# Call Your API Using the Hybrid Flow

This tutorial will help you call your own API using the Hybrid Flow. If you want to learn how the flow works and why you should use it, see Hybrid Flow.

Auth0 makes it easy for your app to implement the Authorization Code Flow using:

- Authentication API: If you prefer to roll your own, keep reading to learn how to call our API directly.

## Prerequisites

**Before beginning this tutorial:**

- Register your Application with Auth0.
    - Select the appropriate **Application Type**.
    - Add an **Allowed Callback URL** of `https://YOUR_APP/callback` .
    - Make sure your Application's Grant Types include **Implicit** and **Authorization Code**.
    - If you want your Application to be able to use Refresh Tokens, make sure the Application's Grant Types include **Refresh Token**.
- Register your API with Auth0
    - If you want your API to receive Refresh Tokens to allow it to obtain new tokens when the previous ones expire, enable **Allow Offline Access**.

## Steps

1. Authorize user: Request the user's authorization and redirect back to your app with an authorization code.
2. Request tokens: Exchange your authorization code for tokens.
3. Call API: Use the retrieved Access Token to call your API.
4. Refresh tokens: Use a Refresh Token to request new tokens when the existing ones expire.

Optional: Explore sample use cases

## Authorize user

To begin the flow, you'll need to get the user's authorization. This step may include one or more of the following processes:

* Authenticating the user; * Redirecting the user to an Identity Provider to handle authentication; * Checking for active Single Sign-on (SSO) sessions; * Obtaining user consent for the requested permission level, unless consent has been previously given.

To authorize the user, your app must send the user to the authorization URL.

### Example authorization URL

```
https://YOUR_DOMAIN/authorize?
    response_type=YOUR_RESPONSE_TYPE&
    response_mode=form_post&
    client_id=YOUR_CLIENT_ID&
    redirect_uri=https://YOUR_APP/callback&
    scope=SCOPE&
    audience=API_AUDIENCE&
    state=STATE&
    nonce=NONCE
```

Was this helpful? Yes / No

#### Parameters

Note that for authorizing a user when calling a custom API, you:

- must include an audience parameter - can include additional scopes supported by the target API

| Parameter Name | Description |
|---|---|
| `response_type` | Denotes the kind of credential that Auth0 will return (code or token). For this flow, the value must include `code`, but may also include `id_token`, `token`, or `id_token token`. Specifically, `id_token` returns an ID Token, and `token` returns an Access Token. |
| `response_mode` | Specifies the method with which response parameters should be returned. For security purposes, the value should be `form_post`. In this mode, response parameters will be encoded as HTML form values that are transmitted via the HTTP POST method and encoded in the body using the `application/x-www-form-urlencoded` format. |
| `client_id` | Your application's Client ID. You can find this value in your Application Settings. |
| `redirect_uri` | The URL to which Auth0 will redirect the browser after authorization has been granted by the user. The Authorization Code will be available in the `code` URL parameter. You must specify this URL as a valid callback URL in your Application Settings.<br><br>**Warning:** Per the OAuth 2.0 Specification, Auth0 removes everything after the hash and does *not* honor any fragments. |
| `scope` | Specifies the scopes for which you want to request authorization, which dictate which claims (or user attributes) you want returned. These must be separated by a space. You can request any of the standard OpenID Connect (OIDC) scopes about users, such as `profile` or `email`, custom claims conforming to a namespaced format, or any scopes supported by the target API (e.g., `read:contacts`). Include `offline_access` to get a Refresh Token (make sure that the **Allow Offline Access** field is enabled in the Application Settings). |
| `audience` | The unique identifier of the API your application wants to access. Use the **Identifier** value on the Settings tab for the API you created as part of the prerequisites for this tutorial. |

| Parameter Name | Description |
|---|---|
| `state` | (recommended) An opaque arbitrary alphanumeric string your app adds to the initial request that Auth0 includes when redirecting back to your application. To see how to use this value to prevent cross-site request forgery (CSRF) attacks, see Mitigate CSRF Attacks With State Parameters. |
| `nonce` | A cryptographically random string that your app adds to the initial request and Auth0 includes inside the ID Token, used to prevent token replay attacks. |

As an example, your HTML snippet for your authorization URL when adding login to your app might look like:

```
<a href="https://YOUR_DOMAIN/authorize?
  response_type=code id_token token&
  client_id=YOUR_CLIENT_ID&
  redirect_uri=https://YOUR_APP/callback&
  scope=appointments%20contacts&
  audience=appointments:api&
  state=xyzABC123&
  nonce=eq...hPmz">
  Sign In
</a>
```

Was this helpful? Yes / No

## Response

If all goes well, you'll receive an `HTTP 302` response. The requested credentials are encoded in the body:

```
HTTP/1.1 302 Found
Content-Type: application/x-www-form-urlencoded
code=AUTHORIZATION_CODE&
access_token=ey...MhPw
&expires_in=7200
&token_type=Bearer
id_token=eyJ...acA&
state=xyzABC123
```

Was this helpful? Yes / No

Note that the returned values depend on what you requested as a `response_type`.

| Response Type | Components |
|---|---|
| code | Authorization code |
| id_token | ID Token |
| token | Access Token (plus `expires_in` and `token_type` values) |
| id_token token | ID Token, Access Token (plus `expires_in` and `token_type` values) |

Auth0 will also return any state value you included in your call to the authorization URL.

⚠ The Access Token that you receive in this transaction is only the first Access Token that you will receive. We do not recommend that it be used to call APIs.

⚠ You should validate your tokens before saving them. To learn how, see Validate ID Tokens and Validate Access Tokens.

When you decode and parse your ID token, you will notice an additional claim, `c_hash`, which contains a hash of the `code`. This claim is mandatory when an ID token is issued at the same time as a `code`, and you should validate it:

1. Using the hash algorithm specified in the `alg` claim in the ID Token header, hash the octets of the ASCII representation of the `code`.
2. Base64url-encode the left-most half of the hash.
3. Check that the result matches the `c_hash` value.

## Request tokens

Now that you have an Authorization Code, you must exchange it for tokens. Using the extracted Authorization Code ( `code` ) from the previous step, you will need to `POST` to the token URL.

The Access Token you receive in this step is the one you should use to call your API. Make sure you keep it separate from the Access Token you received in the previous step of this tutorial.

### Example POST to token URL

cURL C# Go Java Node.JS Obj-C ...

```
curl --request POST \
  --url 'https://YOUR_DOMAIN/oauth/token' \
  --header 'content-type: application/x-www-form-urlencoded' \
  --data grant_type=authorization_code \
  --data 'client_id=YOUR_CLIENT_ID' \
  --data client_secret=YOUR_CLIENT_SECRET \
  --data code=YOUR_AUTHORIZATION_CODE \
  --data 'redirect_uri=https://YOUR_APP/callback'
```

Was this helpful? Yes / No

Parameters

| Parameter Name | Description |
|---|---|
| `grant_type` | Set this to `authorization_code` . |
| `code` | The `authorization_code` retrieved in the previous step of this tutorial. |
| `client_id` | Your application's Client ID. You can find this value in your Application Settings. |
| `client_secret` | Your application's Client Secret. You can find this value in your Application Settings. |
| `redirect_uri` | The valid callback URL set in your Application settings. This must exactly match the `redirect_uri` passed to the authorization URL in the previous step of this tutorial. Note that this must be URL encoded. |

## Response

If all goes well, you'll receive an `HTTP 200` response with a payload containing `access_token`, `refresh_token`, `id_token`, and `token_type` values:

```
{
  "access_token": "eyJz93a...k4laUWw",
  "refresh_token": "GEbRxBN...edjnXbL",
  "id_token": "eyJ0XAi...4faeEoQ",
  "token_type": "Bearer"
}
```

Was this helpful? Yes / No

⚠ You should validate your tokens before saving them. To learn how, see Validate ID Tokens and Validate Access Tokens.

ID Tokens contain user information that must be decoded and extracted.

Access Tokens are used to call the Auth0 Authentication API's /userinfo endpoint or another API. If you are calling your own API, the first thing your API will need to do is verify the Access Token.

Refresh Tokens are used to obtain a new Access Token or ID Token after the previous one has expired. The `refresh_token` will only be present in the response if you included the `offline_access` scope and enabled **Allow Offline Access** for your API in the Dashboard.

⚠ Refresh Tokens must be stored securely since they allow a user to remain authenticated essentially forever.

## Call API

To call your API from a regular web application (or similar cases in which the Client Secret can be safely stored), the application must pass the retrieved Access Token as a Bearer token in the Authorization header of your HTTP request.

cURL C# Go Java Node.JS Obj-C ▪▪▪

```
curl --request GET \
  --url https://myapi.com/api \
  --header 'authorization: Bearer ACCESS_TOKEN' \
  --header 'content-type: application/json'
```

Was this helpful? Yes / No

## Refresh tokens

You have already received a Refresh Token if you've been following this tutorial and completed the following:

- configured your API to allow offline access
- included the `offline_access` scope when you initiated the authentication request through the authorize endpoint.

You can use the Refresh Token to get a new Access Token. Usually, a user will need a new Access Token only after the previous one expires or when gaining access to a new resource for the first time. It's bad practice to call the endpoint to get a new Access Token every time you call an API, and Auth0 maintains rate limits that will throttle the amount of requests to the endpoint that can be executed using the same token from the same IP.

To refresh your token, make a `POST` request to the `/oauth/token` endpoint in the Authentication API, using `grant_type=refresh_token`.

## Example POST to token URL

cURL C# Go Java Node.JS Obj-C ■■■

```
curl --request POST \
  --url 'https://YOUR_DOMAIN/oauth/token' \
  --header 'content-type: application/x-www-form-urlencoded' \
  --data grant_type=refresh_token \
  --data 'client_id=YOUR_CLIENT_ID' \
  --data refresh_token=YOUR_REFRESH_TOKEN
```

Was this helpful? Yes / No

### Parameters

| Parameter Name | Description |
|---|---|
| `grant_type` | Set this to `refresh_token`. |
| `client_id` | Your application's Client ID. You can find this value in your Application Settings. |
| `refresh_token` | The Refresh Token to use. |

| Parameter Name | Description |
|---|---|
| `scope` | (optional) A space-delimited list of requested scope permissions. If not sent, the original scopes will be used; otherwise you can request a reduced set of scopes. Note that this must be URL encoded. |

### Response

If all goes well, you'll receive an `HTTP 200` response with a payload containing a new `access_token`, its lifetime in seconds ( `expires_in` ), granted `scope` values, and `token_type`. If the scope of the initial token included `openid`, then the response will also include a new `id_token`:

```
{
  "access_token": "eyJ...MoQ",
  "expires_in": 86400,
  "scope": "openid offline_access",
  "id_token": "eyJ...0NE",
  "token_type": "Bearer"
}
```

Was this helpful? Yes / No

⚠ You should validate your tokens before saving them. To learn how, see Validate ID Tokens and Validate Access Tokens.

## Sample use cases

## Customize tokens

You can use Rules to change the returned scopes of Access Tokens and/or add claims to Access and ID Tokens. To do so, add the following rule, which will run after the user authenticates:

```
function(user, context, callback) {
  // add custom claims to Access Token and ID Token
  context.accessToken['http://foo/bar'] = 'value';
  context.idToken['http://fiz/baz'] = 'some other value';
  // change scope
  context.accessToken.scope = ['array', 'of', 'strings'];
  callback(null, user, context);
}
```

Was this helpful? Yes / No

Scopes will be available in the token after all rules have run.

⚠ Auth0 returns profile information in a structured claim format as defined by the OpenID Connect (OIDC) specification. This means that custom claims added to ID Tokens or Access Tokens must conform to a namespaced format to avoid possible collisions with standard OIDC claims.

## Keep reading

- OAuth 2.0 Authorization Framework
- OpenID Connect Protocol
- Tokens

Was this article helpful?

✓ YES   ✗ NO

WordPress

CONTACT

10800 NE 8th Street

Suite 600

Bellevue, WA 98004

+1 (888) 235-2699

+1 (425) 312-6521

+44 (0) 33-3234-1966

**Follow** 14 086 **Follow** 5 412 **Like** 14 395

Privacy Policy Terms of Service