Getting Started

Contents

Basic ideas

This section provides a basic explanation of how your code can use the JavaScript client library to interact with a Google service (such as Calendar, Books, or Analytics).

How an application makes an API request using the JavaScript client library

There are many ways to use the JavaScript client library to make API requests, but they all follow the same basic pattern:

1. The application loads the JavaScript client library.
2. The application references its API key, which authenticates the application with Google services.
3. If the application needs access to the user's personal information, it opens a session with a Google auth server. The auth server opens a dialog box which prompts the user to authorize the use of personal information.
4. The application loads the API for the Google service.
5. The application initializes a request object (also called a service object) that specifies the data to be returned by the API.
6. The application executes the request and processes the data returned by the API.

The next section describes shows how these tasks could be handled in a web page's JavaScript.

How it looks in JavaScript

This example code is adapted from authSample.html. It is shown here to give you a quick impression of how the JavaScript client library is used.

For a discussion of this code, see the Authentication page.

```
<!--Add a button for the user to click to initiate auth sequence -->
<button id="authorize-button" style="visibility: hidden">Authorize</button>
<script type="text/javascript">

  var clientId = '837050751313';

  var apiKey = 'AIzaSyAdjHPT5Pb7Nu56WJ_nlrMGOAgUAtKjiPM';
```

```
      var scopes = 'https://www.googleapis.com/auth/plus.me';

      function handleClientLoad() {
        // Step 2: Reference the API key
        gapi.client.setApiKey(apiKey);
        window.setTimeout(checkAuth,1);
      }

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

      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) {
        // Step 3: get authorization to use private data
        gapi.auth.authorize({client_id: clientId, scope: scopes, immediate: false}, handleAut
hResult);
        return false;
      }

      // Load the API and make an API call.  Display the results on the screen.
      function makeApiCall() {
        // Step 4: Load the Google+ API
        gapi.client.load('plus', 'v1').then(function() {
          // Step 5: Assemble the API request
          var request = gapi.client.plus.people.get({
            'userId': 'me'
          });
          // Step 6: Execute the API request
          request.then(function(resp) {
            var heading = document.createElement('h4');
            var image = document.createElement('img');
            image.src = resp.result.image.url;
            heading.appendChild(image);
            heading.appendChild(document.createTextNode(resp.result.displayName));

            document.getElementById('content').appendChild(heading);
          }, function(reason) {
            console.log('Error: ' + reason.result.error.message);
          });
        });
      }
    </script>
    // Step 1: Load JavaScript client library
    <script src="https://apis.google.com/js/client.js?onload=handleClientLoad"></script>
```

> **Note:** Because of the way the onload handler works, it must be defined after `handleClientLoad`.

Setup

Follow the steps in this section to get set up to use the JavaScript client library in your application.

Get a Google Account

First, sign up for a Google Account if you do not already have one.

Choose Google services

Next, choose which of Google's online services your application will use. See the APIs Explorer for information about Google services with APIs that the JavaScript client library can work with.

Get access keys for your application

Google defines two levels of API access:

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 apply for API access at the Google APIs Console:

1. Visit the Google APIs Console. Log in if prompted to do so.
2. Create a project for your application (if you have not already done so) by clicking **Create project**.
3. Select **Services** from the menu. The list of accessible Google services appears.
4. Scroll through the list. For each service you want your application to work with, click the **Status** switch next to the service name (so that it switches from **OFF** to **ON**.
   For some services, the Console will display a **Terms of Service** pane. To go ahead, check the **I agree to these terms** box, then click **Accept**.
5. Scroll back to the top of the page and click **API Access** in the menu.
   The **API Access** pane appears.
6. What you do next depends on what level of access your application needs:
   - For simple API access, note the API key (a long string) in the **Simple API Access** box. That's all you need.
   - For authorized access, go on to the next steps.
7. Click **Create an OAuth 2.0 client ID**.
   The **Create Client ID** dialog appears.
8. Click the **Web application** radio button. Type your site or hostname in the field below.
9. Click **more options** under the radio buttons.
   The **Authorized Redirect URIs** and **Authorized JavaScript Origins** fields appear.
10. Clear any text in the **Authorized Redirect URIs** box. (When using JavaScript, do not specify any redirects.)
11. In the **Authorized JavaScript Origins** box, type the protocol and domain for your site.
    Make sure to enter the domain only, do not include any path value.
    If your site supports both HTTP and HTTPS, you can enter multiple values, one per line.
    Example: if your application supports both HTTP and HTTPS, and your host is `www.example.com`, you should enter the following values:

Authorized Redirect URIs	*leave empty*
Authorized JavaScript Origins	http://www.example.com https://www.example.com

12. Click the **Create client ID** button to complete the process.
    The **Create client ID** dialog disappears. The **Authorized API Access** section now displays your application's OAuth 2.0 credentials.

For information about how to use the OAuth 2.0 Client credential in your application, see the Authentication page.

Where to go from here

The links in the left-hand menu on this page give you access to all the pages in the JavaScript client library docset.

- Developing with the JavaScript client library provides implementation details for the steps described in the Basic ideas section above.
- **Development Topics**: Sub-pages under "Development topics" go into detail about authentication and using CORS, promises, and batch.
- Samples provides code snippets that demonstrate how various features and operations can be implemented.
- Methods and classes provides detailed reference information about each class and method defined for the JavaScript client library.

Last updated October 29, 2014.