)](#)
- [OAuth 2.0 for Identity and Data Access \(Session\)](#)

<Thank You!>



g+ <https://plus.google.com/116812371442532868406>
twitter @ICodeJS