Python Keycloak Client Documentation

Release 0.2.4-dev

Peter Slump

Apr 05, 2020

Contents

The Python Keycloak Client is a set of API clients written in Python to communicate with the different API's which are exposed by Keycloak.

CHAPTER 1

Installation

```
$ pip install python-keycloak-client
```

1.1 Async

```
$ pip install python-keycloak-client[aio]
```

CHAPTER 2

Preparation

Make sure you have created a REALM and Client in Keycloak.

CHAPTER 3

Usage

Everything starts with an instance of `keycloak.realm.KeycloakRealm`

```
from keycloak.realm import KeycloakRealm


realm = KeycloakRealm(server_url='https://example.com', realm_name='my_realm')
```

```
from keycloak.aio.realm import KeycloakRealm


async def main(loop=None):
    realm_params = dict(
        server_url='https://example.com',
        realm_name='my_realm',
        loop=loop
    )
    async with KeycloakRealm(**realm_params) as realm:
        # do something
        print(realm.realm_name)

if __name__ == '__main__':
    import asyncio

    loop = asyncio.get_event_loop()
    loop.run_until_complete(main(loop))
```

3.1 OpenID Connect

The OpenID Connect entry point can be retrieved from the realm object.

```
from keycloak.realm import KeycloakRealm


realm = KeycloakRealm(server_url='https://example.com', realm_name='my_realm')

oidc_client = realm.open_id_connect(client_id='my-client',
                                    client_secret='very-secret-client-secret')
```

3.1.1 Async

```
from keycloak.aio.realm import KeycloakRealm


async def main(loop=None):
    realm_params = dict(
        server_url='https://example.com',
        realm_name='my_realm',
        loop=loop
    )
    async with KeycloakRealm(**realm_params) as realm:
        oidc_client = await realm.open_id_connect(
            client_id='my-client',
            client_secret='very-secret-client-secret'
        )
        # do something


if __name__ == '__main__':
    import asyncio

    loop = asyncio.get_event_loop()
    loop.run_until_complete(main(loop))
```

KeycloakOpenidConnect.**decode_token**(*token*, *key*, *algorithms=None*, ***kwargs*)

A JSON Web Key (JWK) is a JavaScript Object Notation (JSON) data structure that represents a cryptographic key. This specification also defines a JWK Set JSON data structure that represents a set of JWKs. Cryptographic algorithms and identifiers for use with this specification are described in the separate JSON Web Algorithms (JWA) specification and IANA registries established by that specification.

https://tools.ietf.org/html/rfc7517

Parameters

- **token** (*str*) – A signed JWS to be verified.
- **key** (*str*) – A key to attempt to verify the payload with.
- **algorithms** (*str, list*) – (optional) Valid algorithms that should be used to verify the JWS. Defaults to *['RS256']*
- **audience** (*str*) – (optional) The intended audience of the token. If the "aud" claim is included in the claim set, then the audience must be included and must equal the provided claim.
- **issuer** (*str, iterable*) – (optional) Acceptable value(s) for the issuer of the token. If the "iss" claim is included in the claim set, then the issuer must be given and the claim in the token must be among the acceptable values.

- **subject** (*str*) – (optional) The subject of the token. If the "sub" claim is included in the claim set, then the subject must be included and must equal the provided claim.
- **access_token** (*str*) – (optional) An access token returned alongside the id_token during the authorization grant flow. If the "at_hash" claim is included in the claim set, then the access_token must be included, and it must match the "at_hash" claim.
- **options** (*dict*) – (optional) A dictionary of options for skipping validation steps. defaults:

```
{
    'verify_signature': True,
    'verify_aud': True,
    'verify_iat': True,
    'verify_exp': True,
    'verify_nbf': True,
    'verify_iss': True,
    'verify_sub': True,
    'verify_jti': True,
    'leeway': 0,
}
```

Returns The dict representation of the claims set, assuming the signature is valid and all requested data validation passes.

Return type dict

Raises

- **jose.exceptions.JWTError** – If the signature is invalid in any way.
- **jose.exceptions.ExpiredSignatureError** – If the signature has expired.
- **jose.exceptions.JWTClaimsError** – If any claim is invalid in any way.

KeycloakOpenidConnect.**authorization_url**(***kwargs*)

Get authorization URL to redirect the resource owner to.

https://tools.ietf.org/html/rfc6749#section-4.1.1

Parameters

- **redirect_uri** (*str*) – (optional) Absolute URL of the client where the user-agent will be redirected to.
- **scope** (*str*) – (optional) Space delimited list of strings.
- **state** (*str*) – (optional) An opaque value used by the client to maintain state between the request and callback

Returns URL to redirect the resource owner to

Return type str

KeycloakOpenidConnect.**authorization_code**(*code*, *redirect_uri*)

Retrieve access token by *authorization_code* grant.

https://tools.ietf.org/html/rfc6749#section-4.1.3

Parameters

- **code** (*str*) – The authorization code received from the authorization server.
- **redirect_uri** (*str*) – the identical value of the "redirect_uri" parameter in the authorization request.

Return type dict

Returns Access token response

KeycloakOpenidConnect.**client_credentials**(***kwargs*)
Retrieve access token by *client_credentials* grant.

https://tools.ietf.org/html/rfc6749#section-4.4

Parameters **scope** (*str*) – (optional) Space delimited list of strings.

Return type dict

Returns Access token response

KeycloakOpenidConnect.**refresh_token**(*refresh_token*, ***kwargs*)
Refresh an access token

https://tools.ietf.org/html/rfc6749#section-6

Parameters

- **refresh_token** (*str*) –
- **scope** (*str*) – (optional) Space delimited list of strings.

Return type dict

Returns Access token response

KeycloakOpenidConnect.**logout**(*refresh_token*)
The logout endpoint logs out the authenticated user.

Parameters **refresh_token** (*str*) –

KeycloakOpenidConnect.**certs**()
The certificate endpoint returns the public keys enabled by the realm, encoded as a JSON Web Key (JWK). Depending on the realm settings there can be one or more keys enabled for verifying tokens.

https://tools.ietf.org/html/rfc7517

Return type dict

KeycloakOpenidConnect.**userinfo**(*token*)
The UserInfo Endpoint is an OAuth 2.0 Protected Resource that returns Claims about the authenticated End-User. To obtain the requested Claims about the End-User, the Client makes a request to the UserInfo Endpoint using an Access Token obtained through OpenID Connect Authentication. These Claims are normally represented by a JSON object that contains a collection of name and value pairs for the Claims.

http://openid.net/specs/openid-connect-core-1_0.html#UserInfo

Parameters **token** (*str*) –

Return type dict

KeycloakOpenidConnect.**token_exchange**(***kwargs*)
Token exchange is the process of using a set of credentials or token to obtain an entirely different token.

http://www.keycloak.org/docs/latest/securing_apps/index.html #_token-exchange https://www.ietf.org/id/draft-ietf-oauth-token-exchange-12.txt

Parameters

- **subject_token** – A security token that represents the identity of the party on behalf of whom the request is being made. It is required if you are exchanging an existing token for a new one.

- **subject_issuer** – Identifies the issuer of the subject_token. It can be left blank if the token comes from the current realm or if the issuer can be determined from the subject_token_type. Otherwise it is required to be specified. Valid values are the alias of an Identity Provider configured for your realm. Or an issuer claim identifier configured by a specific Identity Provider.
- **subject_token_type** – This parameter is the type of the token passed with the subject_token parameter. This defaults to urn:ietf:params:oauth:token-type:access_token if the subject_token comes from the realm and is an access token. If it is an external token, this parameter may or may not have to be specified depending on the requirements of the subject_issuer.
- **requested_token_type** – This parameter represents the type of token the client wants to exchange for. Currently only oauth and OpenID Connect token types are supported. The default value for this depends on whether the is urn:ietf:params:oauth:token-type:refresh_token in which case you will be returned both an access token and refresh token within the response. Other appropriate values are urn:ietf:params:oauth:token-type:access_token and urn:ietf:params:oauth:token-type:id_token
- **audience** – This parameter specifies the target client you want the new token minted for.
- **requested_issuer** – This parameter specifies that the client wants a token minted by an external provider. It must be the alias of an Identity Provider configured within the realm.
- **requested_subject** – This specifies a username or user id if your client wants to impersonate a different user.

Return type dict

Returns access_token, refresh_token and expires_in

3.2 Authz (Authorization services)

The Authz client can be retrieved from the realm object.

```
from keycloak.realm import KeycloakRealm


realm = KeycloakRealm(server_url='https://example.com', realm_name='my_realm')

authz_client = realm.authz(client_id='my-client')
```

3.2.1 Async

```
from keycloak.aio.realm import KeycloakRealm


async def main(loop=None):
    realm_params = dict(
        server_url='https://example.com',
        realm_name='my_realm',
        loop=loop
    )
    async with KeycloakRealm(**realm_params) as realm:
        authz_client = await realm.authz(client_id='my-client')
```

(continues on next page)

(continued from previous page)

```
        # do something


if __name__ == '__main__':
    import asyncio

    loop = asyncio.get_event_loop()
    loop.run_until_complete(main(loop))
```

KeycloakAuthz.**entitlement**(*token*)

Client applications can use a specific endpoint to obtain a special security token called a requesting party token (RPT). This token consists of all the entitlements (or permissions) for a user as a result of the evaluation of the permissions and authorization policies associated with the resources being requested. With an RPT, client applications can gain access to protected resources at the resource server.

http://www.keycloak.org/docs/latest/authorization_services/index .html#_service_entitlement_api

Return type dict

3.3 Admin API

Manage Realms, Clients, Roles, Users etc.

http://www.keycloak.org/docs-api/3.4/rest-api/index.html

The admin API client get be retrieved from the realm object.

```
from keycloak.realm import KeycloakRealm


realm = KeycloakRealm(server_url='https://example.com', realm_name='my_realm')

admin_client = realm.admin
```

3.3.1 Async

```
from keycloak.aio.realm import KeycloakRealm


async def main(loop=None):
    realm_params = dict(
        server_url='https://example.com',
        realm_name='my_realm',
        loop=loop
    )
    async with KeycloakRealm(**realm_params) as realm:
        admin_client = realm.admin
        # do something


if __name__ == '__main__':
    import asyncio
```

(continues on next page)

(continued from previous page)

```
    loop = asyncio.get_event_loop()
    loop.run_until_complete(main(loop))
```

3.3.2 Realms

Currently there is no actual functionality available for Realm management. However this endpoint is the entrypoint for all other clients.

```
realm = realm.admin.realms.by_name('realm-name')
```

3.3.3 Clients

Manage clients

```
clients = realm.admin.realms.by_name('realm-name').clients
```

The following methods can be accessed:

`Clients.`**`all`**`()`

3.3.4 Roles

Manage client roles

```
roles = realm.admin.realms.by_name('realm-name').clients.by_id('#client id').roles
```

The following methods are available:

Actions on a specific role

```
role = realm.admin.realms.by_name('realm-name').clients.by_id('#client id').roles.by_
↪name('role-name')
```

The following methods are available:

3.3.5 Users

Manage users in a REALM

```
users = realm.admin.realms.by_name('realm-name').users
```

The following methods are available:

`Users.`**`create`**`(`*username*, ***kwargs*`)`

Create a user in Keycloak

http://www.keycloak.org/docs-api/3.4/rest-api/index.html#_users_resource

Parameters

- **`username`** (*`str`*) –
- **`credentials`** (*`object`*) – (optional)

- **first_name** (*str*) – (optional)
- **last_name** (*str*) – (optional)
- **email** (*str*) – (optional)
- **enabled** (*boolean*) – (optional)

3.4 UMA (User-Managed Access)

The UMA client can be retrieved from the realm object.

http://www.keycloak.org/docs/latest/authorization_services/index.html#_service_overview

```
from keycloak.realm import KeycloakRealm


realm = KeycloakRealm(server_url='https://example.com', realm_name='my_realm')

uma_client = realm.uma()
```

3.4.1 Async

```
from keycloak.aio.realm import KeycloakRealm


async def main(loop=None):
    realm_params = dict(
        server_url='https://example.com',
        realm_name='my_realm',
        loop=loop
    )
    async with KeycloakRealm(**realm_params) as realm:
        uma_client = realm.uma()
        # do something


if __name__ == '__main__':
    import asyncio

    loop = asyncio.get_event_loop()
    loop.run_until_complete(main(loop))
```

3.4.2 Resource Set management

KeycloakUMA.**resource_set_create**(*token*, *name*, ***kwargs*)

Create a resource set.

https://docs.kantarainitiative.org/uma/rec-oauth-resource-reg-v1_0_1.html#rfc.section.2.2.1

Parameters

- **token** (*str*) – client access token
- **id** (*str*) – Identifier of the resource set

- **name** (*str*) –
- **uri** (*str*) – (optional)
- **type** (*str*) – (optional)
- **scopes** (*list*) – (optional)
- **icon_uri** (*str*) – (optional)
- **DisplayName** (*str*) – (optional)
- **ownerManagedAccess** (*boolean*) – (optional)
- **owner** (*str*) – (optional)

Return type str

KeycloakUMA.**resource_set_update**(*token*, *id*, *name*, ***kwargs*)

Update a resource set.

https://docs.kantarainitiative.org/uma/rec-oauth-resource-reg-v1_0_1.html#update-resource-set

Parameters

- **token** (*str*) – client access token
- **id** (*str*) – Identifier of the resource set
- **name** (*str*) –
- **uri** (*str*) – (optional)
- **type** (*str*) – (optional)
- **scopes** (*list*) – (optional)
- **icon_uri** (*str*) – (optional)

Return type str

KeycloakUMA.**resource_set_read**(*token*, *id*)

Read a resource set.

https://docs.kantarainitiative.org/uma/rec-oauth-resource-reg-v1_0_1.html#read-resource-set

Parameters

- **token** (*str*) – client access token
- **id** (*str*) – Identifier of the resource set

Return type dict

KeycloakUMA.**resource_set_delete**(*token*, *id*)

Delete a resource set.

https://docs.kantarainitiative.org/uma/rec-oauth-resource-reg-v1_0_1.html#delete-resource-set

Parameters

- **token** (*str*) – client access token
- **id** (*str*) – Identifier of the resource set

KeycloakUMA.**resource_set_list**(*token*, ***kwargs*)

List a resource set.

https://docs.kantarainitiative.org/uma/rec-oauth-resource-reg-v1_0_1.html#list-resource-sets

Parameters

- **token** (*str*) – client access token
- **name** (*str*) – (optional)
- **uri** (*str*) – (optional)
- **owner** (*str*) – (optional)
- **type** (*str*) – (optional)
- **scope** (*str*) – (optional)

Return type list

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index