Last updated May 23, 2017. | Unknown author

Authorizing and Making Authorized Requests

The following sample demonstrates how to get "authorized" access to a Google API using OAuth 2.0. See the full sample at authSample.html.

```
  type="text/javascript"
   function handleClientLoad() {
     // Loads the client library and the auth2 library together for efficiency.
     // Loading the auth2 library is optional here since `gapi.client.init`
function will load
     // it if not already loaded. Loading it upfront can save one network
request.
     gapi.load('client:auth2', initClient);
   }

   function initClient() {
     // Initialize the client with API key and People API, and initialize OAuth
with an
     // OAuth 2.0 client ID and scopes (space delimited string) to request
access.
     gapi.client.init({
         apiKey: 'YOUR_API_KEY',
         discoveryDocs: ["https://people.googleapis.com/$discovery/rest?
version=v1"],
         clientId: 'YOUR_WEB_CLIENT_ID.apps.googleusercontent.com',
         scope: 'profile'
     }).then(function () {
       // Listen for sign-in state changes.
       gapi.auth2.getAuthInstance().isSignedIn.listen(updateSigninStatus);
```

```
      // Handle the initial sign-in state.
      updateSigninStatus(gapi.auth2.getAuthInstance().isSignedIn.get());
    });
  }

  function updateSigninStatus(isSignedIn) {
    // When signin status changes, this function is called.
    // If the signin status is changed to signedIn, we make an API call.
    if (isSignedIn) {
      makeApiCall();
    }
  }

  function handleSignInClick(event) {
    // Ideally the button should only show up after gapi.client.init finishes,
so that this
    // handler won't be called before OAuth is initialized.
    gapi.auth2.getAuthInstance().signIn();
  }

  function handleSignOutClick(event) {
    gapi.auth2.getAuthInstance().signOut();
  }

  function makeApiCall() {
    // Make an API call to the People API, and print the user's given name.
    gapi.client.people.people.get({
      'resourceName': 'people/me',
      'requestMask.includeField': 'person.names'
    }).then(function(response) {
      console.log('Hello, ' + response.result.names[0].givenName);
    }, function(reason) {
      console.log('Error: ' + reason.result.error.message);
    });
  }

 async defer src="https://apis.google.com/js/api.js"
  onload="this.onload=function(){};handleClientLoad()"
  onreadystatechange="if (this.readyState === 'complete') this.onload()"
```

```
id="signin-button" onclick="handleSignInClick()"Sign In
id="signout-button" onclick="handleSignOutClick()"Sign Out
```

It's called "authorized" access because the user must give the application direct authorization to use personal data. Simple web-based applications using JavaScript usually get this authorization the way this example does: by displaying button for the user to click. This action triggers a call to a Google auth server, which pops up a standard authorization dialog. For details, see the Authentication page.

Note: Here we use gapi.load('client:auth2', ...) to load both the client module (for dealing with API requests) and the auth2 module (for dealing with OAuth 2.0) upfront. The gapi.client.init fuction lazily loads auth2 if it is needed. If you are sure your app needs auth, loading the two modules 'client:auth2' together before you call gapi.client.init will save one script load request.

To make gapi.client.init set up OAuth correctly, you would have to assign the clientID variable the client ID generated when you registered your application (again, for instructions see the Getting Started page). The other parameter is scope, which in this case is just the scope for user profile permission.

When the user clicks **Authorize**, the gapi.auth2.getAuthInstance().signIn() function is called, which shows user a popup window to let user authorize. Note that the gapi.auth2.getAuthInstance().signIn() can be only called from a user interaction context for most browsers (i.e. do not call it when your app starts, but call it in a button click handler).

> **Note:** when you authorize your application using Oauth 2.0, you do not also need to set the API key as in the first example. However, it is a good practice to do so, in case your code ever expands to handle unauthorized requests.

Viewed using Just Read