Migrating from OpenID 2.0 to OpenID Connect

Contents

Important: OpenID 2.0 is no longer supported. If your app uses OpenID 2.0, you must migrate your app.

Note: If you are looking for documentation for web applications that use the deprecated OpenID 2.0 for Google login, see OpenID 2.0 (Deprecated). For information about using the deprecated OpenID 2.0 authentication with Google Apps (hosted) accounts, see OpenID API for Google Apps accounts (Deprecated).

Note: To get help on Stack Overflow with migrating your OpenID 2.0 app, tag your questions with 'google-openid'.

This document provides details about migrating your code from OpenID 2.0 to OpenID Connect. We recommend you use Google Sign-in, which is our client library that provides OpenID Connect sign-in for Google Accounts. You can also use our OpenID Connect endpoints directly. This document provides information about both migration strategies.

Migrating to Google Sign-In

If you provide a "sign-in with Google" feature, we recommend using Google Sign-In. Google Sign-In provides OAuth 2.0 (OpenID Connect) authentication with access to additional Google desktop and mobile features. Google Sign-In supports transparent migration and offers widgets and client libraries that make it easy to implement. It supports all users who have a Google account, whether or not they've upgraded to Google+.

If you currently have OpenID 2.0–based software in production, or if you have users who are known only by an OpenID 2.0 identifier, you can switch to Google Sign-In without losing track of your existing users by linking the OpenID 2.0 account identifiers to new Google Sign-In identifiers, then using the new identifiers going forward. For details, see the step-by-step instructions forswitching from OpenID 2.0 to Google Sign-In.

If you switch to Google Sign-In, here are some tips:

- If users have provided email addresses to your app via OpenID 2.0 and you do not want these users to be asked to re-consent when you switch to Google Sign-In, use only the `email` scope.
- To configure Google Sign-In to return profile information in OpenID Connect format, use the `openid` scope and get the user profile by calling the `people.getOpenIdConnect` endpoint.

If your OpenID 2.0–based app is on a platform not supported by Google Sign-In, or if you want to work directly with the OAuth 2.0 REST APIs, then we recommend migrating to Google's OpenID Connect solution, as described below.

Migrating to OpenID Connect

The basic techniques for logging in with OpenID Connect are described inUsing OAuth 2.0 for Login (OpenID Connect). You can introduce OpenID Connect–based authentication into your OpenID 2.0 authentication process without losing track of existing users' accounts by following these steps:

- Step 1: Adjust the authentication request URI that you send to Google by adding `openid.realm` and other parameters.
- Step 2: Update your sign-in code to use an OpenID Connect authentication process.
- Step 3: Map existing OpenID 2.0 identifiers to new OpenID Connect identifiers and add the new identifiers to your user database.

Note: Support for the OpenID 2.0 identifier mapping described above will remain in effect until January 1, 2017.

Step 1: Adjust the authentication request URI that you send to Google

When you construct an authentication request to send to Google, it is in the form of a URI that begins with Google's OAuth 2.0 endpoint, `https://accounts.google.com/o/oauth2/v2/auth`. You construct the rest of the URI by adding parameters as needed. For applications that use OpenID 2.0, the authentication request URI may include an `openid.realm` parameter, and can also include the `login_hint` and other parameters.

If you are migrating your code from OpenID 2.0, set up your OpenID Connect authentication requests as follows:

1. If you have not yet done so, obtain OAuth 2.0 credentials for your project from the Google Developers Console.
2. In the Developers Console, find your application's `redirect_uri` and change it (if necessary) so that it matches your OpenID 2.0 `openid.realm` value, according to the matching rules in section 9.2 of the OpenID 2.0 spec.

   To find the redirect URIs for your OAuth 2.0 credentials, do the following:

   a. Open the Credentials page.

   b. If you haven't done so already, create your OAuth 2.0 credentials by clicking **Add credentials > OAuth 2.0 client ID**.

   c. After you create your credentials, view or edit the redirect URLs by clicking the client ID in the **OAuth 2.0 client IDs** section.
3. When you construct your OpenID Connect authentication URI, include an extra argument, the `openid.realm` parameter. Use the same value that you used for the `openid.realm` parameter in your OpenID 2.0 requests.
4. You must also include a scope parameter that requests access to the user ID. We recommend using the `email` scope, which allows your app to get basic profile information from users who have upgraded to Google+. In some situations when you use the `email` scope, users are not required to re-consent. The user is not presented with a consent screen if all the following are true:

   - The user has provided an email address to your application via OpenID 2.0, and
   - You request only the `email` scope, and
   - The access type for your request is `online`, which is the default. (Note that when OpenID access is granted, it is always for online access, so a refresh token cannot be returned from these requests. For more about `access_type`, see Authentication URI parameters.)

   If these three conditions are met, the user's account information is migrated to the `email` scope.

   Instead of the `email` scope, you can specify `profile` or `plus.login`. The `profile` scope allows your app to get some profile information for users who have not yet upgraded to Google+, but because it expands your app's access to user data, this scope will require users to re-consent. The `plus.login` scope gives you access to write to the user's history and see the people in their circles.
5. If you know the user's email address, include it in the authentication URI as the value of the `login_hint` parameter. If you do not include a `login_hint` and the user is signed into Google with multiple accounts, they will see an "account chooser" asking them to select one account. This might be surprising to them, and they might select an account other than the one your application is trying to authorize, which could increase the complexity of your task.

Step 2: Update your sign-in code

Update your sign-in code so that it launches an OpenID Connect authentication process instead of an OpenID 2.0 authentication process:

- Use the `people.getOpenIdConnect` endpoint, either by using the following HTTP request path, or by using the corresponding client library call:

  `https://www.googleapis.com/plus/v1/people/me/openIdConnect`
- No change is needed to the OpenID Connect scopes `openid`, `profile`, and `email`.

The response includes basic user data in the OpenID Connect format.

Step 3: Map OpenID 2.0 identifiers to OpenID Connect identifiers

When you send an OpenID Connect authentication request URI to Google as described in Step 1, you include an `openid.realm` parameter. The response that is sent to your `redirect_uri` includes an authorization code that your application can use to retrieve an access token and an ID token. (You can also retrieve an ID token directly from the OpenID Connect authentication request by adding `id_token` to the `response_type` parameter, potentially saving a back-end call to the token endpoint.)

The response from that token request includes the usual fields (`access_token` , etc.), plus an `openid_id` field and the standard OpenID Connect `sub` field. The fields you need in this context are `openid_id` and `sub` :

- The **`openid_id`** field holds the OpenID 2.0 identifier. OpenID 2.0 identifiers are strings that begin with `https://www.google.com/accounts/o8/id?id=` .
- The **`sub`** field holds the OpenID Connect identifier (which corresponds to the `id` field of the Google+ `people.get` response). OpenID Connect identifiers are long numeric strings such as `10167301128815079 46` .

To migrate your sign-in infrastructure from OpenID 2.0 to OpenID Connect, add code to your application that does the following for each response that you receive from a token request:

1. Extract the `sub` field from the token-request response. This is the OpenID Connect identifier.
2. In your user database, link the `openid_id` to the new `sub` ID.
3. Use the new IDs going forward.

Tip: If you overwrite a user's OpenID 2.0 identifier (**`openid_id`**) and need to find it again, you can repeat the process of including an**`openid.realm`** parameter in an authentication request and mapping the returned **`openid_id`** to the new **`sub`** ID.

Migrating a hybrid login solution

If your app uses a hybrid OpenID 2.0 / OAuth 1.0 login solution, you must migrate your app by the deadlines given in the migration timetable. We recommend that you switch to Google Sign-In. For step-by-step instructions, see Migrate from OpenID 2.0 or OpenID+OAuth hybrid to Google Sign-In.

If for some reason you need to work directly with the OpenID Connect OAuth 2.0 protocols, it is possible to migrate the two components of your app separately:

- To migrate from OpenID 2.0 to OpenID Connect, see Migrating to OpenID Connect (OAuth 2.0 for login).
- To migrate from OAuth 1.0 to OAuth 2.0, see Migrating from OAuth 1.0 to OAuth 2.0.

The general information in Implementing OAuth with federated login (hybrid protocol) still applies.

Migrating how your app gets email addresses

If you use the deprecated `userinfo` endpoint to get user email addresses, you must migrate how your app gets email addresses.

Last updated December 3, 2015.