
invenio-oauthclient Documentation

Release 1.1.0

CERN

Dec 17, 2018

Contents

1	User's Guide	3
1.1	Installation	3
1.2	Overview	3
1.3	Configuration	5
1.4	Usage	8
1.5	Example applications	11
2	API Reference	15
2.1	API Docs	15
3	Additional Notes	25
3.1	Contributing	25
3.2	Changes	27
3.3	License	27
3.4	Contributors	27
4	Indices and tables	29
	Python Module Index	31

Invenio module that provides OAuth web authorization support.

OAuth client support is typically used to allow features such as social login (e.g. Sign in with Twitter) and access to resources owned by a specific user at a remote service. Both OAuth 1.0 and OAuth 2.0 are supported.

Features:

- Views: OAuth login and authorized endpoints, linked account settings and sign-up handling.
- Client: A client to interact with remote applications.
- Contrib: Ready-to-use GitHub, ORCID, and CERN remote applications.
- Models: Persistence layer for OAuth access tokens including support for storing extra data together with a token.
- Handlers: Customizable handlers for deciding what happens when a user authorizes a request.

Further documentation is available on <https://invenio-oauthclient.readthedocs.io/>

This part of the documentation will show you how to get started in using Invenio-OAuthClient.

1.1 Installation

Invenio-OAuthClient is on PyPI so all you need is:

```
$ pip install invenio-oauthclient
```

1.2 Overview

OAuth 2.0 defines several possible *authorization flows* depending on the type of client you are authorizing (e.g. web application, browser-based app or mobile apps). The *web application client* is the only authorization flow supported by this module.

A typical web application authorization flow involves the following roles:

- **Client** (i.e. a third-party application in this case your Invenio instance).
- **Resource server** (i.e. the remote service).
- **Resource owner** (i.e. the user).

The web application authorization flow is used to e.g. allow sign in with service X. The end result of a completed authorization flow is an *access token* which allows the *client* to access a *resource owner's* resources on the *resource server*.

Before the authorization flow is started, the *client* must be registered with the *resource server*. The resource server will provide a *client key* and *client secret* to the client. Following is an example of the authorization flow with ORCID:

1. The resource owner (i.e. the user) clicks “Sign in with ORCID”:

```
GET /oauth/login/orcid/ HTTP/1.1
```

The *client* redirects the user to the resource server's *authorize URL*.

```
HTTP/1.1 302 FOUND
Location: https://orcid.org/oauth/authorize?response_type=code&
  client_id=<CLIENT KEY>&
  redirect_uri=https://localhost/oauth/authorized/orcid/&
  scope=/authenticate&
  state=...
```

Note, following query parameters in the authorize URL:

- `response_type` - Must be `code` for web application flow (named authorization code grant).
- `client_id` - The client key provided by the resource server when the client was registered.
- `redirect_uri` - The URL the resource server will redirect the resource owner back to after having authorized the request. Usually the redirect URL must be provided when registering the client application with the resource server.
- `scope` - Defines the level of access (defined by the resource server)
- `state` - A token to mitigate against cross-site request forgery (CSRF). In Invenio this state is a JSON Web Signature (JWS) that by default expires after 5 minutes.

2. The *resource server* asks the user to sign-in (if not already signed in).
3. The *resource server* asks the *resource owner* to authorize or reject the client's request for access.
4. If the *resource owner* authorizes the request, the *resource server* redirects the *resource owner* back to the *client* web application (using the `redirect_uri` provided in step 1):

```
HTTP/1.1 302 FOUND
Location: https://localhost/oauth/authorized/orcid/?code=<CODE>&
  state=...
```

Included in the redirect is a one-time *auth code* which is typically only valid for short time (seconds), as well as the `state` token initially provided.

5. The client now exchanges the one time *auth code* for an *access token* using the resource server's *access token URL*:

```
POST https://pub.orcid.org/oauth/token HTTP/1.1
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&
code=<CODE>&
redirect_uri=<REDIRECT_URI>&
client_id=<CLIENT KEY>&
client_secret=<CLIENT SECRET>
```

The resource server replies with an access token:

```
{"access_token": "<ACCESS TOKEN>"}
```

The client stores the access token, and can use it to make authenticated requests to the *resource server*:

```
GET https://api.example.org/ HTTP/1.1
Authorization: Bearer <ACCESS TOKEN>
```


Further reading:

- [RFC6749 - The OAuth 2.0 Authorization Framework](#)
- [OAuth 2 Simplified](#)
- [Flask-OAuthlib](#)
- [OAuthlib](#)

1.3 Configuration

Configuration variables for defining remote applications.

<code>OAUTHCLIENT_REMOTE_APPS</code>	Dictionary of remote applications. See example below. Default: <code>{}</code> .
<code>OAUTHCLIENT_SESSION_KEY_PREFIX</code>	Prefix for the session key used to store the an access token. Default: <code>oauth_token</code> .
<code>OAUTHCLIENT_STATE_EXPIRES</code>	Number of seconds after which the state token expires. Defaults to 300 seconds.
<code>OAUTHCLIENT_REMOTE_APP</code>	Replaces the default remote application class.

Each remote application must be defined in the `OAUTHCLIENT_REMOTE_APPS` dictionary, where the keys are the application names and the values the configuration parameters for the application.

```
OAUTHCLIENT_REMOTE_APPS = dict(
    myapp=dict(
        # configuration values for myapp ...
    ),
)
```

The application name is used in the login, authorized, sign-up and disconnect endpoints:

- Login endpoint: `/oauth/login/<REMOTE_APP>/`.
- Authorized endpoint: `/oauth/authorized/<REMOTE_APP>/`.
- Disconnect endpoint: `/oauth/disconnect/<REMOTE_APP>/`.
- Sign up endpoint: `/oauth/login/<REMOTE_APP>/`.

1.3.1 Remote application

Configuration of a single remote application is a dictionary with the following keys:

- `title` - Title of remote application. Displayed to end-users under Account > Linked accounts.
- `description` - Short description of remote application. Displayed to end-users under Account > Linked accounts.
- `icon` - CSS class for icon of service (e.g. `fa fa-github` for using the Font-Awesome GitHub icon). Displayed to end-users.
- `params` - Flask-OAuthlib remote application parameters..
- `authorized_handler` - Import path to authorized callback handler.
- `disconnect_handler` - Import path to disconnect callback handler.
- `signup_handler` - A dictionary of import path to sign up callback handler.

```
OAuthCLIENT_REMOTE_APPS = dict(
    myapp=dict(
        title='...',
        description='...',
        icon='...',
        authorized_handler="...",
        disconnect_handler="...",
        signup_handler=dict(
            info="...",
            setup="...",
            view="...",
        ),
        params=dict(...),
    )
)
```

1.3.2 Flask-OAuthlib parameters

The Flask-OAuthlib parameters defines the remote application OAuth endpoints as well as the client id and secret. Full description of these parameters are given in the [Flask-OAuthlib documentation](#).

Normally you will have to browse the remote application's API documentation to find which URLs and scopes to use.

Below is an example for GitHub:

```
OAuthCLIENT_REMOTE_APPS = dict(
    github=dict(
        # ...
        params=dict(
            request_token_params={'scope': 'user:email'},
            base_url='https://api.github.com/',
            request_token_url=None,
            access_token_url="https://github.com/login/oauth/access_token",
            access_token_method='POST',
            authorize_url="https://github.com/login/oauth/authorize",
            app_key="GITHUB_APP_CREDENTIALS",
        )
    )
)

GITHUB_APP_CREDENTIALS=dict(
    consumer_key="changeme"
    consumer_secret="changeme"
)
```

The `app_key` parameter allows you to put your sensitive client id and secret in your instance configuration (`var/invenio.base-instance/invenio.cfg`).

1.3.3 Handlers

Handlers allow customizing oauthclient endpoints for each remote application:

- Authorized endpoint: `/oauth/authorized/<REMOTE APP>/`.
- Disconnect endpoint: `/oauth/disconnect/<REMOTE APP>/`.

- Sign up endpoint: /oauth/login/<REMOTE_APP>/.

By default only authorized and disconnect handlers are required, and Invenio provide default implementation that stores the access token in the user session as well as to the database if the user is authenticated:

```

OAUTHCLIENT_REMOTE_APPS = dict(
    myapp=dict(
        # ...
        authorized_handler="invenio_oauthclient.handlers"
                           ":authorized_default_handler",
        disconnect_handler="invenio_oauthclient.handlers"
                           ":disconnect_handler",
    )
    # ...
)

```

If you want to provide sign in/up functionality using oauthclient, Invenio comes with a default handler that will try to find a matching local user for a given authorize request.

```

OAUTHCLIENT_REMOTE_APPS = dict(
    orcid=dict(
        # ...
        authorized_handler="invenio_oauthclient.handlers"
                           ":authorized_signup_handler",
        disconnect_handler="invenio_oauthclient.handlers"
                           ":disconnect_handler",
    )
    signup_handler=dict(
        info="invenio_oauthclient.contrib.orcid:account_info",
        setup="invenio_oauthclient.contrib.orcid:account_setup",
        view="invenio_oauthclient.handlers:signup_handler",
    ),
    # ...
)

```

1.3.4 Custom remote application

Some OAuth services require a specific handling of OAuth requests. If the standard flask-oauthlib.client.OAuthRemoteApp does not support it, it is possible to replace the standard OAuthRemoteApp for all remote application by referring to the custom class with the configuration variable OAUTHCLIENT_REMOTE_APP or for only one remote application by setting remote_app in your remote application configuration.

```

class CustomOAuthRemoteApp(OAuthRemoteApp):
    pass

app.config.update(
    OAUTHCLIENT_REMOTE_APP=
        'myproject.mymodule:CustomOAuthRemoteApp'
)

# OR

app.config.update(
    OAUTHCLIENT_REMOTE_APPS=dict(

```

(continues on next page)

(continued from previous page)

```
        custom_app=dict(
            # ...
            remote_app=
                'myproject.mymodule:CustomOAuthRemoteApp'
        )
    )
)
```

1.4 Usage

1.4.1 GitHub

Pre-configured remote application for enabling sign in/up with GitHub.

1. Ensure you have `github3.py` package installed:

```
cdvirtualenv src/invenio-oauthclient
pip install -e .[github]
```

2. Edit your configuration and add:

```
from invenio_oauthclient.contrib import github

OAUTHCLIENT_REMOTE_APPS = dict(
    github=github.REMOTE_APP,
)

GITHUB_APP_CREDENTIALS = dict(
    consumer_key='changeme',
    consumer_secret='changeme',
)
```

3. Go to GitHub and register a new application: <https://github.com/settings/applications/new>. When registering the application ensure that the *Authorization callback URL* points to: `CFG_SITE_SECURE_URL/oauth/authorized/github/` (e.g. `http://localhost:4000/oauth/authorized/github/` for development).
4. Grab the *Client ID* and *Client Secret* after registering the application and add them to your instance configuration (`invenio.cfg`):

```
GITHUB_APP_CREDENTIALS = dict(
    consumer_key='<CLIENT ID>',
    consumer_secret='<CLIENT SECRET>',
)
```

5. Now go to `CFG_SITE_SECURE_URL/oauth/login/github/` (e.g. <http://localhost:4000/oauth/login/github/>)
6. Also, you should see GitHub listed under Linked accounts: <http://localhost:4000/account/settings/linkedaccounts/>

By default the GitHub module will try first look if a link already exists between a GitHub account and a user. If no link is found, the module tries to retrieve the user email address from GitHub to match it with a local user. If this fails, the user is asked to provide an email address to sign-up.

In templates you can add a sign in/up link:

```
<a href="{{url_for('invenio_oauthclient.login', remote_app='github')}}">
  Sign in with GitHub
</a>
```

For more details you can play with a *working example*.

1.4.2 ORCID

Pre-configured remote application for enabling sign in/up with ORCID.

1. Edit your configuration and add:

```
from invenio_oauthclient.contrib import orcid

OAUTHCLIENT_REMOTE_APPS = dict(
    orcid=orcid.REMOTE_APP,
)

ORCID_APP_CREDENTIALS = dict(
    consumer_key="changeme",
    consumer_secret="changeme",
)
```

Note, if you want to use the ORCID Member API, use `orcid.REMOTE_MEMBER_APP` instead of `orcid.REMOTE_APP`.

In case you want use sandbox: To use the ORCID Public API sandbox, use `orcid.REMOTE_SANDBOX_APP` instead of `orcid.REMOTE_APP`. To use the ORCID Member API sandbox, use `orcid.REMOTE_SANDBOX_MEMBER_APP`.

2. Register a new application with ORCID. When registering the application ensure that the *Redirect URI* points to: `CFG_SITE_URL/oauth/authorized/orcid/` (note, ORCID does not allow localhost to be used, thus testing on development machines is somewhat complicated by this).
3. Grab the *Client ID* and *Client Secret* after registering the application and add them to your instance configuration (`invenio.cfg`):

```
ORCID_APP_CREDENTIALS = dict(
    consumer_key="<CLIENT ID>",
    consumer_secret="<CLIENT SECRET>",
)
```

4. Now go to `CFG_SITE_URL/oauth/login/orcid/` (e.g. <http://localhost:4000/oauth/login/orcid/>)
5. Also, you should see ORCID listed under Linked accounts: <http://localhost:4000/account/settings/linkedaccounts/>

By default the ORCID module will try first look if a link already exists between a ORCID account and a user. If no link is found, the user is asked to provide an email address to sign-up.

In templates you can add a sign in/up link:

```
<a href="{{url_for('invenio_oauthclient.login', remote_app='orcid')}}">
  Sign in with ORCID
</a>
```

For more details you can play with a *working example*.

1.4.3 CERN

Pre-configured remote application for enabling sign in/up with CERN.

1. Edit your configuration and add:

```
import copy

from invenio_oauthclient.contrib import cern

CERN_REMOTE_APP = copy.deepcopy(cern.REMOTE_APP)
CERN_REMOTE_APP["params"].update(dict(request_token_params={
    "resource": "changeme.cern.ch", # replace with your server
    "scope": "Name Email Bio Groups",
}))

OAUTHCLIENT_REMOTE_APPS = dict(
    cern=CERN_REMOTE_APP,
)

CERN_APP_CREDENTIALS = dict(
    consumer_key="changeme",
    consumer_secret="changeme",
)
```

Note, if you want to use the CERN sandbox, use `cern.REMOTE_SANDBOX_APP` instead of `cern.REMOTE_APP`.

2. Register a new application with CERN. When registering the application ensure that the *Redirect URI* points to: `http://localhost:5000/oauth/authorized/cern/` (note, CERN does not allow localhost to be used, thus testing on development machines is somewhat complicated by this).
3. Grab the *Client ID* and *Client Secret* after registering the application and add them to your instance configuration (`invenio.cfg`):

```
CERN_APP_CREDENTIALS = dict(
    consumer_key="<CLIENT ID>",
    consumer_secret="<CLIENT SECRET>",
)
```

4. Now login using CERN OAuth: `http://localhost:5000/oauth/login/cern/`.
5. Also, you should see CERN listed under Linked accounts: `http://localhost:5000/account/settings/linkedaccounts/`

By default the CERN module will try first look if a link already exists between a CERN account and a user. If no link is found, the user is asked to provide an email address to sign-up.

In templates you can add a sign in/up link:

```
<a href="{{ url_for('invenio_oauthclient.login', remote_app='cern') }}">
    Sign in with CERN
</a>
```

For more details you can play with a *working example*.

1.4.4 Globus

Pre-configured remote application for enabling sign in/up with Globus.

1. Edit your configuration and add:

```
from invenio_oauthclient.contrib import globus

OAUTHCLIENT_REMOTE_APPS = dict(
    globus=globus.REMOTE_APP,
)

GLOBUS_APP_CREDENTIALS = dict(
    consumer_key='changeme',
    consumer_secret='changeme',
)
```

2. Register a Globus application at <https://developers.globus.org/> with the *Redirect URL* as <http://localhost:5000/oauth/authorized/globus/>. For full documentation on all app fields, see: <https://docs.globus.org/api/auth/developer-guide/#register-app>
4. Grab the *Client ID* and *Client Secret* after registering the application and add them to your instance configuration (invenio.cfg):

```
GLOBUS_APP_CREDENTIALS = dict(
    consumer_key='<CLIENT ID>',
    consumer_secret='<CLIENT SECRET>',
)
```

5. Now go to your site: <http://localhost:5000/oauth/authorized/globus/>
6. You should see Globus listed under Linked accounts: <http://localhost:5000/account/settings/linkedaccounts/>

1.4.5 Advanced

Advanced usage docs.

1.5 Example applications

1.5.1 GitHub

1. Register a github application with *Authorization callback URL* as <http://localhost:5000/oauth/authorized/github/>
2. Ensure you have github3.py package installed:

```
$ cdvirtualenv src/invenio-oauthclient
$ pip install -e .[github]
```

3. Grab the *Client ID* and *Client Secret* after registering the application and add them to your instance configuration as *consumer_key* and *consumer_secret*.

```
$ export GITHUB_APP_CREDENTIALS_KEY=my_github_client_id
$ export GITHUB_APP_CREDENTIALS_SECRET=my_github_client_secret
```

4. Create database and tables:

```
$ pip install -e .[all]
$ cd examples
```

(continues on next page)

(continued from previous page)

```
$ export FLASK_APP=github_app.py
$ ./app-setup.py
```

You can find the database in *examples/github_app.db*.

5. Run the development server:

```
$ flask run -p 5000 -h '0.0.0.0'
```

6. Open in a browser the page *http://0.0.0.0:5000/github*.

You will be redirected to github to authorize the application.

Click on *Authorize application* and you will be redirected back to *http://localhost:5000/oauth/signup/github/*, where you will be able to finalize the local user registration, inserting email address.

Insert e.g. *fuu@bar.it* as email address and send the form.

Now, you will be again in homepage but this time it say: *hello fuu@bar.it*.

You have completed the user registration.

7. To be able to uninstall the example app:

```
$ ./app-teardown.sh
```

1.5.2 ORCID

1. Register an orcid application with *Authorization callback URL* as *http://localhost:5000/oauth/authorized/orcid/*
2. Install oauthclient:

```
cdvirtualenv src/invenio-oauthclient
pip install -e .[orcid]
```

3. Grab the *Client ID* and *Client Secret* after registering the application and add them to your instance configuration as *consumer_key* and *consumer_secret*.

```
$ export ORCID_APP_CREDENTIALS_KEY=my_orcid_client_id
$ export ORCID_APP_CREDENTIALS_SECRET=my_orcid_client_secret
```

4. Create database and tables:

```
$ pip install -e .[all]
$ cd examples
$ export FLASK_APP=orcid_app.py
$ ./app-setup.py
```

You can find the database in *examples/orcid_app.db*.

5. Run the development server:

```
$ flask -a orcid_app.py run -p 5000 -h '0.0.0.0'
```

6. Open in a browser the page *http://0.0.0.0:5000/orcid*.

You will be redirected to orcid to authorize the application.

Click on *Authorize application* and you will be redirected back to `http://0.0.0.0:5000/oauth/authorized/orcid/`, where you will be able to finalize the local user registration, inserting email address.

Insert e.g. `fuu@bar.it` as email address and send the form.

Now, you will be again in homepage but this time it say: *hello fuu@bar.it*.

You have completed the user registration.

7. To be able to uninstall the example app:

```
$ ./app-teardown.sh
```

1.5.3 CERN

1. Register a CERN application in `https://sso-management.web.cern.ch/OAuth/RegisterOAuthClient.aspx` with `redirect_uri` as `https://localhost:5000/oauth/authorized/cern/` and filling all the other fields:
2. Ensure you have `gunicorn` package installed:

```
cdvirtualenv src/invenio-oauthclient
pip install -e gunicorn
```

3. Ensure you have `openssl` installed in your system (Most of the Linux distributions has it by default.).
3. Grab the `client_id` and `secret_uri` after registering the application and add them to your instance configuration as `consumer_key` and `consumer_secret`.

```
$ export CERN_APP_CREDENTIALS_KEY=my_cern_client_id
$ export CERN_APP_CREDENTIALS_SECRET=my_cern_secret_uri
```

4. Create database and tables:

```
$ pip install -e .[all]
$ cd examples
$ export FLASK_APP=cern_app.py
$ ./app-setup.py
```

You can find the database in `examples/cern_app.db`.

5. Create the key and the certificate in order to run a HTTPS server:

```
$ openssl genrsa 1024 > ssl.key
$ openssl req -new -x509 -nodes -sha1 -key ssl.key > ssl.crt
```

6. Run gunicorn server:

```
$ gunicorn -b :5000 --certfile=ssl.crt --keyfile=ssl.key cern_app:app
```

7. Open in a browser the page `https://localhost:5000/cern`.

You will be redirected to CERN to authorize the application.

Click on *Grant* and you will be redirected back to `https://localhost:5000/oauth/authorized/cern/`

Now, you will be again in homepage but this time it say: *hello youremail@cern.ch*.

You have completed the user authorization.

8. To be able to uninstall the example app:

```
$ ./app-teardown.sh
```

1.5.4 Globus

1. Register a Globus application at <https://developers.globus.org/> with the *Redirect URL* as <http://localhost:5000/oauth/authorized/globus/>. See [here](https://docs.globus.org/api/auth/developer-guide/#register-app) for more documentation: <https://docs.globus.org/api/auth/developer-guide/#register-app>
2. Grab the *Client ID* and *Client Secret* after registering the application and add them to your instance configuration as *consumer_key* and *consumer_secret*.

```
$ export GLOBUS_APP_CREDENTIALS_KEY=my_globus_client_id
$ export GLOBUS_APP_CREDENTIALS_SECRET=my_globus_client_secret
```

3. Create database and tables:

```
$ cdvirtualenv src/invenio-oauthclient
$ pip install -e .[all]
$ cd examples
$ export FLASK_APP=globus_app.py
$ ./app-setup.py
```

You can find the database in *examples/globus_app.db*.

4. Run the development server:

```
$ flask run -p 5000 -h '0.0.0.0'
```

5. Open in a browser the page <http://localhost:5000/globus>.

You will be redirected to globus to authorize the application.

Click on *Allow* and you will be redirected back to <http://localhost:5000/oauth/signup/globus/>, where you will be able to finalize the local user registration.

6. To clean up and drop tables:

```
$ ./app-teardown.sh
```

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

2.1 API Docs

2.1.1 Handlers

Handlers for customizing oauthclient endpoints.

`invenio_oauthclient.handlers.authorized_default_handler(*args, **kwargs)`

Store access token in session.

Default authorized handler.

Parameters

- **remote** – The remote application.
- **resp** – The response.

Returns Redirect response.

`invenio_oauthclient.handlers.authorized_signup_handler(*args, **kwargs)`

Handle sign-in/up functionality.

Parameters

- **remote** – The remote application.
- **resp** – The response.

Returns Redirect response.

`invenio_oauthclient.handlers.disconnect_handler(remote, *args, **kwargs)`

Handle unlinking of remote account.

This default handler will just delete the remote account link. You may wish to extend this module to perform clean-up in the remote service before removing the link (e.g. removing install webhooks).

Parameters `remote` – The remote application.

Returns Redirect response.

`invenio_oauthclient.handlers.get_session_next_url(remote_app)`

Return redirect url stored in session.

Parameters `remote_app` – The remote application.

Returns The redirect URL.

`invenio_oauthclient.handlers.make_handler(f, remote, with_response=True)`

Make a handler for authorized and disconnect callbacks.

Parameters `f` – Callable or an import path to a callable

`invenio_oauthclient.handlers.make_token_getter(remote)`

Make a token getter for a remote application.

`invenio_oauthclient.handlers.oauth1_token_setter(remote, resp, token_type="", extra_data=None)`

Set an OAuth1 token.

Parameters

- **remote** – The remote application.
- **resp** – The response.
- **token_type** – The token type. (Default: '')
- **extra_data** – Extra information. (Default: None)

Returns A `invenio_oauthclient.models.RemoteToken` instance.

`invenio_oauthclient.handlers.oauth2_handle_error(remote, resp, error_code, error_uri, error_description)`

Handle errors during exchange of one-time code for an access tokens.

`invenio_oauthclient.handlers.oauth2_token_setter(remote, resp, token_type="", extra_data=None)`

Set an OAuth2 token.

The `refresh_token` can be used to obtain a new `access_token` after the old one is expired. It is saved in the database for long term use. A `refresh_token` will be present only if `access_type=offline` is included in the authorization code request.

Parameters

- **remote** – The remote application.
- **resp** – The response.
- **token_type** – The token type. (Default: '')
- **extra_data** – Extra information. (Default: None)

Returns A `invenio_oauthclient.models.RemoteToken` instance.

`invenio_oauthclient.handlers.oauth_error_handler(f)`

Decorator to handle exceptions.

`invenio_oauthclient.handlers.oauth_logout_handler(sender_app, user=None)`

Remove all access tokens from session on logout.

`invenio_oauthclient.handlers.response_token_setter(remote, resp)`

Extract token from response and set it for the user.

Parameters

- **remote** – The remote application.
- **resp** – The response.

Raises

- `invenio_oauthclient.errors.AuthClientError` – If authorization with remote service failed.
- `invenio_oauthclient.errors.AuthResponseError` – In case of bad authorized request.

Returns The token.

`invenio_oauthclient.handlers.set_session_next_url(remote_app, url)`
Store redirect url in session for security reasons.

Parameters

- **remote_app** – The remote application.
- **url** – the redirect URL.

`invenio_oauthclient.handlers.signup_handler(remote, *args, **kwargs)`
Handle extra signup information.

Parameters **remote** – The remote application.

Returns Redirect response or the template rendered.

`invenio_oauthclient.handlers.token_delete(remote, token=)`
Remove OAuth access tokens from session.

Parameters

- **remote** – The remote application.
- **token** – Type of token to get. Data passed from `oauth.request()` to identify which token to retrieve. (Default: '')

Returns The token.

`invenio_oauthclient.handlers.token_getter(remote, token=)`
Retrieve OAuth access token.

Used by flask-oauthlib to get the access token when making requests.

Parameters

- **remote** – The remote application.
- **token** – Type of token to get. Data passed from `oauth.request()` to identify which token to retrieve. (Default: '')

Returns The token.

`invenio_oauthclient.handlers.token_session_key(remote_app)`
Generate a session key used to store the token for a remote app.

Parameters **remote_app** – The remote application.

Returns The session key.

`invenio_oauthclient.handlers.token_setter(remote, token, secret="", token_type="", extra_data=None, user=None)`
Set token for user.

Parameters

- **remote** – The remote application.
- **token** – The token to set.
- **token_type** – The token type. (Default: '')
- **extra_data** – Extra information. (Default: None)
- **user** – The user owner of the remote token. If it's not defined, the current user is used automatically. (Default: None)

Returns A *invenio_oauthclient.models.RemoteToken* instance or None.

2.1.2 Models

Models for storing access tokens and links between users and remote apps.

class *invenio_oauthclient.models.RemoteAccount* (**kwargs)

Storage for remote linked accounts.

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

client_id

Client ID of remote application (defined in OAUTHCLIENT_REMOTE_APPS).

classmethod **create** (user_id, client_id, extra_data)

Create new remote account for user.

Parameters

- **user_id** – User id.
- **client_id** – Client id.
- **extra_data** – JSON-serializable dictionary of any extra data that needs to be save together with this link.

Returns A *invenio_oauthclient.models.RemoteAccount* instance.

delete()

Delete remote account together with all stored tokens.

extra_data

Extra data associated with this linked account.

classmethod **get** (user_id, client_id)

Get RemoteAccount object for user.

Parameters

- **user_id** – User id
- **client_id** – Client id.

Returns A *invenio_oauthclient.models.RemoteAccount* instance.

id

Primary key.

user

SQLAlchemy relationship to user.

user_id

Local user linked with a remote app via the access token.

class `invenio_oauthclient.models.RemoteToken` (***kwargs*)

Storage for the access tokens for linked accounts.

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

access_token

Access token to remote application.

classmethod `create` (*user_id, client_id, token, secret, token_type="", extra_data=None*)

Create a new access token.

Note: Creates RemoteAccount as well if it does not exists.

Parameters

- **user_id** – The user id.
- **client_id** – The client id.
- **token** – The token.
- **secret** – The secret key.
- **token_type** – The token type. (Default: '')
- **extra_data** – Extra data to set in the remote account if the remote account doesn't exists. (Default: None)

Returns A `invenio_oauthclient.models.RemoteToken` instance.

classmethod `get` (*user_id, client_id, token_type="", access_token=None*)

Get RemoteToken for user.

Parameters

- **user_id** – The user id.
- **client_id** – The client id.
- **token_type** – The token type. (Default: '')
- **access_token** – If set, will filter also by access token. (Default: None)

Returns A `invenio_oauthclient.models.RemoteToken` instance.

classmethod `get_by_token` (*client_id, access_token, token_type=""*)

Get RemoteAccount object for token.

Parameters

- **client_id** – The client id.
- **access_token** – The access token.

- **token_type** – The token type. (Default: '')

Returns A `invenio_oauthclient.models.RemoteToken` instance.

id_remote_account

Foreign key to account.

remote_account

SQLAlchemy relationship to RemoteAccount objects.

secret

Used only by OAuth 1.

token()

Get token as expected by Flask-OAuthlib.

token_type

Type of token.

update_token(token, secret)

Update token with new values.

Parameters

- **token** – The token value.
- **secret** – The secret key.

class `invenio_oauthclient.models.UserIdentity(**kwargs)`

Represent a UserIdentity record.

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

2.1.3 Views

Blueprints for oauthclient.

Client

Client blueprint used to handle OAuth callbacks.

`invenio_oauthclient.views.client.authorized(remote_app=None)`

Authorized handler callback.

`invenio_oauthclient.views.client.disconnect(remote_app)`

Disconnect user from remote application.

Removes application as well as associated information.

`invenio_oauthclient.views.client.login(remote_app)`

Send user to remote application for authentication.

`invenio_oauthclient.views.client.signup(remote_app)`

Extra signup step.

Settings

Account settings blueprint for oauthclient.

```
invenio_oauthclient.views.settings.index(*args, **kwargs)
```

List linked accounts.

2.1.4 Signals

Signals used together with various handlers.

```
invenio_oauthclient.signals.account_info_received = <blinker.base.NamedSignal object at 0x...
```

Signal is sent after account info handler response.

Example subscriber:

```
from invenio_oauthclient.signals import account_info_received

# During overlay initialization.
@account_info_received.connect
def load_extra_information(remote, token=None, response=None,
                           account_info=None):
    response = remote.get('https://example.org/api/resource')
    # process response
```

```
invenio_oauthclient.signals.account_setup_committed = <blinker.base.NamedSignal object at 0x...
```

Signal is sent after account setup has been committed to database.

Example subscriber:

```
from invenio_oauthclient.signals import account_setup_committed

# During overlay initialization.
@account_setup_committed.connect
def fetch_info(remote):
    response = remote.get('https://example.org/api/resource')
    # process response
```

```
invenio_oauthclient.signals.account_setup_received = <blinker.base.NamedSignal object at 0x...
```

Signal is sent after account info handler response.

Example subscriber:

```
from invenio_oauthclient.signals import account_setup_received

# During overlay initialization.
@account_setup_received.connect
def load_extra_information(remote, token=None, response=None,
                           account_setup=None):
    response = remote.get('https://example.org/api/resource')
    # process response
```

2.1.5 Utils

Utility methods to help find, authenticate or register a remote user.

`invenio_oauthclient.utils.create_csrf_disabled_registrationform()`
 Create a registration form with CSRF disabled.

`invenio_oauthclient.utils.create_registrationform(*args, **kwargs)`
 Make a registration form.

`invenio_oauthclient.utils.fill_form(form, data)`
 Prefill form with data.

Parameters

- **form** – The form to fill.
- **data** – The data to insert in the form.

Returns A pre-filled form.

`invenio_oauthclient.utils.get_safe_redirect_target(arg='next')`
 Get URL to redirect to and ensure that it is local.

Parameters **arg** – URL argument.

Returns The redirect target or None.

`invenio_oauthclient.utils.load_or_import_from_config(key, app=None, default=None)`
 Load or import value from config.

`invenio_oauthclient.utils.oauth_authenticate(client_id, user, require_existing_link=False)`
 Authenticate an oauth authorized callback.

Parameters

- **client_id** – The client id.
- **user** – A user instance.
- **require_existing_link** – If True, check if remote account exists. (Default: False)

Returns True if the user is successfully authenticated.

`invenio_oauthclient.utils.oauth_get_user(client_id, account_info=None, access_token=None)`
 Retrieve user object for the given request.

Uses either the access token or extracted account information to retrieve the user object.

Parameters

- **client_id** – The client id.
- **account_info** – The dictionary with the account info. (Default: None)
- **access_token** – The access token. (Default: None)

Returns A `invenio_accounts.models.User` instance or None.

`invenio_oauthclient.utils.oauth_link_external_id(user, external_id=None)`
 Link a user to an external id.

Parameters

- **user** – A `invenio_accounts.models.User` instance.
- **external_id** – The external id associated with the user. (Default: None)

Raises `invenio_oauthclient.errors.AlreadyLinkedError` – Raised if already exists a link.

`invenio_oauthclient.utils.oauth_register(form)`

Register user if possible.

Parameters `form` – A form instance.

Returns A `invenio_accounts.models.User` instance.

`invenio_oauthclient.utils.oauth_unlink_external_id(external_id)`

Unlink a user from an external id.

Parameters `external_id` – The external id associated with the user.

`invenio_oauthclient.utils.obj_or_import_string(value, default=None)`

Import string or return object.

`invenio_oauthclient.utils.rebuild_access_tokens(old_key)`

Rebuild the access token field when the SECRET_KEY is changed.

Fixes users' login

Parameters `old_key` – the old SECRET_KEY.

2.1.6 Errors

Module level errors.

exception `invenio_oauthclient.errors.AlreadyLinkedError(user, external_id)`

Signifies that an account was already linked to another account.

Initialize exception.

exception `invenio_oauthclient.errors.OAuthClientError(message, remote, response)`

Define OAuth client exception.

Client errors happens when the client (i.e. Invenio) creates an invalid request.

Initialize exception.

Parameters

- **message** – Error message.
- **remote** – Remote application.
- **response** – OAuth response object. Used to extract error, error_uri and error_description.

exception `invenio_oauthclient.errors.OAuthError(message, remote)`

Base class for OAuth exceptions.

Initialize exception.

Parameters

- **message** – Error message.
- **remote** – Remote application.

exception `invenio_oauthclient.errors.OAuthRejectedRequestError(message, remote, response)`

Define exception of rejected response during OAuth process.

Initialize exception.

Parameters

- **message** – Error message.
- **remote** – Remote application.
- **response** – OAuth response object.

exception `invenio_oauthclient.errors.OAuthResponseError` (*message*, *remote*, *re-*
sponse)

Define response exception during OAuth process.

Initialize exception.

Parameters

- **message** – Error message.
- **remote** – Remote application.
- **response** – OAuth response object.

Notes on how to contribute, legal information and changes are here for the interested.

3.1 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

3.1.1 Types of Contributions

Report Bugs

Report bugs at <https://github.com/inveniosoftware/invenio-oauthclient/issues>.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

Write Documentation

invenio-oauthclient could always use more documentation, whether as part of the official invenio-oauthclient docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at <https://github.com/inveniosoftware/invenio-oauthclient/issues>.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

3.1.2 Get Started!

Ready to contribute? Here's how to set up *invenio-oauthclient* for local development.

1. Fork the *inveniosoftware/invenio-oauthclient* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/invenio-oauthclient.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv invenio-oauthclient
$ cd invenio-oauthclient/
$ pip install -e .[all]
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass tests:

```
$ ./run-tests.sh
```

The tests will provide you with test coverage and also check PEP8 (code style), PEP257 (documentation), flake8 as well as build the Sphinx documentation and run doctests.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -s
  -m "component: title without verbs"
  -m "* NEW Adds your new feature."
  -m "* FIX Fixes an existing issue."
  -m "* BETTER Improves and existing feature."
  -m "* Changes something that should not be visible in release notes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

3.1.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests and must not decrease test coverage.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring.
3. The pull request should work for Python 2.7, 3.3, 3.4 and 3.5. Check https://travis-ci.com/inveniosoftware/invenio-oauthclient/pull_requests and make sure that the tests pass for all supported Python versions.

3.2 Changes

Version 1.1.0 (released 2018-12-14)

Version 1.0.0 (released 2018-03-23)

- Initial public release.

3.3 License

MIT License

Copyright (C) 2015-2018 CERN. Copyright (C) 2018 University of Chicago.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Note: In applying this license, CERN does not waive the privileges and immunities granted to it by virtue of its status as an Intergovernmental Organization or submit itself to any jurisdiction.

3.4 Contributors

- Alizee Pace
- Bruno Cuc
- Chiara Bigarella
- Diego Rodriguez

- Dinika Saxena
- Dinos Kousidis
- Harri Hirvonsalo
- Harris Tzovanakis
- Ioannis Tsanaktsidis
- Jacopo Notarstefano
- Javier Delgado
- Javier Martin Montull
- Jiri Kuncar
- Krzysztof Nowak
- Lars Holm Nielsen
- Leonardo Rossi
- Ludmila Marian
- Nicola Tarocco
- Nicolas Harraudeau
- Nikos Filippakis
- Orestis Melkonian
- Pamfilos Fokianos
- Panos Paparrigopoulos
- Paulina Lach
- Sami Hiltunen
- Tibor Simko
- Zacharias Zacharodimos

CHAPTER 4

Indices and tables

- `genindex`
- `modindex`
- `search`

i

- `invenio_oauthclient`, [11](#)
- `invenio_oauthclient.config`, [5](#)
- `invenio_oauthclient.contrib.cern`, [10](#)
- `invenio_oauthclient.contrib.github`, [8](#)
- `invenio_oauthclient.contrib.globus`, [10](#)
- `invenio_oauthclient.contrib.orcid`, [9](#)
- `invenio_oauthclient.errors`, [23](#)
- `invenio_oauthclient.handlers`, [15](#)
- `invenio_oauthclient.models`, [18](#)
- `invenio_oauthclient.signals`, [21](#)
- `invenio_oauthclient.utils`, [21](#)
- `invenio_oauthclient.views`, [20](#)
- `invenio_oauthclient.views.client`, [20](#)
- `invenio_oauthclient.views.settings`, [21](#)

A

access_token (invenio_oauthclient.models.RemoteToken attribute), 19

account_info_received (in module invenio_oauthclient.signals), 21

account_setup_committed (in module invenio_oauthclient.signals), 21

account_setup_received (in module invenio_oauthclient.signals), 21

AlreadyLinkedError, 23

authorized() (in module invenio_oauthclient.views.client), 20

authorized_default_handler() (in module invenio_oauthclient.handlers), 15

authorized_signup_handler() (in module invenio_oauthclient.handlers), 15

C

client_id (invenio_oauthclient.models.RemoteAccount attribute), 18

create() (invenio_oauthclient.models.RemoteAccount class method), 18

create() (invenio_oauthclient.models.RemoteToken class method), 19

create_csrf_disabled_registrationform() (in module invenio_oauthclient.utils), 21

create_registrationform() (in module invenio_oauthclient.utils), 22

D

delete() (invenio_oauthclient.models.RemoteAccount method), 18

disconnect() (in module invenio_oauthclient.views.client), 20

disconnect_handler() (in module invenio_oauthclient.handlers), 15

E

extra_data (invenio_oauthclient.models.RemoteAccount attribute), 18

F

fill_form() (in module invenio_oauthclient.utils), 22

G

get() (invenio_oauthclient.models.RemoteAccount class method), 18

get() (invenio_oauthclient.models.RemoteToken class method), 19

get_by_token() (invenio_oauthclient.models.RemoteToken class method), 19

get_safe_redirect_target() (in module invenio_oauthclient.utils), 22

get_session_next_url() (in module invenio_oauthclient.handlers), 16

I

id (invenio_oauthclient.models.RemoteAccount attribute), 18

id_remote_account (invenio_oauthclient.models.RemoteToken attribute), 20

index() (in module invenio_oauthclient.views.settings), 21

invenio_oauthclient (module), 11

invenio_oauthclient.config (module), 5

invenio_oauthclient.contrib.cern (module), 10

invenio_oauthclient.contrib.github (module), 8

invenio_oauthclient.contrib.globus (module), 10

invenio_oauthclient.contrib.orcid (module), 9

invenio_oauthclient.errors (module), 23

invenio_oauthclient.handlers (module), 15

invenio_oauthclient.models (module), 18

invenio_oauthclient.signals (module), 21

invenio_oauthclient.utils (module), 21

invenio_oauthclient.views (module), 20

invenio_oauthclient.views.client (module), 20

invenio_oauthclient.views.settings (module), 21

L

load_or_import_from_config() (in module invenio_oauthclient.config), 22

nio_oauthclient.utils), 22
login() (in module invenio_oauthclient.views.client), 20

M

make_handler() (in module invenio_oauthclient.handlers), 16
make_token_getter() (in module invenio_oauthclient.handlers), 16

O

oauth1_token_setter() (in module invenio_oauthclient.handlers), 16
oauth2_handle_error() (in module invenio_oauthclient.handlers), 16
oauth2_token_setter() (in module invenio_oauthclient.handlers), 16
oauth_authenticate() (in module invenio_oauthclient.utils), 22
oauth_error_handler() (in module invenio_oauthclient.handlers), 16
oauth_get_user() (in module invenio_oauthclient.utils), 22
oauth_link_external_id() (in module invenio_oauthclient.utils), 22
oauth_logout_handler() (in module invenio_oauthclient.handlers), 16
oauth_register() (in module invenio_oauthclient.utils), 23
oauth_unlink_external_id() (in module invenio_oauthclient.utils), 23
OAuthClientError, 23
OAuthError, 23
OAuthRejectedRequestError, 23
OAuthResponseError, 24
obj_or_import_string() (in module invenio_oauthclient.utils), 23

R

rebuild_access_tokens() (in module invenio_oauthclient.utils), 23
remote_account (invenio_oauthclient.models.RemoteToken attribute), 20
RemoteAccount (class in invenio_oauthclient.models), 18
RemoteToken (class in invenio_oauthclient.models), 19
response_token_setter() (in module invenio_oauthclient.handlers), 16

S

secret (invenio_oauthclient.models.RemoteToken attribute), 20
set_session_next_url() (in module invenio_oauthclient.handlers), 17
signup() (in module invenio_oauthclient.views.client), 20
signup_handler() (in module invenio_oauthclient.handlers), 17

T

token() (invenio_oauthclient.models.RemoteToken method), 20
token_delete() (in module invenio_oauthclient.handlers), 17
token_getter() (in module invenio_oauthclient.handlers), 17
token_session_key() (in module invenio_oauthclient.handlers), 17
token_setter() (in module invenio_oauthclient.handlers), 17
token_type (invenio_oauthclient.models.RemoteToken attribute), 20

U

update_token() (invenio_oauthclient.models.RemoteToken method), 20
user (invenio_oauthclient.models.RemoteAccount attribute), 18
user_id (invenio_oauthclient.models.RemoteAccount attribute), 19
UserIdentity (class in invenio_oauthclient.models), 20