python-keycloak Documentation

Release 0.19.0

Marcos Pereira

May 19, 2022

Contents:

1 Welcome to python-keycloak's documentation! 3

2 Installation 5

3 Dependencies 7
3.1 Tests Dependencies 7

4 Bug reports 9

5 Documentation 11

6 Contributors 13

7 Usage 15

- genindex
- modindex
- search

CHAPTER 1

Welcome to python-keycloak's documentation!

python-keycloak is a Python package providing access to the Keycloak API.

CHAPTER 2

Installation

Via Pypi Package:

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

Manually:

```
$ python setup.py install
```

CHAPTER 3

Dependencies

python-keycloak depends on:

- Python 3
- requests
- python-jose

3.1 Tests Dependencies

- unittest
- httmock

CHAPTER 4

Bug reports

Please report bugs and feature requests at https://github.com/marcospereirampj/python-keycloak/issues

CHAPTER 5

Documentation

The documentation for python-keycloak is available on readthedocs.

CHAPTER 6

Contributors

- Agriness Team
- Marcos Pereira
- Martin Devlin
- Shon T. Urbas
- Markus Spanier
- Remco Kranenburg
- Armin
- Njordr
- Josha Inglis
- Alex
- Ewan Jone

CHAPTER 7

Usage

Main methods:

```
# KEYCLOAK OPENID

from keycloak import KeycloakOpenID

# Configure client
keycloak_openid = KeycloakOpenID(server_url="http://localhost:8080/auth/",
                    client_id="example_client",
                    realm_name="example_realm",
                    client_secret_key="secret",
                    verify=True)

# Optionally, you can pass custom headers that will be added to all HTTP calls
# keycloak_openid = KeycloakOpenID(server_url="http://localhost:8080/auth/",
#                     client_id="example_client",
#                     realm_name="example_realm",
#                     client_secret_key="secret",
#                     verify=True,
#                     custom_headers={'CustomHeader': 'value'})

# Get WellKnow
config_well_know = keycloak_openid.well_know()

# Get Token
token = keycloak_openid.token("user", "password")
token = keycloak_openid.token("user", "password", totp="012345")

# Get Userinfo
userinfo = keycloak_openid.userinfo(token['access_token'])

# Refresh token
token = keycloak_openid.refresh_token(token['refresh_token'])
```

(continues on next page)

(continued from previous page)

```
# Logout
keycloak_openid.logout(token['refresh_token'])

# Get Certs
certs = keycloak_openid.certs()

# Get RPT (Entitlement)
token = keycloak_openid.token("user", "password")
rpt = keycloak_openid.entitlement(token['access_token'], "resource_id")

# Instropect RPT
token_rpt_info = keycloak_openid.introspect(keycloak_openid.introspect(token['access_
↪token'], rpt=rpt['rpt'],
                                        token_type_hint="requesting_party_token"))

# Introspect Token
token_info = keycloak_openid.introspect(token['access_token']))

# Decode Token
KEYCLOAK_PUBLIC_KEY = "secret"
options = {"verify_signature": True, "verify_aud": True, "exp": True}
token_info = keycloak_openid.decode_token(token['access_token'], key=KEYCLOAK_PUBLIC_
↪KEY, options=options)

# Get permissions by token
token = keycloak_openid.token("user", "password")
keycloak_openid.load_authorization_config("example-authz-config.json")
policies = keycloak_openid.get_policies(token['access_token'], method_token_info=
↪'decode', key=KEYCLOAK_PUBLIC_KEY)
permissions = keycloak_openid.get_permissions(token['access_token'], method_token_
↪info='introspect')

# KEYCLOAK ADMIN

from keycloak import KeycloakAdmin

keycloak_admin = KeycloakAdmin(server_url="http://localhost:8080/auth/",
                               username='example-admin',
                               password='secret',
                               realm_name="example_realm",
                               verify=True)

# Optionally, you can pass custom headers that will be added to all HTTP calls
#keycloak_admin = KeycloakAdmin(server_url="http://localhost:8080/auth/",
#                               username='example-admin',
#                               password='secret',
#                               realm_name="example_realm",
#                               verify=True,
#                               custom_headers={'CustomHeader': 'value'})

# Add user
new_user = keycloak_admin.create_user({"email": "example@example.com",
                    "username": "example@example.com",
                    "enabled": True,
                    "firstName": "Example",
                    "lastName": "Example",
                    "realmRoles": ["user_default", ],
```

(continues on next page)

(continued from previous page)

```
                    "attributes": {"example": "1,2,3,3,"}})


# Add user and set password
new_user = keycloak_admin.create_user({"email": "example@example.com",
                    "username": "example@example.com",
                    "enabled": True,
                    "firstName": "Example",
                    "lastName": "Example",
                    "credentials": [{"value": "secret","type": "password",}],
                    "realmRoles": ["user_default", ],
                    "attributes": {"example": "1,2,3,3,"}})

# User counter
count_users = keycloak_admin.users_count()

# Get users Returns a list of users, filtered according to query parameters
users = keycloak_admin.get_users({})

# Get user ID from name
user-id-keycloak = keycloak_admin.get_user_id("example@example.com")

# Get User
user = keycloak_admin.get_user("user-id-keycloak")

# Update User
response = keycloak_admin.update_user(user_id="user-id-keycloak",
                                      payload={'firstName': 'Example Update'})

# Update User Password
response = set_user_password(user_id="user-id-keycloak", password="secret",␣
↪temporary=True)

# Delete User
response = keycloak_admin.delete_user(user_id="user-id-keycloak")

# Get consents granted by the user
consents = keycloak_admin.consents_user(user_id="user-id-keycloak")

# Send User Action
response = keycloak_admin.send_update_account(user_id="user-id-keycloak",
                                              payload=json.dumps(['UPDATE_PASSWORD']))

# Send Verify Email
response = keycloak_admin.send_verify_email(user_id="user-id-keycloak")

# Get sessions associated with the user
sessions = keycloak_admin.get_sessions(user_id="user-id-keycloak")

# Get themes, social providers, auth providers, and event listeners available on this␣
↪server
server_info = keycloak_admin.get_server_info()

# Get clients belonging to the realm Returns a list of clients belonging to the realm
clients = keycloak_admin.get_clients()

# Get client - id (not client-id) from client by name
```

(continues on next page)

(continued from previous page)

```
client_id=keycloak_admin.get_client_id("my-client")

# Get representation of the client - id of client (not client-id)
client = keycloak_admin.get_client(client_id="client_id")

# Get all roles for the realm or client
realm_roles = keycloak_admin.get_realm_roles()

# Get all roles for the client
client_roles = keycloak_admin.get_client_roles(client_id="client_id")

# Get client role
role = keycloak_admin.get_client_role(client_id="client_id", role_name="role_name")

# Warning: Deprecated
# Get client role id from name
role_id = keycloak_admin.get_client_role_id(client_id="client_id", role_name="test")

# Create client role
keycloak_admin.create_client_role(client_id="client_id", {'name': 'roleName',
↪'clientRole': True})

# Get client role id from name
role_id = keycloak_admin.get_client_role_id(client_id=client_id, role_name="test")

# Get all roles for the realm or client
realm_roles = keycloak_admin.get_roles()

# Assign client role to user. Note that BOTH role_name and role_id appear to be␣
↪required.
keycloak_admin.assign_client_role(client_id="client_id", user_id="user_id", role_id=
↪"role_id", role_name="test")

# Assign realm roles to user. Note that BOTH role_name and role_id appear to be␣
↪required.
keycloak_admin.assign_realm_roles(client_id="client_id", user_id="user_id", roles=[{
↪"roles_representation"}])

# Create new group
group = keycloak_admin.create_group(name="Example Group")

# Get all groups
groups = keycloak_admin.get_groups()

# Get group
group = keycloak_admin.get_group(group_id='group_id')

# Get group by path
group = keycloak_admin.get_group_by_path(path='/group/subgroup', search_in_
↪subgroups=True)

# Function to trigger user sync from provider
sync_users(storage_id="storage_di", action="action")
```