# Cochrane Account Single Sign-on

- Introduction
- Registration
- Endpoints
    - Primary endpoints
- Authorization Code Grant (server-side) flow
    - Step A: Authorization code grant
    - Step B: Access token request
- Implicit Grant (client-side) flow
- Resource Owner Password Credentials
- Client Credentials Grant flow
- Validating the access token
    - Using Introspection Endpoint
    - Local validation
- Refreshing an access token
    - Access and refresh tokens configuration
- Using an access token to access the Archie API
    - Checking permissions for a user
    - Checking permissions for a single review

## Introduction

All major changes with introducing new SSO system are listed here.

The OAuth 2.0 authorization framework enables a third-party application (the client) to obtain limited access to an HTTP service on behalf of a resource owner (the end user). For instance, a tool for generating advanced meta analysis graphs (the client) could obtain access to read and possibly modify a review author's (the end user's) reviews in Archie.

The full specification of the OAuth 2.0 authorization framework can be found at: http://tools.ietf.org/html/rfc6749. In the following this is referred to as "the specification".

OAuth 2.0 is the framework to build authentication protocols, actual protocol used in Cochrane project is OpenID Connect (OIDC), which is implemented in Keycloak. For more information on OIDC implementation in Keycloak please go to: https://www.keycloak.org/docs/9.0/server_admin/index.html#_oidc.

OIDC itself uses JSON Web Token (JWT) standards to define identity token format and ways to digitally sign and encryt data. More information about JWT can be found at: https://tools.ietf.org/html/rfc7519.

Keycloak is implementing all four authorization grant types or 'flows' described in the specification:

1. **Authorization Code Grant** flow (section 4.1 of the specification), informally known as the server-side flow.
2. **Implicit Grant** flow (section 4.2 of the specification), informally known as the client-side flow.
3. **Resource Owner Password Credentials** (section 4.3 of the specification).
4. **Client Credentials Grant** (section 4.4 of the specification), also known as the client credentials flow.

The aim of each supported flow is for the client to obtain an access token that can be used in subsequent API calls to read and/or modify resources in Archie or Review DB in the context of the end user. This is done in a way (using HTTP redirects) where the end user never has to enter his or her Cochrane Account password into the client application but only on the common SSO login page.

OAuth 2.0 can also be used for single sign on, i.e. the ability for the end user to log into the client using his or her Cochrane Account, by utilising the authentication part of the OAuth flow only, and not using the access token for further API calls.

## Registration

Before a client can use the system it must be registered in Keycloak with the following information (see Keycloack documentation for details):

- Client ID;
- Client name (for display);
- Client secret (password - for confidential clients only, see below) - generated by Keycloak;
- One or more redirect URIs (to which the end user's user agent (typically a browser) is redirected after completing the Cochrane Account authentication);
- A URI pointing to a landing page for the client. This is used on various Cochrane Account pages (except login) to return the user to the client;
- The resource scopes that the client should have access to (see below);
- A client type.

In addition, for the client credentials flow, an Keycloak user must be associated with the client (see documentation for details). Keycloak automatically generates a user. It must be configured to be associated with an Archie user, which may grant it permissions in Archie.

There are two different types of client, confidential and public:

- A **confidential client** is one that is capable of keeping its client credentials secret and the access token hidden from the end user. This typically means that a server has to be involved where the access token can be stored in a user session object. A confidential client is using the server-side flow to obtain the access token server-to-server without involving the browser, and in this process it authenticates itself to Keycloak using its

client ID and client secret. Since this is the most secure type of client it has more privileges than a public client, e.g. the ability to refresh access tokens that are about to expire. Confidential clients may also use the client credentials flow to obtain an access token server-to-server without involving an end user.
- The typical example of a **public client** is a pure HTML5/JavaScript app. Since the source code and memory can be inspected by the end user in the browser this type of client is not capable of keeping the access token hidden from the end user. And since the client secret would be visible in the source code there's no point in assigning a secret to a public client. A public client is using the client-side flow to obtain short lived (e.g. 4 hours) access tokens.

The registration of redirect URIs is vital for the security of the system, and Keycloak only allows complete URIs to be registered (see Keycloak documentation for details, part *Valid Redirect URIs*).

## Endpoints

| Environment | KEYCLOAK_BASEURL | ARCHIE_BASEURL |
|---|---|---|
| Test | https://test-login.cochrane.org/ | https://test-archie.cochrane.org/ |
| Training | https://training-login.cochrane.org/ | https://training-archie.cochrane.org/ |
| Production | https://login.cochrane.org/ | https://archie.cochrane.org/ |

All endpoints require a secure connection (HTTPS). If plain HTTP is used an error status code 403 (forbidden) is returned.

### Primary endpoints

Authorization endpoint:

```
${KEYCLOAK_BASEURL}/auth/realms/cochrane/protocol/openid-connect/auth
```

Token endpoint:

```
${KEYCLOAK_BASEURL}/auth/realms/cochrane/protocol/openid-connect/token
```

## Authorization Code Grant (server-side) flow

### Step A: Authorization code grant

The server-side flow is optimised for confidential clients, although in theory it could be use by a public client. The first step is for the client to obtain an authorization code from Keycloak.

The client directs the end user's browser (e.g. in a pop-up window) to the authorization endpoint with the following query parameters ("application/x-www-form-urlencoded" format) added to the endpoint URI:

**response_type** (required): Must be set to "code" for this flow.

**client_id** (required): The ID obtained when registering the client.

**redirect_uri** (optional): A redirect URI registered for the client. Required if multiple URIs have been registered, otherwise defaults to the single registered URI.

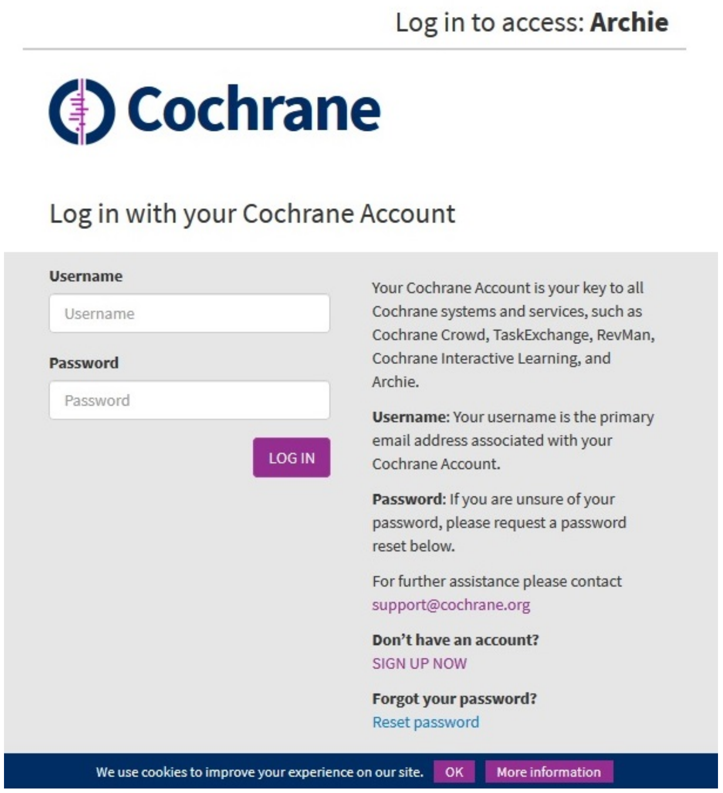
**state** (highly recommended): An (often randomly) generated value used by the client to maintain state between the request and callback. The parameter will be returned unchanged in the callback.

**scope** (optional): The scope of the resources that the access token should provide access to. Possible values are:

- "none": The default, single sign on only, no access to data
- "person: Manage person records
- "group": Manage groups
- "document": Manage documents and reviews
- "workflow": Manage tasks and workflows
- "crs": Manage studies in CRS
- "crso": Manage studies in CRSO
- "linked_data": Access to linked data resources
- "all": Manage any resource type

Multiple scopes can be provided as a space separated list, e.g. "person document", but "all" or "none" must stand alone. If no scope is specified, "none" is used (note: the default is **not** all scopes registered with the client). Only scopes registered with the client are allowed.

At the authorization endpoint the end user will have to log into Cochrane Account if he or she is not logged in already. After the authentication, provided that the scope of the request is different from "none", the end user may be presented with a consent screen where he or she has to agree to give the client access to resources in Archie within the given scope before the flow may continue. Note: the consent screen is not shown for clients hosted on *.cochrane.org domains.



Log in to access: **Archie**

Log in with your Cochrane Account

**Username**

Username

**Password**

Password

LOG IN

Your Cochrane Account is your key to all Cochrane systems and services, such as Cochrane Crowd, TaskExchange, RevMan, Cochrane Interactive Learning, and Archie.

**Username:** Your username is the primary email address associated with your Cochrane Account.

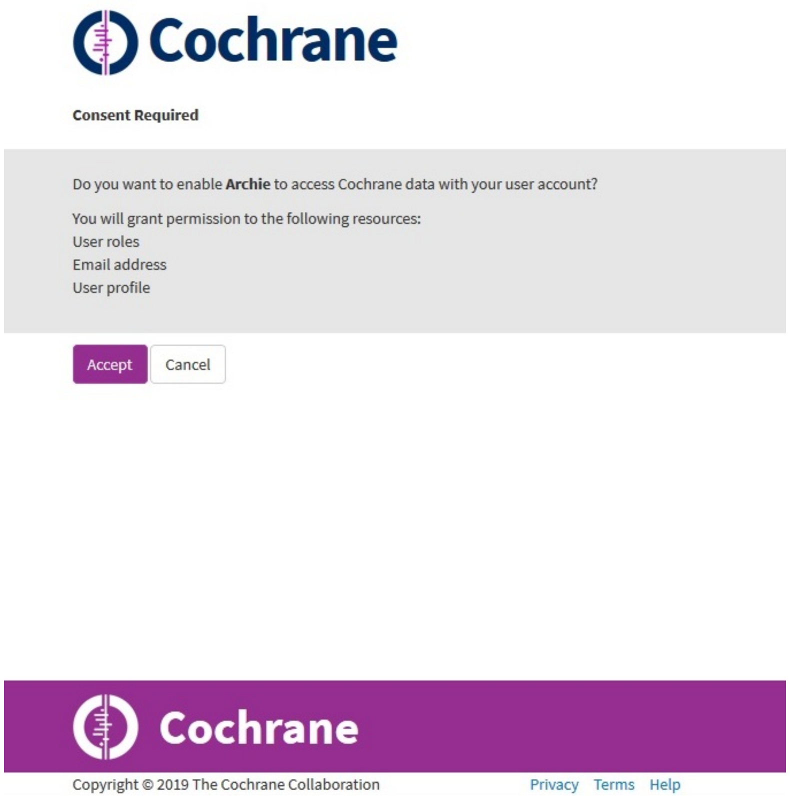
**Password:** If you are unsure of your password, please request a password reset below.

For further assistance please contact support@cochrane.org

**Don't have an account?**
SIGN UP NOW

**Forgot your password?**
Reset password

We use cookies to improve your experience on our site. OK More information



**Consent Required**

Do you want to enable **Archie** to access Cochrane data with your user account?

You will grant permission to the following resources:
User roles
Email address
User profile

Accept Cancel

Copyright © 2019 The Cochrane Collaboration

Privacy Terms Help

Once the authentication and possible consent is in order, i.e. the end user has authorized the client to access his or her data, the browser is redirected back to the redirect URI (using a HTTP 302 status code) with the following query parameters added:

**code**: The authorization code generated by Keycloak.

**state** (optional): The state parameter provided by the client, if any.

**session_state**: A unique value that identifies the current user session.

In case of an error condition, depending on the error type, an error message will be displayed either directly to the end user (if there is a problem with the client ID or redirect URI), or it will be provided as an **error** parameter in the redirect URI to the client. For instance, if the end user does not give his or her consent, an "access_denied" error is returned to the client. See the specification section 4.1.2.1 for details.

## Step B: Access token request

In the second step the client exchanges the authorization code with an access token server-to-server without involving the user's browser, and it requires client credentials as Basic auth.

The client's server makes a POST request to the token endpoint with the following parameters included ("application/x-www-form-urlencoded" format):

**grant_type** (required): Must be set to "authorization_code" for this flow.

**code** (required): The authorization code obtained in step A.

**redirect_uri** (optional): Required if the **redirect_uri** parameter was provided in step A, in which case the parameter must have the exact same value. This parameter is not used for redirecting at this step, only for added security.

### Client authentication

For any call to the token endpoint, including this, the client must provide its credentials as Basic HTTP authentication with client ID as user name and client secret as password. The authentication must be provided in the Authorization header for a confidential client.

Before building the header for Basic authentication, client IDs and client secrets containing special characters including %, +, and all non-ASCII characters  must be encoded using the UTF-8 character encoding scheme first; the resulting octet sequence then needs to be further application/x-www-form-urlencoded (see section 2.3.1 of the specification). To build the header, the encoded credentials are separated by a colon, the resulting string is base 64 encoded, and the result is prefixed with "Basic ".

### Access code response

The response to the POST request is a JSON object (MIME type "application/json") like this:

```
{
    "access_token":<JWT>,
    "expires_in":300,
    "refresh_expires_in":1800,
    "refresh_token":<REFRESH_TOKEN>,
    "token_type":"bearer",
    "not-before-policy":1549438553,
    "session_state":"db478a1b-3acd-4737-bcce-bf83bc0d9eda",
    "scope":"person email"
}
```

The first field is the actual access token which should be kept on the server and never passed to the end user's browser. The **token_type** will always be "Bearer" in this implementation. The number of seconds that the token is valid is given in **expires_in**.

### Error response

In case of an error condition Keycloak responds with a HTTP 400 status code with the error message included in the body of the response as a JSON object, e.g.:

```
{
    "error": "invalid_grant",
    "error_description": "Invalid authorization code."
}
```

For further details see section 5.2 of the specification.

# Implicit Grant (client-side) flow

The client-side flow is optimised for public clients such as pure HTML5/JavaScript clients.

The client directs the end user's browser (e.g. in a pop-up window or IFRAME) to the authorization endpoint with the following query parameters added ("application/x-www-form-urlencoded" format) to the endpoint URI:

**response_type** (required): Must be set to "token" for this flow.

**client_id** (required): The ID obtained when registering the client.

**redirect_uri** (optional): A redirect URI registered for the client. Required if multiple URIs have been registered, otherwise defaults to the single registered URI.

**state** (highly recommended): A (often randomly) generated value used by the client to maintain state between the request and callback. The parameter will be returned unchanged in the callback.

**scope** (optional) (the same as in scope description above)

Multiple scopes can be provided as a space separated list, e.g. "person document", but "all" or "none" must stand alone. If no scope is specified, "none" is used (note: the default is not all scopes registered with the client). Only scopes registered with the client are allowed.

The authentication and consent screen is displayed as for the server-side flow. Once completed, the browser is redirected back to the redirect URI (using a HTTP 302 status code) with the following parameters added to the fragment component of the redirect URI, i.e. after the hash sign:

**access_token**: The access token issued by Keycloak.

**token_type**: Always "Bearer" .

**expires_in**: The number of seconds that the token is valid.

**state** (optional): The state parameter provided by the client, if any.

In case of an error condition, depending on the error type, an error message will either be displayed directly to the end user (if there is a problem with the client ID or redirect URI), or it will be provided as an **error** parameter in the fragment component of the redirect URI to the client. See the specification section 4.2.2.1 for details.

The format of the access token and error response objects is as described above for the Authorization Code Grant. However, no refresh token is supplied.

## Resource Owner Password Credentials

The Resource Owner Password Credentials flow allows client to obtain an access token using resource owner's credentials (username and password). The Password grant is one of the simplest OAuth grants and involves only one step: client uses resource owner's username and password to make POST request to the server to exchange password for access token. This authorization grant type requires that the application collect the user's password.

POST request is done with follwoing parameters:

**grant_type** (required): Must be set to "password" for this flow.

**username(required):** Must be set to resource owner usrename.

**password(required):** Must be set to resource owner password.

**client_id(required):** Must be set to identifier of the client that was obtained during client registration.

**scope** (optional) (the same in scope description above).

The format of the access token and error response objects is as described above for the Authorization Code Grant.

## Client Credentials Grant flow

The client credentials flow allows a confidential client (server) to obtain an access token server-to-server without involving an end user. Before this flow is used, an Keycloak user must be associated with the client. The access token obtained with this flow will be associated with the user and the user's permissions.

The server makes a POST request to the token endpoint with the following parameters included ("application/x-www-form-urlencoded" format):

**grant_type** (required): Must be set to "client_credentials" for this flow.

**scope** (optional) (the same as in scope description above).

Multiple scopes can be provided as a space separated list, e.g. "person document", but "all" or "none" must stand alone. If no scope is specified, "none" is used (note: the default is **not** all scopes registered with the client). Only scopes registered with the client are allowed.

Authentication must be provided in the Authorization header as described for the Authorization Code Grant under client authentication.

The format of the access token and error response objects is as described above for the Authorization Code Grant.

# Validating the access token

Keycloack provides a mechanism for validating access token. There are two ways to approach it - using Introspection Endpoint and validating tokens locally. See Keycloack documentation for more details.

## Using Introspection Endpoint

In order to manually validate a token issued by Keycloack, invoke Introspection Endpoint  /realms/{realm-name}/protocol/openid-connect/token/introspect

Please note that this endpoint can only be invoked by **confidential** clients (see OAuth 2 documentation for details).

To validate a token, make a POST request with parameters sent as "application/x-www-form-urlencoded" data to the Introspection Endpoint (see above) with the following query parameters included:

**introspection**: takes one of two possible values: access_token or refresh_token.

The response is a JSON object (MIME type "application/json") like this:

```
{
    "active": true,
    "scope": "document",
    "client_id": "l238j323ds-23ij4",

    "user_name": "jdoe",

    "token_type": "bearer"

    "exp": 1419356238,

    "iat": 1419356238,

    "nbf": 1419350238

}
```

## Local validation

Due to the fact that Keycloack issued tokens are JWT digitally signed and encoded using JWS, access token can be locally validated using public key of the issueing realm. The realm's public key can be be looked-up and cached using the certificate endpoint with the Key ID (KID) embedded within the JWS.

Certificate endpoint is: /realms/{realm-name}/protocol/openid-connect/certs. It returns public keys enabled by realm encoded as JSON Web Key (JWK).

Accurate clock synchronisation is required for local token validation, so it should not be used in environments where this can't be guaranteed (e.g. end-user web browser).

# Refreshing an access token

Refresh tokens can be used to obtain a new access token without asking for the end user's consent. Access tokens expire after 4 hours unless they are revoked. Refresh tokens are valid for 180 days or until they are used to obtain a new access token.

To refresh an access token (server-side flow only) the client's server makes a POST request to the token endpoint with the following parameters included ("application/x-www-form-urlencoded" format):

**grant_type** (required): Must be set to "refresh_token".

**refresh_token** (required): The previously obtained refresh token.

The format of the response is identical to the one returned when the original access token was obtained.

Note that a new refresh token is also issued with the response, so the old refresh token must be discarded by the client and replaced by the new token.

In case of an error condition, e.g. if the refresh token is no longer valid, the response will follow the same structure as for other token endpoint errors. The client should discard the refresh token and obtain a new access token using the server-side flow.

## Access and refresh tokens configuration

Keycloak provides a possibility to configure different lifespan for access token and refresh token. This is all done on the Tokens tab in the Keyclock. Expiration time can be configured for:

- Access token in *Access Token Lifespan* field
- Refresh token in *SSO Session Idle* field

See Session and Token Timeouts for more detailed information.

## Using an access token to access the Archie API

The access token must be passed to the API in the Authorization header, e.g.:

GET ${ARCHIE_BASEURL}/rest/reviews/CD000004   HTTP/1.1

Authorization: Bearer 3e8ec1a3d43c983b57df0616b498c04807b466e919999aa0f3f3aabca1dd48cc

The result, if the call succeeds, is the review in XML format (MIME type "application/xml"). If the call fails an HTTP error status code will be returned.

See Review Document API for more examples.

### Checking permissions for a user

The access token must be passed to the API in the Authorization header, e.g.:

GET  ${ARCHIE_BASEURL}/oauth2/permissions HTTP/1.1

Authorization: Bearer 3e8ec1a3d43c983b57df0616b498c04807b466e919999aa0f3f3aabca1dd48cc

The response is a JSON object (MIME type "application/json") like this:

```
[
    {
        "resource": "folder",
        "entity": "all",
        "grants": ["delete", "view_folder", "read_files", "write"]
    }
    {
        "resource": "person",
        "entity": "all",
        "grants": ["view", "view_private", "write", "delete", "mark_duplicate", "select_topics",
"affiliation_crud"]
    }
]
```

### Checking permissions for a single review

As another example, the API contains a method to get the user's permissions for a single review:

GET  ${ARCHIE_BASEURL}/rest/reviews/CD000004/permissions HTTP/1.1

Authorization: Bearer 3e8ec1a3d43c983b57df0616b498c04807b466e919999aa0f3f3aabca1dd48cc

The result, if the call succeeds, is a JSON object (MIME type "application/json") listing the user's permissions, e.g.:

```
{
 "permissions": ["view", "read_published", "read_to_be_published", "read_author_drafts", "read_shared_drafts",
"write_authoring", "view_author_roles"],
 "read": true
 "write": true
 "delete": false
}
```

The "read", "write" and "delete" properties in the response that take the current phase of the review into account. For instance, *"write": true* could be returned if the user can write in authoring phase ("write_authoring" permission) AND the review is in authoring phase.

If the call fails an HTTP error status code will be returned.