Google code

Authentication and Authorization for Google APIs

Using OAuth 2.0 to Access Google APIs (Experimental)

Google supports a recent draft of the OAuth 2.0 protocol with bearer tokens for authorizing access to private user data. The spec is close to settling down, and we intend to update our code to match the final OAuth 2.0 and bearer token standards. OAuth has often been described as a "valet key for the web." It lets applications ask users for access to just the data they need and no more.

The core concepts are simple:

- Your application asks for a particular **scope** of access
- Google displays an **OAuth dialog** to users, asking for consent to authorize access to your application
- If the user approves, your application will get a short-lived **access token** that you can use to validate requests for the user's data

Today Google supports three **flows** of OAuth 2.0:

- The **client-side** flow for JavaScript applications running in a browser
- The **server-side** flow for web applications with servers that can securely store persistent information
- The **native application** flow for desktop and mobile applications

The rest of this document walks through these flows.

Contents

OAuth 2.0 flows

OAuth 2.0 for client-side web applications

This flow is meant for JavaScript-based web applications that can't maintain state over time. For those who prefer to send and receive HTTP directly without using a client library, this section walks through how to use OAuth 2.0 for client-side web applications with Google.

Getting a user's permission

Every flow of OAuth 2.0 begins by sending the user to our OAuth dialog at

```
https://accounts.google.com/o/oauth2/auth
```

with the following query parameters:

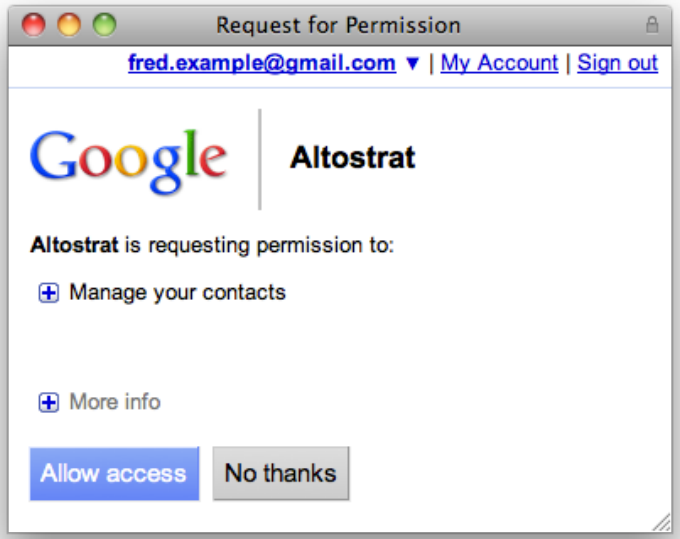
Parameter	Description
`client_id`	(required) This is how Google identifies your application—Google will give you a `client_id` when you register your app with Google.
`redirect_uri`	(required) The URL on your site that will handle OAuth responses after the user takes an action on the dialog. You'll need to register the `redirect_uri` you'd like to use in advance. See the Registering your app with Google section for details on how to register.
`scope`	(required) URL identifying the Google service to be accessed. See the documentation for the API you'd like to use for what scope to specify. To specify more than one scope, list each one separated with a space.
`response_type`	(required) Either `code` or `token`. Use `code` for the server-side flow. For the client-side flow, use `token`.
`state`	(optional) A string used to maintain state between the request and redirect. This value will

	be appended to your redirect_uri after the user takes an action on the OAuth dialog.

Here's an example URL for a hypothetical app called "Altostrat" running on www.example.com:

```
https://accounts.google.com/o/oauth2/auth?
  client_id=21302922996.apps.googleusercontent.com&
  redirect_uri=https://www.example.com/back&
  scope=https://www.google.com/m8/feeds/&
  response_type=token
```

Try out that example URL now to get a sense for how it works. Because this is just an example and no app exists at example.com, you should see a 404 error response.



After the user approves access or chooses not to, we'll redirect to the `redirect_uri` you pass us. If the user denies access, we'll append an error code:

```
https://www.example.com/back?error=access_denied
```

If the user approves access we'll append a short-lived access token in the hash fragment of the URL. Access tokens typically are valid for a bit more than an hour, but implementations shouldn't depend on that behavior. In the example above, your browser should be redirected to something like:

```
https://www.example.com/back#access_token=1/QbIbRMWW&expires_in=4301
```

JavaScript running on that page can grab that access token from the `window.location.hash` and either store it in a cookie or POST it to a server.

Using your access token

You can use your access token to call a protected API by including it in an `oauth_token` query parameter or an Authorization header. For example, you can use the curl command-line application to call the Contacts API with the access token you got from the link above:

```
curl https://www.google.com/m8/feeds/contacts/default/full?
oauth_token=1/QbIbRMWW
```

See the Calling a protected API section for the full details.

Getting additional access tokens

When your access token expires, our API endpoints will respond with HTTP 401 Unauthorized. At any time, you can send your user to the same authorization endpoint you used in the previous step. If the user has already authorized your application for the scopes you're requesting, we won't show the OAuth dialog and will immediately redirect to the `redirect_uri` you pass us with a new access token.

OAuth 2.0 for server-side web applications

This flow is meant for web applications with servers that can keep secrets and maintain state. For those who prefer to send and receive HTTP directly without using a client library, this section walks through how to use OAuth 2.0 for server-side web applications with Google.

The server-side flow has two parts. In the first part, your application asks the user for permission to access their data—if the user approves, instead of sending an access token directly as in the client-side flow, Google will send your application an **authorization code**. In the second part, your application will POST that code along with its **client secret** to Google in order to get both an access token and a long-lived **refresh token**.

Your application can get additional access tokens for that user by POSTing the refresh token back to the same place.

Getting an access token

Redirect your users to `https://accounts.google.com/o/oauth2/auth` with your `client_id`, `redirect_uri` and a list of scopes you'd like access to. For the server-side flow, also set `response_type=code`. For example:

```
https://accounts.google.com/o/oauth2/auth?
  client_id=824390819211.apps.googleusercontent.com&
  redirect_uri=https://www.example.com/back&
  scope=https://www.google.com/m8/feeds/&
  response_type=code
```

Try out an example URL now to get a sense for how it works. Because this is just an example and no app exists at example.com, you should see a 404 error response.

Just as in the client-side flow, you'll need to register your application with Google to get a `client_id` and record the redirect_uri you'd like to use. See the Registering your app with Google section for details on how to register.

After the user approves access or chooses not to, we'll redirect to the `redirect_uri` you passed us and append either an authorization code or an error message in a query parameter. For example, in the example above if a user denies access, we'll redirect to:

```
https://www.example.com/back?error=access_denied
```

And if the user approves access:

```
https://www.example.com/back?code=4/P7q7W91a-oMsCeLvIaQm6bTrgtp6
```

Your webserver should swap that authorization code for a refresh token and access token pair by POSTing it along with your `client_id`, `client_secret` and `grant_type=authorization_code` to our OAuth 2.0 token endpoint:

```
https://accounts.google.com/o/oauth2/token
```

This request might look like:

```
POST /o/oauth2/token HTTP/1.1
Host: accounts.google.com
Content-Type: application/x-www-form-urlencoded

code=4/P7q7W91a-oMsCeLvIaQm6bTrgtp6&
client_id=21302922996.apps.googleusercontent.com&
client_secret=XTHhXh1SlUNgvyWGwDk1EjXB&
redirect_uri=https://www.example.com/back&
grant_type=authorization_code
```

You can try out that call using the curl command line application, substituting the code you got from the above example URL—note that you may need to urlencode your redirect and secret, as curl won't automatically do that for you:

```
curl https://accounts.google.com/o/oauth2/token -d
"code=4/P7q7W91a-
oMsCeLvIaQm6bTrgtp6&client_id=21302922996.apps.googleusercontent.com&client_se
cret=XTHhXh1SlUNgvyWGwDk1EjXB&redirect_uri=https://www.example.com/back&grant_
type=authorization_code"
```

We'll return a refresh token and a short-lived access token in a JSON object:

```
{
  "access_token":"1/fFAGRNJru1FTz70BzhT3Zg",
  "expires_in":3920,
  "refresh_token":"1/6BMfW9j53gdGImsixUH6kU5RsR4zwI9lUVX-tqf8JXQ"
}
```

Using your access token

You can use your access token to call a protected API by including it in an `oauth_token` query parameter or an Authorization header. For example, you can use the curl command-line application to call the Contacts API with the access token you got above:

```
curl https://www.google.com/m8/feeds/contacts/default/full?
oauth_token=1/fFAGRNJru1FTz70BzhT3Zg
```

See the Calling a protected API section for the full details.

Getting additional access tokens

When your access token expires, our API endpoints will respond with HTTP 401 Unauthorized. At any time, you can use the token endpoint with your refresh token, client credentials and the parameter `grant_type=refresh_token` to get a new short-lived access token:

```
POST /o/oauth2/token HTTP/1.1
Host: accounts.google.com
Content-Type: application/x-www-form-urlencoded

client_id=21302922996.apps.googleusercontent.com&
client_secret=XTHhXh1SlUNgvyWGwDk1EjXB&
refresh_token=1/6BMfW9j53gdGImsixUH6kU5RsR4zwI9lUVX-tqf8JXQ&
grant_type=refresh_token
```

OAuth 2.0 for native applications

This flow is meant for mobile, and desktop installed applications that want access to user data. For those who prefer to send and receive HTTP directly without using a client library, this section walks through how to use OAuth 2.0 for native apps with Google.

The native app flow is identical to the server-side flow with one exception: a special redirect_uri. Instead of specifying a URL on your site, use the special string: `urn:ietf:wg:oauth:2.0:oob`. The "oob" part stands for "out of band" and the rest of the string identifies it as a part of the OAuth 2.0 standard.

When you use this redirect_uri, instead of redirecting the user's browser to a page on your site with an authorization code, Google will display the authorization code or error response in the title of the page and a text field with instructions for the user to copy and paste it in to your application. Your application can either monitor the window title of a web-view or browser window it launches and close it once the authorization code appears or prompt the user to copy and paste the code.

Getting an access token

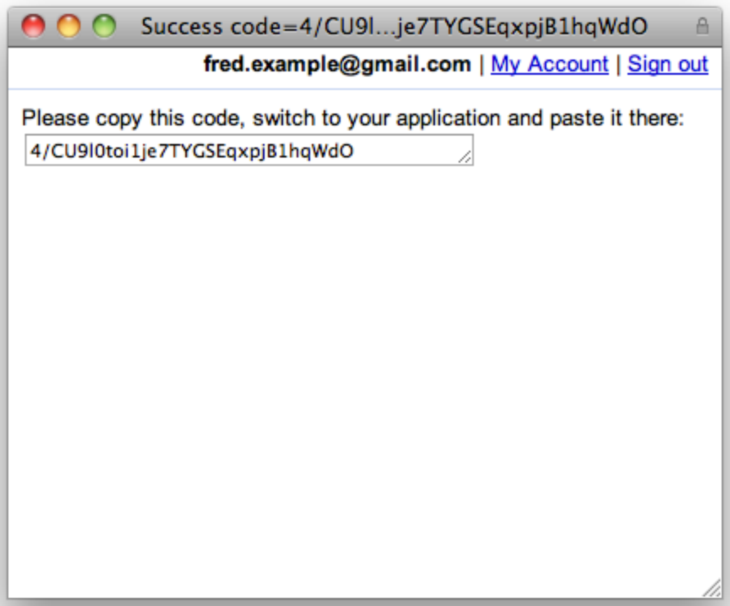
Open a browser or a webview to the OAuth dialog at `https://accounts.google.com/o/oauth2/auth` with your `client_id`, `redirect_uri=urn:ietf:wg:oauth:2.0:oob` and a list of scopes you'd like access to. For the native app flow, also set `response_type=code`. For example:

```
https://accounts.google.com/o/oauth2/auth?
  client_id=21302922996.apps.googleusercontent.com&
  redirect_uri=urn:ietf:wg:oauth:2.0:oob&
  scope=https://www.google.com/m8/feeds/&
  response_type=code
```

Just as in the client-side flow, you'll need to register your application with Google to get a client_id and record the special redirect_uri: `urn:ietf:wg:oauth:2.0:oob`. See the Registering your app with Google section for details on how to register.

Try out an example URL now to get a sense for how it works. To see the special native app page below, you'll need to approve access—don't worry, nobody will actually get access to your contacts if you click Allow.

In the native app flow, after the user approves access, we'll display the authorization code in the title of the page and in a text input with instructions for the user to copy and paste the code to your application


Success code=4/CU9l...je7TYGSEqxpjB1hqWdO
fred.example@gmail.com | My Account | Sign out

Please copy this code, switch to your application and paste it there:

4/CU9l0toi1je7TYGSEqxpjB1hqWdO

On many platforms, your application should be able to monitor the window title of a browser window it creates and close the window when it sees a valid response. If your platform doesn't support that, you can instruct users to copy and paste the code to your application.

Your native app should swap that authorization code for a refresh token and access token pair by POSTing it along with your `client_id`, `client_secret` and `grant_type=authorization_code` to our OAuth 2.0 token endpoint:

```
https://accounts.google.com/o/oauth2/token
```

This request might look like:

```
POST /o/oauth2/token HTTP/1.1
Host: accounts.google.com
Content-Type: application/x-www-form-urlencoded

client_id=21302922996.apps.googleusercontent.com&
client_secret=XTHhXh1SlUNgvyWGwDk1EjXB&
code=4/P7q7W91a-oMsCeLvIaQm6bTrgtp6&
redirect_uri=urn:ietf:wg:oauth:2.0:oob&
grant_type=authorization_code
```

We'll return a refresh token and a short-lived access token in a JSON object:

```
{
  "access_token":"1/fFAGRNJru1FTz70BzhT3Zg",
  "expires_in":3920,
  "refresh_token":"1/6BMfW9j53gdGImsixUH6kU5RsR4zwI9lUVX-tqf8JXQ"
}
```

Using your access token

You can use your access token to call a protected API by including it in an `oauth_token` query parameter or an Authorization header. For example, you can use the curl command-line application to call the Contacts API with the access token you got above:

```
curl https://www.google.com/m8/feeds/contacts/default/full?
oauth_token=1/fFAGRNJru1FTz70BzhT3Zg
```

See the Calling a protected API section for the full details.

Getting additional access tokens

When your access token expires, our API endpoints will respond with HTTP 401 Unauthorized. At any point, you can use the token endpoint with your refresh token, client credentials and the parameter `grant_type=refresh_token` to get a new short-lived access token:

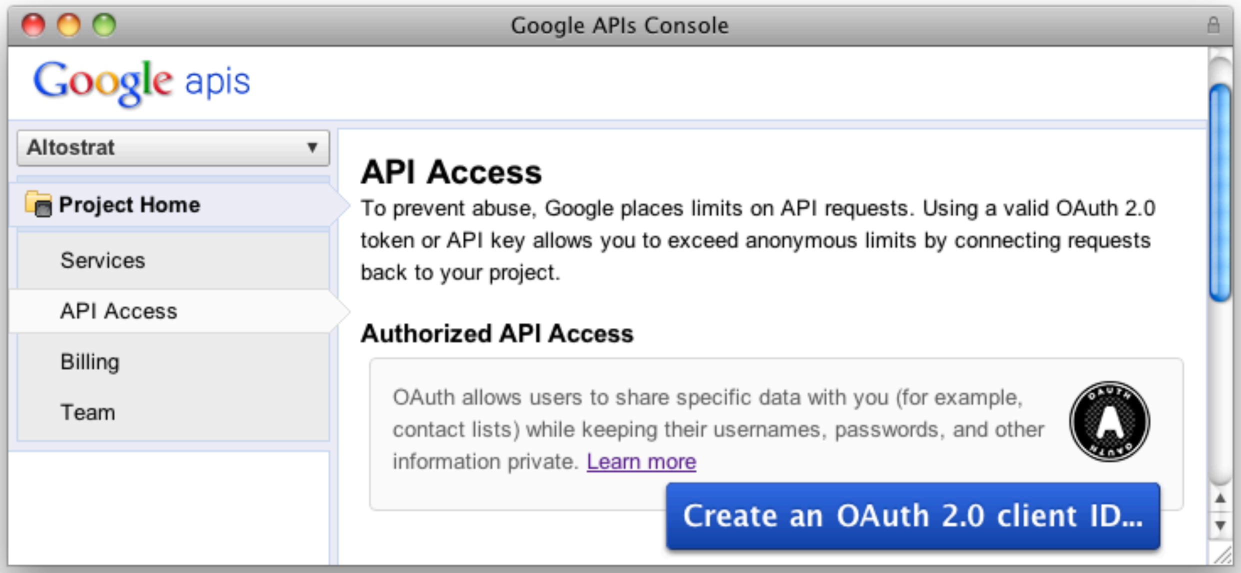
```
POST /o/oauth2/token HTTP/1.1
Host: accounts.google.com
Content-Type: application/x-www-form-urlencoded

client_id=21302922996.apps.googleusercontent.com&
client_secret=XTHhXh1SlUNgvyWGwDk1EjXB&
refresh_token=1/6BMfW9j53gdGImsixUH6kU5RsR4zwI9lUVX-tqf8JXQ&
grant_type=refresh_token
```

Registering your app with Google

Application registration in OAuth 2.0 is different from the registration process for earlier protocols. For OAuth 2.0, all applications need to register their name and callback URL (the `redirect_uri` parameter) with Google in the API Console, but are not required to verify domain ownership.

To register your application, create a project in the APIs Console, then click the "Create an OAuth 2.0 client ID" button in the API Access tab.



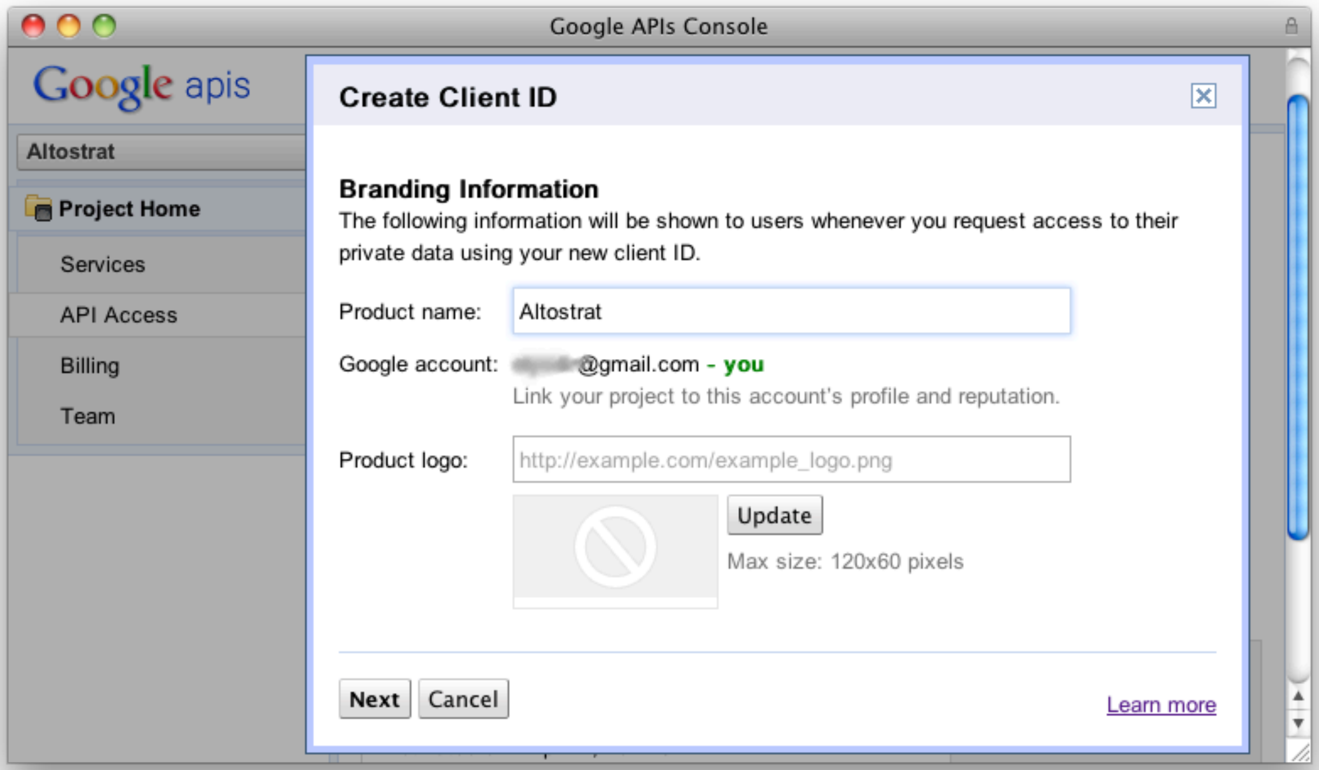
API Access

To prevent abuse, Google places limits on API requests. Using a valid OAuth 2.0 token or API key allows you to exceed anonymous limits by connecting requests back to your project.

Authorized API Access

OAuth allows users to share specific data with you (for example, contact lists) while keeping their usernames, passwords, and other information private. Learn more

We'll ask you to give an application name ("Product name") to display to users in the OAuth dialog. In addition, we'll use your email address as the "support email" for your application. You can change that email address by signing in to your APIs Console project with a different account whose email address you've verified. To learn more about verifying your Google account, see the help center.



Branding Information

The following information will be shown to users whenever you request access to their private data using your new client ID.

Product name: Altostrat

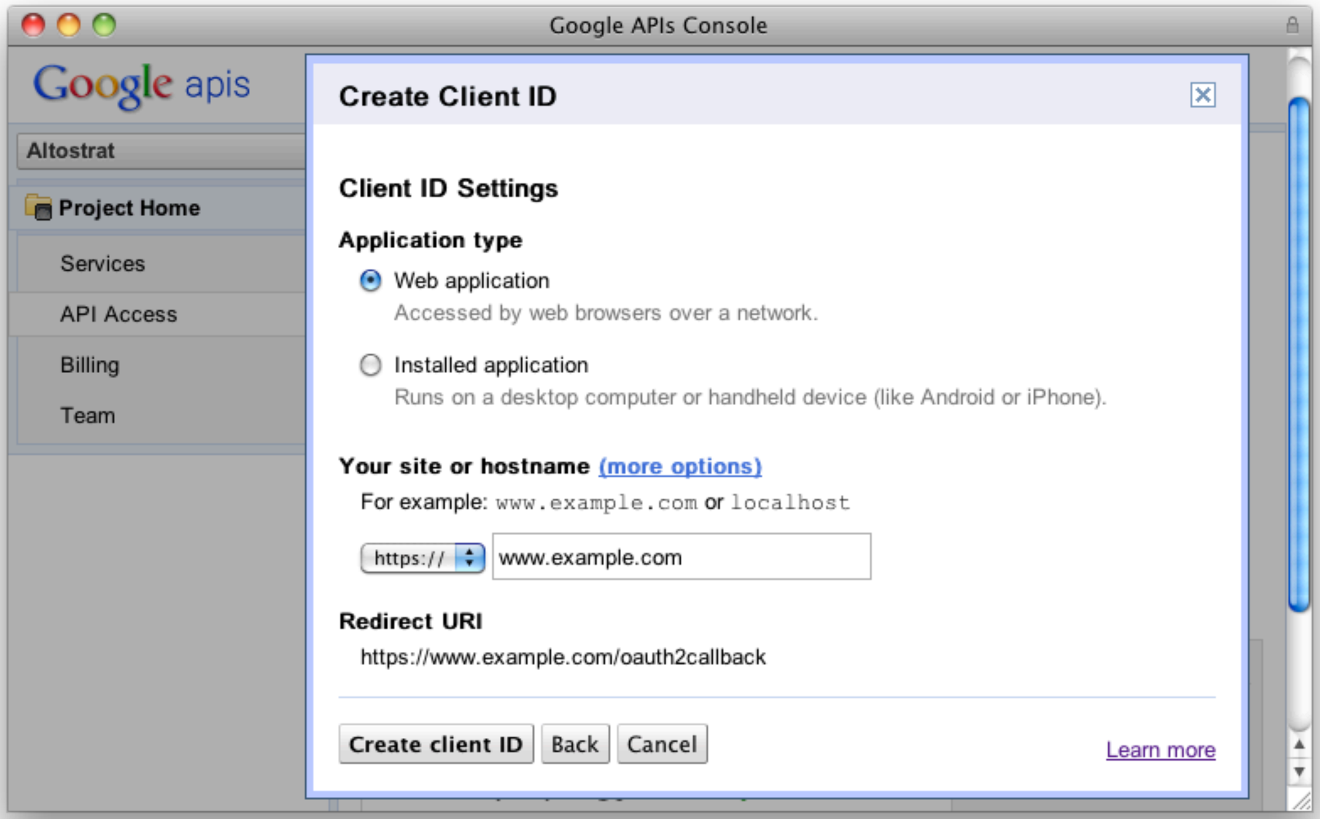
Google account: [redacted]@gmail.com - you

Link your project to this account's profile and reputation.

Product logo: http://example.com/example_logo.png

Max size: 120x60 pixels

Next, indicate whether your application is a web application or an installed application. If it's a web application, provide a domain or hostname.



Client ID Settings

Application type

[✓ Web application]
Accessed by web browsers over a network.

Installed application
Runs on a desktop computer or handheld device (like Android or iPhone).

Your site or hostname

For example: www.example.com or localhost

https:// www.example.com

Redirect URI

https://www.example.com/oauth2callback

Note that the default `redirect_uri` is your host name followed by `oauth2callback`. To change that, or to add an authorized JavaScript origin, click the "More options" link. Registering the exact `redirect_uri` you'll be using helps us be sure that we're actually passing tokens to your application and not an attacker.

Calling a protected API

After your application has obtained an access token, your app can use it to access APIs by including it in either an **oauth_token** query parameter or an **Authorization: OAuth** header. Because we use bearer tokens in our implementation of OAuth 2.0, no additional credentials are required.

For example, a call to the contacts API using the oauth_token query parameter might look like the following:

```
GET https://www.google.com/m8/feeds/contacts/default/full?
oauth_token=YOUR_ACCESS_TOKEN
```

and using the Authorization: OAuth header:

```
GET /m8/feeds/contacts/default/full HTTP/1.1
Authorization: OAuth YOUR_ACCESS_TOKEN
Host: accounts.google.com
```

You can try either out with curl, a command-line application for HTTP. Here's an example of the query parameter:

```
curl https://www.google.com/m8/feeds/contacts/default/full?
oauth_token=YOUR_ACCESS_TOKEN
```

And with the Authorization: OAuth header:

```
curl -H "Authorization: OAuth YOUR_ACCESS_TOKEN"
https://www.google.com/m8/feeds/contacts/default/full
```

If your token has expired, you'll get an HTTP 401 Unauthorized response. See the documentation on each flow for how to get additional access tokens.

Client libraries

The following client libraries make implementing OAuth 2.0 even simpler by integrating with popular frameworks:

- Google APIs Client Library for Python
- Google APIs Client Library for .NET
- Google APIs Client Library for Ruby
- Google APIs Client Library for PHP
- OAuth 2.0 Library for Google Web Toolkit
- Google Toolbox for Mac OAuth 2.0 Controllers