Adding the HTML sign-in button to your page

The easiest approach to getting up and running with Google+ Sign-In is to add the HTML-based sign-in button to your page. The sign-in button is automatically rendered for you based on the attributes that you define for the button.

Try it

The button below triggers the OAuth 2.0 sign-in flow and will output the authorization result object. You can try granting access, denying access, or closing the authorization dialog to see the results of each action.

Sign in

Reload the example or open in a new window

You can also render a button using JavaScript or you can initiate the sign-in flow with JavaScript. These options are also useful when you require a custom button or when you need to time the creation of the button if your site loads its content dynamically.

Add the HTML sign-in button

When your users sign in with Google, your app gets an *access token*, which you can use to make calls to the Google+ API on behalf of your user while the user is active on your site.

Adding a Google+ Sign-In button to your page requires you to:

Step 1: Create a client ID and client secret.
Step 2: Include the Google+ script on your page.
Step 3: Choose your OAuth scopes.
Step 4: Add a sign-in button to your page.
Step 5: Handle the sign in with a JavaScript callback.

Step 1: Create a client ID and client secret

To create a client ID and client secret, create a Google Developers Console project, enable the Google+ API, create an OAuth 2.0 client ID, and register your JavaScript origins:

1. Go to the Google Developers Console.
2. Select a project, or create a new one by clicking **Create Project**:

> **Note:** Use a single project to hold all platform instances of your app (Android, iOS, web, etc.), each with a different Client ID.

   a. In the **Project name** field, type in a name for your project.
   b. In the **Project ID** field, optionally type in a project ID for your project or use the one that the console has created for you. This ID must be unique world-wide.
   c. Click the **Create** button and wait for the project to be created.

   d. Click on the new project name in the list to start editing the project.
3. In the left sidebar, select the **APIs** item below "APIs & auth". A list of Google web services appears.
4. Find the **Google+ API** service and set its status to **ON**—notice that this action moves the service to the top of the list. You can turn off the Google Cloud services if you don't need them. Enable any other APIs that your app requires.
5. In the sidebar under "APIs & auth", select **Consent screen**.
   a. Choose an **Email Address** and specify a **Product Name**.
6. In the sidebar under "APIs & auth", select **Credentials**.
7. Click **Create a new Client ID** — a dialog box appears.
   Register the origins from which your app is allowed to access the Google APIs, as follows. An origin is a unique combination of protocol, hostname, and port.
   a. In the **Application type** section of the dialog, select **Web application**.
   b. In the **Authorized JavaScript origins** field, enter the origin for your app. You can enter multiple origins to allow for your app to run on different protocols, domains, or subdomains. Wildcards are not allowed. In the example below, the second URL could be a production URL.

   ```
   http://localhost:8080
   https://myproductionurl.example.com
   ```

   c. In the **Authorized redirect URI** field, delete the default value. It is not used for this case.
   d. Click the **Create Client ID** button.
8. In the resulting **Client ID for web application** section, copy the **Client ID** and **Client secret** that your app will need to use to access the APIs.

Step 2: Include the Google+ script on your page

Include the following script in the `<head>` section of your web page:

```
<script src="https://apis.google.com/js/client:platform.js" async defer></script>
```

Step 3: Choose your OAuth scopes

The Google+ Sign-In button allows you to specify a list of scopes that designate the data and APIs that your app will be able to access on behalf of an authorized user. Specify the scope value in your scope attribute in the next step.

- If you need access to the user's identity, social graph and the ability to write app activities, set the scope to `https://www.googleapis.com/auth/plus.login`
- If you just need access to basic profile information, set the scope to `profile`.

When specifying scopes, be aware that your app can request additional scopes anytime after your app's initial scopes have been granted. It is common to request these additional scopes at the point just prior to when your app requires them. This practice presents users with simpler and less overwhelming consent screens.

Next, you will add the sign-in button to your app.

Step 4: Add a Google+ Sign-In button to your page

The following example demonstrates adding an HTML tag that is transformed into the Google+ Sign-In button when the API loads. In your code, replace `CLIENT_ID` with the value that you created earlier:

```
<span id="signinButton">
  <span
    class="g-signin"
    data-callback="signinCallback"
    data-clientid="CLIENT_ID"
    data-cookiepolicy="single_host_origin"
    data-requestvisibleactions="http://schema.org/AddAction"
```

```
    data-scope="https://www.googleapis.com/auth/plus.login">
  </span>
</span>
```

The data-scope attribute indicates what kind of access to a user's data your app requires. The data-requestvisibleactions attribute defined the type of moments that your app is requesting to write on behalf of the user. See the button attributes for more information on customizing the button.

When the user clicks the button, they are prompted to authorize the app to access to this data. The callback function signinCallback will handle the result of the authorization. You will next implement the callback function.

When specifying scopes, be aware that your app can request additional scopes anytime after your app's initial scopes have been granted. It is common to request these additional scopes at the point just prior to when your app requires them. This practice presents users with simpler and less overwhelming consent screens.

Next, you must add the callback function to listen for the result of the sign-in flow.

Step 5: Handle the sign-in

Your callback function is a JavaScript function written by you that is triggered after the sign-in button is loaded. The function is passed an object that represents the authorization result.

When sign-in is successful, the authorization object will contain an access token in the access_token field. If the user is not signed-in you can inspect the error field to determine the cause.

Callback example:

```
function signinCallback(authResult) {
  if (authResult['status']['signed_in']) {
    // Update the app to reflect a signed in user
    // Hide the sign-in button now that the user is authorized, for example:
    document.getElementById('signinButton').setAttribute('style', 'display: none');
  } else {
    // Update the app to reflect a signed out user
    // Possible error values:
    //   "user_signed_out" - User is signed-out
    //   "access_denied" - User denied access to your app
    //   "immediate_failed" - Could not automatically log in the user
    console.log('Sign-in state: ' + authResult['error']);
  }
}
```

This example checks if the user is signed in to your website. The access token is automatically passed to the gapi.auth.setToken method, which enables you to make subsequent REST API calls.

Last updated October 24, 2014.