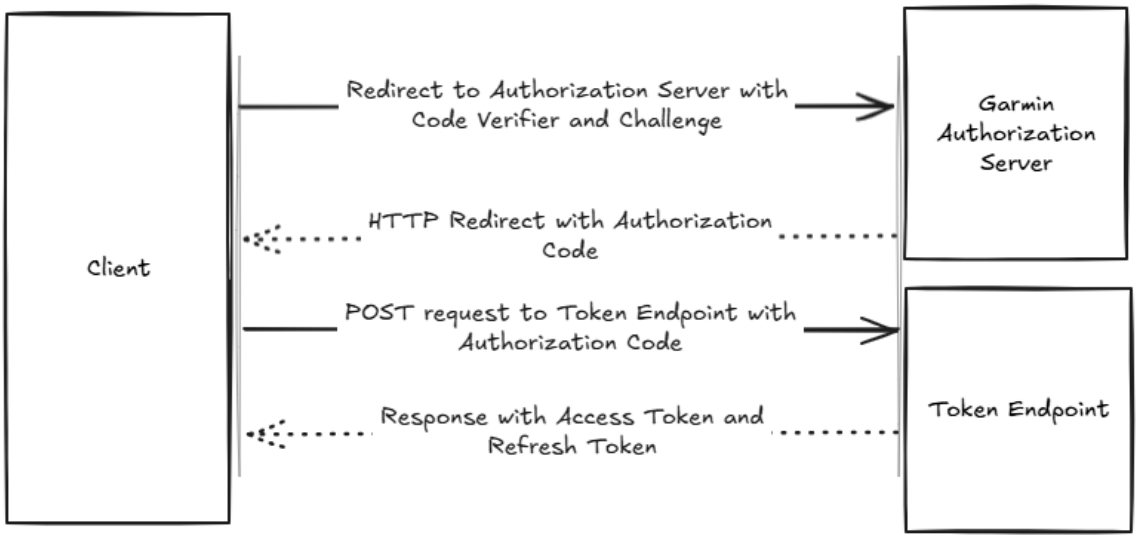
Garmin Connect Developer Program
OAuth2.0 PKCE Specification

Garmin Connect Developer Program uses the OAuth2.0 PKCE version. PKCE, which stands for “Proof of Key Code Exchange,” is an extension of the OAuth 2.0 protocol that helps prevent code interception attacks. OAuth 2.0 allows users to share their data securely between different applications, and PKCE provides an additional security layer.
The client (third-party application) initiates the OAuth flow by prompting the user to log in to the Garmin Connect account and consent to the third-party app. Users can control what permissions are granted.
After the user agrees or declines to share data, they will be redirected back to the third-party application. If they decide to share data, the client will receive an authorization code that can be exchanged for the access token. After receiving the token, the application can fetch the User ID and permissions granted.

Garmin Connect Developer Oauth2 tool example flow (https://apis.garmin.com/tools/oauth2/authorizeUser)



Contents

STEP 1: Authorization Request

The client generates a code verifier and challenge before redirecting the user to the authorization server. The code verifier is a cryptographically random string with characters A-Z, a-z, 0-9, and punctuation characters (hyphen, period, underscore, and tilde) between 43 and 128 characters long. Once the client has generated the code verifier, a code challenge is created (SHA-256 hashed version of code_verifier - base64url(sha256(code_verifier))).
CORS pre-flight requests (OPTIONS) are not supported.

URL(GET): https://connect.garmin.com/oauth2Confirm

Parameters:
response_type=code, *required*
client_id=<client id (consumer key)>, *required*
code_challenge=<SHA-256 hashed version of code_verifier>, *required*
code_challenge_method=S256, *required*
redirect_uri=<uri to redirect the user to during the authorization request>, *optional*
state=<unique string to tie to the authorization code>, *optional; it can be used to ensure that the redirect back to your site or app wasn't spoofed.*

Request Example:

```
curl --request GET --url 'https://connect.garmin.com/oauth2Confirm?
client_id=600a2456-ab5f-4b1f-8eb3-8cc8dc86264d
&response_type=code
&state=TUUVBcUVHd2Q0V0FWZUZqWEs5aHl6IdkIrOWFMMkJVRUgzZGN3UnoxSlBQcHZxeXhZRWg%3D
&redirect_uri=https://localhost.com/redirect
&code_challenge=UrJZ-JcnGMxHS8Fnmxf1TbTP22-RymoMZTsa6H1D5ZU
&code_challenge_method=S256'/
```

Response after user confirmed to share data:

<Redirect URL>?code=<code>&state=<state>

STEP 2: Access Token Request

URL (POST) https://diauth.garmin.com/di-oauth2-service/oauth/token

Parameters:
grant_type=authorization_code, *required*
client_id=<client id (consumer key)>, *required*
client_secret=<client secret (consumer secret)>, *required*
code=<the authorization code returned in Step 1>, *required*
code_verifier=<the code verifier generated by the client that was hashed to create the code challenge used in Step 1>, *required*
redirect_uri=<must match the value provided in Step 1 if applicable>, *not required if was not used in step 1*

Request Example:

```
curl --request POST https://diauth.garmin.com/di-oauth2-service/oauth/token? \
  --header 'Content-Type: application/x-www-form-urlencoded' \
  --data grant_type=authorization_code \
  --data redirect_uri=https://localhost.com/redirect \
  --data code=d6a8b166c9b74e6fade0e081e26f2212 \
  --data code_verifier=BCTmANU0ltGBqu7QgiFJBMNhGYOk_L_dS0qTt1-uGftiGuE1 \
  --data client_id=600a2456-ab5f-4b1f-8eb3-8cc8dc86264d \
  --data client_secret=7inlEErpIzyRmXViU33gxgzzhQOTXxSQ0uyYyFH6uxk
```

Response:

```
{
"access_token": "VTkc5JilK0dd8w_s0FJGabMqSFyjSXyNHIb0lUgFJyIr2YZxhey-KMzDzBCI2LJc6yC5NGbC",
"expires_in": 86400,
"token_type": "bearer",
"refresh_token": "xUEA805jTCcGd4b7rs-SkOJP=="
"scope": "PARTNER_WRITE PARTNER_READ CONNECT_READ CONNECT_WRITE",
“jti”: "f9eb2316-9b9d-495a-8732-e16c4b5bcafd",
“refresh_token_expires_in": 7775998
 }
```

* Scope is a default API scope and cannot be modified. API sections are managed during the app creation process. Users control permissions granted during the authorization step.

* We recommend subtracting 600 seconds or more from the expiration value to account for any network issues and ensure the token can be refreshed in time.

Refresh Token

Access tokens expire three months after they are created, so they must be refreshed for an application to maintain access to a user's resources. We return a new refresh token every time you get a new access token. If you need to request a new access token, we recommend checking to see if the short-lived access tokens have expired. If they have expired, request a new short-lived access token with the last refresh token received.

URL (POST) https://diauth.garmin.com/di-oauth2-service/oauth/token

```
curl --request POST \

  --url https://diauth.garmin.com/di-oauth2-service/oauth/token?=' \
  --header 'Content-Type: application/x-www-form-urlencoded' \
  --data grant_type=refresh_token \
 --data client_id=600a2456-ab5f-4b1f-8eb3-8cc8dc86264d \
--data client_secret=7inlEErpIzyRmXViU33gxgzzhQOTXxSQ0uyYyFH6uxk
  --data refresh_token=eyJyZWZyZXNoV…kzLTRmMzEtYmM3NC1lYWU0NjczMzhiZGIifQ== \
```

Response:

```
{
"access_token": "VTkc5JilK0dd8w_s0FJGabMqSFyjSX….lUgFJyIr2YZxhey-KMzDzBCI2LJc6yC5NGbC",
"token_type": "bearer",
"refresh_token": "xUEA805jTCcGd4b7rs-SkOJP",
"expires_in": 86400,
"scope": "PARTNER_WRITE PARTNER_READ CONNECT_READ CONNECT_WRITE",
“jti”: "f9eb2316-9b9d-495a-8732-e16c4b5bcafd",
“refresh_token_expires_in": 7775998
  }
```

Accessing the API Using an Access Token

Applications must use unexpired access tokens to make requests and can be included by specifying the Authorization: Bearer {access_token} header.

```
curl --request GET \
--url https://apis.garmin.com/wellness-api/rest/user/permissions \
--header 'Authorization: Bearer eyJhbGciOiJS.............tpZCI6ImRpLW9hdXRoLXNpZ25lci1' \
```

User ID

Each Garmin Connect user has a unique API ID associated with them that will persist across multiple UATs/consumers. For instance, if a user deletes their association through Garmin Connect and then completes the OAuth process to generate a new User Access Token with the same Garmin Connect account, the second token will still have the same API User ID as the first token. Similarly, if a partner manages multiple programs and the user signs up for each of them, the API User ID returned for each UAT will match.
PUSH and PING notifications will contain the User ID so the partner can match data to a user on their platform.

Request URL to fetch API User ID
GET https://apis.garmin.com/wellness-api/rest/user/id

No parameters are required for this request.

Response: `{"userId": "d3315b1072421d0dd7c8f6b8e1de4df8"}`

Delete User Registration

This endpoint **must** be called if the partner website or application provides a "Delete My Account" or "Disconnect" mechanism outside of the normal Garmin Connect consent removal process at the account information or in any other case where the user would reasonably believe the partner program is allowing them to remove their consent to share Garmin data.

Request URL to delete a user registration

DELETE: https://apis.garmin.com/wellness-api/rest/user/registration

User Permission endpoint

Consumers can set up multiple permissions, such as "Activity Export" and "Workout Import." However, while signing up, the user may only opt in for fewer permissions, so this endpoint helps fetch the permissions for that particular user after the user has consented. Post consent changes will be notified via the User_Permission webhook.

Method & URL: GET https://apis.garmin.com/wellness-api/rest/user/permissions

Response body: The user permissions that were retrieved in JSON.

Example response:

```
{["ACTIVITY_EXPORT",
    "WORKOUT_IMPORT",
    "HEALTH_EXPORT",
    "COURSE_IMPORT",
    "MCT_EXPORT"]}
```