keystoneauth1 Documentation

Release 5.9.2.dev27

Openstack Developers

Feb 18, 2025

CONTENTS

CHAPTER ONE

USING SESSIONS

1.1 Introduction

The `keystoneauth1.session.Session` class was introduced into keystoneauth1 as an attempt to bring a unified interface to the various OpenStack clients that share common authentication and request parameters between a variety of services.

The model for using a Session and auth plugin as well as the general terms used have been heavily inspired by the requests library. However neither the Session class nor any of the authentication plugins rely directly on those concepts from the requests library so you should not expect a direct translation.

1.1.1 Features

- Common client authentication

  Authentication is handled by one of a variety of authentication plugins and then this authentication information is shared between all the services that use the same Session object.

- Security maintenance

  Security code is maintained in a single place and reused between all clients such that in the event of problems it can be fixed in a single location.

- Standard service and version discovery

  Clients are not expected to have any knowledge of an identity token or any other form of identification credential. Service, endpoint, major version discovery, and microversion support discovery are handled by the Session and plugins. Discovery information is automatically cached in memory, so the user need not worry about excessive use of discovery metadata.

- Safe logging of HTTP interactions

  Clients need to be able to enable logging of the HTTP interactions, but some things, such as the token or secrets, need to be ommitted.

1.2 Sessions for Users

The Session object is the contact point to your OpenStack cloud services. It stores the authentication credentials and connection information required to communicate with OpenStack such that it can be reused to communicate with many services. When creating services this Session object is passed to the client so that it may use this information.

A Session will authenticate on demand. When a request that requires authentication passes through the Session the authentication plugin will be asked for a valid token. If a valid token is available it will be used otherwise the authentication plugin may attempt to contact the authentication service and fetch a new one.

An example using keystoneclient to wrap a Session:

```
>>> from keystoneauth1.identity import v3
>>> from keystoneauth1 import session
>>> from keystoneclient.v3 import client

>>> auth = v3.Password(auth_url='https://my.keystone.com:5000/v3',
...                    username='myuser',
...                    password='mypassword',
...                    project_name='proj',
...                    user_domain_id='default',
...                    project_domain_id='default')
>>> sess = session.Session(auth=auth,
...                        verify='/path/to/ca.cert')
>>> ks = client.Client(session=sess)
>>> users = ks.users.list()
```

As other OpenStack client libraries adopt this means of operating they will be created in a similar fashion by passing the Session object to the clients constructor.

1.2.1 Sharing Authentication Plugins

A Session can only contain one authentication plugin. However, there is nothing that specifically binds the authentication plugin to that Session - a new Session can be created that reuses the existing authentication plugin:

```
>>> new_sess = session.Session(auth=sess.auth,
                               verify='/path/to/different-cas.cert')
```

In this case we cannot know which Session object will be used when the plugin performs the authentication call so the command must be able to succeed with either.

Authentication plugins can also be provided on a per-request basis. This will be beneficial in a situation where a single Session is juggling multiple authentication credentials:

```
>>> sess.get('https://my.keystone.com:5000/v3',
             auth=my_auth_plugin)
```

If an auth plugin is provided via parameter then it will override any auth plugin on the Session.

1.3 Sessions for Client Developers

Sessions are intended to take away much of the hassle of dealing with authentication data and token formats. Clients should be able to specify filter parameters for selecting the endpoint and have the parsing of the catalog managed for them.

1.3.1 Major Version Discovery and Microversion Support

In OpenStack, the root URLs of available services are distributed to the user in an object called the Service Catalog, which is part of the token they receive. Clients are expected to use the URLs from the Service Catalog rather than have them provided. The root URL of a given service is referred to as the *endpoint* of the service. The URL of a specific version of a service is referred to as a *versioned endpoint*. REST requests for a service are made against a given *versioned endpoint*.

The topic of Major API versions and microversions can be confusing. As *keystoneauth* provides facilities for discovery of versioned endpoints associated with a Major API Version and for fetching information about the microversions that versioned endpoint supports, it is important to be aware of the distinction between the two.

Conceptually the most important thing to understand is that a Major API Version describes the URL of a discrete versioned endpoint, while a given versioned endpoint might have properties that express that it supports a range of microversions.

When a user wants to make a REST request against a service, the user expresses the Major API version and the type of service so that the appropriate versioned endpoint can be found and used. For example, a user might request version 2 of the compute service from cloud.example.com and end up with a versioned endpoint of `https://compute.example.com/v2`.

Each service provides a discovery document at the root of each versioned endpoint that contains information about that versioned endpoint. Each service also provides a document at the root of the unversioned endpoint that contains a list of the discovery documents for all of the available versioned endpoints. By examining these documents, it is possible to find the versioned endpoint that corresponds with the users desired Major API version.

Each of those documents may also indicate that the given versioned endpoint supports microversions by listing a minimum and maximum microversion that it understands. As a result of having found the versioned endpoint for the requested Major API version, the user will also know which microversions, if any, may be used in requests to that versioned endpoint.

When a client makes REST requests to the Major API versions endpoint, the client can, optionally, on a request-by-request basis, include a header specifying that the individual request use the behavior defined by the given microversion. If a client does not request a microversion, the service will behave as if the minimum supported microversion was specified.

The overall transaction then has three parts:

- What is the endpoint for a given Major API version of a given service?
- What are the minimum and maximum microversions supported at that endpoint?
- Which one of that range of microversions, if any, does the user want to use for a given request?

keystoneauth provides facilities for discovering the endpoint for a given Major API of a given service, as well as reporting the available microversion ranges that endpoint supports, if any.

More information is available in the API-WG Specs on the topics of Microversions and Consuming the Catalog.

1.3.2 Authentication

When making a request with a Session object you can simply pass the keyword parameter `authenticated` to indicate whether the argument should contain a token, by default a token is included if an authentication plugin is available:

```
>>> # In keystone this route is unprotected by default
>>> resp = sess.get('https://my.keystone.com:5000/v3',
                     authenticated=False)
```

1.3.3 Service Discovery

In general a client does not need to know the full URL for the server that they are communicating with, simply that it should send a request to a path belonging to the correct service.

This is controlled by the `endpoint_filter` parameter to a request which contains all the information an authentication plugin requires to determine the correct URL to which to send a request. When using this mode only the path for the request needs to be specified:

```
>>> resp = session.get('/users',
                       endpoint_filter={'service_type': 'identity',
                                        'interface': 'admin',
                                        'region_name': 'myregion',
                                        'min_version': '2.0',
                                        'max_version': '3.4',
                                        'discover_versions': False})
```

Note

The min_version and max_version arguments in this example indicate acceptable range for finding the endpoint for the given Major API versions. They are in the endpoint_filter, they are not requesting the call to `/users` be made at a specific microversion.

endpoint_filter accepts a number of arguments with which it can determine an endpoint url:

service_type
: the type of service. For example `identity`, `compute`, `volume` or many other predefined identifiers.

interface
: the network exposure the interface has. Can also be a list, in which case the first matching interface will be used. Valid values are:

  - `public`: An endpoint that is available to the wider internet or network.
  - `internal`: An endpoint that is only accessible within the private network.
  - `admin`: An endpoint to be used for administrative tasks.

region_name
: the name of the region where the endpoint resides.

version
: the minimum version, restricted to a given Major API. For instance, a *version* of `2.2` will match `2.2` and `2.3` but not `2.1` or `3.0`. Mutually exclusive with *min_version* and *max_version*.

min_version
: the minimum version of a given API, intended to be used as the lower bound of a range with *max_version*. See *max_version* for examples. Mutually exclusive with *version*.

max_version

the maximum version of a given API, intended to be used as the upper bound of a range with *min_version*. For example:

```
'min_version': '2.2',
'max_version': '3.3'
```

will match `2.2`, `2.10`, `3.0`, and `3.3`, but not `1.42`, `2.1`, or `3.20`. Mutually exclusive with *version*.

> **Note**
>
> version, min_version and max_version are all used to help determine the endpoint for a given Major API version of a service.

discover_versions

whether or not version discovery should be run, even if not strictly necessary. It is often possible to fulfill an endpoint request purely from the catalog, meaning the version discovery API is a potentially wasted additional call. However, its possible that running discovery instead of inference is desired. Defaults to `True`.

All version arguments (*version*, *min_version* and *max_version*) can be given as:

- string: `'2.0'`
- int: `2`
- float: `2.0`
- tuple of ints: `(2, 0)`

version and *max_version* can also be given the string `latest`, which indicates that the highest available version should be used.

The endpoint filter is a simple key-value filter and can be provided with any number of arguments. It is then up to the auth plugin to correctly use the parameters it understands.

If you want to further limit your service discovery by allowing experimental APIs or disallowing deprecated APIs, you can use the `allow` parameter:

```
>>> resp = session.get('/<project-id>/volumes',
                       endpoint_filter={'service_type': 'volume',
                                        'interface': 'public',
                                        'version': 1},
                       allow={'allow_deprecated': False})
```

The discoverable types of endpoints that *allow* can recognize are:

- *allow_deprecated*: Allow deprecated version endpoints.
- *allow_experimental*: Allow experimental version endpoints.
- *allow_unknown*: Allow endpoints with an unrecognised status.

The Session object creates a valid request by determining the URL matching the filters and appending it to the provided path. If multiple URL matches are found then any one may be chosen.

While authentication plugins will endeavour to maintain a consistent set of arguments for an `endpoint_filter` the concept of an authentication plugin is purposefully generic. A specific mechanism may not know how to interpret certain arguments in which case it may ignore them. For example the `keystoneauth1.token_endpoint.Token` plugin (which is used when you want to always use a specific endpoint and token combination) will always return the same endpoint regardless of the parameters to `endpoint_filter` or a custom OpenStack authentication mechanism may not have the concept of multiple `interface` options and choose to ignore that parameter.

There is some expectation on the user that they understand the limitations of the authentication system they are using.

1.3.4 Using Adapters

If the developer would prefer not to provide *endpoint_filter* with every API call, a `keystoneauth1.adapter.Adapter` can be created. The *Adapter* constructor takes the same arguments as *endpoint_filter*, as well as a *Session*. An *Adapter* behaves much like a *Session*, with the same REST methods, but is mounted on the endpoint that would be found by *endpoint_filter*.

```
adapter = keystoneauth1.adapter.Adapter(
    session=session,
    service_type='volume',
    interface='public',
    version=1)
response = adapter.get('/volumes')
```

As with `endpoint_filter` on a Session, the `version`, `min_version` and `max_version` parameters exist to help determine the appropriate endpoint for a Major API of a service.

1.3.5 Endpoint Metadata

Both `keystoneauth1.adapter.Adapter` and `keystoneauth1.session.Session` have a method for getting metadata about the endpoint found for a given service: `get_endpoint_data`.

On the `keystoneauth1.session.Session` it takes the same arguments as *endpoint_filter*.

On the `keystoneauth1.adapter.Adapter` it does not take arguments, as it returns the information for the Endpoint the Adapter is mounted on.

`get_endpoint_data` returns an `keystoneauth1.discovery.EndpointData` object. This object can be used to find information about the Endpoint, including which major *api_version* was found, or which *interface* in case of ranges, lists of input values or `latest` version.

It can also be used to determine the *min_microversion* and *max_microversion* supported by the API. If an API does not support microversions, the values for both will be `None`. It will also contain values for *next_min_version* and *not_before* if they exist for the endpoint, or `None` if they do not. The `keystoneauth1.discovery.EndpointData` object will always contain microversion related attributes regardless of whether the REST document does or not.

`get_endpoint_data` makes use of the same cache as the rest of the discovery process, so calling it should incur no undue expense. By default it will make at least one version discovery call so that it can fetch microversion metadata. If the user knows a service does not support microversions and is merely curious as to which major version was discovered, `discover_versions` can be set to `False` to prevent fetching microversion metadata.

1.3.6 Requesting a Microversion

A user who wants to specify a microversion for a given request can pass it to the `microversion` parameter of the *request* method on the `keystoneauth1.session.Session` object, or the `keystoneauth1.adapter.Adapter` object. This will cause *keystoneauth* to pass the appropriate header to the service informing the service of the microversion the user wants.

```
resp = session.get('/volumes',
                   microversion='3.15',
                   endpoint_filter={'service_type': 'volume',
                                    'interface': 'public',
                                    'min_version': '3',
                                    'max_version': 'latest'})
```

If the user is using a `keystoneauth1.adapter.Adapter`, the *service_type*, which is a part of the data sent in the microversion header, will be taken from the Adapters *service_type*.

```
adapter = keystoneauth1.adapter.Adapter(
    session=session,
    service_type='compute',
    interface='public',
    min_version='2.1')
response = adapter.get('/servers', microversion='2.38')
```

The user can also provide a `default_microversion` parameter to the Adapter constructor which will be used on all requests where an explicit microversion is not requested.

```
adapter = keystoneauth1.adapter.Adapter(
    session=session,
    service_type='compute',
    interface='public',
    min_version='2.1',
    default_microversion='2.38')
response = adapter.get('/servers')
```

If the user is using a `keystoneauth1.session.Session`, the *service_type* will be taken from the *service_type* in *endpoint_filter*.

If the *service_type* is the incorrect value to use for the microversion header for the service in question, the parameter *microversion_service_type* can be given. For instance, although keystoneauth already knows about Cinder, the *service_type* for Cinder is `block-storage` but the microversion header expects `volume`.

```
# Interactions with cinder do not need to explicitly override the
# microversion_service_type - it is only being used as an example for the
# use of the parameter.
resp = session.get('/volumes',
                   microversion='3.15',
                   microversion_service_type='volume',
                   endpoint_filter={'service_type': 'block-storage',
                                    'interface': 'public',
                                    'min_version': '3',
                                    'max_version': 'latest'})
```

1.4 Logging

The logging system uses standard python logging rooted on the `keystoneauth` namespace as would be expected. There are two possibilities of where log messages about HTTP interactions will go.

By default, all messages will go to the `keystoneauth.session` logger.

If the `split_loggers` option on the `keystoneauth1.session.Session` constructor is set to `True`, the HTTP content will be split across four subloggers to allow for fine-grained control of what is logged and how:

keystoneauth.session.request-id
Emits a log entry at the `DEBUG` level for every http request including information about the URL, `service-type` and `request-id`.

keystoneauth.session.request
Emits a log entry at the `DEBUG` level for every http request including a curl formatted string of the request.

keystoneauth.session.response
Emits a log entry at the `DEBUG` level for every http response received, including the status code, and the headers received.

keystoneauth.session.body
Emits a log entry at the `DEBUG` level containing the contents of the response body if the `content-type` is either `text` or `json`.

1.4.1 Using loggers

A full description of how to consume python logging is out of scope of this document, but a few simple examples are provided.

If you would like to configure logging to log keystoneuath at the `INFO` level with no `DEBUG` messages:

```python
import keystoneauth1
import logging

logger = logging.getLogger('keystoneauth')
logger.addHandler(logging.StreamHandler())
logger.setLevel(logging.INFO)
```

If you would like to get a full HTTP debug trace including bodies:

```python
import keystoneauth1
import logging

logger = logging.getLogger('keystoneauth')
logger.addHandler(logging.StreamHandler())
logger.setLevel(logging.DEBUG)
```

If you would like to get a full HTTP debug trace bug with no bodies:

```python
import keystoneauth1
import keystoneauth1.session
import logging
```

(continues on next page)

(continued from previous page)

```
logger = logging.getLogger('keystoneauth')
logger.addHandler(logging.StreamHandler())
logger.setLevel(logging.DEBUG)
body_logger = logging.getLogger('keystoneauth.session.body')
body_logger.setLevel(logging.WARN)
session = keystoneauth1.session.Session(split_loggers=True)
```

Finally, if you would like to log request-ids and response headers to one file, request commands, response headers and response bodies to a different file, and everything else to the console:

```
import keystoneauth1
import keystoneauth1.session
import logging

# Create a handler that outputs only outputs INFO level messages to stdout
stream_handler = logging.StreamHandler()
stream_handler.setLevel(logging.INFO)

# Configure the default behavior of all keystoneauth logging to log at the
# INFO level.
logger = logging.getLogger('keystoneauth')
logger.setLevel(logging.INFO)

# Emit INFO messages from all keystoneauth loggers to stdout
logger.addHandler(stream_handler)

# Create an output formatter that includes logger name and timestamp.
formatter = logging.Formatter('%(asctime)s %(name)s %(message)s')

# Create a file output for request ids and response headers
request_handler = logging.FileHandler('request.log')
request_handler.setFormatter(formatter)

# Create a file output for request commands, response headers and bodies
body_handler = logging.FileHandler('response-body.log')
body_handler.setFormatter(formatter)

# Log all HTTP interactions at the DEBUG level
session_logger = logging.getLogger('keystoneauth.session')
session_logger.setLevel(logging.DEBUG)

# Emit request ids to the request log
request_id_logger = logging.getLogger('keystoneauth.session.request-id')
request_id_logger.addHandler(request_handler)

# Emit response headers to both the request log and the body log
header_logger = logging.getLogger('keystoneauth.session.response')
header_logger.addHandler(request_handler)
header_logger.addHandler(body_handler)
```

(continues on next page)

(continued from previous page)

```
# Emit request commands to the body log
request_logger = logging.getLogger('keystoneauth.session.request')
request_logger.addHandler(body_handler)

# Emit bodies only to the body log
body_logger = logging.getLogger('keystoneauth.session.body')
body_logger.addHandler(body_handler)

session = keystoneauth1.session.Session(split_loggers=True)
```

The above will produce messages like the following in request.log:

```
2017-09-19 22:10:09,466 keystoneauth.session.request-id  GET call to volumev2␣
↪for http://cloud.example.com/volume/v2/137155c35fb34172a284a3c2540c92ab/
↪volumes/detail used request id req-f4f2058a-9308-4c4a-94e6-5ee1cd6c78bd
2017-09-19 22:10:09,751 keystoneauth.session.response    [200] Date: Tue, 19␣
↪Sep 2017 22:10:09 GMT Server: Apache/2.4.18 (Ubuntu) x-compute-request-id:␣
↪req-2e9181d2-9f3e-404e-a12f-1f1566736ab3 Content-Type: application/json␣
↪Content-Length: 15 x-openstack-request-id: req-2e9181d2-9f3e-404e-a12f-
↪1f1566736ab3 Connection: close
```

And content like the following into response-body.log:

```
2017-09-19 22:10:09,490 keystoneauth.session.request     curl -g -i -X GET␣
↪http://cloud.example.com/volume/v2/137155c35fb34172a284a3c2540c92ab/volumes/
↪detail?marker=34cd00cf-bf67-4667-a900-5ce233e383d5 -H "User-Agent: os-
↪client-config/1.28.0 shade/1.23.1 keystoneauth1/3.2.0 python-requests/2.18.
↪4 CPython/2.7.12" -H "X-Auth-Token: {SHA1}
↪a1d03d2a4cbee590a55f1786d452e1027d5fd781"
2017-09-19 22:10:09,751 keystoneauth.session.response    [200] Date: Tue, 19␣
↪Sep 2017 22:10:09 GMT Server: Apache/2.4.18 (Ubuntu) x-compute-request-id:␣
↪req-2e9181d2-9f3e-404e-a12f-1f1566736ab3 Content-Type: application/json␣
↪Content-Length: 15 x-openstack-request-id: req-2e9181d2-9f3e-404e-a12f-
↪1f1566736ab3 Connection: close
2017-09-19 22:10:09,751 keystoneauth.session.body        {"volumes": []}
```

1.4.2 User Provided Loggers

The HTTP methods (request, get, post, put, etc) on *keystoneauth1.session.Session* and *keystoneauth1.adapter.Adapter* all support a `logger` parameter. A user can provide their own logger which will override the session loggers mentioned above. If a single logger is provided in this manner, request, response and body content will all be logged to that logger at the `DEBUG` level, and the strings `REQ:`, `RESP:` and `RESP BODY:` will be pre-pended as appropriate.

CHAPTER

TWO

AUTHENTICATION PLUGINS

2.1 Introduction

Authentication plugins provide a generic means by which to extend the authentication mechanisms known to OpenStack clients.

In the vast majority of cases the authentication plugins used will be those written for use with the OpenStack Identity Service (Keystone), however this is not the only possible case, and the mechanisms by which authentication plugins are used and implemented should be generic enough to cover completely customized authentication solutions.

The subset of authentication plugins intended for use with an OpenStack Identity server (such as Keystone) are called Identity Plugins.

2.2 Available Plugins

Keystoneauth ships with a number of plugins and particularly Identity Plugins.

2.2.1 V2 Identity Plugins

Standard V2 identity plugins are defined in the module: `keystoneauth1.identity.v2`

They include:

- `Password`: Authenticate against a V2 identity service using a username and password.
- `Token`: Authenticate against a V2 identity service using an existing token.

V2 identity plugins must use an *auth_url* that points to the root of a V2 identity server URL, i.e.: `http://hostname:5000/v2.0`.

2.2.2 V3 Identity Plugins

Standard V3 identity plugins are defined in the module `keystoneauth1.identity.v3`.

V3 Identity plugins are slightly different from their V2 counterparts as a V3 authentication request can contain multiple authentication methods. To handle this V3 defines a number of different `AuthMethod` classes:

- `PasswordMethod`: Authenticate against a V3 identity service using a username and password.
- `TokenMethod`: Authenticate against a V3 identity service using an existing token.
- `ReceiptMethod`: Authenticate against a V3 identity service using an existing auth-receipt. This method has to be used in conjunction with at least one other method.

- `TOTPMethod`: Authenticate against a V3 identity service using Time-Based One-Time Password (TOTP).
- `TokenlessAuth`: Authenticate against a V3 identity service using tokenless authentication.
- `ApplicationCredentialMethod`: Authenticate against a V3 identity service using an application credential.
- `KerberosMethod`: Authenticate against a V3 identity service using Kerberos.
- `OAuth2ClientCredentialMethod`: Authenticate against a V3 identity service using an OAuth2.0 client credential.
- `OAuth2mTlsClientCredential`: Authenticate against a V3 identity service using an OAuth2.0 Mutual-TLS client credentials.

The `AuthMethod` objects are then passed to the `Auth` plugin:

```
>>> from keystoneauth1 import session
>>> from keystoneauth1.identity import v3
>>> password = v3.PasswordMethod(username='user',
...                              password='password',
...                              user_domain_name='default')
>>> auth = v3.Auth(auth_url='http://my.keystone.com:5000/v3',
...                auth_methods=[password],
...                project_id='projectid')
>>> sess = session.Session(auth=auth)
```

You can even add additional methods to an existing auth instance after it has been created:

```
>>> totp = v3.TOTPMethod(username='user',
...                      passcode='123456',
...                      user_domain_name='default')
>>> auth.add_method(totp)
```

Or use the `MultiFactor` helper plugin to do it all simply in one go, an example of whichs exists in the section below.

For the common cases where you will only want to use one `AuthMethod` there are also helper authentication plugins for the various `AuthMethod` which can be used more like the V2 plugins:

- `Password`: Authenticate using only a `PasswordMethod`.
- `Token`: Authenticate using only a `TokenMethod`.
- `TOTP`: Authenticate using only a `TOTPMethod`.
- `Kerberos`: Authenticate using only a `KerberosMethod`.

```
>>> auth = v3.Password(auth_url='http://my.keystone.com:5000/v3',
...                    username='username',
...                    password='password',
...                    project_id='projectid',
...                    user_domain_name='default')
>>> sess = session.Session(auth=auth)
```

This will have exactly the same effect as using the single `PasswordMethod` above.

V3 identity plugins must use an *auth_url* that points to the root of a V3 identity server URL, i.e.: `http://hostname:5000/v3`.

2.2.3 Multi-Factor with V3 Identity Plugins

The basic example of multi-factor authentication is when you supply all the needed auth methods up front.

This can be done by building an Auth class with method instances:

```
from keystoneauth1 import session
from keystoneauth1.identity import v3

auth = v3.Auth(
    auth_url='http://my.keystone.com:5000/v3',
    auth_methods=[
        v3.PasswordMethod(
            username='user',
            password='password',
            user_domain_id="default",
        ),
        v3.TOTPMethod(
            username='user',
            passcode='123456',
            user_domain_id="default",
        )
    ],
    project_id='projectid',
)
sess = session.Session(auth=auth)
```

Or by letting the helper plugin do it for you:

```
from keystoneauth1 import session
from keystoneauth1.identity import v3

auth = v3.MultiFactor(
    auth_url='http://my.keystone.com:5000/v3',
    auth_methods=['v3password', 'v3totp'],
    username='user',
    password='password',
    passcode='123456',
    user_domain_id="default",
    project_id='projectid',
)
sess = session.Session(auth=auth)
```

Note: The `MultiFactor` helper does not support auth receipts as an option in auth_methods, but one can be added with *auth.add_method*.

When you supply just one method when multiple are needed, a `MissingAuthMethods` error will be raised. This can be caught, and you can infer based on the error what the missing methods were, and from it extract the receipt to continue authentication:

```
auth = v3.Password(auth_url='http://my.keystone.com:5000/v3',
                   username='username',
                   password='password',
                   project_id='projectid',
                   user_domain_id='default')
sess = session.Session(auth=auth)
try:
   sess.get_token()
except exceptions.MissingAuthMethods as e:
    receipt = e.receipt
    methods = e.methods
    required_methods = e.required_auth_methods
```

Once you know what auth methods are needed to continue, you can extend the existing auth plugin with additional methods:

```
auth.add_method(
    v3.TOTPMethod(
        username='user',
        passcode='123456',
        user_domain_id='default',
    )
)
sess.get_token()
```

Or if you do not have the existing auth method, but have the receipt you can continue as well:

```
auth = v3.TOTP(
    auth_url='http://my.keystone.com:5000/v3',
    username='user',
    passcode='123456',
    user_domain_id='default',
    project_id='projectid',
)
auth.add_method(v3.ReceiptMethod(receipt=receipt))
sess = session.Session(auth=auth)
sess.get_token()
```

2.2.4 Standalone Plugins

Services can be deployed in a standalone environment where there is no integration with an identity service. The following plugins are provided to support standalone services:

- `HTTPBasicAuth`: HTTP Basic authentication
- `NoAuth`: No authentication

Standalone plugins must be given an *endpoint* that points to the URL of the one service being used, since there is no service catalog to look up endpoints:

```
from keystoneauth1 import session
from keystoneauth1 import noauth
```

(continues on next page)

(continued from previous page)

```
auth = noauth.NoAuth(endpoint='http://hostname:6385/')
sess = session.Session(auth=auth)
```

`HTTPBasicAuth` also requres a *username* and *password*:

```
from keystoneauth1 import session
from keystoneauth1 import http_basic
auth = http_basic.HTTPBasicAuth(endpoint='http://hostname:6385/',
                                username='myUser',
                                password='myPassword')
sess = session.Session(auth=auth)
```

2.3 Federation

The following V3 plugins are provided to support federation:

- `MappedKerberos`: Federated (mapped) Kerberos.
- `Password`: SAML2 password authentication.
- `v3:OpenIDConnectAccessToken`: Plugin to reuse an existing OpenID Connect access token.
- `v3:OpenIDConnectAuthorizationCode`: OpenID Connect Authorization Code grant type.
- `v3:OpenIDConnectClientCredentials`: OpenID Connect Client Credentials grant type.
- `v3:OpenIDConnectPassword`: OpenID Connect Resource Owner Password Credentials grant type.
- `Keystone2Keystone`: Keystone to Keystone Federation.

The Keystone2Keystone plugin is special as it takes a Password auth for one keystone instance acting as an Identity Provider as input in order to create a session on the keystone acting as a Service Provider, for example:

```
from keystoneauth1 import session
from keystoneauth1.identity import v3
from keystoneauth1.identity.v3 import k2k

pwauth = v3.Password(auth_url='http://my.keystone.com:5000/v3',
                     username='username',
                     password='password',
                     project_id='projectid',
                     user_domain_name='Default')
k2kauth = k2k.Keystone2Keystone(pwauth, 'mysp',
                                project_id='federated_projectid')
k2ksession = session.Session(auth=k2kauth)
```

The *OpenIDConnectPassword* plugin also supports OTP. This option is required in cases when the Identity Provider requires more than a password to authenticate the user. As the OTP usually is a short-lived code that continually changes, then, when this option is active, the user will be requested to input the OTP code when executing the authentication process.

To enable this option, the user will need to export the environment variable OS_IDP_OTP_KEY with the OTP key used by the Identity Providers authentication API.

E.g.: If the Identity Providers authentication API requires some JSON like:

```
{
    "username": "user1",
    "password": "passwd",
    "totp": "763907"
}
```

Then, you will use the totp value in your OS_IDP_OTP_KEY, something like export OS_IDP_OTP_KEY=totp.

After the configuration of the OS_IDP_OTP_KEY environment variable, every time that you will log in through the python openstack-client, a prompt will be displayed requesting to you to input your OTP code.

2.3.1 Version Independent Identity Plugins

Standard version independent identity plugins are defined in the module `keystoneauth1.identity.generic`.

For the cases of plugins that exist under both the identity V2 and V3 APIs there is an abstraction to allow the plugin to determine which of the V2 and V3 APIs are supported by the server and use the most appropriate API.

These plugins are:

- `Password`: Authenticate using a user/password against either v2 or v3 API.
- `Token`: Authenticate using an existing token against either v2 or v3 API.

These plugins work by first querying the identity server to determine available versions and so the *auth_url* used with the plugins should point to the base URL of the identity server to use. If the *auth_url* points to either a V2 or V3 endpoint it will restrict the plugin to only working with that version of the API.

2.3.2 Simple Plugins

In addition to the Identity plugins a simple plugin that will always use the same provided token and endpoint is available. This is useful for situations where you have a token and want to bypass authentication to obtain a new token for subsequent requests. Testing, proxies, and service-to-service authentication on behalf of a user are good examples use cases for this authentication plugin.

It can be found at `keystoneauth1.token_endpoint.Token`.

For example:

```
>>> from keystoneauth1 import token_endpoint
>>> from keystoneauth1 import session
>>> a = token_endpoint.Token('http://my.keystone.com:5000/v3',
...                          token=token)
>>> s = session.Session(auth=a)
```

2.3.3 V3 OAuth 1.0a Plugins

There also exists a plugin for OAuth 1.0a authentication. We provide a helper authentication plugin at: `V3OAuth1`. The plugin requires the OAuth consumers key and secret, as well as the OAuth access tokens key and secret. For example:

```
>>> from keystoneauth1.extras import oauth1
>>> from keystoneauth1 import session
>>> a = oauth1.V3OAuth1('http://my.keystone.com:5000/v3',
...                     consumer_key=consumer_id,
...                     consumer_secret=consumer_secret,
...                     access_key=access_token_key,
...                     access_secret=access_token_secret)
>>> s = session.Session(auth=a)
```

2.4 Application Credentials

There is a specific authentication method for interacting with Identity servers that support application credential authentication. Since application credentials are associated to a user on a specific project, some parameters are not required as they would be with traditional password authentication. The following method can be used to authenticate for a token using an application credential:

- `ApplicationCredential`:

The following example shows the method usage with a session:

```
>>> from keystoneauth1 import session
>>> from keystone.identity import v3
>>> auth = v3.ApplicationCredential(
        application_credential_secret='application_credential_secret',
        application_credential_id='c2872b920853478292623be94b657090'
    )
>>> sess = session.Session(auth=auth)
```

2.5 OAuth2.0 Client Credentials

Warning

The access token must be only added for the requests using HTTPS according to RFC6749.

There is a specific authentication method for interacting with Identity servers that support OAuth2.0 Client Credential Grant. The notable difference from the other authentication method is that, after passing the authentication, the `session` will add Authorization header with an OAuth2.0 access token to sent subsequent requests. The following method can be used to authenticate for a token using OAuth2.0 client credentials:

- `OAuth2ClientCredential`:

The following example shows the method usage with a session:

```
>>> from keystoneauth1 import session
>>> from keystone.identity import v3
>>> auth = v3.OAuth2ClientCredential(
        oauth2_endpoint='https://keystone.host/identity/v3/OS-OAUTH2/token'
        oauth2_client_id='f96a2fec117141a6b5fbaa0485632244',
        oauth2_client_secret='client_credential_secret'
    )
>>> sess = session.Session(auth=auth)
```

2.6 OAuth2.0 Mutual-TLS Client Credentials

Warning

The access token must be only added for the requests using mutual TLS according to RFC8705.

There is a specific authentication method for interacting with Identity servers that support OAuth 2.0 Mutual-TLS Client Authentication. The notable difference from the other authentication method is that, after passing the authentication, the `session` will add Authorization header with an OAuth2.0 Certificate-Bound Access Tokens to sent subsequent requests. The following method can be used to authenticate for a token using OAuth2.0 Mutual-TLS client credentials:

- `OAuth2mTlsClientCredential`:

The following example shows the method usage with a session:

```
>>> from keystoneauth1 import session
>>> from keystone.identity import v3
>>> auth = v3.OAuth2mTlsClientCredential(
        auth_url='http://keystone.host:5000/v3'
        oauth2_endpoint='https://keystone.host/identity/v3/OS-OAUTH2/token'
        oauth2_client_id='f96a2fec117141a6b5fbaa0485632244'
    )
>>> sess = session.Session(auth=auth)
```

2.7 Tokenless Auth

A plugin for tokenless authentication also exists. It provides a means to authorize client operations within the Identity server by using an X.509 TLS client certificate without having to issue a token. We provide a tokenless authentication plugin at:

- `TokenlessAuth`

It is mostly used by service clients for token validation and here is an example of how this plugin would be used in practice:

```
>>> from keystoneauth1 import session
>>> from keystoneauth1.identity import v3
>>> auth = v3.TokenlessAuth(auth_url='https://keystone:5000/v3',
...                         domain_name='my_service_domain')
```

(continues on next page)

(continued from previous page)

```
>>> sess = session.Session(
...                     auth=auth,
...                     cert=('/opt/service_client.crt',
...                           '/opt/service_client.key'),
...                     verify='/opt/ca.crt')
```

2.8 Loading Plugins by Name

In auth_token middleware and for some service to service communication it is possible to specify a plugin to load via name. The authentication options that are available are then specific to the plugin that you specified. Currently the authentication plugins that are available in *keystoneauth* are:

- http_basic: `keystoneauth1.http_basic.HTTPBasicAuth`
- none: `keystoneauth1.noauth.NoAuth`
- password: `keystoneauth1.identity.generic.Password`
- token: `keystoneauth1.identity.generic.Token`
- v2password: `keystoneauth1.identity.v2.Password`
- v2token: `keystoneauth1.identity.v2.Token`
- v3applicationcredential: `keystoneauth1.identity.v3.ApplicationCredential`
- v3password: `keystoneauth1.identity.v3.Password`
- v3token: `keystoneauth1.identity.v3.Token`
- v3fedkerb: `keystoneauth1.extras.kerberos.MappedKerberos`
- v3kerberos: `keystoneauth1.extras.kerberos.Kerberos`
- v3oauth1: `keystoneauth1.extras.oauth1.v3.OAuth1`
- v3oidcaccesstoken: `keystoneauth1.identity.v3:OpenIDConnectAccessToken`
- v3oidcauthcode: `keystoneauth1.identity.v3:OpenIDConnectAuthorizationCode`
- v3oidcdeviceauthz: `keystoneauth1.loading._plugins.identity.v3:OpenIDConnectDeviceAuthorization`
- v3oidcclientcredentials: `keystoneauth1.identity.v3:OpenIDConnectClientCredentials`
- v3oidcpassword: `keystoneauth1.identity.v3:OpenIDConnectPassword`
- v3samlpassword: `keystoneauth1.extras._saml2.v3.Password`
- v3tokenlessauth: `keystoneauth1.identity.v3.TokenlessAuth`
- v3totp: `keystoneauth1.identity.v3.TOTP`
- v3oauth2clientcredential: `keystoneauth1.identity.v3.OAuth2ClientCredential`
- v3oauth2mtlsclientcredential: `keystoneauth1.identity.v3.OAuth2mTlsClientCredential`

2.9 Creating Authentication Plugins

2.9.1 Creating an Identity Plugin

If you have implemented a new authentication mechanism into the Identity service then you will be able to reuse a lot of the infrastructure available for the existing Identity mechanisms. As the V2 identity API is essentially frozen, it is expected that new plugins are for the V3 API.

To implement a new V3 plugin that can be combined with others you should implement the base `keystoneauth1.identity.v3.AuthMethod` class and implement the `get_auth_data()` function. If your Plugin cannot be used in conjunction with existing `keystoneauth1.identity.v3.AuthMethod` then you should just override `keystoneauth1.identity.v3.Auth` directly.

The new `AuthMethod` should take all the required parameters via `__init__()` and return from `get_auth_data()` a tuple with the unique identifier of this plugin (e.g. *password*) and a dictionary containing the payload of values to send to the authentication server. The session, calling auth object and request headers are also passed to this function so that the plugin may use or manipulate them.

You should also provide a class that inherits from `keystoneauth1.identity.v3.Auth` with an instance of your new `AuthMethod` as the *auth_methods* parameter to `keystoneauth1.identity.v3.Auth`.

By convention (and like above) these are named *PluginType* and *PluginTypeMethod* (for example `Password` and `PasswordMethod`).

2.9.2 Creating a Custom Plugin

To implement an entirely new plugin you should implement the base class `keystoneauth1.plugin.BaseAuthPlugin` and provide the `get_endpoint()`, `get_token()` and `invalidate()` methods.

`get_token()` is called to retrieve the string token from a plugin. It is intended that a plugin will cache a received token and so if the token is still valid then it should be re-used rather than fetching a new one. A session object is provided with which the plugin can contact its server. (Note: use *authenticated=False* when making those requests or it will end up being called recursively). The return value should be the token as a string.

`get_endpoint()` is called to determine a base URL for a particular services requests. The keyword arguments provided to the function are those that are given by the *endpoint_filter* variable in `keystoneauth1.session.Session.request()`. A session object is also provided so that the plugin may contact an external source to determine the endpoint. Again this will be generally be called once per request and so it is up to the plugin to cache these responses if appropriate. The return value should be the base URL to communicate with.

`invalidate()` should also be implemented to clear the current user credentials so that on the next `get_token()` call a new token can be retrieved.

The most simple example of a plugin is the `keystoneauth1.token_endpoint.Token` plugin.

CHAPTER

THREE

PLUGIN OPTIONS

3.1 Usage

3.1.1 Using plugins via CLI

Plugins can be configured via CLI options, using argparses `ArgumentParser`. This is commonly used to produce client tooling that communicates with OpenStack APIs and therefore needs to allow authentication. For example, `openstackclient` allows configuration using CLI options.

When using auth plugins via CLI you can specify parameters via CLI options or via environment configuration, with CLI options superseding environment configuration. CLI options are specified with the pattern `--os-` and the parameter name. For example, to use the *password* plugin via CLI options you can specify:

```
openstack --os-auth-type password \
          --os-auth-url http://keystone.example.com:5000/ \
          --os-username myuser \
          --os-password mypassword \
          --os-project-name myproject \
          --os-default-domain-name mydomain \
          operation
```

Environment variables are specified using the pattern `OS_` followed by the uppercase parameter name replacing `-` with `_`. Using the *password* example again:

```
export OS_AUTH_TYPE=password
export OS_AUTH_URL=http://keystone.example.com:5000/
export OS_USERNAME=myuser
export OS_PASSWORD=mypassword
export OS_PROJECT_NAME=myproject
export OS_DEFAULT_DOMAIN_NAME=mydomain
```

3.1.2 Using plugins via `clouds.yaml`

Plugins can be configured via `clouds.yaml` files, which are supported by `openstacksdk`. When using a `clouds.yaml`, you specify the plugin name as `auth_type` within the cloud entry and then specify all plugin options within the `auth` key of the cloud entry. For example, to use the *password* plugin for a cloud entry `mycloud` in a `clouds.yaml` file you can specify:

```
clouds:
  mycloud:
```

(continues on next page)

(continued from previous page)

```
    auth_type: password
    auth:
      auth_url: http://keystone.example.com:5000/
      username: myuser
      password: mypassword
      project_name: myproject
      default_domain_name: mydomain
```

3.1.3 Using plugins via config file

Plugins can be configured using INI-style configuration file, using oslo.config. This is commonly used to allow OpenStack service to talk to each other though it can be used for any service that wishes to authenticate against Keystone and uses oslo.config. For example, this configuration style is used to allow the Compute service (Nova) to talk to the Networking service (Neutron), Block Storage service (Cinder), and others.

When using the plugins via config file you define the plugin name as `auth_type`. The options of the plugin are then specified while replacing `-` with `_` to be valid in configuration.

For example to use the *password* plugin in a config file you would specify:

```
[section]
auth_type = password
auth_url = http://keystone.example.com:5000/
username = myuser
password = mypassword
project_name = myproject
default_domain_name = mydomain
```

3.1.4 Using plugins via other mechanisms

Beyond the three configuration mechanisms described here, different services may implement loaders in their own way and you should consult their relevant documentation. However, the same auth options will always be available.

3.2 Built-in Plugins

This is a listing of all included plugins and the options that they accept. Plugins are listed alphabetically and not in any order of priority.

3.2.1 admin_token

Authenticate with an existing token and a known endpoint.

This plugin is primarily useful for development or for use with identity service ADMIN tokens. Because this token is used directly there is no fetching a service catalog or determining scope information and so it cannot be used by clients that expect use this scope information.

Because there is no service catalog the endpoint that is supplied with initialization is used for all operations performed with this plugin so must be the full base URL to an actual service.

endpoint
The endpoint that will always be used

CLI options
`--os-endpoint`, `--os-url`

Environment variables
`OS_ENDPOINT`, `OS_URL`

token
The token that will always be used

CLI options
`--os-token`

Environment variables
`OS_TOKEN`

3.2.2 http_basic

Use HTTP Basic authentication to perform requests.

This can be used to instantiate clients for services deployed in standalone mode.

There is no fetching a service catalog or determining scope information and so it cannot be used by clients that expect to use this scope information.

---

endpoint
The endpoint that will always be used

CLI options
`--os-endpoint`

Environment variables
`OS_ENDPOINT`

password
Users password

CLI options
`--os-password`

Environment variables
`OS_PASSWORD`

username
Username

CLI options
`--os-username`, `--os-user-name`

Environment variables
`OS_USERNAME`, `OS_USER_NAME`

3.2.3 none

Use no tokens to perform requests.

This can be used to instantiate clients for services deployed in noauth/standalone mode.

There is no fetching a service catalog or determining scope information and so it cannot be used by clients that expect to use this scope information.

---

endpoint
The endpoint that will always be used

CLI options
`--os-endpoint`

Environment variables
`OS_ENDPOINT`

3.2.4 password

Authenticate with a username and password.

Authenticate to the identity service using the provided username and password. This is the standard and most common form of authentication.

As a generic plugin this plugin is identity version independent and will discover available versions before use. This means it expects to be provided an unversioned URL to operate against.

---

auth-url
Authentication URL (**mandatory**)

CLI options
`--os-auth-url`

Environment variables
`OS_AUTH_URL`

default-domain-id
Optional domain ID to use with v3 and v2 parameters. It will be used for both the user and project domain in v3 and ignored in v2 authentication.

CLI options
`--os-default-domain-id`

Environment variables
`OS_DEFAULT_DOMAIN_ID`

default-domain-name
Optional domain name to use with v3 API and v2 parameters. It will be used for both the user and project domain in v3 and ignored in v2 authentication.

CLI options
`--os-default-domain-name`

Environment variables
`OS_DEFAULT_DOMAIN_NAME`

domain-id
Domain ID to scope to

CLI options
`--os-domain-id`

Environment variables
`OS_DOMAIN_ID`

domain-name
Domain name to scope to

CLI options
`--os-domain-name`

Environment variables
`OS_DOMAIN_NAME`

password
Users password

CLI options
`--os-password`

Environment variables
`OS_PASSWORD`

project-domain-id
Domain ID containing project

CLI options
`--os-project-domain-id`

Environment variables
`OS_PROJECT_DOMAIN_ID`

project-domain-name
Domain name containing project

CLI options
`--os-project-domain-name`

Environment variables
`OS_PROJECT_DOMAIN_NAME`

project-id
Project ID to scope to

CLI options
`--os-project-id`, `--os-tenant-id`

Environment variables
`OS_PROJECT_ID`, `OS_TENANT_ID`

project-name
Project name to scope to

CLI options
`--os-project-name`, `--os-tenant-name`

Environment variables
`OS_PROJECT_NAME`, `OS_TENANT_NAME`

system-scope
Scope for system operations

CLI options
`--os-system-scope`

Environment variables
`OS_SYSTEM_SCOPE`

trust-id
ID of the trust to use as a trustee use

CLI options
`--os-trust-id`

Environment variables
`OS_TRUST_ID`

user-domain-id
Users domain id

CLI options
`--os-user-domain-id`

Environment variables
`OS_USER_DOMAIN_ID`

user-domain-name
Users domain name

CLI options
`--os-user-domain-name`

Environment variables
`OS_USER_DOMAIN_NAME`

user-id
User id

CLI options
`--os-user-id`

Environment variables
`OS_USER_ID`

username
Username

CLI options
`--os-username`, `--os-user-name`

Environment variables
`OS_USERNAME`, `OS_USER_NAME`

3.2.5 token

Given an existing token rescope it to another target.

Use the Identity services rescope mechanism to get a new token based upon an existing token. Because an auth plugin requires a service catalog and scope information it is often easier to fetch a new token based on an existing one than validate and reuse the one you already have.

As a generic plugin this plugin is identity version independent and will discover available versions before use. This means it expects to be provided an unversioned URL to operate against.

---

auth-url
Authentication URL (**mandatory**)

CLI options
`--os-auth-url`

Environment variables
`OS_AUTH_URL`

default-domain-id
Optional domain ID to use with v3 and v2 parameters. It will be used for both the user and project domain in v3 and ignored in v2 authentication.

CLI options
`--os-default-domain-id`

Environment variables
`OS_DEFAULT_DOMAIN_ID`

default-domain-name
Optional domain name to use with v3 API and v2 parameters. It will be used for both the user and project domain in v3 and ignored in v2 authentication.

CLI options
`--os-default-domain-name`

Environment variables
`OS_DEFAULT_DOMAIN_NAME`

domain-id
Domain ID to scope to

CLI options
`--os-domain-id`

Environment variables
`OS_DOMAIN_ID`

domain-name
Domain name to scope to

CLI options
`--os-domain-name`

Environment variables
`OS_DOMAIN_NAME`

project-domain-id
Domain ID containing project

CLI options
`--os-project-domain-id`

Environment variables
`OS_PROJECT_DOMAIN_ID`

project-domain-name
Domain name containing project

CLI options
`--os-project-domain-name`

Environment variables
`OS_PROJECT_DOMAIN_NAME`

project-id
Project ID to scope to

CLI options
`--os-project-id`, `--os-tenant-id`

Environment variables
`OS_PROJECT_ID`, `OS_TENANT_ID`

project-name
Project name to scope to

CLI options
`--os-project-name`, `--os-tenant-name`

Environment variables
`OS_PROJECT_NAME`, `OS_TENANT_NAME`

system-scope
Scope for system operations

CLI options
`--os-system-scope`

Environment variables
`OS_SYSTEM_SCOPE`

token
Token to authenticate with

CLI options
`--os-token`

Environment variables
`OS_TOKEN`

trust-id
ID of the trust to use as a trustee use

CLI options
`--os-trust-id`

Environment variables
`OS_TRUST_ID`

3.2.6 v2password

Authenticate with a username and password.

Authenticate to the identity service using the provided username and password. This is the standard and most common form of authentication.

---

auth-url
Authentication URL (**mandatory**)

CLI options
`--os-auth-url`

Environment variables
`OS_AUTH_URL`

password
Password to use

CLI options
`--os-password`

Environment variables
`OS_PASSWORD`

tenant-id
Tenant ID

CLI options
`--os-tenant-id`

Environment variables
`OS_TENANT_ID`

tenant-name
Tenant Name

CLI options
`--os-tenant-name`

Environment variables
`OS_TENANT_NAME`

trust-id
ID of the trust to use as a trustee use

CLI options
`--os-trust-id`

Environment variables
`OS_TRUST_ID`

user-id
User ID to login with

CLI options
`--os-user-id`

Environment variables
`OS_USER_ID`

username
Username to login with

CLI options
`--os-username`, `--os-user-name`

Environment variables
`OS_USERNAME`, `OS_USER_NAME`

3.2.7 v2token

Given an existing token rescope it to another target.

Use the Identity services rescope mechanism to get a new token based upon an existing token. Because an auth plugin requires a service catalog and scope information it is often easier to fetch a new token based on an existing one than validate and reuse the one you already have.

---

auth-url
Authentication URL (**mandatory**)

CLI options
`--os-auth-url`

Environment variables
`OS_AUTH_URL`

tenant-id
Tenant ID

CLI options
`--os-tenant-id`

Environment variables
`OS_TENANT_ID`

tenant-name
Tenant Name

CLI options
`--os-tenant-name`

Environment variables
`OS_TENANT_NAME`

token
Token

CLI options
`--os-token`

Environment variables
`OS_TOKEN`

trust-id
ID of the trust to use as a trustee use

CLI options
`--os-trust-id`

Environment variables
`OS_TRUST_ID`

3.2.8 v3adfspassword

auth-url
Authentication URL **(mandatory)**

CLI options
`--os-auth-url`

Environment variables
`OS_AUTH_URL`

domain-id
Domain ID to scope to

CLI options
`--os-domain-id`

Environment variables
`OS_DOMAIN_ID`

domain-name
Domain name to scope to

CLI options
`--os-domain-name`

Environment variables
`OS_DOMAIN_NAME`

identity-provider
Identity Providers name **(mandatory)**

CLI options
`--os-identity-provider`

Environment variables
`OS_IDENTITY_PROVIDER`

identity-provider-url
An Identity Provider URL, where the SAML authentication request will be sent. **(mandatory)**

CLI options
`--os-identity-provider-url`

Environment variables
`OS_IDENTITY_PROVIDER_URL`

password
Password **(mandatory)**

CLI options
`--os-password`

Environment variables
`OS_PASSWORD`

project-domain-id
Domain ID containing project

CLI options
`--os-project-domain-id`

Environment variables
`OS_PROJECT_DOMAIN_ID`

project-domain-name
Domain name containing project

CLI options
`--os-project-domain-name`

Environment variables
`OS_PROJECT_DOMAIN_NAME`

project-id
Project ID to scope to

CLI options
`--os-project-id`

Environment variables
`OS_PROJECT_ID`

project-name
Project name to scope to

CLI options
`--os-project-name`

Environment variables
`OS_PROJECT_NAME`

protocol
Protocol for federated plugin (**mandatory**)

CLI options
`--os-protocol`

Environment variables
`OS_PROTOCOL`

service-provider-endpoint
Service Providers Endpoint (**mandatory**)

CLI options
`--os-service-provider-endpoint`

Environment variables
`OS_SERVICE_PROVIDER_ENDPOINT`

service-provider-entity-id
Service Providers SAML Entity ID **(mandatory)**

CLI options
`--os-service-provider-entity-id`

Environment variables
`OS_SERVICE_PROVIDER_ENTITY_ID`

system-scope
Scope for system operations

CLI options
`--os-system-scope`

Environment variables
`OS_SYSTEM_SCOPE`

trust-id
ID of the trust to use as a trustee use

CLI options
`--os-trust-id`

Environment variables
`OS_TRUST_ID`

username
Username **(mandatory)**

CLI options
`--os-username`

Environment variables
`OS_USERNAME`

3.2.9 v3applicationcredential

Authenticate with an application credential.

Authenticate to the identity service using the provided application credential secret and ID or name. If a name is used, you must also provide a username and user domain to assist in lookup.

---

application_credential_id
Application credential ID

CLI options
`--os-application_credential_id`

Environment variables
`OS_APPLICATION_CREDENTIAL_ID`

application_credential_name
Application credential name

CLI options
`--os-application_credential_name`

Environment variables
`OS_APPLICATION_CREDENTIAL_NAME`

application_credential_secret
Application credential auth secret **(mandatory)**

CLI options
`--os-application_credential_secret`

Environment variables
`OS_APPLICATION_CREDENTIAL_SECRET`

auth-url
Authentication URL **(mandatory)**

CLI options
`--os-auth-url`

Environment variables
`OS_AUTH_URL`

domain-id
Domain ID to scope to

CLI options
`--os-domain-id`

Environment variables
`OS_DOMAIN_ID`

domain-name
Domain name to scope to

CLI options
`--os-domain-name`

Environment variables
`OS_DOMAIN_NAME`

project-domain-id
Domain ID containing project

CLI options
`--os-project-domain-id`

Environment variables
`OS_PROJECT_DOMAIN_ID`

project-domain-name
Domain name containing project

CLI options
`--os-project-domain-name`

Environment variables
`OS_PROJECT_DOMAIN_NAME`

project-id
Project ID to scope to

CLI options
`--os-project-id`

Environment variables
`OS_PROJECT_ID`

project-name
Project name to scope to

CLI options
`--os-project-name`

Environment variables
`OS_PROJECT_NAME`

system-scope
Scope for system operations

CLI options
`--os-system-scope`

Environment variables
`OS_SYSTEM_SCOPE`

trust-id
ID of the trust to use as a trustee use

CLI options
`--os-trust-id`

Environment variables
`OS_TRUST_ID`

user-domain-id
Users domain ID

CLI options
`--os-user-domain-id`

Environment variables
`OS_USER_DOMAIN_ID`

user-domain-name
Users domain name

CLI options
`--os-user-domain-name`

Environment variables
`OS_USER_DOMAIN_NAME`

user-id
Users user ID

CLI options
`--os-user-id`

Environment variables
`OS_USER_ID`

username
Users username

CLI options
`--os-username`, `--os-user-name`

Environment variables
`OS_USERNAME`, `OS_USER_NAME`

3.2.10 v3fedkerb

auth-url
Authentication URL (**mandatory**)

CLI options
`--os-auth-url`

Environment variables
`OS_AUTH_URL`

domain-id
Domain ID to scope to

CLI options
`--os-domain-id`

Environment variables
`OS_DOMAIN_ID`

domain-name
Domain name to scope to

CLI options
`--os-domain-name`

Environment variables
`OS_DOMAIN_NAME`

identity-provider
Identity Providers name (**mandatory**)

CLI options
`--os-identity-provider`

Environment variables
`OS_IDENTITY_PROVIDER`

mutual-auth
Configures Kerberos Mutual Authentication

CLI options
`--os-mutual-auth`

Environment variables
`OS_MUTUAL_AUTH`

project-domain-id
Domain ID containing project

CLI options
`--os-project-domain-id`

Environment variables
`OS_PROJECT_DOMAIN_ID`

project-domain-name
Domain name containing project

CLI options
`--os-project-domain-name`

Environment variables
`OS_PROJECT_DOMAIN_NAME`

project-id
Project ID to scope to

CLI options
`--os-project-id`

Environment variables
`OS_PROJECT_ID`

project-name
Project name to scope to

CLI options
`--os-project-name`

Environment variables
`OS_PROJECT_NAME`

protocol
Protocol for federated plugin **(mandatory)**

CLI options
`--os-protocol`

Environment variables
`OS_PROTOCOL`

system-scope
Scope for system operations

CLI options
`--os-system-scope`

Environment variables
`OS_SYSTEM_SCOPE`

trust-id
ID of the trust to use as a trustee use

CLI options
`--os-trust-id`

Environment variables
`OS_TRUST_ID`

3.2.11 v3kerberos

auth-url
Authentication URL (**mandatory**)

CLI options
`--os-auth-url`

Environment variables
`OS_AUTH_URL`

domain-id
Domain ID to scope to

CLI options
`--os-domain-id`

Environment variables
`OS_DOMAIN_ID`

domain-name
Domain name to scope to

CLI options
`--os-domain-name`

Environment variables
`OS_DOMAIN_NAME`

mutual-auth
Configures Kerberos Mutual Authentication

CLI options
`--os-mutual-auth`

Environment variables
`OS_MUTUAL_AUTH`

project-domain-id
Domain ID containing project

CLI options
`--os-project-domain-id`

Environment variables
`OS_PROJECT_DOMAIN_ID`

project-domain-name
Domain name containing project

CLI options
`--os-project-domain-name`

Environment variables
`OS_PROJECT_DOMAIN_NAME`

project-id
Project ID to scope to

CLI options
`--os-project-id`

Environment variables
`OS_PROJECT_ID`

project-name
Project name to scope to

CLI options
`--os-project-name`

Environment variables
`OS_PROJECT_NAME`

system-scope
Scope for system operations

CLI options
`--os-system-scope`

Environment variables
`OS_SYSTEM_SCOPE`

trust-id
ID of the trust to use as a trustee use

CLI options
`--os-trust-id`

Environment variables
`OS_TRUST_ID`

3.2.12 v3multifactor

Authenticate using multiple factors.

Authenticate to the identity service using a combination of factors, such as username/password and a TOTP code.

---

auth-url
Authentication URL **(mandatory)**

CLI options
`--os-auth-url`

Environment variables
`OS_AUTH_URL`

auth_methods
Methods to authenticate with. **(mandatory)**

CLI options
`--os-auth_methods`

Environment variables
`OS_AUTH_METHODS`

domain-id
Domain ID to scope to

CLI options
`--os-domain-id`

Environment variables
`OS_DOMAIN_ID`

domain-name
Domain name to scope to

CLI options
`--os-domain-name`

Environment variables
`OS_DOMAIN_NAME`

project-domain-id
Domain ID containing project

CLI options
`--os-project-domain-id`

Environment variables
`OS_PROJECT_DOMAIN_ID`

project-domain-name
Domain name containing project

CLI options
`--os-project-domain-name`

Environment variables
`OS_PROJECT_DOMAIN_NAME`

project-id
Project ID to scope to

CLI options
`--os-project-id`

Environment variables
`OS_PROJECT_ID`

project-name
Project name to scope to

CLI options
`--os-project-name`

Environment variables
`OS_PROJECT_NAME`

system-scope
Scope for system operations

CLI options
`--os-system-scope`

Environment variables
`OS_SYSTEM_SCOPE`

trust-id
ID of the trust to use as a trustee use

CLI options
`--os-trust-id`

Environment variables
`OS_TRUST_ID`

3.2.13 v3oauth1

access-key
OAuth Access Key **(mandatory)**

CLI options
`--os-access-key`

Environment variables
`OS_ACCESS_KEY`

access-secret
OAuth Access Secret **(mandatory)**

CLI options
`--os-access-secret`

Environment variables
`OS_ACCESS_SECRET`

auth-url
Authentication URL **(mandatory)**

CLI options
`--os-auth-url`

Environment variables
`OS_AUTH_URL`

consumer-key
OAuth Consumer ID/Key **(mandatory)**

CLI options
`--os-consumer-key`

Environment variables
`OS_CONSUMER_KEY`

consumer-secret
OAuth Consumer Secret **(mandatory)**

CLI options
`--os-consumer-secret`

Environment variables
`OS_CONSUMER_SECRET`

3.2.14 v3oauth2clientcredential

Authenticate with an OAuth2.0 client credential.

---

auth-url
Authentication URL (**mandatory**)

CLI options
`--os-auth-url`

Environment variables
`OS_AUTH_URL`

domain-id
Domain ID to scope to

CLI options
`--os-domain-id`

Environment variables
`OS_DOMAIN_ID`

domain-name
Domain name to scope to

CLI options
`--os-domain-name`

Environment variables
`OS_DOMAIN_NAME`

oauth2_client_id
Client id for OAuth2.0 (**mandatory**)

CLI options
`--os-oauth2_client_id`

Environment variables
`OS_OAUTH2_CLIENT_ID`

oauth2_client_secret
Client secret for OAuth2.0 (**mandatory**)

CLI options
`--os-oauth2_client_secret`

Environment variables
`OS_OAUTH2_CLIENT_SECRET`

oauth2_endpoint
Endpoint for OAuth2.0 (**mandatory**)

CLI options
`--os-oauth2_endpoint`

Environment variables
`OS_OAUTH2_ENDPOINT`

project-domain-id
Domain ID containing project

CLI options
`--os-project-domain-id`

Environment variables
`OS_PROJECT_DOMAIN_ID`

project-domain-name
Domain name containing project

CLI options
`--os-project-domain-name`

Environment variables
`OS_PROJECT_DOMAIN_NAME`

project-id
Project ID to scope to

CLI options
`--os-project-id`

Environment variables
`OS_PROJECT_ID`

project-name
Project name to scope to

CLI options
`--os-project-name`

Environment variables
`OS_PROJECT_NAME`

system-scope
Scope for system operations

CLI options
`--os-system-scope`

Environment variables
`OS_SYSTEM_SCOPE`

trust-id
ID of the trust to use as a trustee use

CLI options
`--os-trust-id`

Environment variables
`OS_TRUST_ID`

3.2.15 v3oauth2mtlsclientcredential

Authenticate with an OAuth2.0 mTLS client credential.

---

auth-url
Authentication URL **(mandatory)**

CLI options
`--os-auth-url`

Environment variables
`OS_AUTH_URL`

domain-id
Domain ID to scope to

CLI options
`--os-domain-id`

Environment variables
`OS_DOMAIN_ID`

domain-name
Domain name to scope to

CLI options
`--os-domain-name`

Environment variables
`OS_DOMAIN_NAME`

oauth2-client-id
Client credential ID for OAuth2.0 Mutual-TLS Authorization **(mandatory)**

CLI options
`--os-oauth2-client-id`

Environment variables
`OS_OAUTH2_CLIENT_ID`

oauth2-endpoint
Endpoint for OAuth2.0 Mutual-TLS Authorization **(mandatory)**

CLI options
`--os-oauth2-endpoint`

Environment variables
`OS_OAUTH2_ENDPOINT`

project-domain-id
Domain ID containing project

CLI options
`--os-project-domain-id`

Environment variables
`OS_PROJECT_DOMAIN_ID`

project-domain-name
Domain name containing project

CLI options
`--os-project-domain-name`

Environment variables
`OS_PROJECT_DOMAIN_NAME`

project-id
Project ID to scope to

CLI options
`--os-project-id`

Environment variables
`OS_PROJECT_ID`

project-name
Project name to scope to

CLI options
`--os-project-name`

Environment variables
`OS_PROJECT_NAME`

system-scope
Scope for system operations

CLI options
`--os-system-scope`

Environment variables
`OS_SYSTEM_SCOPE`

trust-id
ID of the trust to use as a trustee use

CLI options
`--os-trust-id`

Environment variables
`OS_TRUST_ID`

3.2.16 v3oidcaccesstoken

Authenticate with the OIDC Access Token flow.

---

access-token
OAuth 2.0 Access Token **(mandatory)**

CLI options
`--os-access-token`

Environment variables
`OS_ACCESS_TOKEN`

auth-url
Authentication URL **(mandatory)**

CLI options
`--os-auth-url`

Environment variables
`OS_AUTH_URL`

domain-id
Domain ID to scope to

CLI options
`--os-domain-id`

Environment variables
`OS_DOMAIN_ID`

domain-name
Domain name to scope to

CLI options
`--os-domain-name`

Environment variables
`OS_DOMAIN_NAME`

identity-provider
Identity Providers name **(mandatory)**

CLI options
`--os-identity-provider`

Environment variables
`OS_IDENTITY_PROVIDER`

project-domain-id
Domain ID containing project

CLI options
`--os-project-domain-id`

Environment variables
`OS_PROJECT_DOMAIN_ID`

project-domain-name
Domain name containing project

CLI options
`--os-project-domain-name`

Environment variables
`OS_PROJECT_DOMAIN_NAME`

project-id
Project ID to scope to

CLI options
`--os-project-id`

Environment variables
`OS_PROJECT_ID`

project-name
Project name to scope to

CLI options
`--os-project-name`

Environment variables
`OS_PROJECT_NAME`

protocol
Protocol for federated plugin **(mandatory)**

CLI options
`--os-protocol`

Environment variables
`OS_PROTOCOL`

system-scope
Scope for system operations

CLI options
`--os-system-scope`

Environment variables
`OS_SYSTEM_SCOPE`

trust-id
ID of the trust to use as a trustee use

CLI options
`--os-trust-id`

Environment variables
`OS_TRUST_ID`

3.2.17 v3oidcauthcode

Authenticate with the OIDC Authorization Code flow.

---

access-token-endpoint
OpenID Connect Provider Token Endpoint. Note that if a discovery document is being passed this option will override the endpoint provided by the server in the discovery document.

CLI options
`--os-access-token-endpoint`

Environment variables
`OS_ACCESS_TOKEN_ENDPOINT`

access-token-type
OAuth 2.0 Authorization Server Introspection token type, it is used to decide which type of token will be used when processing token introspection. Valid values are: access_token or id_token

CLI options
`--os-access-token-type`

Environment variables
`OS_ACCESS_TOKEN_TYPE`

auth-url
Authentication URL **(mandatory)**

CLI options
`--os-auth-url`

Environment variables
`OS_AUTH_URL`

client-id
OAuth 2.0 Client ID

CLI options
`--os-client-id`

Environment variables
`OS_CLIENT_ID`

client-secret
OAuth 2.0 Client Secret

CLI options
`--os-client-secret`

Environment variables
`OS_CLIENT_SECRET`

code
OAuth 2.0 Authorization Code **(mandatory)**

CLI options
`--os-code`, `--os-authorization-code`

Environment variables
`OS_CODE`, `OS_AUTHORIZATION_CODE`

discovery-endpoint
OpenID Connect Discovery Document URL. The discovery document will be used to obtain the values of the access token endpoint and the authentication endpoint. This URL should look like https://idp.example.org/.well-known/openid-configuration

CLI options
`--os-discovery-endpoint`

Environment variables
`OS_DISCOVERY_ENDPOINT`

domain-id
Domain ID to scope to

CLI options
`--os-domain-id`

Environment variables
`OS_DOMAIN_ID`

domain-name
Domain name to scope to

CLI options
`--os-domain-name`

Environment variables
`OS_DOMAIN_NAME`

identity-provider
Identity Providers name (**mandatory**)

CLI options
`--os-identity-provider`

Environment variables
`OS_IDENTITY_PROVIDER`

openid-scope
OpenID Connect scope that is requested from authorization server. Note that the OpenID Connect specification states that openid must be always specified.

CLI options
`--os-openid-scope`

Environment variables
`OS_OPENID_SCOPE`

project-domain-id
Domain ID containing project

CLI options
`--os-project-domain-id`

Environment variables
`OS_PROJECT_DOMAIN_ID`

project-domain-name
Domain name containing project

CLI options
`--os-project-domain-name`

Environment variables
`OS_PROJECT_DOMAIN_NAME`

project-id
Project ID to scope to

CLI options
`--os-project-id`

Environment variables
`OS_PROJECT_ID`

project-name
Project name to scope to

CLI options
`--os-project-name`

Environment variables
`OS_PROJECT_NAME`

protocol
Protocol for federated plugin (**mandatory**)

CLI options
`--os-protocol`

Environment variables
`OS_PROTOCOL`

redirect-uri
OpenID Connect Redirect URL

CLI options
`--os-redirect-uri`

Environment variables
`OS_REDIRECT_URI`

system-scope
Scope for system operations

CLI options
`--os-system-scope`

Environment variables
`OS_SYSTEM_SCOPE`

trust-id
ID of the trust to use as a trustee use

CLI options
`--os-trust-id`

Environment variables
`OS_TRUST_ID`

3.2.18 v3oidcclientcredentials

Authenticate with the OIDC Client Credentials flow.

---

access-token-endpoint
OpenID Connect Provider Token Endpoint. Note that if a discovery document is being passed this option will override the endpoint provided by the server in the discovery document.

CLI options
`--os-access-token-endpoint`

Environment variables
`OS_ACCESS_TOKEN_ENDPOINT`

access-token-type
OAuth 2.0 Authorization Server Introspection token type, it is used to decide which type of token will be used when processing token introspection. Valid values are: access_token or id_token

CLI options
`--os-access-token-type`

Environment variables
`OS_ACCESS_TOKEN_TYPE`

auth-url
Authentication URL **(mandatory)**

CLI options
`--os-auth-url`

Environment variables
`OS_AUTH_URL`

client-id
OAuth 2.0 Client ID

CLI options
`--os-client-id`

Environment variables
`OS_CLIENT_ID`

client-secret
OAuth 2.0 Client Secret

CLI options
`--os-client-secret`

Environment variables
`OS_CLIENT_SECRET`

discovery-endpoint
OpenID Connect Discovery Document URL. The discovery document will be used to obtain the values of the access token endpoint and the authentication endpoint. This URL should look like https://idp.example.org/.well-known/openid-configuration

CLI options
`--os-discovery-endpoint`

Environment variables
`OS_DISCOVERY_ENDPOINT`

domain-id
Domain ID to scope to

CLI options
`--os-domain-id`

Environment variables
`OS_DOMAIN_ID`

domain-name
Domain name to scope to

CLI options
`--os-domain-name`

Environment variables
`OS_DOMAIN_NAME`

identity-provider
Identity Providers name **(mandatory)**

CLI options
`--os-identity-provider`

Environment variables
`OS_IDENTITY_PROVIDER`

openid-scope
OpenID Connect scope that is requested from authorization server. Note that the OpenID Connect specification states that openid must be always specified.

CLI options
`--os-openid-scope`

Environment variables
`OS_OPENID_SCOPE`

project-domain-id
Domain ID containing project

CLI options
`--os-project-domain-id`

Environment variables
`OS_PROJECT_DOMAIN_ID`

project-domain-name
Domain name containing project

CLI options
`--os-project-domain-name`

Environment variables
`OS_PROJECT_DOMAIN_NAME`

project-id
Project ID to scope to

CLI options
`--os-project-id`

Environment variables
`OS_PROJECT_ID`

project-name
Project name to scope to

CLI options
`--os-project-name`

Environment variables
`OS_PROJECT_NAME`

protocol
Protocol for federated plugin **(mandatory)**

CLI options
`--os-protocol`

Environment variables
`OS_PROTOCOL`

system-scope
Scope for system operations

CLI options
`--os-system-scope`

Environment variables
`OS_SYSTEM_SCOPE`

trust-id
ID of the trust to use as a trustee use

CLI options
`--os-trust-id`

Environment variables
`OS_TRUST_ID`

3.2.19 v3oidcdeviceauthz

Authenticate with the OAuth 2.0 Device Authorization flow.

---

access-token-endpoint
OpenID Connect Provider Token Endpoint. Note that if a discovery document is being passed this option will override the endpoint provided by the server in the discovery document.

CLI options
`--os-access-token-endpoint`

Environment variables
`OS_ACCESS_TOKEN_ENDPOINT`

auth-url
Authentication URL (**mandatory**)

CLI options
`--os-auth-url`

Environment variables
`OS_AUTH_URL`

client-id
OAuth 2.0 Client ID

CLI options
`--os-client-id`

Environment variables
`OS_CLIENT_ID`

client-secret
OAuth 2.0 Client Secret

CLI options
`--os-client-secret`

Environment variables
`OS_CLIENT_SECRET`

code-challenge-method
PKCE Challenge Method (RFC 7636)

CLI options
`--os-code-challenge-method`

Environment variables
`OS_CODE_CHALLENGE_METHOD`

device-authorization-endpoint
OAuth 2.0 Device Authorization Endpoint. Note that if a discovery document is being passed this option will override the endpoint provided by the server in the discovery document.

CLI options
`--os-device-authorization-endpoint`

Environment variables
`OS_DEVICE_AUTHORIZATION_ENDPOINT`

discovery-endpoint
OpenID Connect Discovery Document URL. The discovery document will be used to obtain the values of the access token endpoint and the authentication endpoint. This URL should look like https://idp.example.org/.well-known/openid-configuration

CLI options
`--os-discovery-endpoint`

Environment variables
`OS_DISCOVERY_ENDPOINT`

domain-id
Domain ID to scope to

CLI options
`--os-domain-id`

Environment variables
`OS_DOMAIN_ID`

domain-name
Domain name to scope to

CLI options
`--os-domain-name`

Environment variables
`OS_DOMAIN_NAME`

identity-provider
Identity Providers name (**mandatory**)

CLI options
`--os-identity-provider`

Environment variables
`OS_IDENTITY_PROVIDER`

openid-scope
OpenID Connect scope that is requested from authorization server. Note that the OpenID Connect specification states that openid must be always specified.

CLI options
`--os-openid-scope`

Environment variables
`OS_OPENID_SCOPE`

project-domain-id
Domain ID containing project

CLI options
`--os-project-domain-id`

Environment variables
`OS_PROJECT_DOMAIN_ID`

project-domain-name
Domain name containing project

CLI options
`--os-project-domain-name`

Environment variables
`OS_PROJECT_DOMAIN_NAME`

project-id
Project ID to scope to

CLI options
`--os-project-id`

Environment variables
`OS_PROJECT_ID`

project-name
Project name to scope to

CLI options
`--os-project-name`

Environment variables
`OS_PROJECT_NAME`

protocol
Protocol for federated plugin **(mandatory)**

CLI options
`--os-protocol`

Environment variables
`OS_PROTOCOL`

system-scope
Scope for system operations

CLI options
`--os-system-scope`

Environment variables
`OS_SYSTEM_SCOPE`

trust-id
ID of the trust to use as a trustee use

CLI options
`--os-trust-id`

Environment variables
`OS_TRUST_ID`

3.2.20 v3oidcpassword

Authenticate with the OIDC Resource Owner Password Credentials flow.

---

access-token-endpoint
OpenID Connect Provider Token Endpoint. Note that if a discovery document is being passed this option will override the endpoint provided by the server in the discovery document.

CLI options
`--os-access-token-endpoint`

Environment variables
`OS_ACCESS_TOKEN_ENDPOINT`

access-token-type
OAuth 2.0 Authorization Server Introspection token type, it is used to decide which type of token will be used when processing token introspection. Valid values are: access_token or id_token

CLI options
`--os-access-token-type`

Environment variables
`OS_ACCESS_TOKEN_TYPE`

auth-url
Authentication URL (**mandatory**)

CLI options
`--os-auth-url`

Environment variables
`OS_AUTH_URL`

client-id
OAuth 2.0 Client ID

CLI options
`--os-client-id`

Environment variables
`OS_CLIENT_ID`

client-secret
OAuth 2.0 Client Secret

CLI options
`--os-client-secret`

Environment variables
`OS_CLIENT_SECRET`

discovery-endpoint
OpenID Connect Discovery Document URL. The discovery document will be used to obtain the values of the access token endpoint and the authentication endpoint. This URL should look like https://idp.example.org/.well-known/openid-configuration

CLI options
`--os-discovery-endpoint`

Environment variables
`OS_DISCOVERY_ENDPOINT`

domain-id
Domain ID to scope to

CLI options
`--os-domain-id`

Environment variables
`OS_DOMAIN_ID`

domain-name
Domain name to scope to

CLI options
`--os-domain-name`

Environment variables
`OS_DOMAIN_NAME`

identity-provider
Identity Providers name (**mandatory**)

CLI options
`--os-identity-provider`

Environment variables
`OS_IDENTITY_PROVIDER`

idp_otp_key
A key to be used in the Identity Provider access token endpoint to pass the OTP value. E.g. totp

CLI options
`--os-idp_otp_key`

Environment variables
`OS_IDP_OTP_KEY`

openid-scope
OpenID Connect scope that is requested from authorization server. Note that the OpenID Connect specification states that openid must be always specified.

CLI options
`--os-openid-scope`

Environment variables
`OS_OPENID_SCOPE`

password
Password (**mandatory**)

CLI options
`--os-password`

Environment variables
`OS_PASSWORD`

project-domain-id
Domain ID containing project

CLI options
`--os-project-domain-id`

Environment variables
`OS_PROJECT_DOMAIN_ID`

project-domain-name
Domain name containing project

CLI options
`--os-project-domain-name`

Environment variables
`OS_PROJECT_DOMAIN_NAME`

project-id
Project ID to scope to

CLI options
`--os-project-id`

Environment variables
`OS_PROJECT_ID`

project-name
Project name to scope to

CLI options
`--os-project-name`

Environment variables
`OS_PROJECT_NAME`

protocol
Protocol for federated plugin (**mandatory**)

CLI options
`--os-protocol`

Environment variables
`OS_PROTOCOL`

system-scope
Scope for system operations

CLI options
`--os-system-scope`

Environment variables
`OS_SYSTEM_SCOPE`

trust-id
ID of the trust to use as a trustee use

CLI options
`--os-trust-id`

Environment variables
`OS_TRUST_ID`

username
Username **(mandatory)**

CLI options
`--os-username`

Environment variables
`OS_USERNAME`

3.2.21 v3password

Authenticate with a username and password.

Authenticate to the identity service using the provided username and password. This is the standard and most common form of authentication.

---

auth-url
Authentication URL **(mandatory)**

CLI options
`--os-auth-url`

Environment variables
`OS_AUTH_URL`

domain-id
Domain ID to scope to

CLI options
`--os-domain-id`

Environment variables
`OS_DOMAIN_ID`

domain-name
Domain name to scope to

CLI options
`--os-domain-name`

Environment variables
`OS_DOMAIN_NAME`

password
Users password

CLI options
`--os-password`

Environment variables
`OS_PASSWORD`

project-domain-id
Domain ID containing project

CLI options
`--os-project-domain-id`

Environment variables
`OS_PROJECT_DOMAIN_ID`

project-domain-name
Domain name containing project

CLI options
`--os-project-domain-name`

Environment variables
`OS_PROJECT_DOMAIN_NAME`

project-id
Project ID to scope to

CLI options
`--os-project-id`

Environment variables
`OS_PROJECT_ID`

project-name
Project name to scope to

CLI options
`--os-project-name`

Environment variables
`OS_PROJECT_NAME`

system-scope
Scope for system operations

CLI options
`--os-system-scope`

Environment variables
`OS_SYSTEM_SCOPE`

trust-id
ID of the trust to use as a trustee use

CLI options
`--os-trust-id`

Environment variables
`OS_TRUST_ID`

user-domain-id
Users domain ID

CLI options
`--os-user-domain-id`

Environment variables
`OS_USER_DOMAIN_ID`

user-domain-name
Users domain name

CLI options
`--os-user-domain-name`

Environment variables
`OS_USER_DOMAIN_NAME`

user-id
Users user ID

CLI options
`--os-user-id`

Environment variables
`OS_USER_ID`

username
Users username

CLI options
`--os-username`, `--os-user-name`

Environment variables
`OS_USERNAME`, `OS_USER_NAME`

3.2.22 v3samlpassword

auth-url
Authentication URL **(mandatory)**

CLI options
`--os-auth-url`

Environment variables
`OS_AUTH_URL`

domain-id
Domain ID to scope to

CLI options
`--os-domain-id`

Environment variables
`OS_DOMAIN_ID`

domain-name
Domain name to scope to

CLI options
`--os-domain-name`

Environment variables
`OS_DOMAIN_NAME`

identity-provider
Identity Providers name **(mandatory)**

CLI options
`--os-identity-provider`

Environment variables
`OS_IDENTITY_PROVIDER`

identity-provider-url
An Identity Provider URL, where the SAML2 authentication request will be sent. **(mandatory)**

CLI options
`--os-identity-provider-url`

Environment variables
`OS_IDENTITY_PROVIDER_URL`

password
Password **(mandatory)**

CLI options
`--os-password`

Environment variables
`OS_PASSWORD`

project-domain-id
Domain ID containing project

CLI options
`--os-project-domain-id`

Environment variables
`OS_PROJECT_DOMAIN_ID`

project-domain-name
Domain name containing project

CLI options
`--os-project-domain-name`

Environment variables
`OS_PROJECT_DOMAIN_NAME`

project-id
Project ID to scope to

CLI options
`--os-project-id`

Environment variables
`OS_PROJECT_ID`

project-name
Project name to scope to

CLI options
`--os-project-name`

Environment variables
`OS_PROJECT_NAME`

protocol
Protocol for federated plugin **(mandatory)**

CLI options
`--os-protocol`

Environment variables
`OS_PROTOCOL`

system-scope
Scope for system operations

CLI options
`--os-system-scope`

Environment variables
`OS_SYSTEM_SCOPE`

trust-id
ID of the trust to use as a trustee use

CLI options
`--os-trust-id`

Environment variables
`OS_TRUST_ID`

username
Username **(mandatory)**

CLI options
`--os-username`

Environment variables
`OS_USERNAME`

3.2.23 v3token

Given an existing token rescope it to another target.

Use the Identity services rescope mechanism to get a new token based upon an existing token. Because an auth plugin requires a service catalog and scope information it is often easier to fetch a new token based on an existing one than validate and reuse the one you already have.

---

auth-url
Authentication URL **(mandatory)**

CLI options
`--os-auth-url`

Environment variables
`OS_AUTH_URL`

domain-id
Domain ID to scope to

CLI options
`--os-domain-id`

Environment variables
`OS_DOMAIN_ID`

domain-name
Domain name to scope to

CLI options
`--os-domain-name`

Environment variables
`OS_DOMAIN_NAME`

project-domain-id
Domain ID containing project

CLI options
`--os-project-domain-id`

Environment variables
`OS_PROJECT_DOMAIN_ID`

project-domain-name
Domain name containing project

CLI options
`--os-project-domain-name`

Environment variables
`OS_PROJECT_DOMAIN_NAME`

project-id
Project ID to scope to

CLI options
`--os-project-id`

Environment variables
`OS_PROJECT_ID`

project-name
Project name to scope to

CLI options
`--os-project-name`

Environment variables
`OS_PROJECT_NAME`

system-scope
Scope for system operations

CLI options
`--os-system-scope`

Environment variables
`OS_SYSTEM_SCOPE`

token
Token to authenticate with

CLI options
`--os-token`

Environment variables
`OS_TOKEN`

trust-id
ID of the trust to use as a trustee use

CLI options
`--os-trust-id`

Environment variables
`OS_TRUST_ID`

3.2.24 v3tokenlessauth

Authenticate without a token, using an X.509 certificate.

---

auth-url
Authentication URL (**mandatory**)

CLI options
`--os-auth-url`

Environment variables
`OS_AUTH_URL`

domain-id
Domain ID to scope to

CLI options
`--os-domain-id`

Environment variables
`OS_DOMAIN_ID`

domain-name
Domain name to scope to

CLI options
`--os-domain-name`

Environment variables
`OS_DOMAIN_NAME`

project-domain-id
Domain ID containing project

CLI options
`--os-project-domain-id`

Environment variables
`OS_PROJECT_DOMAIN_ID`

project-domain-name
Domain name containing project

CLI options
`--os-project-domain-name`

Environment variables
`OS_PROJECT_DOMAIN_NAME`

project-id
Project ID to scope to

CLI options
`--os-project-id`

Environment variables
`OS_PROJECT_ID`

project-name
Project name to scope to

CLI options
`--os-project-name`

Environment variables
`OS_PROJECT_NAME`

3.2.25 v3totp

Authenticate with a Time-based One-Time Password.

Authenticate to the identity service using a time-based one-time password. This is typically used in combination with another plugin as part of a multi-factor configuration.

---

auth-url
Authentication URL **(mandatory)**

CLI options
`--os-auth-url`

Environment variables
`OS_AUTH_URL`

domain-id
Domain ID to scope to

CLI options
`--os-domain-id`

Environment variables
`OS_DOMAIN_ID`

domain-name
Domain name to scope to

CLI options
`--os-domain-name`

Environment variables
`OS_DOMAIN_NAME`

passcode
Users TOTP passcode

CLI options
`--os-passcode`

Environment variables
`OS_PASSCODE`

project-domain-id
Domain ID containing project

CLI options
`--os-project-domain-id`

Environment variables
`OS_PROJECT_DOMAIN_ID`

project-domain-name
Domain name containing project

CLI options
`--os-project-domain-name`

Environment variables
`OS_PROJECT_DOMAIN_NAME`

project-id
Project ID to scope to

CLI options
`--os-project-id`

Environment variables
`OS_PROJECT_ID`

project-name
Project name to scope to

CLI options
`--os-project-name`

Environment variables
`OS_PROJECT_NAME`

system-scope
Scope for system operations

CLI options
`--os-system-scope`

Environment variables
`OS_SYSTEM_SCOPE`

trust-id
ID of the trust to use as a trustee use

CLI options
`--os-trust-id`

Environment variables
`OS_TRUST_ID`

user-domain-id
Users domain ID

CLI options
`--os-user-domain-id`

Environment variables
`OS_USER_DOMAIN_ID`

user-domain-name
Users domain name

CLI options
`--os-user-domain-name`

Environment variables
`OS_USER_DOMAIN_NAME`

user-id
Users user ID

CLI options
`--os-user-id`

Environment variables
`OS_USER_ID`

username
Users username

CLI options
`--os-username`, `--os-user-name`

Environment variables
`OS_USERNAME`, `OS_USER_NAME`

3.3 Additional Plugins

keystoneauth is designed to be pluggable and Python packages exist that provide additional plugins.

CHAPTER FOUR

EXTRAS

The extensibility of keystoneauth plugins is purposefully designed to allow a range of different authentication mechanisms that dont have to reside in the upstream packages. There are however a number of plugins that upstream supports that involve additional dependencies that the keystoneauth package cannot depend upon directly.

To get around this we utilize setuptools extras dependencies for additional plugins. To use a plugin like the kerberos plugin that has additional dependencies you must install the additional dependencies like:

```
pip install keystoneauth1[kerberos]
```

By convention (not a requirement) extra plugins have a module located in the keystoneauth1.extras module with the same name as the dependency. eg:

```
from keystoneauth1.extras import kerberos
```

There is no keystoneauth specific check that the correct dependencies are installed for accessing a module. You would expect to see standard python ImportError when the required dependencies are not found.

4.1 Examples

All extras plugins follow the pattern:

1. import plugin module
2. instantiate the plugin
3. call get_token method of the plugin passing it a session object to get a token

4.1.1 Kerberos

Get domain-scoped token using `Kerberos`:

```
from keystoneauth1.extras import kerberos
from keystoneauth1 import session

plugin = kerberos.Kerberos('http://example.com:5000/v3')
sess = session.Session(plugin)
token = plugin.get_token(sess)
```

Get unscoped federated token:

```
from keystoneauth1.extras import kerberos
from keystoneauth1 import session

plugin = kerberos.MappedKerberos(
    auth_url='http://example.com:5000/v3', protocol='example_protocol',
    identity_provider='example_identity_provider')

sess = session.Session()
token = plugin.get_token(sess)
```

Get project scoped federated token:

```
from keystoneauth1.extras import kerberos
from keystoneauth1 import session

plugin = kerberos.MappedKerberos(
    auth_url='http://example.com:5000/v3', protocol='example_protocol',
    identity_provider='example_identity_provider',
    project_id='example_project_id')

sess = session.Session()
token = plugin.get_token(sess)
project_id = plugin.get_project_id(sess)
```

CHAPTER

FIVE

MIGRATING FROM KEYSTONECLIENT

When keystoneauth was extracted from keystoneclient the basic usage of the session, adapter and auth plugins purposefully did not change. If you are using them in a supported fashion from keystoneclient then the transition should be fairly simple.

5.1 Authentication Plugins

The authentication plugins themselves changed very little however there were changes to the way plugins are loaded and some of the supporting classes.

5.1.1 Plugin Loading

In keystoneclient auth plugin loading is managed by the class itself. This method proved useful in allowing the plugin to control the way it was loaded however it linked the authentication logic with the config and CLI loading.

In keystoneauth this has been severed and the auth plugin is handled separately from the mechanism that loads it.

Authentication plugins still implement the base authentication class `BaseAuthPlugin`. To make the plugins capable of being loaded from CLI or CONF file you should implement the base `BaseLoader` class which is loaded when *os-auth-type* is used. This class handles the options that are presented, and then constructs the authentication plugin for use by the application.

Largely the options that are returned will be the same as what was used in keystoneclient however in keystoneclient the options used `oslo_config.cfg.Opt` objects. Due to trying to keep minimal dependencies there is no direct dependency from keystoneauth on oslo.config and instead options should be specified as `Opt` objects.

To ensure distinction between the plugins, the setuptools entrypoints that plugins register at has been updated to reflect keystoneauth1 and should now be: keystoneauth1.plugin

5.1.2 AccessInfo Objects

AccessInfo objects are a representation of the information stored within a token. In keystoneclient these objects were dictionaries of the token data with property accessors. In keystoneauth the dictionary interface has been removed and just the property accessors are available.

The creation function has also changed. The `keystoneclient.access.AccessInfo.factory()` method has been removed and replaced with the `keystoneauth1.access.create()`.

5.1.3 Step-by-step migration example

Add `keystoneauth1` to requirements.txt

In the code do the following change:

```
-from keystoneclient import auth
+from keystoneauth1 import plugin
```

consequently:

```
-auth.BaseAuthPlugin
+plugin.BaseAuthPlugin
```

To import service catalog:

```
-from keystoneclient import service_catalog
+from keystoneauth1.access import service_catalog
```

To get url using service catalog *endpoint_type* parameter was changed to *interface*:

```
-service_catalog.ServiceCatalogV2(sc).service_catalog.url_for(..., endpoint_
↪type=interface)
+service_catalog.ServiceCatalogV2(sc).service_catalog.url_for(...,␣
↪interface=interface)
```

Obtaining the session:

```
-from keystoneclient import session
+from keystoneauth1 import loading as ks_loading

-_SESSION = session.Session.load_from_conf_options(
-auth_plugin = auth.load_from_conf_options(conf, NEUTRON_GROUP)
+_SESSION = ks_loading.load_session_from_conf_options(
+auth_plugin = ks_loading.load_auth_from_conf_options(conf, NEUTRON_GROUP)
```

Mocking session for test purposes:

```
-@mock.patch('keystoneclient.session.Session')
+@mock.patch('keystoneauth1.session.Session')
```

Token fixture imports havent change much:

```
-from keystoneclient.fixture import V2Token
+from keystoneauth1.fixture import V2Token
```

CHAPTER

SIX

KEYSTONEAUTH1

6.1 keystoneauth1 package

6.1.1 Subpackages

keystoneauth1.access package

Submodules

keystoneauth1.access.access module

class keystoneauth1.access.access.**AccessInfo**(*body: dict[str, Any], auth_token: str | None = None*)

Bases: `object`

Encapsulates a raw authentication token from keystone.

Provides helper methods for extracting useful values from that token.

`__annotations__ = {'_data': typing.Any, '_service_catalog': 'ty.Optional[service_catalog.ServiceCatalog]', '_service_catalog_class': typing.Type[keystoneauth1.access.service_catalog.ServiceCatalog], '_service_providers': 'ty.Optional[service_providers.ServiceProviders]'}`

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.access', '__annotations__': {'_service_catalog_class': typing.Type[keystoneauth1.access.service_catalog.ServiceCatalog], '_data': typing.Any, '_service_catalog': 'ty.Optional[service_catalog.ServiceCatalog]', '_service_providers': 'ty.Optional[service_providers.ServiceProviders]'}, '__doc__': 'Encapsulates a raw authentication token from keystone.\n\n Provides helper methods for extracting useful values from that token.\n ', '__init__': <function AccessInfo.__init__>, 'service_catalog': <property object>, 'will_expire_soon': <function AccessInfo.will_expire_soon>, 'has_service_catalog': <function AccessInfo.has_service_catalog>, 'auth_token': <property object>, 'expires': <property object>, 'issued': <property object>, 'username': <property object>, 'user_id': <property object>, 'user_domain_id': <property object>, 'user_domain_name': <property object>, 'role_ids': <property object>, 'role_names': <property object>, 'domain_name': <property object>, 'domain_id': <property object>, 'project_name': <property object>, 'tenant_name': <property object>, 'scoped': <property object>, 'project_scoped': <property object>, 'domain_scoped': <property object>, 'system_scoped': <property object>, 'trust_id': <property object>, 'trust_scoped': <property object>, 'trustee_user_id': <property object>, 'trustor_user_id': <property object>, 'project_id': <property object>, 'tenant_id': <property object>, 'project_domain_id': <property object>, 'project_domain_name': <property object>, 'oauth_access_token_id': <property object>, 'oauth_consumer_id': <property object>, 'is_federated': <property object>, 'is_admin_project': <property object>, 'audit_id': <property object>, 'audit_chain_id': <property object>, 'initial_audit_id': <property object>, 'service_providers': <property object>, 'bind': <property object>, 'project_is_domain': <property object>, '__dict__': <attribute '__dict__' of 'AccessInfo' objects>, '__weakref__': <attribute '__weakref__' of 'AccessInfo' objects>})`

`__doc__ = 'Encapsulates a raw authentication token from keystone.\n\n Provides helper methods for extracting useful values from that token.\n '`

`__init__`(*body: dict[str, Any], auth_token: str | None = None*)

`__module__ = 'keystoneauth1.access.access'`

`__weakref__`

list of weak references to the object (if defined)

`_data: Any`

`_service_catalog: ServiceCatalog | None`

`_service_catalog_class: Type[ServiceCatalog]`

`_service_providers: ServiceProviders | None`

`property audit_chain_id: str | None`

Return the audit chain ID if present.

In the event that a token was rescoped then this ID will be the `audit_id` of the initial token. Returns None if no value present.

Returns
str or None.

property audit_id: str | None

Return the audit ID if present.

Returns
str or None.

property auth_token: str | None

Return the token_id associated with the auth request.

To be used in headers for authenticating OpenStack API requests.

Returns
str

property bind: dict[str, Any] | None

Information about external mechanisms the token is bound to.

If a token is bound to an external authentication mechanism it can only be used in conjunction with that mechanism. For example if bound to a kerberos principal it may only be accepted if there is also kerberos authentication performed on the request.

Returns
A dictionary or None. The key will be the bind type the value is a dictionary that is specific to the format of the bind type. Returns None if there is no bind information in the token.

property domain_id: str | None

Return the domain id associated with the auth request.

Returns
str or None (if no domain associated with the token)

property domain_name: str | None

Return the domain name associated with the auth request.

Returns
str or None (if no domain associated with the token)

property domain_scoped: bool

Return true if the auth token was scoped to a domain.

Returns
bool

property expires: datetime | None

Return the token expiration (as datetime object).

Returns
datetime

has_service_catalog() → bool

Return true if the auth token has a service catalog.

Returns
boolean

property initial_audit_id: str | None

The audit ID of the initially requested token.

This is the `audit_chain_id` if present or the `audit_id`.

property is_admin_project: bool

Return true if the current project scope is the admin project.

For backwards compatibility purposes if there is nothing specified in the token we always assume we are in the admin project, so this will default to True.

:returns boolean

property is_federated: bool

Return true if federation was used to get the token.

Returns
boolean

property issued: datetime | None

Return the token issue time (as datetime object).

Returns
datetime

property oauth_access_token_id: str | None

Return the access token ID if OAuth authentication used.

Returns
str or None.

property oauth_consumer_id: str | None

Return the consumer ID if OAuth authentication used.

Returns
str or None.

property project_domain_id: str | None

Return the projects domain id associated with the auth request.

Returns
str

property project_domain_name: str | None

Return the projects domain name associated with the auth request.

Returns
str

property project_id: str | None

Return the project ID associated with the auth request.

This returns None if the auth token wasnt scoped to a project.

Returns
str or None (if no project associated with the token)

property project_is_domain: bool | None

Return if a project act as a domain.

Returns
bool

property project_name: str | None

Return the project name associated with the auth request.

Returns
str or None (if no project associated with the token)

property project_scoped: bool

Return true if the auth token was scoped to a tenant (project).

Returns
bool

property role_ids: list[str] | None

Return a list of users role ids associated with the auth request.

Returns
a list of strings of role ids

property role_names: list[str] | None

Return a list of users role names associated with the auth request.

Returns
a list of strings of role names

property scoped: bool

Return true if the auth token was scoped.

Returns true if scoped to a tenant(project) or domain, and contains a populated service catalog.

This is deprecated, use project_scoped instead.

Returns
bool

property service_catalog: ServiceCatalog

property service_providers: ServiceProviders | None

Return an object representing the list of trusted service providers.

Used for Keystone2Keystone federating-out.

Returns
`keystoneauth1.service_providers.ServiceProviders` or None

property system_scoped: bool

Return true if the auth token was scoped to the system.

Returns
bool

property tenant_id: str | None

Synonym for project_id.

property tenant_name: str | None

Synonym for project_name.

property trust_id: str | None

Return the trust id associated with the auth request.

Returns
str or None (if no trust associated with the token)

property trust_scoped: bool

Return true if the auth token was scoped from a delegated trust.

The trust delegation is via the OS-TRUST v3 extension.

Returns
bool

property trustee_user_id: str | None

Return the trustee user id associated with a trust.

Returns
str or None (if no trust associated with the token)

property trustor_user_id: str | None

Return the trustor user id associated with a trust.

Returns
str or None (if no trust associated with the token)

property user_domain_id: str | None

Return the users domain id associated with the auth request.

Returns
str

property user_domain_name: str | None

Return the users domain name associated with the auth request.

Returns
str

property user_id: str | None

Return the user id associated with the auth request.

Returns
str

property username: str | None

Return the username associated with the auth request.

Follows the pattern defined in the V2 API of first looking for name, returning that if available, and falling back to username if name is unavailable.

Returns
str

will_expire_soon(*stale_duration: int = 30*) → bool

Determine if expiration is about to occur.

Returns
true if expiration is within the given duration

Return type
boolean

class `keystoneauth1.access.access.AccessInfoV2`(*body: dict[str, Any]*, *auth_token: str | None = None*)

Bases: `AccessInfo`

An object for encapsulating raw v2 auth token from identity service.

`__annotations__ = {'_data': <class 'keystoneauth1.access.types.TokenResponseV2'>, '_service_catalog': 'ty.Optional[service_catalog.ServiceCatalog]', '_service_catalog_class': 'ty.Type[service_catalog.ServiceCatalog]', '_service_providers': 'ty.Optional[service_providers.ServiceProviders]'}`

`__doc__ = 'An object for encapsulating raw v2 auth token from identity service.'`

`__module__ = 'keystoneauth1.access.access'`

`_data: TokenResponseV2`

`_service_catalog: ServiceCatalog | None`

`_service_catalog_class`

alias of `ServiceCatalogV2`

`_service_providers: ServiceProviders | None`

`property _token: TokenV2`

`property _trust: TrustV2 | None`

`property _user: UserV2`

`property audit_chain_id: str | None`

Return the audit chain ID if present.

In the event that a token was rescoped then this ID will be the `audit_id` of the initial token. Returns None if no value present.

Returns
str or None.

`property audit_id: str | None`

Return the audit ID if present.

Returns
str or None.

`property auth_token: str | None`

Return the token_id associated with the auth request.

To be used in headers for authenticating OpenStack API requests.

Returns
str

property bind: dict[str, Any] | None

Information about external mechanisms the token is bound to.

If a token is bound to an external authentication mechanism it can only be used in conjunction with that mechanism. For example if bound to a kerberos principal it may only be accepted if there is also kerberos authentication performed on the request.

Returns
A dictionary or None. The key will be the bind type the value is a dictionary that is specific to the format of the bind type. Returns None if there is no bind information in the token.

property domain_id: str | None

Return the domain id associated with the auth request.

Returns
str or None (if no domain associated with the token)

property domain_name: str | None

Return the domain name associated with the auth request.

Returns
str or None (if no domain associated with the token)

property domain_scoped: bool

Return true if the auth token was scoped to a domain.

Returns
bool

property expires: datetime | None

Return the token expiration (as datetime object).

Returns
datetime

has_service_catalog() → bool

Return true if the auth token has a service catalog.

Returns
boolean

property is_admin_project: bool

Return true if the current project scope is the admin project.

For backwards compatibility purposes if there is nothing specified in the token we always assume we are in the admin project, so this will default to True.

:returns boolean

property is_federated: bool

Return true if federation was used to get the token.

Returns
boolean

property issued: datetime | None

Return the token issue time (as datetime object).

Returns
datetime

property oauth_access_token_id: str | None

Return the access token ID if OAuth authentication used.

Returns
str or None.

property oauth_consumer_id: str | None

Return the consumer ID if OAuth authentication used.

Returns
str or None.

property project_domain_id: str | None

Return the projects domain id associated with the auth request.

Returns
str

property project_domain_name: str | None

Return the projects domain name associated with the auth request.

Returns
str

property project_id: str | None

Return the project ID associated with the auth request.

This returns None if the auth token wasnt scoped to a project.

Returns
str or None (if no project associated with the token)

property project_is_domain: bool | None

Return if a project act as a domain.

Returns
bool

property project_name: str | None

Return the project name associated with the auth request.

Returns
str or None (if no project associated with the token)

property role_ids: list[str] | None

Return a list of users role ids associated with the auth request.

Returns
a list of strings of role ids

property role_names: list[str] | None

Return a list of users role names associated with the auth request.

Returns
a list of strings of role names

property service_providers: ServiceProviders | None

Return an object representing the list of trusted service providers.

Used for Keystone2Keystone federating-out.

Returns
`keystoneauth1.service_providers.ServiceProviders` or None

property system_scoped: bool

Return true if the auth token was scoped to the system.

Returns
bool

property trust_id: str | None

Return the trust id associated with the auth request.

Returns
str or None (if no trust associated with the token)

property trust_scoped: bool

Return true if the auth token was scoped from a delegated trust.

The trust delegation is via the OS-TRUST v3 extension.

Returns
bool

property trustee_user_id: str | None

Return the trustee user id associated with a trust.

Returns
str or None (if no trust associated with the token)

property trustor_user_id: str | None

Return the trustor user id associated with a trust.

Returns
str or None (if no trust associated with the token)

property user_domain_id: str | None

Return the users domain id associated with the auth request.

Returns
str

property user_domain_name: str | None

Return the users domain name associated with the auth request.

Returns
str

property user_id: str | None

Return the user id associated with the auth request.

Returns
str

property username: str | None

Return the username associated with the auth request.

Follows the pattern defined in the V2 API of first looking for name, returning that if available, and falling back to username if name is unavailable.

Returns
str

version = 'v2.0'

class keystoneauth1.access.access.**AccessInfoV3**(*body: dict[str, Any]*, *auth_token: str | None = None*)

Bases: `AccessInfo`

An object encapsulating raw v3 auth token from identity service.

__annotations__ = {'_data': <class 'keystoneauth1.access.types.TokenResponseV3'>, '_service_catalog': 'ty.Optional[service_catalog.ServiceCatalog]', '_service_catalog_class': 'ty.Type[service_catalog.ServiceCatalog]', '_service_providers': 'ty.Optional[service_providers.ServiceProviders]'}

__doc__ = 'An object encapsulating raw v3 auth token from identity service.'

__module__ = 'keystoneauth1.access.access'

property _application_credential: ApplicationCredentialV3 | None

_data: TokenResponseV3

property _domain: DomainV3 | None

property _oauth: OAuth1V3 | None

property _oauth2_credential: OAuth2V3 | None

property _project: ProjectV3 | None

property _project_domain: ProjectDomainV3 | None

_service_catalog: ServiceCatalog | None

_service_catalog_class

alias of `ServiceCatalogV3`

_service_providers: ServiceProviders | None

property _token: TokenV3

property _trust: TrustV3 | None

property _user: UserV3

property _user_domain: UserDomainV3

property application_credential: ApplicationCredentialV3

property application_credential_access_rules: list[ApplicationCredentialAccessRuleV3] | None

property application_credential_id: str | None

property audit_chain_id: str | None

Return the audit chain ID if present.

In the event that a token was rescoped then this ID will be the `audit_id` of the initial token. Returns None if no value present.

Returns
str or None.

property audit_id: str | None

Return the audit ID if present.

Returns
str or None.

property bind: dict[str, Any] | None

Information about external mechanisms the token is bound to.

If a token is bound to an external authentication mechanism it can only be used in conjunction with that mechanism. For example if bound to a kerberos principal it may only be accepted if there is also kerberos authentication performed on the request.

Returns
A dictionary or None. The key will be the bind type the value is a dictionary that is specific to the format of the bind type. Returns None if there is no bind information in the token.

property domain_id: str | None

Return the domain id associated with the auth request.

Returns
str or None (if no domain associated with the token)

property domain_name: str | None

Return the domain name associated with the auth request.

Returns
str or None (if no domain associated with the token)

property domain_scoped: bool

Return true if the auth token was scoped to a domain.

Returns
bool

property expires: datetime | None

Return the token expiration (as datetime object).

Returns
datetime

has_service_catalog() → bool

Return true if the auth token has a service catalog.

Returns
boolean

property is_admin_project: bool

Return true if the current project scope is the admin project.

For backwards compatibility purposes if there is nothing specified in the token we always assume we are in the admin project, so this will default to True.

:returns boolean

property is_federated: bool

Return true if federation was used to get the token.

Returns
boolean

property issued: datetime | None

Return the token issue time (as datetime object).

Returns
datetime

property oauth2_credential: OAuth2V3

property oauth2_credential_thumbprint: str | None

property oauth_access_token_id: str | None

Return the access token ID if OAuth authentication used.

Returns
str or None.

property oauth_consumer_id: str | None

Return the consumer ID if OAuth authentication used.

Returns
str or None.

property project_domain_id: str | None

Return the projects domain id associated with the auth request.

Returns
str

property project_domain_name: str | None

Return the projects domain name associated with the auth request.

Returns
str

property project_id: str | None

Return the project ID associated with the auth request.

This returns None if the auth token wasnt scoped to a project.

Returns
str or None (if no project associated with the token)

property project_is_domain: bool | None

Return if a project act as a domain.

Returns
bool

property project_name: str | None

Return the project name associated with the auth request.

Returns
str or None (if no project associated with the token)

property role_ids: list[str] | None

Return a list of users role ids associated with the auth request.

Returns
a list of strings of role ids

property role_names: list[str] | None

Return a list of users role names associated with the auth request.

Returns
a list of strings of role names

property service_providers: ServiceProviders | None

Return an object representing the list of trusted service providers.

Used for Keystone2Keystone federating-out.

Returns
`keystoneauth1.service_providers.ServiceProviders` or None

property system: SystemV3 | None

property system_scoped: bool

Return true if the auth token was scoped to the system.

Returns
bool

property trust_id: str | None

Return the trust id associated with the auth request.

Returns
str or None (if no trust associated with the token)

property trust_scoped: bool

Return true if the auth token was scoped from a delegated trust.

The trust delegation is via the OS-TRUST v3 extension.

Returns
bool

property trustee_user_id: str | None

Return the trustee user id associated with a trust.

Returns
str or None (if no trust associated with the token)

property trustor_user_id: str | None

Return the trustor user id associated with a trust.

Returns
str or None (if no trust associated with the token)

property user_domain_id: str | None

Return the users domain id associated with the auth request.

Returns
str

property user_domain_name: str | None

Return the users domain name associated with the auth request.

Returns
str

property user_id: str | None

Return the user id associated with the auth request.

Returns
str

property username: str | None

Return the username associated with the auth request.

Follows the pattern defined in the V2 API of first looking for name, returning that if available, and falling back to username if name is unavailable.

Returns
str

version = 'v3'

keystoneauth1.access.access.**create**(*resp: Response | None = None, body: dict[str, object] | None = None, auth_token: str | None = None*) → AccessInfo

keystoneauth1.access.service_catalog module

class keystoneauth1.access.service_catalog.**ServiceCatalog**(*catalog: list[dict[str, Any]]*)

Bases: object

Helper methods for dealing with a Keystone Service Catalog.

__abstractmethods__ = frozenset({'from_token', 'is_interface_match'})

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.service_catalog', '__doc__': 'Helper methods for dealing with a Keystone Service Catalog.', '__init__': <function ServiceCatalog.__init__>, 'from_token': <classmethod(<function ServiceCatalog.from_token>)>, '_get_endpoint_region': <function ServiceCatalog._get_endpoint_region>, 'catalog': <property object>, 'is_interface_match': <function ServiceCatalog.is_interface_match>, 'normalize_interface': <staticmethod(<function ServiceCatalog.normalize_interface>)>, '_normalize_endpoints': <function ServiceCatalog._normalize_endpoints>, '_denormalize_endpoints': <function ServiceCatalog._denormalize_endpoints>, 'normalize_catalog': <function ServiceCatalog.normalize_catalog>, '_get_interface_list': <function ServiceCatalog._get_interface_list>, 'get_endpoints_data': <function ServiceCatalog.get_endpoints_data>, '_endpoints_by_type': <function ServiceCatalog._endpoints_by_type>, 'get_endpoints': <function ServiceCatalog.get_endpoints>, 'get_endpoint_data_list': <function ServiceCatalog.get_endpoint_data_list>, 'get_urls': <function ServiceCatalog.get_urls>, 'url_for': <function ServiceCatalog.url_for>, 'endpoint_data_for': <function ServiceCatalog.endpoint_data_for>, '__dict__': <attribute '__dict__' of 'ServiceCatalog' objects>, '__weakref__': <attribute '__weakref__' of 'ServiceCatalog' objects>, '__abstractmethods__': frozenset({'from_token', 'is_interface_match'}), '_abc_impl': <_abc._abc_data object>, '__annotations__': {}})`

`__doc__ = 'Helper methods for dealing with a Keystone Service Catalog.'`

`__init__`(*catalog: list[dict[str, Any]]*)

`__module__ = 'keystoneauth1.access.service_catalog'`

`__weakref__`

list of weak references to the object (if defined)

`_abc_impl = <_abc._abc_data object>`

`_denormalize_endpoints`(*endpoints: list[EndpointData]*) → list[str | None]

Return original endpoint description dicts.

Takes a list of EndpointData objects and returns the original dict that was returned from the catalog.

Parameters

endpoints (*list*) List of *keystoneauth1.discover.EndpointData*

Returns

List of endpoint description dicts in original catalog format

`_endpoints_by_type`(*requested: str | None, endpoints: dict[str, list[EndpointData]]*) → dict[str, list[EndpointData]]

Get the approrpriate endpoints from the list of given endpoints.

Per the service type alias rules:

If a user requests a service by its proper name and that matches, win.

If a user requests a service by its proper name and only a single alias matches, win.

If a user requests a service by its proper name and more than one alias matches, choose the first alias from the list given.

Do the first alias match after the other filters, as they might limit the number of choices for us otherwise.

Parameters

- **requested** (*str*) The service_type as requested by the user.
- **endpoints** (*dict*) A dictionary keyed by found service_type. Values are opaque to this method.

Returns

Dict of service_type/endpoints filtered for the appropriate service_type based on alias matching rules.

_get_endpoint_region(*endpoint: dict[str, str]*) → str | None

_get_interface_list(*interface: str | list[str] | tuple[str] | set[str] | None*) → list[str]

_normalize_endpoints(*endpoints: list[dict[str, Any]]*) → list[dict[str, Any]]

Translate endpoint description dicts into v3 form.

Takes a raw endpoint description from the catalog and changes it to be in v3 format. It also saves a copy of the data in raw_endpoint so that it can be returned by methods that expect the actual original data.

Parameters

endpoints (*list*) List of endpoint description dicts

Returns

List of endpoint description dicts in v3 format

property catalog: list[dict[str, Any]]

Return the raw service catalog content, mostly useful for debugging.

Applications should avoid this and use accessor methods instead. However, there are times when inspecting the raw catalog can be useful for analysis and other reasons.

endpoint_data_for(*service_type: str | None = None, interface: str | list[str] | tuple[str] | set[str] | None = 'public', region_name: str | None = None, service_name: str | None = None, service_id: str | None = None, endpoint_id: str | None = None*) → EndpointData

Fetch endpoint data from the service catalog.

Fetch the specified endpoint data from the service catalog for a particular endpoint attribute. If no attribute is given, return the first endpoint of the specified type.

Valid interface types: *public* or *publicURL*,

internal or *internalURL*, *admin* or adminURL'

Parameters

- **service_type** (*string*) Service type of the endpoint.
- **interface** Type of endpoint. Can be a single value or a list of values. If its a list of values, they will be looked for in order of preference.
- **region_name** (*string*) Region of the endpoint.

- **service_name** (*string*) The assigned name of the service.
- **service_id** (*string*) The identifier of a service.
- **endpoint_id** (*string*) The identifier of an endpoint.

abstract classmethod from_token(*token: dict[str, Any]*) → Self

Retrieve the service catalog from a token.

Parameters
token

Returns
A service catalog.

get_endpoint_data_list(*service_type: str | None = None, interface: str | list[str] | tuple[str] | set[str] | None = 'public', region_name: str | None = None, service_name: str | None = None, service_id: str | None = None, endpoint_id: str | None = None*) → list[EndpointData]

Fetch a flat list of matching EndpointData objects.

Fetch the endpoints from the service catalog for a particular endpoint attribute. If no attribute is given, return the first endpoint of the specified type.

Valid interface types: *public* or *publicURL*,
internal or *internalURL*, *admin* or adminURL'

Parameters

- **service_type** (*string*) Service type of the endpoint.
- **interface** Type of endpoint. Can be a single value or a list of values. If its a list of values, they will be looked for in order of preference.
- **region_name** (*string*) Region of the endpoint.
- **service_name** (*string*) The assigned name of the service.
- **service_id** (*string*) The identifier of a service.
- **endpoint_id** (*string*) The identifier of an endpoint.

Returns
a list of matching EndpointData objects

Return type
list(*keystoneauth1.discover.EndpointData*)

get_endpoints(*service_type: str | None = None, interface: str | list[str] | tuple[str] | set[str] | None = None, region_name: str | None = None, service_name: str | None = None, service_id: str | None = None, endpoint_id: str | None = None*) → dict[str, list[str | None]]

Fetch and filter endpoint data for the specified service(s).

Returns endpoints for the specified service (or all) containing the specified type (or all) and region (or all) and service name.

If there is no name in the service catalog the service_name check will be skipped. This allows compatibility with services that existed before the name was available in the catalog.

Returns a dict keyed by service_type with a list of endpoint dicts

get_endpoints_data(*service_type: str | None = None*, *interface: str | list[str] | tuple[str] | set[str] | None = None*, *region_name: str | None = None*, *service_name: str | None = None*, *service_id: str | None = None*, *endpoint_id: str | None = None*) → dict[str, list[EndpointData]]

Fetch and filter endpoint data for the specified service(s).

Returns endpoints for the specified service (or all) containing the specified type (or all) and region (or all) and service name.

If there is no name in the service catalog the service_name check will be skipped. This allows compatibility with services that existed before the name was available in the catalog.

Valid interface types: *public* or *publicURL*,
internal or *internalURL*, *admin* or adminURL'

Parameters

- **service_type** (*string*) Service type of the endpoint.
- **interface** Type of endpoint. Can be a single value or a list of values. If its a list of values, they will be looked for in order of preference.
- **region_name** (*string*) Region of the endpoint.
- **service_name** (*string*) The assigned name of the service.
- **service_id** (*string*) The identifier of a service.
- **endpoint_id** (*string*) The identifier of an endpoint.

Returns
a dict, keyed by service_type, of lists of EndpointData

get_urls(*service_type: str | None = None*, *interface: str | list[str] | tuple[str] | set[str] | None = 'public'*, *region_name: str | None = None*, *service_name: str | None = None*, *service_id: str | None = None*, *endpoint_id: str | None = None*) → tuple[str | None, ...]

Fetch endpoint urls from the service catalog.

Fetch the urls of endpoints from the service catalog for a particular endpoint attribute. If no attribute is given, return the url of the first endpoint of the specified type.

Valid interface types: *public* or *publicURL*,
internal or *internalURL*, *admin* or adminURL'

Parameters

- **service_type** (*string*) Service type of the endpoint.
- **interface** Type of endpoint. Can be a single value or a list of values. If its a list of values, they will be looked for in order of preference.
- **region_name** (*string*) Region of the endpoint.
- **service_name** (*string*) The assigned name of the service.
- **service_id** (*string*) The identifier of a service.
- **endpoint_id** (*string*) The identifier of an endpoint.

Returns
tuple of urls

abstract is_interface_match(*endpoint: dict[str, str]*, *interface: str*) → bool

Helper function to normalize endpoint matching across v2 and v3.

Returns
True if the provided endpoint matches the required interface otherwise False.

normalize_catalog() → list[dict[str, Any]]

Return the catalog normalized into v3 format.

static normalize_interface(*interface: str*) → str

Handle differences in the way v2 and v3 catalogs specify endpoint.

Both v2 and v3 must be able to handle the endpoint style of the other. For example v2 must be able to handle a public interface and v3 must be able to handle a publicURL interface.

Returns
the endpoint string in the format appropriate for this service catalog.

url_for(*service_type: str | None = None*, *interface: str | list[str] | tuple[str] | set[str] | None = 'public'*, *region_name: str | None = None*, *service_name: str | None = None*, *service_id: str | None = None*, *endpoint_id: str | None = None*) → str | None

Fetch an endpoint from the service catalog.

Fetch the specified endpoint from the service catalog for a particular endpoint attribute. If no attribute is given, return the first endpoint of the specified type.

Valid interface types: *public* or *publicURL*,
internal or *internalURL*, *admin* or adminURL'

Parameters

- **service_type** (*string*) Service type of the endpoint.
- **interface** Type of endpoint. Can be a single value or a list of values. If its a list of values, they will be looked for in order of preference.
- **region_name** (*string*) Region of the endpoint.
- **service_name** (*string*) The assigned name of the service.
- **service_id** (*string*) The identifier of a service.
- **endpoint_id** (*string*) The identifier of an endpoint.

class keystoneauth1.access.service_catalog.**ServiceCatalogV2**(*catalog: list[dict[str, Any]]*)

Bases: `ServiceCatalog`

An object for encapsulating the v2 service catalog.

The object is created using raw v2 auth token from Keystone.

__abstractmethods__ = frozenset({})

__annotations__ = {}

`__doc__ = 'An object for encapsulating the v2 service catalog.\n\n The object is created using raw v2 auth token from Keystone.\n '`

`__module__ = 'keystoneauth1.access.service_catalog'`

`_abc_impl = <_abc._abc_data object>`

`_denormalize_endpoints`(*endpoints: list[EndpointData]*) → list[str | None]

Return original endpoint description dicts.

Takes a list of EndpointData objects and returns the original dict that was returned from the catalog.

Parameters

`endpoints` (*list*) List of *keystoneauth1.discover.EndpointData*

Returns

List of endpoint description dicts in original catalog format

`_normalize_endpoints`(*endpoints: list[dict[str, Any]]*) → list[dict[str, Any]]

Translate endpoint description dicts into v3 form.

Takes a raw endpoint description from the catalog and changes it to be in v3 format. It also saves a copy of the data in raw_endpoint so that it can be returned by methods that expect the actual original data.

Parameters

`endpoints` (*list*) List of endpoint description dicts

Returns

List of endpoint description dicts in v3 format

`classmethod` **`from_token`**(*token: dict[str, Any]*) → Self

Retrieve the service catalog from a token.

Parameters

`token`

Returns

A service catalog.

`is_interface_match`(*endpoint: dict[str, str]*, *interface: str*) → bool

Helper function to normalize endpoint matching across v2 and v3.

Returns

True if the provided endpoint matches the required interface otherwise False.

`static` **`normalize_interface`**(*interface: str*) → str

Handle differences in the way v2 and v3 catalogs specify endpoint.

Both v2 and v3 must be able to handle the endpoint style of the other. For example v2 must be able to handle a public interface and v3 must be able to handle a publicURL interface.

Returns

the endpoint string in the format appropriate for this service catalog.

`class keystoneauth1.access.service_catalog.`**`ServiceCatalogV3`**(*catalog: list[dict[str, Any]]*)

Bases: `ServiceCatalog`

An object for encapsulating the v3 service catalog.

The object is created using raw v3 auth token from Keystone.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {}`

`__doc__ = 'An object for encapsulating the v3 service catalog.\n\n The object is created using raw v3 auth token from Keystone.\n '`

`__module__ = 'keystoneauth1.access.service_catalog'`

`_abc_impl = <_abc._abc_data object>`

`classmethod from_token`(*token: dict[str, Any]*) → Self

Retrieve the service catalog from a token.

Parameters
`token`

Returns
A service catalog.

`is_interface_match`(*endpoint: dict[str, str]*, *interface: str*) → bool

Helper function to normalize endpoint matching across v2 and v3.

Returns
True if the provided endpoint matches the required interface otherwise False.

`static normalize_interface`(*interface: str*) → str

Handle differences in the way v2 and v3 catalogs specify endpoint.

Both v2 and v3 must be able to handle the endpoint style of the other. For example v2 must be able to handle a public interface and v3 must be able to handle a publicURL interface.

Returns
the endpoint string in the format appropriate for this service catalog.

keystoneauth1.access.service_providers module

`class` `keystoneauth1.access.service_providers.`**`ServiceProviders`**(*service_providers: list[ServiceProviderV3]*)

Bases: `object`

Helper methods for dealing with Service Providers.

__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.service_providers', '__doc__': 'Helper methods for dealing with Service Providers.', '__init__': <function ServiceProviders.__init__>, 'from_token': <classmethod(<function ServiceProviders.from_token>)>, '_get_service_provider': <function ServiceProviders._get_service_provider>, 'get_sp_url': <function ServiceProviders.get_sp_url>, 'get_auth_url': <function ServiceProviders.get_auth_url>, '__dict__': <attribute '__dict__' of 'ServiceProviders' objects>, '__weakref__': <attribute '__weakref__' of 'ServiceProviders' objects>, '__annotations__': {}})

__doc__ = 'Helper methods for dealing with Service Providers.'

__init__(*service_providers: list[ServiceProviderV3]*)

__module__ = 'keystoneauth1.access.service_providers'

__weakref__

list of weak references to the object (if defined)

_get_service_provider(*sp_id: str*) → ServiceProviderV3

classmethod **from_token**(*token: TokenResponseV3*) → ServiceProviders

get_auth_url(*sp_id: str*) → str

get_sp_url(*sp_id: str*) → str

keystoneauth1.access.types module

class keystoneauth1.access.types.**AccessV2**

Bases: TypedDict

__annotations__ = {'metadata': typing_extensions.NotRequired[keystoneauth1.access.types.MetadataV2], 'serviceCatalog': typing_extensions.NotRequired[list[keystoneauth1.access.types.CatalogServiceV2]], 'token': <class 'keystoneauth1.access.types.TokenV2'>, 'trust': typing_extensions.NotRequired[keystoneauth1.access.types.TrustV2], 'user': <class 'keystoneauth1.access.types.UserV2'>}

__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'token': <class 'keystoneauth1.access.types.TokenV2'>, 'serviceCatalog': typing_extensions.NotRequired[list[keystoneauth1.access.types.CatalogServiceV2]], 'user': <class 'keystoneauth1.access.types.UserV2'>, 'metadata': typing_extensions.NotRequired[keystoneauth1.access.types.MetadataV2], 'trust': typing_extensions.NotRequired[keystoneauth1.access.types.TrustV2]}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'AccessV2' objects>, '__weakref__': <attribute '__weakref__' of 'AccessV2' objects>, '__doc__': None, '__required_keys__': frozenset({'token', 'metadata', 'trust', 'serviceCatalog', 'user'}), '__optional_keys__': frozenset(), '__total__': True})

__doc__ = None

__module__ = 'keystoneauth1.access.types'

__optional_keys__ = frozenset({})

__orig_bases__ = (<function TypedDict>,)

__required_keys__ = frozenset({'metadata', 'serviceCatalog', 'token', 'trust', 'user'})

__total__ = True

__weakref__

list of weak references to the object (if defined)

metadata: NotRequired[MetadataV2]

serviceCatalog: NotRequired[list[CatalogServiceV2]]

token: TokenV2

trust: NotRequired[TrustV2]

user: UserV2

class keystoneauth1.access.types.ApplicationCredentialAccessRuleV3

Bases: TypedDict

__annotations__ = {'id': <class 'str'>}

__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'id': <class 'str'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'ApplicationCredentialAccessRuleV3' objects>, '__weakref__': <attribute '__weakref__' of 'ApplicationCredentialAccessRuleV3' objects>, '__doc__': None, '__required_keys__': frozenset({'id'}), '__optional_keys__': frozenset(), '__total__': True})

`__doc__ = None`

`__module__ = 'keystoneauth1.access.types'`

`__optional_keys__ = frozenset({})`

`__orig_bases__ = (<function TypedDict>,)`

`__required_keys__ = frozenset({'id'})`

`__total__ = True`

`__weakref__`

list of weak references to the object (if defined)

`id: str`

`class keystoneauth1.access.types.ApplicationCredentialV3`

Bases: `TypedDict`

`__annotations__ = {'access_rules': typing_extensions.NotRequired[list[keystoneauth1.access.types.ApplicationCredentialAccessRuleV3]], 'id': <class 'str'>, 'name': <class 'str'>, 'restricted': <class 'bool'>}`

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'access_rules': typing_extensions.NotRequired[list[keystoneauth1.access.types.ApplicationCredentialAccessRuleV3]], 'id': <class 'str'>, 'name': <class 'str'>, 'restricted': <class 'bool'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'ApplicationCredentialV3' objects>, '__weakref__': <attribute '__weakref__' of 'ApplicationCredentialV3' objects>, '__doc__': None, '__required_keys__': frozenset({'name', 'access_rules', 'restricted', 'id'}), '__optional_keys__': frozenset(), '__total__': True})`

`__doc__ = None`

`__module__ = 'keystoneauth1.access.types'`

`__optional_keys__ = frozenset({})`

`__orig_bases__ = (<function TypedDict>,)`

`__required_keys__ = frozenset({'access_rules', 'id', 'name', 'restricted'})`

`__total__ = True`

`__weakref__`

list of weak references to the object (if defined)

`access_rules: NotRequired[list[ApplicationCredentialAccessRuleV3]]`

`id: str`

name: str

restricted: bool

class keystoneauth1.access.types.**CatalogServiceV2**

Bases: TypedDict

__annotations__ = {'endpoints': list[keystoneauth1.access.types.EndpointV2], 'endpoints_links': list[typing.Any], 'name': <class 'str'>, 'type': <class 'str'>}

__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'endpoints': list[keystoneauth1.access.types.EndpointV2], 'endpoints_links': list[typing.Any], 'type': <class 'str'>, 'name': <class 'str'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'CatalogServiceV2' objects>, '__weakref__': <attribute '__weakref__' of 'CatalogServiceV2' objects>, '__doc__': None, '__required_keys__': frozenset({'name', 'type', 'endpoints', 'endpoints_links'}), '__optional_keys__': frozenset(), '__total__': True})

__doc__ = None

__module__ = 'keystoneauth1.access.types'

__optional_keys__ = frozenset({})

__orig_bases__ = (<function TypedDict>,)

__required_keys__ = frozenset({'endpoints', 'endpoints_links', 'name', 'type'})

__total__ = True

__weakref__

list of weak references to the object (if defined)

endpoints: list[EndpointV2]

endpoints_links: list[Any]

name: str

type: str

class keystoneauth1.access.types.**DomainV3**

Bases: TypedDict

__annotations__ = {'id': <class 'str'>, 'name': <class 'str'>}

__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'id': <class 'str'>, 'name': <class 'str'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'DomainV3' objects>, '__weakref__': <attribute '__weakref__' of 'DomainV3' objects>, '__doc__': None, '__required_keys__': frozenset({'name', 'id'}), '__optional_keys__': frozenset(), '__total__': True})

`__doc__ = None`

`__module__ = 'keystoneauth1.access.types'`

`__optional_keys__ = frozenset({})`

`__orig_bases__ = (<function TypedDict>,)`

`__required_keys__ = frozenset({'id', 'name'})`

`__total__ = True`

`__weakref__`

list of weak references to the object (if defined)

`id: str`

`name: str`

`class keystoneauth1.access.types.EndpointV2`

Bases: `TypedDict`

`__annotations__ = {'adminURL': <class 'str'>, 'id': <class 'str'>, 'internalURL': <class 'str'>, 'publicURL': <class 'str'>, 'region': <class 'str'>}`

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'adminURL': <class 'str'>, 'region': <class 'str'>, 'internalURL': <class 'str'>, 'id': <class 'str'>, 'publicURL': <class 'str'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'EndpointV2' objects>, '__weakref__': <attribute '__weakref__' of 'EndpointV2' objects>, '__doc__': None, '__required_keys__': frozenset({'publicURL', 'adminURL', 'id', 'region', 'internalURL'}), '__optional_keys__': frozenset(), '__total__': True})`

`__doc__ = None`

`__module__ = 'keystoneauth1.access.types'`

`__optional_keys__ = frozenset({})`

`__orig_bases__ = (<function TypedDict>,)`

`__required_keys__ = frozenset({'adminURL', 'id', 'internalURL', 'publicURL', 'region'})`

`__total__ = True`

`__weakref__`

list of weak references to the object (if defined)

`adminURL: str`

`id: str`

internalURL: str

publicURL: str

region: str

class keystoneauth1.access.types.**EndpointV3**

Bases: TypedDict

__annotations__ = {'id': <class 'str'>, 'interface': <class 'str'>, 'region': <class 'str'>, 'region_id': <class 'str'>, 'url': <class 'str'>}

__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'id': <class 'str'>, 'interface': <class 'str'>, 'region': <class 'str'>, 'region_id': <class 'str'>, 'url': <class 'str'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'EndpointV3' objects>, '__weakref__': <attribute '__weakref__' of 'EndpointV3' objects>, '__doc__': None, '__required_keys__': frozenset({'id', 'url', 'region_id', 'region', 'interface'}), '__optional_keys__': frozenset(), '__total__': True})

__doc__ = None

__module__ = 'keystoneauth1.access.types'

__optional_keys__ = frozenset({})

__orig_bases__ = (<function TypedDict>,)

__required_keys__ = frozenset({'id', 'interface', 'region', 'region_id', 'url'})

__total__ = True

__weakref__

list of weak references to the object (if defined)

id: str

interface: str

region: str

region_id: str

url: str

class keystoneauth1.access.types.**FederationGroupV3**

Bases: TypedDict

__annotations__ = {'id': <class 'str'>}

__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'id': <class 'str'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'FederationGroupV3' objects>, '__weakref__': <attribute '__weakref__' of 'FederationGroupV3' objects>, '__doc__': None, '__required_keys__': frozenset({'id'}), '__optional_keys__': frozenset(), '__total__': True})

__doc__ = None

__module__ = 'keystoneauth1.access.types'

__optional_keys__ = frozenset({})

__orig_bases__ = (<function TypedDict>,)

__required_keys__ = frozenset({'id'})

__total__ = True

__weakref__

list of weak references to the object (if defined)

id: str

class keystoneauth1.access.types.**FederationProtocolV3**

Bases: TypedDict

__annotations__ = {'id': <class 'str'>}

__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'id': <class 'str'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'FederationProtocolV3' objects>, '__weakref__': <attribute '__weakref__' of 'FederationProtocolV3' objects>, '__doc__': None, '__required_keys__': frozenset({'id'}), '__optional_keys__': frozenset(), '__total__': True})

__doc__ = None

__module__ = 'keystoneauth1.access.types'

__optional_keys__ = frozenset({})

__orig_bases__ = (<function TypedDict>,)

__required_keys__ = frozenset({'id'})

__total__ = True

__weakref__

list of weak references to the object (if defined)

id: str

class keystoneauth1.access.types.**FederationProviderV3**

Bases: TypedDict

`__annotations__ = {'id': <class 'str'>}`

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'id': <class 'str'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'FederationProviderV3' objects>, '__weakref__': <attribute '__weakref__' of 'FederationProviderV3' objects>, '__doc__': None, '__required_keys__': frozenset({'id'}), '__optional_keys__': frozenset(), '__total__': True})`

`__doc__ = None`

`__module__ = 'keystoneauth1.access.types'`

`__optional_keys__ = frozenset({})`

`__orig_bases__ = (<function TypedDict>,)`

`__required_keys__ = frozenset({'id'})`

`__total__ = True`

`__weakref__`

list of weak references to the object (if defined)

`id: str`

`class keystoneauth1.access.types.FederationV3`

Bases: `TypedDict`

`__annotations__ = {'groups': list[keystoneauth1.access.types.FederationGroupV3], 'identity_provider': <class 'keystoneauth1.access.types.FederationProviderV3'>, 'protocol': <class 'keystoneauth1.access.types.FederationProtocolV3'>}`

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'groups': list[keystoneauth1.access.types.FederationGroupV3], 'identity_provider': <class 'keystoneauth1.access.types.FederationProviderV3'>, 'protocol': <class 'keystoneauth1.access.types.FederationProtocolV3'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'FederationV3' objects>, '__weakref__': <attribute '__weakref__' of 'FederationV3' objects>, '__doc__': None, '__required_keys__': frozenset({'groups', 'protocol', 'identity_provider'}), '__optional_keys__': frozenset(), '__total__': True})`

`__doc__ = None`

`__module__ = 'keystoneauth1.access.types'`

`__optional_keys__ = frozenset({})`

`__orig_bases__ = (<function TypedDict>,)`

`__required_keys__ = frozenset({'groups', 'identity_provider', 'protocol'})`

__total__ = True

__weakref__

list of weak references to the object (if defined)

groups: list[FederationGroupV3]

identity_provider: FederationProviderV3

protocol: FederationProtocolV3

class keystoneauth1.access.types.**MetadataV2**

Bases: TypedDict

__annotations__ = {'is_admin': <class 'int'>, 'roles': list[str]}

__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'is_admin': <class 'int'>, 'roles': list[str]}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'MetadataV2' objects>, '__weakref__': <attribute '__weakref__' of 'MetadataV2' objects>, '__doc__': None, '__required_keys__': frozenset({'roles', 'is_admin'}), '__optional_keys__': frozenset(), '__total__': True})

__doc__ = None

__module__ = 'keystoneauth1.access.types'

__optional_keys__ = frozenset({})

__orig_bases__ = (<function TypedDict>,)

__required_keys__ = frozenset({'is_admin', 'roles'})

__total__ = True

__weakref__

list of weak references to the object (if defined)

is_admin: int

roles: list[str]

class keystoneauth1.access.types.**OAuth1V3**

Bases: TypedDict

__annotations__ = {'access_token_id': <class 'str'>, 'consumer_id': <class 'str'>}

__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'access_token_id': <class 'str'>, 'consumer_id': <class 'str'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'OAuth1V3' objects>, '__weakref__': <attribute '__weakref__' of 'OAuth1V3' objects>, '__doc__': None, '__required_keys__': frozenset({'access_token_id', 'consumer_id'}), '__optional_keys__': frozenset(), '__total__': True})

`__doc__ = None`

`__module__ = 'keystoneauth1.access.types'`

`__optional_keys__ = frozenset({})`

`__orig_bases__ = (<function TypedDict>,)`

`__required_keys__ = frozenset({'access_token_id', 'consumer_id'})`

`__total__ = True`

`__weakref__`

list of weak references to the object (if defined)

`access_token_id: str`

`consumer_id: str`

`class keystoneauth1.access.types.OAuth2V3`

Bases: `dict`

`__annotations__ = {'x5t#S256': <class 'str'>}`

`__dict__ = mappingproxy({'__annotations__': {'x5t#S256': <class 'str'>}, '__module__': 'keystoneauth1.access.types', '__dict__': <attribute '__dict__' of 'OAuth2V3' objects>, '__weakref__': <attribute '__weakref__' of 'OAuth2V3' objects>, '__doc__': None, '__required_keys__': frozenset({'x5t#S256'}), '__optional_keys__': frozenset(), '__total__': True})`

`__doc__ = None`

`__module__ = 'keystoneauth1.access.types'`

`__optional_keys__ = frozenset({})`

`__required_keys__ = frozenset({'x5t#S256'})`

`__total__ = True`

`__weakref__`

list of weak references to the object (if defined)

`class keystoneauth1.access.types.ProjectDomainV3`

Bases: `TypedDict`

`__annotations__ = {'id': <class 'str'>, 'name': <class 'str'>}`

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'id': <class 'str'>, 'name': <class 'str'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'ProjectDomainV3' objects>, '__weakref__': <attribute '__weakref__' of 'ProjectDomainV3' objects>, '__doc__': None, '__required_keys__': frozenset({'name', 'id'}), '__optional_keys__': frozenset(), '__total__': True})`

`__doc__ = None`

`__module__ = 'keystoneauth1.access.types'`

`__optional_keys__ = frozenset({})`

`__orig_bases__ = (<function TypedDict>,)`

`__required_keys__ = frozenset({'id', 'name'})`

`__total__ = True`

`__weakref__`

list of weak references to the object (if defined)

`id: str`

`name: str`

`class keystoneauth1.access.types.ProjectV3`

Bases: `TypedDict`

`__annotations__ = {'domain': <class 'keystoneauth1.access.types.ProjectDomainV3'>, 'id': <class 'str'>, 'name': <class 'str'>}`

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'domain': <class 'keystoneauth1.access.types.ProjectDomainV3'>, 'id': <class 'str'>, 'name': <class 'str'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'ProjectV3' objects>, '__weakref__': <attribute '__weakref__' of 'ProjectV3' objects>, '__doc__': None, '__required_keys__': frozenset({'domain', 'name', 'id'}), '__optional_keys__': frozenset(), '__total__': True})`

`__doc__ = None`

`__module__ = 'keystoneauth1.access.types'`

`__optional_keys__ = frozenset({})`

`__orig_bases__ = (<function TypedDict>,)`

`__required_keys__ = frozenset({'domain', 'id', 'name'})`

`__total__ = True`

`__weakref__`

list of weak references to the object (if defined)

`domain: ProjectDomainV3`

`id: str`

`name: str`

class keystoneauth1.access.types.**RoleV2**

Bases: TypedDict

__annotations__ = {'name': <class 'str'>}

__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'name': <class 'str'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'RoleV2' objects>, '__weakref__': <attribute '__weakref__' of 'RoleV2' objects>, '__doc__': None, '__required_keys__': frozenset({'name'}), '__optional_keys__': frozenset(), '__total__': True})

__doc__ = None

__module__ = 'keystoneauth1.access.types'

__optional_keys__ = frozenset({})

__orig_bases__ = (<function TypedDict>,)

__required_keys__ = frozenset({'name'})

__total__ = True

__weakref__

list of weak references to the object (if defined)

name: str

class keystoneauth1.access.types.**RoleV3**

Bases: TypedDict

__annotations__ = {'id': <class 'str'>, 'name': <class 'str'>}

__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'id': <class 'str'>, 'name': <class 'str'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'RoleV3' objects>, '__weakref__': <attribute '__weakref__' of 'RoleV3' objects>, '__doc__': None, '__required_keys__': frozenset({'name', 'id'}), '__optional_keys__': frozenset(), '__total__': True})

__doc__ = None

__module__ = 'keystoneauth1.access.types'

__optional_keys__ = frozenset({})

__orig_bases__ = (<function TypedDict>,)

__required_keys__ = frozenset({'id', 'name'})

__total__ = True

__weakref__

list of weak references to the object (if defined)

id: str

name: str

class keystoneauth1.access.types.**ServiceProviderV3**

Bases: TypedDict

__annotations__ = {'auth_url': <class 'str'>, 'id': <class 'str'>, 'sp_url': <class 'str'>}

__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'auth_url': <class 'str'>, 'id': <class 'str'>, 'sp_url': <class 'str'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'ServiceProviderV3' objects>, '__weakref__': <attribute '__weakref__' of 'ServiceProviderV3' objects>, '__doc__': None, '__required_keys__': frozenset({'auth_url', 'sp_url', 'id'}), '__optional_keys__': frozenset(), '__total__': True})

__doc__ = None

__module__ = 'keystoneauth1.access.types'

__optional_keys__ = frozenset({})

__orig_bases__ = (<function TypedDict>,)

__required_keys__ = frozenset({'auth_url', 'id', 'sp_url'})

__total__ = True

__weakref__

list of weak references to the object (if defined)

auth_url: str

id: str

sp_url: str

class keystoneauth1.access.types.**ServiceV3**

Bases: TypedDict

__annotations__ = {'endpoints': list[keystoneauth1.access.types.EndpointV3], 'id': <class 'str'>, 'name': <class 'str'>, 'type': <class 'str'>}

__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'endpoints': list[keystoneauth1.access.types.EndpointV3], 'id': <class 'str'>, 'name': <class 'str'>, 'type': <class 'str'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'ServiceV3' objects>, '__weakref__': <attribute '__weakref__' of 'ServiceV3' objects>, '__doc__': None, '__required_keys__': frozenset({'name', 'endpoints', 'type', 'id'}), '__optional_keys__': frozenset(), '__total__': True})

`__doc__ = None`

`__module__ = 'keystoneauth1.access.types'`

`__optional_keys__ = frozenset({})`

`__orig_bases__ = (<function TypedDict>,)`

`__required_keys__ = frozenset({'endpoints', 'id', 'name', 'type'})`

`__total__ = True`

`__weakref__`

list of weak references to the object (if defined)

`endpoints: list[EndpointV3]`

`id: str`

`name: str`

`type: str`

`class keystoneauth1.access.types.SystemV3`

Bases: `TypedDict`

`__annotations__ = {'all': <class 'bool'>}`

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'all': <class 'bool'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'SystemV3' objects>, '__weakref__': <attribute '__weakref__' of 'SystemV3' objects>, '__doc__': None, '__required_keys__': frozenset({'all'}), '__optional_keys__': frozenset(), '__total__': True})`

`__doc__ = None`

`__module__ = 'keystoneauth1.access.types'`

`__optional_keys__ = frozenset({})`

`__orig_bases__ = (<function TypedDict>,)`

`__required_keys__ = frozenset({'all'})`

`__total__ = True`

`__weakref__`

list of weak references to the object (if defined)

`all: bool`

`class keystoneauth1.access.types.TenantV2`

Bases: `TypedDict`

`__annotations__ = {'description': typing_extensions.NotRequired[typing.Optional[str]], 'enabled': typing_extensions.NotRequired[bool], 'id': <class 'str'>, 'name': <class 'str'>}`

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'description': typing_extensions.NotRequired[typing.Optional[str]], 'enabled': typing_extensions.NotRequired[bool], 'id': <class 'str'>, 'name': <class 'str'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'TenantV2' objects>, '__weakref__': <attribute '__weakref__' of 'TenantV2' objects>, '__doc__': None, '__required_keys__': frozenset({'name', 'description', 'enabled', 'id'}), '__optional_keys__': frozenset(), '__total__': True})`

`__doc__ = None`

`__module__ = 'keystoneauth1.access.types'`

`__optional_keys__ = frozenset({})`

`__orig_bases__ = (<function TypedDict>,)`

`__required_keys__ = frozenset({'description', 'enabled', 'id', 'name'})`

`__total__ = True`

`__weakref__`

list of weak references to the object (if defined)

`description: NotRequired[str | None]`

`enabled: NotRequired[bool]`

`id: str`

`name: str`

class keystoneauth1.access.types.**TokenResponseV2**

Bases: `TypedDict`

`__annotations__ = {'access': <class 'keystoneauth1.access.types.AccessV2'>}`

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'access': <class 'keystoneauth1.access.types.AccessV2'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'TokenResponseV2' objects>, '__weakref__': <attribute '__weakref__' of 'TokenResponseV2' objects>, '__doc__': None, '__required_keys__': frozenset({'access'}), '__optional_keys__': frozenset(), '__total__': True})`

`__doc__ = None`

`__module__ = 'keystoneauth1.access.types'`

`__optional_keys__ = frozenset({})`

`__orig_bases__ = (<function TypedDict>,)`

`__required_keys__ = frozenset({'access'})`

`__total__ = True`

`__weakref__`

list of weak references to the object (if defined)

`access: AccessV2`

class `keystoneauth1.access.types.`**`TokenResponseV3`**

Bases: `TypedDict`

`__annotations__ = {'token': <class 'keystoneauth1.access.types.TokenV3'>}`

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'token': <class 'keystoneauth1.access.types.TokenV3'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'TokenResponseV3' objects>, '__weakref__': <attribute '__weakref__' of 'TokenResponseV3' objects>, '__doc__': None, '__required_keys__': frozenset({'token'}), '__optional_keys__': frozenset(), '__total__': True})`

`__doc__ = None`

`__module__ = 'keystoneauth1.access.types'`

`__optional_keys__ = frozenset({})`

`__orig_bases__ = (<function TypedDict>,)`

`__required_keys__ = frozenset({'token'})`

`__total__ = True`

`__weakref__`

list of weak references to the object (if defined)

`token: TokenV3`

class `keystoneauth1.access.types.`**`TokenV2`**

Bases: `TypedDict`

`__annotations__ = {'audit_ids': list[str], 'bind': typing_extensions.NotRequired[dict[str, typing.Any]], 'expires': <class 'str'>, 'id': <class 'str'>, 'issued_at': typing_extensions.NotRequired[str], 'tenant': typing_extensions.NotRequired[keystoneauth1.access.types.TenantV2]}`

__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'audit_ids': list[str], 'bind': typing_extensions.NotRequired[dict[str, typing.Any]], 'expires': <class 'str'>, 'id': <class 'str'>, 'issued_at': typing_extensions.NotRequired[str], 'tenant': typing_extensions.NotRequired[keystoneauth1.access.types.TenantV2]}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'TokenV2' objects>, '__weakref__': <attribute '__weakref__' of 'TokenV2' objects>, '__doc__': None, '__required_keys__': frozenset({'bind', 'expires', 'tenant', 'id', 'audit_ids', 'issued_at'}), '__optional_keys__': frozenset(), '__total__': True})

__doc__ = None

__module__ = 'keystoneauth1.access.types'

__optional_keys__ = frozenset({})

__orig_bases__ = (<function TypedDict>,)

__required_keys__ = frozenset({'audit_ids', 'bind', 'expires', 'id', 'issued_at', 'tenant'})

__total__ = True

__weakref__

list of weak references to the object (if defined)

audit_ids: list[str]

bind: NotRequired[dict[str, Any]]

expires: str

id: str

issued_at: NotRequired[str]

tenant: NotRequired[TenantV2]

class keystoneauth1.access.types.**TokenV3**

Bases: dict

```
__annotations__ = {'OS-OAUTH1':
typing_extensions.NotRequired[keystoneauth1.access.types.OAuth1V3],
'OS-TRUST:trust':
typing_extensions.NotRequired[keystoneauth1.access.types.TrustV3],
'application_credential': typing_extensions.NotRequired[keystoneauth1.
access.types.ApplicationCredentialV3], 'audit_ids': list[str], 'bind':
typing_extensions.NotRequired[dict[str, typing.Any]], 'catalog':
typing_extensions.NotRequired[list[keystoneauth1.access.types.ServiceV3]],
'domain':
typing_extensions.NotRequired[keystoneauth1.access.types.DomainV3],
'expires_at': <class 'str'>, 'is_admin_project':
typing_extensions.NotRequired[bool], 'is_domain':
typing_extensions.NotRequired[bool], 'issued_at': <class 'str'>,
'methods': list[str], 'oauth2_credential':
typing_extensions.NotRequired[keystoneauth1.access.types.OAuth2V3],
'project': <class 'keystoneauth1.access.types.ProjectV3'>, 'roles':
list[keystoneauth1.access.types.RoleV3], 'service_providers':
typing_extensions.NotRequired[list[keystoneauth1.access.types.
ServiceProviderV3]], 'system':
typing_extensions.NotRequired[keystoneauth1.access.types.SystemV3],
'user': <class 'keystoneauth1.access.types.UserV3'>}
```

```
__dict__ = mappingproxy({'__annotations__': {'application_credential':
typing_extensions.NotRequired[keystoneauth1.access.types.
ApplicationCredentialV3], 'audit_ids': list[str], 'bind':
typing_extensions.NotRequired[dict[str, typing.Any]], 'catalog':
typing_extensions.NotRequired[list[keystoneauth1.access.types.ServiceV3]],
'domain':
typing_extensions.NotRequired[keystoneauth1.access.types.DomainV3],
'expires_at': <class 'str'>, 'is_admin_project':
typing_extensions.NotRequired[bool], 'is_domain':
typing_extensions.NotRequired[bool], 'issued_at': <class 'str'>,
'methods': list[str], 'oauth2_credential':
typing_extensions.NotRequired[keystoneauth1.access.types.OAuth2V3],
'project': <class 'keystoneauth1.access.types.ProjectV3'>, 'roles':
list[keystoneauth1.access.types.RoleV3], 'service_providers':
typing_extensions.NotRequired[list[keystoneauth1.access.types.
ServiceProviderV3]], 'system':
typing_extensions.NotRequired[keystoneauth1.access.types.SystemV3],
'user': <class 'keystoneauth1.access.types.UserV3'>, 'OS-OAUTH1':
typing_extensions.NotRequired[keystoneauth1.access.types.OAuth1V3],
'OS-TRUST:trust':
typing_extensions.NotRequired[keystoneauth1.access.types.TrustV3]},
'__module__': 'keystoneauth1.access.types', '__dict__': <attribute
'__dict__' of 'TokenV3' objects>, '__weakref__': <attribute '__weakref__'
of 'TokenV3' objects>, '__doc__': None, '__required_keys__':
frozenset({'domain', 'catalog', 'expires_at', 'methods', 'user',
'OS-OAUTH1', 'service_providers', 'oauth2_credential', 'issued_at',
'bind', 'is_domain', 'audit_ids', 'is_admin_project', 'project', 'roles',
'system', 'OS-TRUST:trust', 'application_credential'}),
'__optional_keys__': frozenset(), '__total__': True})
```

`__doc__ = None`

`__module__ = 'keystoneauth1.access.types'`

`__optional_keys__ = frozenset({})`

`__required_keys__ = frozenset({'OS-OAUTH1', 'OS-TRUST:trust', 'application_credential', 'audit_ids', 'bind', 'catalog', 'domain', 'expires_at', 'is_admin_project', 'is_domain', 'issued_at', 'methods', 'oauth2_credential', 'project', 'roles', 'service_providers', 'system', 'user'})`

`__total__ = True`

`__weakref__`

list of weak references to the object (if defined)

`application_credential: NotRequired[ApplicationCredentialV3]`

`audit_ids: list[str]`

`bind: NotRequired[dict[str, Any]]`

`catalog: NotRequired[list[ServiceV3]]`

`domain: NotRequired[DomainV3]`

`expires_at: str`

`is_admin_project: NotRequired[bool]`

`is_domain: NotRequired[bool]`

`issued_at: str`

`methods: list[str]`

`oauth2_credential: NotRequired[OAuth2V3]`

`project: ProjectV3`

`roles: list[RoleV3]`

`service_providers: NotRequired[list[ServiceProviderV3]]`

`system: NotRequired[SystemV3]`

`user: UserV3`

`class keystoneauth1.access.types.TrustV2`

Bases: `TypedDict`

`__annotations__ = {'id': <class 'str'>, 'impersonation': <class 'bool'>, 'trustee_user_id': <class 'str'>, 'trustor_user_id': <class 'str'>}`

__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'id': <class 'str'>, 'impersonation': <class 'bool'>, 'trustee_user_id': <class 'str'>, 'trustor_user_id': <class 'str'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'TrustV2' objects>, '__weakref__': <attribute '__weakref__' of 'TrustV2' objects>, '__doc__': None, '__required_keys__': frozenset({'trustor_user_id', 'trustee_user_id', 'impersonation', 'id'}), '__optional_keys__': frozenset(), '__total__': True})

__doc__ = None

__module__ = 'keystoneauth1.access.types'

__optional_keys__ = frozenset({})

__orig_bases__ = (<function TypedDict>,)

__required_keys__ = frozenset({'id', 'impersonation', 'trustee_user_id', 'trustor_user_id'})

__total__ = True

__weakref__

list of weak references to the object (if defined)

id: str

impersonation: bool

trustee_user_id: str

trustor_user_id: str

class keystoneauth1.access.types.**TrustV3**

Bases: TypedDict

__annotations__ = {'id': <class 'str'>, 'impersonation': <class 'bool'>, 'trustee_user': <class 'keystoneauth1.access.types.TrusteeUser'>, 'trustor_user': <class 'keystoneauth1.access.types.TrustorUser'>}

__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'id': <class 'str'>, 'impersonation': <class 'bool'>, 'trustee_user': <class 'keystoneauth1.access.types.TrusteeUser'>, 'trustor_user': <class 'keystoneauth1.access.types.TrustorUser'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'TrustV3' objects>, '__weakref__': <attribute '__weakref__' of 'TrustV3' objects>, '__doc__': None, '__required_keys__': frozenset({'trustee_user', 'impersonation', 'id', 'trustor_user'}), '__optional_keys__': frozenset(), '__total__': True})

__doc__ = None

__module__ = 'keystoneauth1.access.types'

`__optional_keys__ = frozenset({})`

`__orig_bases__ = (<function TypedDict>,)`

`__required_keys__ = frozenset({'id', 'impersonation', 'trustee_user', 'trustor_user'})`

`__total__ = True`

`__weakref__`

list of weak references to the object (if defined)

`id: str`

`impersonation: bool`

`trustee_user: TrusteeUser`

`trustor_user: TrustorUser`

`class keystoneauth1.access.types.TrusteeUser`

Bases: TypedDict

`__annotations__ = {'id': <class 'str'>}`

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'id': <class 'str'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'TrusteeUser' objects>, '__weakref__': <attribute '__weakref__' of 'TrusteeUser' objects>, '__doc__': None, '__required_keys__': frozenset({'id'}), '__optional_keys__': frozenset(), '__total__': True})`

`__doc__ = None`

`__module__ = 'keystoneauth1.access.types'`

`__optional_keys__ = frozenset({})`

`__orig_bases__ = (<function TypedDict>,)`

`__required_keys__ = frozenset({'id'})`

`__total__ = True`

`__weakref__`

list of weak references to the object (if defined)

`id: str`

`class keystoneauth1.access.types.TrustorUser`

Bases: TypedDict

`__annotations__ = {'id': <class 'str'>}`

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'id': <class 'str'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'TrustorUser' objects>, '__weakref__': <attribute '__weakref__' of 'TrustorUser' objects>, '__doc__': None, '__required_keys__': frozenset({'id'}), '__optional_keys__': frozenset(), '__total__': True})`

`__doc__ = None`

`__module__ = 'keystoneauth1.access.types'`

`__optional_keys__ = frozenset({})`

`__orig_bases__ = (<function TypedDict>,)`

`__required_keys__ = frozenset({'id'})`

`__total__ = True`

`__weakref__`

list of weak references to the object (if defined)

`id: str`

`class keystoneauth1.access.types.UserDomainV3`

Bases: TypedDict

`__annotations__ = {'id': <class 'str'>, 'name': <class 'str'>}`

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'id': <class 'str'>, 'name': <class 'str'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'UserDomainV3' objects>, '__weakref__': <attribute '__weakref__' of 'UserDomainV3' objects>, '__doc__': None, '__required_keys__': frozenset({'name', 'id'}), '__optional_keys__': frozenset(), '__total__': True})`

`__doc__ = None`

`__module__ = 'keystoneauth1.access.types'`

`__optional_keys__ = frozenset({})`

`__orig_bases__ = (<function TypedDict>,)`

`__required_keys__ = frozenset({'id', 'name'})`

`__total__ = True`

`__weakref__`

list of weak references to the object (if defined)

`id: str`

`name: str`

class keystoneauth1.access.types.**UserV2**

Bases: TypedDict

__annotations__ = {'id': <class 'str'>, 'name': <class 'str'>, 'role_links': list[typing.Any], 'roles': list[keystoneauth1.access.types.RoleV2], 'tenantId': typing_extensions.NotRequired[str], 'tenantName': typing_extensions.NotRequired[str], 'username': <class 'str'>}

__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.types', '__annotations__': {'id': <class 'str'>, 'name': <class 'str'>, 'role_links': list[typing.Any], 'roles': list[keystoneauth1.access.types.RoleV2], 'tenantId': typing_extensions.NotRequired[str], 'tenantName': typing_extensions.NotRequired[str], 'username': <class 'str'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'UserV2' objects>, '__weakref__': <attribute '__weakref__' of 'UserV2' objects>, '__doc__': None, '__required_keys__': frozenset({'role_links', 'roles', 'username', 'id', 'name', 'tenantName', 'tenantId'}), '__optional_keys__': frozenset(), '__total__': True})

__doc__ = None

__module__ = 'keystoneauth1.access.types'

__optional_keys__ = frozenset({})

__orig_bases__ = (<function TypedDict>,)

__required_keys__ = frozenset({'id', 'name', 'role_links', 'roles', 'tenantId', 'tenantName', 'username'})

__total__ = True

__weakref__

list of weak references to the object (if defined)

id: str

name: str

role_links: list[Any]

roles: list[RoleV2]

tenantId: NotRequired[str]

tenantName: NotRequired[str]

username: str

class keystoneauth1.access.types.**UserV3**

Bases: dict

`__annotations__ = {'OS-FEDERATION': typing_extensions.NotRequired[keystoneauth1.access.types.FederationV3], 'domain': <class 'keystoneauth1.access.types.UserDomainV3'>, 'id': <class 'str'>, 'name': <class 'str'>, 'password_expires_at': typing_extensions.NotRequired[str]}`

`__dict__ = mappingproxy({'__annotations__': {'domain': <class 'keystoneauth1.access.types.UserDomainV3'>, 'id': <class 'str'>, 'name': <class 'str'>, 'password_expires_at': typing_extensions.NotRequired[str], 'OS-FEDERATION': typing_extensions.NotRequired[keystoneauth1.access.types.FederationV3]}, '__module__': 'keystoneauth1.access.types', '__dict__': <attribute '__dict__' of 'UserV3' objects>, '__weakref__': <attribute '__weakref__' of 'UserV3' objects>, '__doc__': None, '__required_keys__': frozenset({'domain', 'id', 'name', 'password_expires_at', 'OS-FEDERATION'}), '__optional_keys__': frozenset(), '__total__': True})`

`__doc__ = None`

`__module__ = 'keystoneauth1.access.types'`

`__optional_keys__ = frozenset({})`

`__required_keys__ = frozenset({'OS-FEDERATION', 'domain', 'id', 'name', 'password_expires_at'})`

`__total__ = True`

`__weakref__`

list of weak references to the object (if defined)

`domain: UserDomainV3`

`id: str`

`name: str`

`password_expires_at: NotRequired[str]`

Module contents

class `keystoneauth1.access.AccessInfo`(*body: dict[str, Any], auth_token: str | None = None*)

Bases: `object`

Encapsulates a raw authentication token from keystone.

Provides helper methods for extracting useful values from that token.

`__annotations__ = {'_data': typing.Any, '_service_catalog_class': typing.Type[keystoneauth1.access.service_catalog.ServiceCatalog]}`

__dict__ = mappingproxy({'__module__': 'keystoneauth1.access.access', '__annotations__': {'_service_catalog_class': typing.Type[keystoneauth1.access.service_catalog.ServiceCatalog], '_data': typing.Any, '_service_catalog': 'ty.Optional[service_catalog.ServiceCatalog]', '_service_providers': 'ty.Optional[service_providers.ServiceProviders]'}, '__doc__': 'Encapsulates a raw authentication token from keystone.\n\n Provides helper methods for extracting useful values from that token.\n ', '__init__': <function AccessInfo.__init__>, 'service_catalog': <property object>, 'will_expire_soon': <function AccessInfo.will_expire_soon>, 'has_service_catalog': <function AccessInfo.has_service_catalog>, 'auth_token': <property object>, 'expires': <property object>, 'issued': <property object>, 'username': <property object>, 'user_id': <property object>, 'user_domain_id': <property object>, 'user_domain_name': <property object>, 'role_ids': <property object>, 'role_names': <property object>, 'domain_name': <property object>, 'domain_id': <property object>, 'project_name': <property object>, 'tenant_name': <property object>, 'scoped': <property object>, 'project_scoped': <property object>, 'domain_scoped': <property object>, 'system_scoped': <property object>, 'trust_id': <property object>, 'trust_scoped': <property object>, 'trustee_user_id': <property object>, 'trustor_user_id': <property object>, 'project_id': <property object>, 'tenant_id': <property object>, 'project_domain_id': <property object>, 'project_domain_name': <property object>, 'oauth_access_token_id': <property object>, 'oauth_consumer_id': <property object>, 'is_federated': <property object>, 'is_admin_project': <property object>, 'audit_id': <property object>, 'audit_chain_id': <property object>, 'initial_audit_id': <property object>, 'service_providers': <property object>, 'bind': <property object>, 'project_is_domain': <property object>, '__dict__': <attribute '__dict__' of 'AccessInfo' objects>, '__weakref__': <attribute '__weakref__' of 'AccessInfo' objects>})

__doc__ = 'Encapsulates a raw authentication token from keystone.\n\n Provides helper methods for extracting useful values from that token.\n '

__init__(*body: dict[str, Any], auth_token: str | None = None*)

__module__ = 'keystoneauth1.access.access'

__weakref__

list of weak references to the object (if defined)

_data: Any

_service_catalog_class: Type[ServiceCatalog]

property audit_chain_id: str | None

Return the audit chain ID if present.

In the event that a token was rescoped then this ID will be the `audit_id` of the initial token. Returns None if no value present.

Returns
str or None.

property audit_id: str | None

Return the audit ID if present.

Returns
str or None.

property auth_token: str | None

Return the token_id associated with the auth request.

To be used in headers for authenticating OpenStack API requests.

Returns
str

property bind: dict[str, Any] | None

Information about external mechanisms the token is bound to.

If a token is bound to an external authentication mechanism it can only be used in conjunction with that mechanism. For example if bound to a kerberos principal it may only be accepted if there is also kerberos authentication performed on the request.

Returns
A dictionary or None. The key will be the bind type the value is a dictionary that is specific to the format of the bind type. Returns None if there is no bind information in the token.

property domain_id: str | None

Return the domain id associated with the auth request.

Returns
str or None (if no domain associated with the token)

property domain_name: str | None

Return the domain name associated with the auth request.

Returns
str or None (if no domain associated with the token)

property domain_scoped: bool

Return true if the auth token was scoped to a domain.

Returns
bool

property expires: datetime | None

Return the token expiration (as datetime object).

Returns
datetime

has_service_catalog() → bool

Return true if the auth token has a service catalog.

Returns
boolean

property initial_audit_id: str | None

The audit ID of the initially requested token.

This is the `audit_chain_id` if present or the `audit_id`.

property is_admin_project: bool

Return true if the current project scope is the admin project.

For backwards compatibility purposes if there is nothing specified in the token we always assume we are in the admin project, so this will default to True.

:returns boolean

property is_federated: bool

Return true if federation was used to get the token.

Returns
boolean

property issued: datetime | None

Return the token issue time (as datetime object).

Returns
datetime

property oauth_access_token_id: str | None

Return the access token ID if OAuth authentication used.

Returns
str or None.

property oauth_consumer_id: str | None

Return the consumer ID if OAuth authentication used.

Returns
str or None.

property project_domain_id: str | None

Return the projects domain id associated with the auth request.

Returns
str

property project_domain_name: str | None

Return the projects domain name associated with the auth request.

Returns
str

property project_id: str | None

Return the project ID associated with the auth request.

This returns None if the auth token wasnt scoped to a project.

Returns
str or None (if no project associated with the token)

property project_is_domain: bool | None

Return if a project act as a domain.

Returns
bool

property project_name: str | None

Return the project name associated with the auth request.

Returns
str or None (if no project associated with the token)

property project_scoped: bool

Return true if the auth token was scoped to a tenant (project).

Returns
bool

property role_ids: list[str] | None

Return a list of users role ids associated with the auth request.

Returns
a list of strings of role ids

property role_names: list[str] | None

Return a list of users role names associated with the auth request.

Returns
a list of strings of role names

property scoped: bool

Return true if the auth token was scoped.

Returns true if scoped to a tenant(project) or domain, and contains a populated service catalog.

This is deprecated, use project_scoped instead.

Returns
bool

property service_catalog: ServiceCatalog

property service_providers: ServiceProviders | None

Return an object representing the list of trusted service providers.

Used for Keystone2Keystone federating-out.

Returns
`keystoneauth1.service_providers.ServiceProviders` or None

property system_scoped: bool

Return true if the auth token was scoped to the system.

Returns
bool

property tenant_id: str | None

Synonym for project_id.

property tenant_name: str | None

Synonym for project_name.

property trust_id: str | None

Return the trust id associated with the auth request.

Returns
str or None (if no trust associated with the token)

property trust_scoped: bool

Return true if the auth token was scoped from a delegated trust.

The trust delegation is via the OS-TRUST v3 extension.

Returns
bool

property trustee_user_id: str | None

Return the trustee user id associated with a trust.

Returns
str or None (if no trust associated with the token)

property trustor_user_id: str | None

Return the trustor user id associated with a trust.

Returns
str or None (if no trust associated with the token)

property user_domain_id: str | None

Return the users domain id associated with the auth request.

Returns
str

property user_domain_name: str | None

Return the users domain name associated with the auth request.

Returns
str

property user_id: str | None

Return the user id associated with the auth request.

Returns
str

property username: str | None

Return the username associated with the auth request.

Follows the pattern defined in the V2 API of first looking for name, returning that if available, and falling back to username if name is unavailable.

Returns
str

will_expire_soon(*stale_duration: int = 30*) → bool

Determine if expiration is about to occur.

Returns
true if expiration is within the given duration

Return type
boolean

class `keystoneauth1.access.`**`AccessInfoV2`**(*body: dict[str, Any], auth_token: str | None = None*)

Bases: `AccessInfo`

An object for encapsulating raw v2 auth token from identity service.

`__annotations__ = {'_data': <class 'keystoneauth1.access.types.TokenResponseV2'>}`

`__doc__ = 'An object for encapsulating raw v2 auth token from identity service.'`

`__module__ = 'keystoneauth1.access.access'`

`_data: TokenResponseV2`

`_service_catalog_class`

alias of `ServiceCatalogV2`

`property _token: TokenV2`

`property _trust: TrustV2 | None`

`property _user: UserV2`

`property audit_chain_id: str | None`

Return the audit chain ID if present.

In the event that a token was rescoped then this ID will be the `audit_id` of the initial token. Returns None if no value present.

Returns
str or None.

`property audit_id: str | None`

Return the audit ID if present.

Returns
str or None.

`property auth_token: str | None`

Return the token_id associated with the auth request.

To be used in headers for authenticating OpenStack API requests.

Returns
str

`property bind: dict[str, Any] | None`

Information about external mechanisms the token is bound to.

If a token is bound to an external authentication mechanism it can only be used in conjunction with that mechanism. For example if bound to a kerberos principal it may only be accepted if there is also kerberos authentication performed on the request.

Returns
A dictionary or None. The key will be the bind type the value is a dictionary that is specific to the format of the bind type. Returns None if there is no bind information in the token.

property domain_id: str | None

Return the domain id associated with the auth request.

Returns
str or None (if no domain associated with the token)

property domain_name: str | None

Return the domain name associated with the auth request.

Returns
str or None (if no domain associated with the token)

property domain_scoped: bool

Return true if the auth token was scoped to a domain.

Returns
bool

property expires: datetime | None

Return the token expiration (as datetime object).

Returns
datetime

has_service_catalog() → bool

Return true if the auth token has a service catalog.

Returns
boolean

property is_admin_project: bool

Return true if the current project scope is the admin project.

For backwards compatibility purposes if there is nothing specified in the token we always assume we are in the admin project, so this will default to True.

:returns boolean

property is_federated: bool

Return true if federation was used to get the token.

Returns
boolean

property issued: datetime | None

Return the token issue time (as datetime object).

Returns
datetime

property oauth_access_token_id: str | None

Return the access token ID if OAuth authentication used.

Returns
str or None.

property oauth_consumer_id: `str | None`

Return the consumer ID if OAuth authentication used.

Returns
str or None.

property project_domain_id: `str | None`

Return the projects domain id associated with the auth request.

Returns
str

property project_domain_name: `str | None`

Return the projects domain name associated with the auth request.

Returns
str

property project_id: `str | None`

Return the project ID associated with the auth request.

This returns None if the auth token wasnt scoped to a project.

Returns
str or None (if no project associated with the token)

property project_is_domain: `bool | None`

Return if a project act as a domain.

Returns
bool

property project_name: `str | None`

Return the project name associated with the auth request.

Returns
str or None (if no project associated with the token)

property role_ids: `list[str] | None`

Return a list of users role ids associated with the auth request.

Returns
a list of strings of role ids

property role_names: `list[str] | None`

Return a list of users role names associated with the auth request.

Returns
a list of strings of role names

property service_providers: `ServiceProviders | None`

Return an object representing the list of trusted service providers.

Used for Keystone2Keystone federating-out.

Returns
`keystoneauth1.service_providers.ServiceProviders` or None

property system_scoped: bool

Return true if the auth token was scoped to the system.

Returns
bool

property trust_id: str | None

Return the trust id associated with the auth request.

Returns
str or None (if no trust associated with the token)

property trust_scoped: bool

Return true if the auth token was scoped from a delegated trust.

The trust delegation is via the OS-TRUST v3 extension.

Returns
bool

property trustee_user_id: str | None

Return the trustee user id associated with a trust.

Returns
str or None (if no trust associated with the token)

property trustor_user_id: str | None

Return the trustor user id associated with a trust.

Returns
str or None (if no trust associated with the token)

property user_domain_id: str | None

Return the users domain id associated with the auth request.

Returns
str

property user_domain_name: str | None

Return the users domain name associated with the auth request.

Returns
str

property user_id: str | None

Return the user id associated with the auth request.

Returns
str

property username: str | None

Return the username associated with the auth request.

Follows the pattern defined in the V2 API of first looking for name, returning that if available, and falling back to username if name is unavailable.

Returns
str

version = 'v2.0'

class keystoneauth1.access.**AccessInfoV3**(*body: dict[str, Any], auth_token: str | None = None*)

Bases: `AccessInfo`

An object encapsulating raw v3 auth token from identity service.

__annotations__ = {'_data': <class 'keystoneauth1.access.types.TokenResponseV3'>}

__doc__ = 'An object encapsulating raw v3 auth token from identity service.'

__module__ = 'keystoneauth1.access.access'

property _application_credential: ApplicationCredentialV3 | None

_data: TokenResponseV3

property _domain: DomainV3 | None

property _oauth: OAuth1V3 | None

property _oauth2_credential: OAuth2V3 | None

property _project: ProjectV3 | None

property _project_domain: ProjectDomainV3 | None

_service_catalog_class

alias of `ServiceCatalogV3`

property _token: TokenV3

property _trust: TrustV3 | None

property _user: UserV3

property _user_domain: UserDomainV3

property application_credential: ApplicationCredentialV3

property application_credential_access_rules: list[ApplicationCredentialAccessRuleV3] | None

property application_credential_id: str | None

property audit_chain_id: str | None

Return the audit chain ID if present.

In the event that a token was rescoped then this ID will be the `audit_id` of the initial token. Returns None if no value present.

Returns

str or None.

`property audit_id: str | None`

Return the audit ID if present.

Returns
str or None.

`property bind: dict[str, Any] | None`

Information about external mechanisms the token is bound to.

If a token is bound to an external authentication mechanism it can only be used in conjunction with that mechanism. For example if bound to a kerberos principal it may only be accepted if there is also kerberos authentication performed on the request.

Returns
A dictionary or None. The key will be the bind type the value is a dictionary that is specific to the format of the bind type. Returns None if there is no bind information in the token.

`property domain_id: str | None`

Return the domain id associated with the auth request.

Returns
str or None (if no domain associated with the token)

`property domain_name: str | None`

Return the domain name associated with the auth request.

Returns
str or None (if no domain associated with the token)

`property domain_scoped: bool`

Return true if the auth token was scoped to a domain.

Returns
bool

`property expires: datetime | None`

Return the token expiration (as datetime object).

Returns
datetime

`has_service_catalog()` → bool

Return true if the auth token has a service catalog.

Returns
boolean

`property is_admin_project: bool`

Return true if the current project scope is the admin project.

For backwards compatibility purposes if there is nothing specified in the token we always assume we are in the admin project, so this will default to True.

:returns boolean

`property is_federated: bool`

Return true if federation was used to get the token.

Returns
boolean

property issued: datetime | None

Return the token issue time (as datetime object).

Returns
datetime

property oauth2_credential: OAuth2V3

property oauth2_credential_thumbprint: str | None

property oauth_access_token_id: str | None

Return the access token ID if OAuth authentication used.

Returns
str or None.

property oauth_consumer_id: str | None

Return the consumer ID if OAuth authentication used.

Returns
str or None.

property project_domain_id: str | None

Return the projects domain id associated with the auth request.

Returns
str

property project_domain_name: str | None

Return the projects domain name associated with the auth request.

Returns
str

property project_id: str | None

Return the project ID associated with the auth request.

This returns None if the auth token wasnt scoped to a project.

Returns
str or None (if no project associated with the token)

property project_is_domain: bool | None

Return if a project act as a domain.

Returns
bool

property project_name: str | None

Return the project name associated with the auth request.

Returns
str or None (if no project associated with the token)

property role_ids: list[str] | None

Return a list of users role ids associated with the auth request.

Returns
a list of strings of role ids

property role_names: list[str] | None

Return a list of users role names associated with the auth request.

Returns
a list of strings of role names

property service_providers: ServiceProviders | None

Return an object representing the list of trusted service providers.

Used for Keystone2Keystone federating-out.

Returns
`keystoneauth1.service_providers.ServiceProviders` or None

property system: SystemV3 | None

property system_scoped: bool

Return true if the auth token was scoped to the system.

Returns
bool

property trust_id: str | None

Return the trust id associated with the auth request.

Returns
str or None (if no trust associated with the token)

property trust_scoped: bool

Return true if the auth token was scoped from a delegated trust.

The trust delegation is via the OS-TRUST v3 extension.

Returns
bool

property trustee_user_id: str | None

Return the trustee user id associated with a trust.

Returns
str or None (if no trust associated with the token)

property trustor_user_id: str | None

Return the trustor user id associated with a trust.

Returns
str or None (if no trust associated with the token)

property user_domain_id: str | None

Return the users domain id associated with the auth request.

Returns
str

property user_domain_name: str | None

Return the users domain name associated with the auth request.

Returns
str

property user_id: str | None

Return the user id associated with the auth request.

Returns
str

property username: str | None

Return the username associated with the auth request.

Follows the pattern defined in the V2 API of first looking for name, returning that if available, and falling back to username if name is unavailable.

Returns
str

version = 'v3'

keystoneauth1.access.**create**(*resp: Response* | *None* = *None*, *body: dict[str, object]* | *None* = *None*, *auth_token: str* | *None* = *None*) → AccessInfo

keystoneauth1.exceptions package

Submodules

keystoneauth1.exceptions.auth module

exception keystoneauth1.exceptions.auth.**AuthorizationFailure**(*message: str* | *None* = *None*)

Bases: ClientException

__doc__ = None

__module__ = 'keystoneauth1.exceptions.auth'

message = 'Cannot authorize API client.'

exception keystoneauth1.exceptions.auth.**MissingAuthMethods**(*response: Response*)

Bases: ClientException

__annotations__ = {}

__doc__ = None

__init__(*response: Response*)

__module__ = 'keystoneauth1.exceptions.auth'

message = 'Not all required auth rules were satisfied'

keystoneauth1.exceptions.auth_plugins module

exception `keystoneauth1.exceptions.auth_plugins.`**`AuthPluginException`**(*message: str | None = None*)

Bases: `ClientException`

`__annotations__ = {}`

`__doc__ = None`

`__module__ = 'keystoneauth1.exceptions.auth_plugins'`

`message = 'Unknown error with authentication plugins.'`

exception `keystoneauth1.exceptions.auth_plugins.`**`MissingAuthPlugin`**(*message: str | None = None*)

Bases: `AuthPluginException`

`__annotations__ = {}`

`__doc__ = None`

`__module__ = 'keystoneauth1.exceptions.auth_plugins'`

`message = 'An authenticated request is required but no plugin available.'`

exception `keystoneauth1.exceptions.auth_plugins.`**`MissingRequiredOptions`**(*options: Sequence[loading.Opt]*)

Bases: `OptionError`

One or more required options were not provided.

Parameters
options (*`list`(`keystoneauth1.loading.Opt`)*) Missing options.

`options`

List of the missing options.

`__annotations__ = {}`

`__doc__ = 'One or more required options were not provided.\n\n :param list(keystoneauth1.loading.Opt) options:  Missing options.\n\n .. py:attribute::  options\n\n List of the missing options.\n '`

`__init__`(*options: Sequence[loading.Opt]*)

`__module__ = 'keystoneauth1.exceptions.auth_plugins'`

exception `keystoneauth1.exceptions.auth_plugins.`**`NoMatchingPlugin`**(*name: str*)

Bases: `AuthPluginException`

No auth plugins could be created from the parameters provided.

Parameters
name (*`str`*) The name of the plugin that was attempted to load.

name

The name of the plugin that was attempted to load.

__annotations__ = {}

__doc__ = 'No auth plugins could be created from the parameters provided.\n\n :param str name: The name of the plugin that was attempted to load.\n\n .. py:attribute:: name\n\n The name of the plugin that was attempted to load.\n '

__init__(*name: str*)

__module__ = 'keystoneauth1.exceptions.auth_plugins'

exception `keystoneauth1.exceptions.auth_plugins.`**OptionError**(*message: str | None = None*)

Bases: `AuthPluginException`

A requirement of this plugin loader was not met.

This error can be raised by a specific plugin loader during the load_from_options stage to indicate a parameter problem that can not be handled by the generic options loader.

The intention here is that a plugin can do checks like if a name parameter is provided then a domain parameter must also be provided, but that Opt checking doesnt handle.

__annotations__ = {}

__doc__ = "A requirement of this plugin loader was not met.\n\n This error can be raised by a specific plugin loader during the\n load_from_options stage to indicate a parameter problem that can not be\n handled by the generic options loader.\n\n The intention here is that a plugin can do checks like if a name parameter\n is provided then a domain parameter must also be provided, but that Opt\n checking doesn't handle.\n "

__module__ = 'keystoneauth1.exceptions.auth_plugins'

exception `keystoneauth1.exceptions.auth_plugins.`**UnsupportedParameters**(*names: Sequence[str]*)

Bases: `AuthPluginException`

A parameter that was provided or returned is not supported.

Parameters

names (*list(str)*) Names of the unsupported parameters.

names

Names of the unsupported parameters.

__annotations__ = {}

__doc__ = 'A parameter that was provided or returned is not supported.\n\n :param list(str) names: Names of the unsupported parameters.\n\n .. py:attribute:: names\n\n Names of the unsupported parameters.\n '

__init__(*names: Sequence[str]*)

__module__ = 'keystoneauth1.exceptions.auth_plugins'

keystoneauth1.exceptions.base module

exception keystoneauth1.exceptions.base.**ClientException**(*message: str | None = None*)

Bases: `Exception`

The base exception for everything to do with clients.

`__annotations__ = {}`

`__doc__ = 'The base exception for everything to do with clients.'`

`__init__`(*message: str | None = None*)

`__module__ = 'keystoneauth1.exceptions.base'`

`__weakref__`

list of weak references to the object (if defined)

`message = 'ClientException'`

keystoneauth1.exceptions.catalog module

exception keystoneauth1.exceptions.catalog.**CatalogException**(*message: str | None = None*)

Bases: `ClientException`

`__annotations__ = {}`

`__doc__ = None`

`__module__ = 'keystoneauth1.exceptions.catalog'`

`message = 'Unknown error with service catalog.'`

exception keystoneauth1.exceptions.catalog.**EmptyCatalog**(*message: str | None = None*)

Bases: `EndpointNotFound`

`__annotations__ = {}`

`__doc__ = None`

`__module__ = 'keystoneauth1.exceptions.catalog'`

`message = 'The service catalog is empty.'`

exception keystoneauth1.exceptions.catalog.**EndpointNotFound**(*message: str | None = None*)

Bases: `CatalogException`

`__annotations__ = {}`

`__doc__ = None`

`__module__ = 'keystoneauth1.exceptions.catalog'`

`message = 'Could not find requested endpoint in Service Catalog.'`

keystoneauth1.exceptions.connection module

exception keystoneauth1.exceptions.connection.**ConnectFailure**(*message: str | None = None*)

Bases: ConnectionError, RetriableConnectionFailure

__annotations__ = {}

__doc__ = None

__module__ = 'keystoneauth1.exceptions.connection'

message = 'Connection failure that may be retried.'

exception keystoneauth1.exceptions.connection.**ConnectTimeout**(*message: str | None = None*)

Bases: ConnectionError, RetriableConnectionFailure

__annotations__ = {}

__doc__ = None

__module__ = 'keystoneauth1.exceptions.connection'

message = 'Timed out connecting to service.'

exception keystoneauth1.exceptions.connection.**ConnectionError**(*message: str | None = None*)

Bases: ClientException

__annotations__ = {}

__doc__ = None

__module__ = 'keystoneauth1.exceptions.connection'

message = 'Cannot connect to API service.'

exception keystoneauth1.exceptions.connection.**RetriableConnectionFailure**

Bases: Exception

A mixin class that implies you can retry the most recent request.

__annotations__ = {}

__doc__ = 'A mixin class that implies you can retry the most recent request.'

__module__ = 'keystoneauth1.exceptions.connection'

__weakref__

list of weak references to the object (if defined)

exception keystoneauth1.exceptions.connection.**SSLError**(*message: str | None = None*)

Bases: ConnectionError

__annotations__ = {}

__doc__ = None

__module__ = 'keystoneauth1.exceptions.connection'

message = 'An SSL error occurred.'

exception keystoneauth1.exceptions.connection.**UnknownConnectionError**(*msg: str, original: Exception*)

Bases: ConnectionError

An error was encountered but we dont know what it is.

__annotations__ = {}

__doc__ = "An error was encountered but we don't know what it is."

__init__(*msg: str, original: Exception*)

__module__ = 'keystoneauth1.exceptions.connection'

keystoneauth1.exceptions.discovery module

exception keystoneauth1.exceptions.discovery.**DiscoveryFailure**(*message: str | None = None*)

Bases: ClientException

__annotations__ = {}

__doc__ = None

__module__ = 'keystoneauth1.exceptions.discovery'

message = 'Discovery of client versions failed.'

exception keystoneauth1.exceptions.discovery.**ImpliedMaxVersionMismatch**(*service_type: str, implied: tuple[int | float, ...], given: str*)

Bases: ImpliedVersionMismatch

__doc__ = None

__module__ = 'keystoneauth1.exceptions.discovery'

label = 'max_version'

exception keystoneauth1.exceptions.discovery.**ImpliedMinVersionMismatch**(*service_type: str, implied: tuple[int | float, ...], given: str*)

Bases: `ImpliedVersionMismatch`

`__annotations__` `= {}`

`__doc__` `= None`

`__module__` `= 'keystoneauth1.exceptions.discovery'`

`label` `= 'min_version'`

exception `keystoneauth1.exceptions.discovery.`**`ImpliedVersionMismatch`**(*service_type: str*, *implied: tuple[int | float, ...]*, *given: str*)

Bases: `ValueError`

`__annotations__` `= {}`

`__doc__` `= None`

`__init__`(*service_type: str*, *implied: tuple[int | float, ...]*, *given: str*)

`__module__` `= 'keystoneauth1.exceptions.discovery'`

`__weakref__`

list of weak references to the object (if defined)

`label` `= 'version'`

exception `keystoneauth1.exceptions.discovery.`**`VersionNotAvailable`**(*message: str | None = None*)

Bases: `DiscoveryFailure`

`__annotations__` `= {}`

`__doc__` `= None`

`__module__` `= 'keystoneauth1.exceptions.discovery'`

`message` `= 'Discovery failed. Requested version is not available.'`

`keystoneauth1.exceptions.discovery.`**`_PARSED_VERSION_T`**

alias of `tuple`[`int` | `float`,]

keystoneauth1.exceptions.http module

HTTP Exceptions used by keystoneauth1.

exception `keystoneauth1.exceptions.http.`**`BadGateway`**(*message: str | None = None*, *details: str | None = None*, *response: Response | None = None*, *request_id: str | None = None*, *url: str | None = None*, *method: str | None = None*, *http_status: int | None = None*, *retry_after: int = 0*)

Bases: `HttpServerError`

HTTP 502 - Bad Gateway.

The server was acting as a gateway or proxy and received an invalid response from the upstream server.

`__annotations__ = {}`

`__doc__ = 'HTTP 502 - Bad Gateway.\n\n The server was acting as a gateway or proxy and received an invalid\n response from the upstream server.\n '`

`__module__ = 'keystoneauth1.exceptions.http'`

`http_status = 502`

`message = 'Bad Gateway'`

exception `keystoneauth1.exceptions.http.`**`BadRequest`**(*message: str | None = None*, *details: str | None = None*, *response: Response | None = None*, *request_id: str | None = None*, *url: str | None = None*, *method: str | None = None*, *http_status: int | None = None*, *retry_after: int = 0*)

Bases: `HTTPClientError`

HTTP 400 - Bad Request.

The request cannot be fulfilled due to bad syntax.

`__annotations__ = {}`

`__doc__ = 'HTTP 400 - Bad Request.\n\n The request cannot be fulfilled due to bad syntax.\n '`

`__module__ = 'keystoneauth1.exceptions.http'`

`http_status = 400`

`message = 'Bad Request'`

exception `keystoneauth1.exceptions.http.`**`Conflict`**(*message: str | None = None*, *details: str | None = None*, *response: Response | None = None*, *request_id: str | None = None*, *url: str | None = None*, *method: str | None = None*, *http_status: int | None = None*, *retry_after: int = 0*)

Bases: `HTTPClientError`

HTTP 409 - Conflict.

Indicates that the request could not be processed because of conflict in the request, such as an edit conflict.

__annotations__ = {}

__doc__ = 'HTTP 409 - Conflict.\n\n Indicates that the request could not be processed because of conflict\n in the request, such as an edit conflict.\n '

__module__ = 'keystoneauth1.exceptions.http'

http_status = 409

message = 'Conflict'

exception keystoneauth1.exceptions.http.**ExpectationFailed**(*message: str | None = None*, *details: str | None = None*, *response: Response | None = None*, *request_id: str | None = None*, *url: str | None = None*, *method: str | None = None*, *http_status: int | None = None*, *retry_after: int = 0*)

Bases: HTTPClientError

HTTP 417 - Expectation Failed.

The server cannot meet the requirements of the Expect request-header field.

__annotations__ = {}

__doc__ = 'HTTP 417 - Expectation Failed.\n\n The server cannot meet the requirements of the Expect request-header field.\n '

__module__ = 'keystoneauth1.exceptions.http'

http_status = 417

message = 'Expectation Failed'

exception keystoneauth1.exceptions.http.**Forbidden**(*message: str | None = None*, *details: str | None = None*, *response: Response | None = None*, *request_id: str | None = None*, *url: str | None = None*, *method: str | None = None*, *http_status: int | None = None*, *retry_after: int = 0*)

Bases: HTTPClientError

HTTP 403 - Forbidden.

The request was a valid request, but the server is refusing to respond to it.

__annotations__ = {}

__doc__ = 'HTTP 403 - Forbidden.\n\n The request was a valid request, but the server is refusing to respond\n to it.\n '

__module__ = 'keystoneauth1.exceptions.http'

http_status = 403

message = 'Forbidden'

exception keystoneauth1.exceptions.http.**GatewayTimeout**(*message: str | None = None, details: str | None = None, response: Response | None = None, request_id: str | None = None, url: str | None = None, method: str | None = None, http_status: int | None = None, retry_after: int = 0*)

Bases: `HttpServerError`

HTTP 504 - Gateway Timeout.

The server was acting as a gateway or proxy and did not receive a timely response from the upstream server.

__annotations__ = {}

__doc__ = 'HTTP 504 - Gateway Timeout.\n\n The server was acting as a gateway or proxy and did not receive a timely\n response from the upstream server.\n '

__module__ = 'keystoneauth1.exceptions.http'

http_status = 504

message = 'Gateway Timeout'

exception keystoneauth1.exceptions.http.**Gone**(*message: str | None = None, details: str | None = None, response: Response | None = None, request_id: str | None = None, url: str | None = None, method: str | None = None, http_status: int | None = None, retry_after: int = 0*)

Bases: `HTTPClientError`

HTTP 410 - Gone.

Indicates that the resource requested is no longer available and will not be available again.

__annotations__ = {}

__doc__ = 'HTTP 410 - Gone.\n\n Indicates that the resource requested is no longer available and will\n not be available again.\n '

__module__ = 'keystoneauth1.exceptions.http'

http_status = 410

message = 'Gone'

exception keystoneauth1.exceptions.http.**HTTPClientError**(*message: str | None = None, details: str | None = None, response: Response | None = None, request_id: str | None = None, url: str | None = None, method: str | None = None, http_status: int | None = None, retry_after: int = 0*)

Bases: HttpError

Client-side HTTP error.

Exception for cases in which the client seems to have erred.

__annotations__ = {}

__doc__ = 'Client-side HTTP error.\n\n Exception for cases in which the client seems to have erred.\n '

__module__ = 'keystoneauth1.exceptions.http'

message = 'HTTP Client Error'

exception keystoneauth1.exceptions.http.**HttpError**(*message: str | None = None, details: str | None = None, response: Response | None = None, request_id: str | None = None, url: str | None = None, method: str | None = None, http_status: int | None = None, retry_after: int = 0*)

Bases: ClientException

The base exception class for all HTTP exceptions.

__annotations__ = {}

__doc__ = 'The base exception class for all HTTP exceptions.'

__init__(*message: str | None = None, details: str | None = None, response: Response | None = None, request_id: str | None = None, url: str | None = None, method: str | None = None, http_status: int | None = None, retry_after: int = 0*)

__module__ = 'keystoneauth1.exceptions.http'

http_status = 0

message = 'HTTP Error'

exception keystoneauth1.exceptions.http.**HttpNotImplemented**(*message: str | None = None, details: str | None = None, response: Response | None = None, request_id: str | None = None, url: str | None = None, method: str | None = None, http_status: int | None = None, retry_after: int = 0*)

Bases: `HttpServerError`

HTTP 501 - Not Implemented.

The server either does not recognize the request method, or it lacks the ability to fulfill the request.

`__annotations__ = {}`

`__doc__ = 'HTTP 501 - Not Implemented.\n\n The server either does not recognize the request method, or it lacks\n the ability to fulfill the request.\n '`

`__module__ = 'keystoneauth1.exceptions.http'`

`http_status = 501`

`message = 'Not Implemented'`

exception keystoneauth1.exceptions.http.**HttpServerError**(*message: str | None = None, details: str | None = None, response: Response | None = None, request_id: str | None = None, url: str | None = None, method: str | None = None, http_status: int | None = None, retry_after: int = 0*)

Bases: `HttpError`

Server-side HTTP error.

Exception for cases in which the server is aware that it has erred or is incapable of performing the request.

`__annotations__ = {}`

`__doc__ = 'Server-side HTTP error.\n\n Exception for cases in which the server is aware that it has\n erred or is incapable of performing the request.\n '`

`__module__ = 'keystoneauth1.exceptions.http'`

`message = 'HTTP Server Error'`

exception keystoneauth1.exceptions.http.**HttpVersionNotSupported**(*message: str | None = None, details: str | None = None, response: Response | None = None, request_id: str | None = None, url: str | None = None, method: str | None = None, http_status: int | None = None, retry_after: int = 0*)

Bases: `HttpServerError`

HTTP 505 - HttpVersion Not Supported.

The server does not support the HTTP protocol version used in the request.

`__annotations__ = {}`

`__doc__ = 'HTTP 505 - HttpVersion Not Supported.\n\n The server does not support the HTTP protocol version used in the request.\n '`

`__module__ = 'keystoneauth1.exceptions.http'`

`http_status = 505`

`message = 'HTTP Version Not Supported'`

exception keystoneauth1.exceptions.http.**InternalServerError**(*message: str | None = None, details: str | None = None, response: Response | None = None, request_id: str | None = None, url: str | None = None, method: str | None = None, http_status: int | None = None, retry_after: int = 0*)

Bases: `HttpServerError`

HTTP 500 - Internal Server Error.

A generic error message, given when no more specific message is suitable.

`__annotations__ = {}`

`__doc__ = 'HTTP 500 - Internal Server Error.\n\n A generic error message, given when no more specific message is suitable.\n '`

`__module__ = 'keystoneauth1.exceptions.http'`

http_status = 500

message = 'Internal Server Error'

exception keystoneauth1.exceptions.http.**LengthRequired**(*message: str | None = None, details: str | None = None, response: Response | None = None, request_id: str | None = None, url: str | None = None, method: str | None = None, http_status: int | None = None, retry_after: int = 0*)

Bases: `HTTPClientError`

HTTP 411 - Length Required.

The request did not specify the length of its content, which is required by the requested resource.

__annotations__ = {}

__doc__ = 'HTTP 411 - Length Required.\n\n The request did not specify the length of its content, which is\n required by the requested resource.\n '

__module__ = 'keystoneauth1.exceptions.http'

http_status = 411

message = 'Length Required'

exception keystoneauth1.exceptions.http.**MethodNotAllowed**(*message: str | None = None, details: str | None = None, response: Response | None = None, request_id: str | None = None, url: str | None = None, method: str | None = None, http_status: int | None = None, retry_after: int = 0*)

Bases: `HTTPClientError`

HTTP 405 - Method Not Allowed.

A request was made of a resource using a request method not supported by that resource.

__annotations__ = {}

__doc__ = 'HTTP 405 - Method Not Allowed.\n\n A request was made of a resource using a request method not supported\n by that resource.\n '

__module__ = 'keystoneauth1.exceptions.http'

http_status = 405

message = 'Method Not Allowed'

exception keystoneauth1.exceptions.http.**NotAcceptable**(*message: str | None = None, details: str | None = None, response: Response | None = None, request_id: str | None = None, url: str | None = None, method: str | None = None, http_status: int | None = None, retry_after: int = 0*)

Bases: `HTTPClientError`

HTTP 406 - Not Acceptable.

The requested resource is only capable of generating content not acceptable according to the Accept headers sent in the request.

`__annotations__ = {}`

`__doc__ = 'HTTP 406 - Not Acceptable.\n\n The requested resource is only capable of generating content not\n acceptable according to the Accept headers sent in the request.\n '`

`__module__ = 'keystoneauth1.exceptions.http'`

`http_status = 406`

`message = 'Not Acceptable'`

exception keystoneauth1.exceptions.http.**NotFound**(*message: str | None = None, details: str | None = None, response: Response | None = None, request_id: str | None = None, url: str | None = None, method: str | None = None, http_status: int | None = None, retry_after: int = 0*)

Bases: `HTTPClientError`

HTTP 404 - Not Found.

The requested resource could not be found but may be available again in the future.

`__annotations__ = {}`

`__doc__ = 'HTTP 404 - Not Found.\n\n The requested resource could not be found but may be available again\n in the future.\n '`

`__module__ = 'keystoneauth1.exceptions.http'`

`http_status = 404`

`message = 'Not Found'`

exception keystoneauth1.exceptions.http.**PaymentRequired**(*message: str | None = None, details: str | None = None, response: Response | None = None, request_id: str | None = None, url: str | None = None, method: str | None = None, http_status: int | None = None, retry_after: int = 0*)

Bases: `HTTPClientError`

HTTP 402 - Payment Required.

Reserved for future use.

`__annotations__ = {}`

`__doc__ = 'HTTP 402 - Payment Required.\n\n Reserved for future use.\n '`

`__module__ = 'keystoneauth1.exceptions.http'`

`http_status = 402`

`message = 'Payment Required'`

exception keystoneauth1.exceptions.http.**PreconditionFailed**(*message: str | None = None, details: str | None = None, response: Response | None = None, request_id: str | None = None, url: str | None = None, method: str | None = None, http_status: int | None = None, retry_after: int = 0*)

Bases: `HTTPClientError`

HTTP 412 - Precondition Failed.

The server does not meet one of the preconditions that the requester put on the request.

`__annotations__ = {}`

`__doc__ = 'HTTP 412 - Precondition Failed.\n\n The server does not meet one of the preconditions that the requester\n put on the request.\n '`

`__module__ = 'keystoneauth1.exceptions.http'`

`http_status = 412`

`message = 'Precondition Failed'`

exception `keystoneauth1.exceptions.http.`**`ProxyAuthenticationRequired`**(*message: str | None = None, details: str | None = None, response: Response | None = None, request_id: str | None = None, url: str | None = None, method: str | None = None, http_status: int | None = None, retry_after: int = 0*)

Bases: `HTTPClientError`

HTTP 407 - Proxy Authentication Required.

The client must first authenticate itself with the proxy.

`__annotations__ = {}`

`__doc__ = 'HTTP 407 - Proxy Authentication Required.\n\n The client must first authenticate itself with the proxy.\n '`

`__module__ = 'keystoneauth1.exceptions.http'`

`http_status = 407`

`message = 'Proxy Authentication Required'`

exception `keystoneauth1.exceptions.http.`**`RequestEntityTooLarge`**(*message: str | None = None, details: str | None = None, response: Response | None = None, request_id: str | None = None, url: str | None = None, method: str | None = None, http_status: int | None = None, retry_after: int = 0*)

Bases: `HTTPClientError`

HTTP 413 - Request Entity Too Large.

The request is larger than the server is willing or able to process.

__annotations__ = {}

__doc__ = 'HTTP 413 - Request Entity Too Large.\n\n The request is larger than the server is willing or able to process.\n '

__init__(*message: str | None = None*, *details: str | None = None*, *response: Response | None = None*, *request_id: str | None = None*, *url: str | None = None*, *method: str | None = None*, *http_status: int | None = None*, *retry_after: int = 0*)

__module__ = 'keystoneauth1.exceptions.http'

http_status = 413

message = 'Request Entity Too Large'

exception keystoneauth1.exceptions.http.**RequestTimeout**(*message: str | None = None*, *details: str | None = None*, *response: Response | None = None*, *request_id: str | None = None*, *url: str | None = None*, *method: str | None = None*, *http_status: int | None = None*, *retry_after: int = 0*)

Bases: HTTPClientError

HTTP 408 - Request Timeout.

The server timed out waiting for the request.

__annotations__ = {}

__doc__ = 'HTTP 408 - Request Timeout.\n\n The server timed out waiting for the request.\n '

__module__ = 'keystoneauth1.exceptions.http'

http_status = 408

message = 'Request Timeout'

exception keystoneauth1.exceptions.http.**RequestUriTooLong**(*message: str | None = None*, *details: str | None = None*, *response: Response | None = None*, *request_id: str | None = None*, *url: str | None = None*, *method: str | None = None*, *http_status: int | None = None*, *retry_after: int = 0*)

Bases: HTTPClientError

HTTP 414 - Request-URI Too Long.

The URI provided was too long for the server to process.

__annotations__ = {}

__doc__ = 'HTTP 414 - Request-URI Too Long.\n\n The URI provided was too long for the server to process.\n '

__module__ = 'keystoneauth1.exceptions.http'

http_status = 414

message = 'Request-URI Too Long'

exception keystoneauth1.exceptions.http.**RequestedRangeNotSatisfiable**(*message: str | None = None*, *details: str | None = None*, *response: Response | None = None*, *request_id: str | None = None*, *url: str | None = None*, *method: str | None = None*, *http_status: int | None = None*, *retry_after: int = 0*)

Bases: HTTPClientError

HTTP 416 - Requested Range Not Satisfiable.

The client has asked for a portion of the file, but the server cannot supply that portion.

__annotations__ = {}

__doc__ = 'HTTP 416 - Requested Range Not Satisfiable.\n\n The client has asked for a portion of the file, but the server cannot\n supply that portion.\n '

__module__ = 'keystoneauth1.exceptions.http'

http_status = 416

message = 'Requested Range Not Satisfiable'

exception keystoneauth1.exceptions.http.**ServiceUnavailable**(*message: str | None = None, details: str | None = None, response: Response | None = None, request_id: str | None = None, url: str | None = None, method: str | None = None, http_status: int | None = None, retry_after: int = 0*)

Bases: `HttpServerError`

HTTP 503 - Service Unavailable.

The server is currently unavailable.

`__annotations__ = {}`

`__doc__ = 'HTTP 503 - Service Unavailable.\n\n The server is currently unavailable.\n '`

`__module__ = 'keystoneauth1.exceptions.http'`

`http_status = 503`

`message = 'Service Unavailable'`

exception keystoneauth1.exceptions.http.**Unauthorized**(*message: str | None = None, details: str | None = None, response: Response | None = None, request_id: str | None = None, url: str | None = None, method: str | None = None, http_status: int | None = None, retry_after: int = 0*)

Bases: `HTTPClientError`

HTTP 401 - Unauthorized.

Similar to 403 Forbidden, but specifically for use when authentication is required and has failed or has not yet been provided.

`__annotations__ = {}`

`__doc__ = 'HTTP 401 - Unauthorized.\n\n Similar to 403 Forbidden, but specifically for use when authentication\n is required and has failed or has not yet been provided.\n '`

`__module__ = 'keystoneauth1.exceptions.http'`

`http_status = 401`

`message = 'Unauthorized'`

exception keystoneauth1.exceptions.http.**UnprocessableEntity**(*message: str | None = None, details: str | None = None, response: Response | None = None, request_id: str | None = None, url: str | None = None, method: str | None = None, http_status: int | None = None, retry_after: int = 0*)

Bases: `HTTPClientError`

HTTP 422 - Unprocessable Entity.

The request was well-formed but was unable to be followed due to semantic errors.

`__annotations__ = {}`

`__doc__ = 'HTTP 422 - Unprocessable Entity.\n\n The request was well-formed but was unable to be followed due to semantic\n errors.\n '`

`__module__ = 'keystoneauth1.exceptions.http'`

`http_status = 422`

`message = 'Unprocessable Entity'`

exception keystoneauth1.exceptions.http.**UnsupportedMediaType**(*message: str | None = None, details: str | None = None, response: Response | None = None, request_id: str | None = None, url: str | None = None, method: str | None = None, http_status: int | None = None, retry_after: int = 0*)

Bases: `HTTPClientError`

HTTP 415 - Unsupported Media Type.

The request entity has a media type which the server or resource does not support.

`__annotations__ = {}`

`__doc__ = 'HTTP 415 - Unsupported Media Type.\n\n The request entity has a media type which the server or resource does\n not support.\n '`

`__module__ = 'keystoneauth1.exceptions.http'`

`http_status = 415`

message = **'Unsupported Media Type'**

keystoneauth1.exceptions.http.**from_response**(*response: Response*, *method: str*, *url: str*) → HttpError | MissingAuthMethods

Return an instance of `HttpError` or subclass based on response.

Parameters

- **response** instance of *requests.Response* class
- **method** HTTP method used for request
- **url** URL used for request

keystoneauth1.exceptions.oidc module

exception keystoneauth1.exceptions.oidc.**InvalidDiscoveryEndpoint**(*message: str | None = None*)

Bases: AuthPluginException

__annotations__ = **{}**

__doc__ = **None**

__module__ = **'keystoneauth1.exceptions.oidc'**

message = **'OpenID Connect Discovery Document endpoint not set.'**

exception keystoneauth1.exceptions.oidc.**InvalidOidcDiscoveryDocument**(*message: str | None = None*)

Bases: AuthPluginException

__annotations__ = **{}**

__doc__ = **None**

__module__ = **'keystoneauth1.exceptions.oidc'**

message = **'OpenID Connect Discovery Document is not valid JSON.'**

exception keystoneauth1.exceptions.oidc.**OidcAccessTokenEndpointNotFound**(*message: str | None = None*)

Bases: AuthPluginException

__annotations__ = **{}**

__doc__ = **None**

__module__ = **'keystoneauth1.exceptions.oidc'**

message = **'OpenID Connect access token endpoint not provided.'**

exception keystoneauth1.exceptions.oidc.**OidcAuthorizationEndpointNotFound**(*message: str | None = None*)

Bases: AuthPluginException

__annotations__ = {}

__doc__ = None

__module__ = 'keystoneauth1.exceptions.oidc'

message = 'OpenID Connect authorization endpoint not provided.'

exception keystoneauth1.exceptions.oidc.**OidcDeviceAuthorizationEndpointNotFound**(*message: str | None = None*)

Bases: AuthPluginException

__annotations__ = {}

__doc__ = None

__module__ = 'keystoneauth1.exceptions.oidc'

message = 'OpenID Connect device authorization endpoint not provided.'

exception keystoneauth1.exceptions.oidc.**OidcDeviceAuthorizationTimeOut**(*message: str | None = None*)

Bases: AuthPluginException

__annotations__ = {}

__doc__ = None

__module__ = 'keystoneauth1.exceptions.oidc'

message = 'Timeout for OpenID Connect device authorization.'

exception keystoneauth1.exceptions.oidc.**OidcInvalidCodeChallengeMethod**(*message: str | None = None*)

Bases: AuthPluginException

__annotations__ = {}

__doc__ = None

__module__ = 'keystoneauth1.exceptions.oidc'

message = 'Invalid code challenge method.'

exception keystoneauth1.exceptions.oidc.**OidcPluginNotSupported**(*message: str | None = None*)

Bases: AuthPluginException

__annotations__ = {}

__doc__ = None

__module__ = 'keystoneauth1.exceptions.oidc'

message = 'OpenID Connect grant type not supported by provider.'

keystoneauth1.exceptions.response module

exception keystoneauth1.exceptions.response.**InvalidResponse**(*response: Response*)

Bases: ClientException

__annotations__ = {}

__doc__ = None

__init__(*response: Response*)

__module__ = 'keystoneauth1.exceptions.response'

message = 'Invalid response from server.'

keystoneauth1.exceptions.service_providers module

exception keystoneauth1.exceptions.service_providers.**ServiceProviderNotFound**(*sp_id: str*)

Bases: ClientException

A Service Provider cannot be found.

__annotations__ = {}

__doc__ = 'A Service Provider cannot be found.'

__init__(*sp_id: str*)

__module__ = 'keystoneauth1.exceptions.service_providers'

Module contents

keystoneauth1.extras package

Subpackages

keystoneauth1.extras.kerberos package

Module contents

Kerberos authentication plugins.

Warning

This module requires installation of an extra package (*requests_kerberos*) not installed by default. Without the extra package an import error will occur. The extra package can be installed using:

```
$ pip install keystoneauth1[kerberos]
```

class keystoneauth1.extras.kerberos.**Kerberos**(*auth_url: str, mutual_auth: str | None = None, *, unscoped: bool = False, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

Bases: Auth

__abstractmethods__ = frozenset({})

__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}

__doc__ = None

__init__(*auth_url: str, mutual_auth: str | None = None, *, unscoped: bool = False, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*) → None

__module__ = 'keystoneauth1.extras.kerberos'

_abc_impl = <_abc._abc_data object>

_auth_method_class

alias of KerberosMethod

class keystoneauth1.extras.kerberos.**KerberosMethod**(**, mutual_auth: str | None = None*)

Bases: AuthMethod

__abstractmethods__ = frozenset({})

__annotations__ = {'_method_parameters': 'ty.Optional[list[str]]', 'mutual_auth': typing.Optional[str]}

__doc__ = None

__init__(**, mutual_auth: str | None = None*) → None

`__module__ = 'keystoneauth1.extras.kerberos'`

`_abc_impl = <_abc._abc_data object>`

`get_auth_data`(*session: Session*, *auth: Auth*, *headers: dict[str, str]*, *request_kwargs: dict[str, object]*) → tuple[None, None] | tuple[str, Mapping[str, object]]

Return the authentication section of an auth plugin.

Parameters

- **session** (*keystoneauth1.session.Session*) The communication session.
- **auth** (*base.Auth*) The auth plugin calling the method.
- **headers** (*dict*) The headers that will be sent with the auth request if a plugin needs to add to them.

Returns

The identifier of this plugin and a dict of authentication data for the auth type.

Return type

tuple(string, dict)

`mutual_auth: str | None`

class `keystoneauth1.extras.kerberos.`**`MappedKerberos`**(*auth_url: str*, *identity_provider: str*, *protocol: str*, *mutual_auth: str | None = None*, ***, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*)

Bases: `FederationBaseAuth`

Authenticate using Kerberos via the keystone federation mechanisms.

This uses the OS-FEDERATION extension to gain an unscoped token and then use the standard keystone auth process to scope that to any given project.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}`

`__doc__ = 'Authenticate using Kerberos via the keystone federation mechanisms.\n\n This uses the OS-FEDERATION extension to gain an unscoped token and then\n use the standard keystone auth process to scope that to any given project.\n '`

__init__(*auth_url: str*, *identity_provider: str*, *protocol: str*, *mutual_auth: str | None = None*, ***, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*) → None

__module__ = **'keystoneauth1.extras.kerberos'**

_abc_impl = **<_abc._abc_data object>**

get_unscoped_auth_ref(*session: Session*) → AccessInfoV3

Fetch unscoped federated token.

keystoneauth1.extras.kerberos.**_dependency_check**() → None

keystoneauth1.extras.kerberos.**_mutual_auth**(*value: str | None*) → str

keystoneauth1.extras.kerberos.**_requests_auth**(*mutual_authentication: str | None*) → requests_kerberos.HTTPKerberosAuth

keystoneauth1.extras.oauth1 package

Submodules

keystoneauth1.extras.oauth1.v3 module

Oauth authentication plugins.

Warning

This module requires installation of an extra package (*oauthlib*) not installed by default. Without the extra package an import error will occur. The extra package can be installed using:

```
$ pip install keystoneauth['oauth1']
```

class keystoneauth1.extras.oauth1.v3.**OAuth1**(*auth_url: str*, *consumer_key: str*, *consumer_secret: str*, *access_key: str*, *access_secret: str*, ***, *unscoped: bool = False*, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*)

Bases: Auth

__abstractmethods__ = **frozenset({})**

__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}

__doc__ = None

__init__(*auth_url: str*, *consumer_key: str*, *consumer_secret: str*, *access_key: str*, *access_secret: str*, ***, *unscoped: bool = False*, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*) → None

__module__ = 'keystoneauth1.extras.oauth1.v3'

_abc_impl = <_abc._abc_data object>

_auth_method_class

alias of `OAuth1Method`

class keystoneauth1.extras.oauth1.v3.**OAuth1Method**(***, *access_key: str*, *access_secret: str*, *consumer_key: str*, *consumer_secret: str*)

Bases: `AuthMethod`

OAuth based authentication method.

Parameters

- **access_key** (*string*) Access token key.
- **access_secret** (*string*) Access token secret.
- **consumer_key** (*string*) Consumer key.
- **consumer_secret** (*string*) Consumer secret.

__abstractmethods__ = frozenset({})

__annotations__ = {'_method_parameters': 'ty.Optional[list[str]]', 'access_key': <class 'str'>, 'access_secret': <class 'str'>, 'consumer_key': <class 'str'>, 'consumer_secret': <class 'str'>}

__doc__ = 'OAuth based authentication method.\n\n :param string access_key: Access token key.\n :param string access_secret: Access token secret.\n :param string consumer_key: Consumer key.\n :param string consumer_secret: Consumer secret.\n '

__init__(***, *access_key: str*, *access_secret: str*, *consumer_key: str*, *consumer_secret: str*) → None

__module__ = 'keystoneauth1.extras.oauth1.v3'

_abc_impl = <_abc._abc_data object>

access_key: str

access_secret: str

consumer_key: str

consumer_secret: str

get_auth_data(*session: Session*, *auth: Auth*, *headers: dict[str, str]*, *request_kwargs: dict[str, object]*) → tuple[None, None] | tuple[str, Mapping[str, object]]

Return the authentication section of an auth plugin.

Parameters

- **session** (*keystoneauth1.session.Session*) The communication session.
- **auth** (*base.Auth*) The auth plugin calling the method.
- **headers** (*dict*) The headers that will be sent with the auth request if a plugin needs to add to them.

Returns

The identifier of this plugin and a dict of authentication data for the auth type.

Return type

tuple(string, dict)

get_cache_id_elements() → dict[str, str | None]

Get the elements for this auth method that make it unique.

These elements will be used as part of the `keystoneauth1.plugin.BaseIdentityPlugin.get_cache_id()` to allow caching of the auth plugin.

Plugins should override this if they want to allow caching of their state.

To avoid collision or overrides the keys of the returned dictionary should be prefixed with the plugin identifier. For example the password plugin returns its username value as password_username.

Module contents

keystoneauth1.extras.oauth1.**V3OAuth1**

alias of `OAuth1`

keystoneauth1.extras.oauth1.**V3OAuth1Method**

alias of `OAuth1Method`

Module contents

keystoneauth1.fixture package

Submodules

keystoneauth1.fixture.discovery module

class keystoneauth1.fixture.discovery.**DiscoveryBase**(*id*, *status=None*, *updated=None*)

Bases: `dict[str, Any]`

The basic version discovery structure.

All version discovery elements should have access to these values.

Parameters

- **id** (*string*) The version id for this version entry.
- **status** (*string*) The status of this entry.
- **updated** (*DateTime*) When the API was last updated.

`__annotations__ = {}`

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.fixture.discovery', '__doc__': 'The basic version discovery structure.\n\n All version discovery elements should have access to these values.\n\n :param string id: The version id for this version entry.\n :param string status: The status of this entry.\n :param DateTime updated: When the API was last updated.\n ', '__init__': <function DiscoveryBase.__init__>, 'id': <property object>, 'status': <property object>, 'links': <property object>, 'updated_str': <property object>, 'updated': <property object>, 'add_link': <function DiscoveryBase.add_link>, 'media_types': <property object>, 'add_media_type': <function DiscoveryBase.add_media_type>, '__orig_bases__': (dict[str, typing.Any],), '__dict__': <attribute '__dict__' of 'DiscoveryBase' objects>, '__weakref__': <attribute '__weakref__' of 'DiscoveryBase' objects>, '__annotations__': {}})`

`__doc__ = 'The basic version discovery structure.\n\n All version discovery elements should have access to these values.\n\n :param string id: The version id for this version entry.\n :param string status: The status of this entry.\n :param DateTime updated: When the API was last updated.\n '`

`__init__`(*id*, *status=None*, *updated=None*)

`__module__ = 'keystoneauth1.fixture.discovery'`

`__orig_bases__ = (dict[str, typing.Any],)`

`__weakref__`

list of weak references to the object (if defined)

`add_link`(*href*, *rel='self'*, *type=None*)

`add_media_type`(*base*, *type*)

`property id`

`property links`

`property media_types`

`property status`

`property updated`

`property updated_str`

`class keystoneauth1.fixture.discovery.DiscoveryList`(*href=None*, *v2=True*, *v3=True*, *v2_id=None*, *v3_id=None*, *v2_status=None*, *v2_updated=None*, *v2_html=True*, *v2_pdf=True*, *v3_status=None*, *v3_updated=None*, *v3_json=True*, *v3_xml=True*)

Bases: `dict`[`str`, `Any`]

A List of version elements.

Creates a correctly structured list of identity service endpoints for use in testing with discovery.

Parameters

- **href** (*string*) The url that this should be based at.
- **v2** (*bool*) Add a v2 element.
- **v3** (*bool*) Add a v3 element.
- **v2_status** (*string*) The status to use for the v2 element.
- **v2_updated** (*DateTime*) The update time to use for the v2 element.
- **v2_html** (*bool*) True to add a html link to the v2 element.
- **v2_pdf** (*bool*) True to add a pdf link to the v2 element.
- **v3_status** (*string*) The status to use for the v3 element.
- **v3_updated** (*DateTime*) The update time to use for the v3 element.
- **v3_json** (*bool*) True to add a html link to the v2 element.
- **v3_xml** (*bool*) True to add a pdf link to the v2 element.

`TEST_URL = 'http://keystone.host:5000/'`

`__annotations__ = {}`

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.fixture.discovery', '__doc__': 'A List of version elements.\n\n Creates a correctly structured list of identity service endpoints for\n use in testing with discovery.\n\n :param string href: The url that this should be based at.\n :param bool v2: Add a v2 element.\n :param bool v3: Add a v3 element.\n :param string v2_status: The status to use for the v2 element.\n :param DateTime v2_updated: The update time to use for the v2 element.\n :param bool v2_html: True to add a html link to the v2 element.\n :param bool v2_pdf: True to add a pdf link to the v2 element.\n :param string v3_status: The status to use for the v3 element.\n :param DateTime v3_updated: The update time to use for the v3 element.\n :param bool v3_json: True to add a html link to the v2 element.\n :param bool v3_xml: True to add a pdf link to the v2 element.\n ', 'TEST_URL': 'http://keystone.host:5000/', '__init__': <function DiscoveryList.__init__>, 'versions': <property object>, 'add_version': <function DiscoveryList.add_version>, 'add_v2': <function DiscoveryList.add_v2>, 'add_v3': <function DiscoveryList.add_v3>, 'add_microversion': <function DiscoveryList.add_microversion>, 'add_nova_microversion': <function DiscoveryList.add_nova_microversion>, '__orig_bases__': (dict[str, typing.Any],), '__dict__': <attribute '__dict__' of 'DiscoveryList' objects>, '__weakref__': <attribute '__weakref__' of 'DiscoveryList' objects>, '__annotations__': {}})`

`__doc__ = 'A List of version elements.\n\n Creates a correctly structured list of identity service endpoints for\n use in testing with discovery.\n\n :param string href: The url that this should be based at.\n :param bool v2: Add a v2 element.\n :param bool v3: Add a v3 element.\n :param string v2_status: The status to use for the v2 element.\n :param DateTime v2_updated: The update time to use for the v2 element.\n :param bool v2_html: True to add a html link to the v2 element.\n :param bool v2_pdf: True to add a pdf link to the v2 element.\n :param string v3_status: The status to use for the v3 element.\n :param DateTime v3_updated: The update time to use for the v3 element.\n :param bool v3_json: True to add a html link to the v2 element.\n :param bool v3_xml: True to add a pdf link to the v2 element.\n '`

`__init__`(*href=None*, *v2=True*, *v3=True*, *v2_id=None*, *v3_id=None*, *v2_status=None*, *v2_updated=None*, *v2_html=True*, *v2_pdf=True*, *v3_status=None*, *v3_updated=None*, *v3_json=True*, *v3_xml=True*)

`__module__ = 'keystoneauth1.fixture.discovery'`

`__orig_bases__ = (dict[str, typing.Any],)`

`__weakref__`

list of weak references to the object (if defined)

`add_microversion`(*href*, *id*, ***kwargs*)

Add a microversion version to the list.

The parameters are the same as MicroversionDiscovery.

add_nova_microversion(*href*, *id*, ***kwargs*)

Add a nova microversion version to the list.

The parameters are the same as NovaMicroversionDiscovery.

add_v2(*href*, ***kwargs*)

Add a v2 version to the list.

The parameters are the same as V2Discovery.

add_v3(*href*, ***kwargs*)

Add a v3 version to the list.

The parameters are the same as V3Discovery.

add_version(*version*)

Add a new version structure to the list.

Parameters

version (*dict*) A new version structure to add to the list.

property versions

class keystoneauth1.fixture.discovery.**MicroversionDiscovery**(*href*, *id*, *min_version=''*, *max_version=''*, ***kwargs*)

Bases: DiscoveryBase

A Version element that has microversions.

Provides some default values and helper methods for creating a microversion endpoint version structure. Clients should use this instead of creating their own structures.

Parameters

- **href** (*string*) The url that this entry should point to.
- **id** (*string*) The version id that should be reported.
- **min_version** (*string*) The minimum supported microversion. (optional)
- **max_version** (*string*) The maximum supported microversion. (optional)

__annotations__ = {}

__doc__ = 'A Version element that has microversions.\n\n Provides some default values and helper methods for creating a microversion\n endpoint version structure. Clients should use this instead of creating\n their own structures.\n\n :param string href: The url that this entry should point to.\n :param string id: The version id that should be reported.\n :param string min_version: The minimum supported microversion. (optional)\n :param string max_version: The maximum supported microversion. (optional)\n '

__init__(*href*, *id*, *min_version=''*, *max_version=''*, ***kwargs*)

__module__ = 'keystoneauth1.fixture.discovery'

property max_version

property min_version

class keystoneauth1.fixture.discovery.**NovaMicroversionDiscovery**(*href*, *id*, *min_version=None*, *version=None*, ***kwargs*)

Bases: DiscoveryBase

A Version element with nova-style microversions.

Provides some default values and helper methods for creating a microversion endpoint version structure. Clients should use this instead of creating their own structures.

Parameters

- **href** The url that this entry should point to.
- **id** (*string*) The version id that should be reported.
- **min_version** (*string*) The minimum microversion supported. (optional)
- **version** (*string*) The maximum microversion supported. (optional)

__annotations__ = {}

__doc__ = 'A Version element with nova-style microversions.\n\n Provides some default values and helper methods for creating a microversion\n endpoint version structure. Clients should use this instead of creating\n their own structures.\n\n :param href: The url that this entry should point to.\n :param string id: The version id that should be reported.\n :param string min_version: The minimum microversion supported. (optional)\n :param string version: The maximum microversion supported. (optional)\n '

__init__(*href*, *id*, *min_version=None*, *version=None*, ***kwargs*)

__module__ = 'keystoneauth1.fixture.discovery'

property min_version

property version

class keystoneauth1.fixture.discovery.**V2Discovery**(*href*, *id=None*, *html=True*, *pdf=True*, ***kwargs*)

Bases: DiscoveryBase

A Version element for a V2 identity service endpoint.

Provides some default values and helper methods for creating a v2.0 endpoint version structure. Clients should use this instead of creating their own structures.

Parameters

- **href** (*string*) The url that this entry should point to.
- **id** (*string*) The version id that should be reported. (optional) Defaults to v2.0.

- **html** (*bool*) Add HTML describedby links to the structure.
- **pdf** (*bool*) Add PDF describedby links to the structure.

`_DESC_URL = 'https://developer.openstack.org/api-ref/identity/v2/'`

`__annotations__ = {}`

`__doc__ = "A Version element for a V2 identity service endpoint.\n\n Provides some default values and helper methods for creating a v2.0\n endpoint version structure. Clients should use this instead of creating\n their own structures.\n\n :param string href: The url that this entry should point to.\n :param string id: The version id that should be reported. (optional)\n Defaults to 'v2.0'.\n :param bool html: Add HTML describedby links to the structure.\n :param bool pdf: Add PDF describedby links to the structure.\n\n "`

`__init__`(*href*, *id=None*, *html=True*, *pdf=True*, ***kwargs*)

`__module__ = 'keystoneauth1.fixture.discovery'`

`add_html_description()`

Add the HTML described by links.

The standard structure includes a link to a HTML document with the API specification. Add it to this entry.

`add_pdf_description()`

Add the PDF described by links.

The standard structure includes a link to a PDF document with the API specification. Add it to this entry.

class `keystoneauth1.fixture.discovery.V3Discovery`(*href*, *id=None*, *json=True*, *xml=True*, ***kwargs*)

Bases: `DiscoveryBase`

A Version element for a V3 identity service endpoint.

Provides some default values and helper methods for creating a v3 endpoint version structure. Clients should use this instead of creating their own structures.

Parameters

- **href** The url that this entry should point to.
- **id** (*string*) The version id that should be reported. (optional) Defaults to v3.0.
- **json** (*bool*) Add JSON media-type elements to the structure.
- **xml** (*bool*) Add XML media-type elements to the structure.

`__annotations__ = {}`

`__doc__ = "A Version element for a V3 identity service endpoint.\n\n Provides some default values and helper methods for creating a v3\n endpoint version structure. Clients should use this instead of creating\n their own structures.\n\n :param href: The url that this entry should point to.\n :param string id: The version id that should be reported. (optional)\n Defaults to 'v3.0'.\n :param bool json: Add JSON media-type elements to the structure.\n :param bool xml: Add XML media-type elements to the structure.\n "`

`__init__`(*href*, *id=None*, *json=True*, *xml=True*, ***kwargs*)

`__module__` `= 'keystoneauth1.fixture.discovery'`

`add_json_media_type`()

Add the JSON media-type links.

The standard structure includes a list of media-types that the endpoint supports. Add JSON to the list.

`add_xml_media_type`()

Add the XML media-type links.

The standard structure includes a list of media-types that the endpoint supports. Add XML to the list.

`class keystoneauth1.fixture.discovery.`**`VersionDiscovery`**(*href*, *id*, ***kwargs*)

Bases: `DiscoveryBase`

A Version element for non-keystone services without microversions.

Provides some default values and helper methods for creating a microversion endpoint version structure. Clients should use this instead of creating their own structures.

Parameters

- **href** (*string*) The url that this entry should point to.
- **id** (*string*) The version id that should be reported.

`__annotations__` `= {}`

`__doc__ = 'A Version element for non-keystone services without microversions.\n\n Provides some default values and helper methods for creating a microversion\n endpoint version structure. Clients should use this instead of creating\n their own structures.\n\n :param string href: The url that this entry should point to.\n :param string id: The version id that should be reported.\n '`

`__init__`(*href*, *id*, ***kwargs*)

`__module__` `= 'keystoneauth1.fixture.discovery'`

keystoneauth1.fixture.exception module

exception `keystoneauth1.fixture.exception.`**`FixtureValidationError`**

Bases: `Exception`

The token you created is not legitimate.

The data contained in the token that was generated is not valid and would not have been returned from a keystone server. You should not do testing with this token.

`__annotations__ = {}`

`__doc__ = 'The token you created is not legitimate.\n\n The data contained in the token that was generated is not valid and would\n not have been returned from a keystone server. You should not do testing\n with this token.\n '`

`__module__ = 'keystoneauth1.fixture.exception'`

`__weakref__`

list of weak references to the object (if defined)

keystoneauth1.fixture.hooks module

Custom hooks for betamax and keystoneauth.

Module providing a set of hooks specially designed for interacting with clouds and keystone authentication.

author

Yolanda Robla

`keystoneauth1.fixture.hooks.`**`mask_fixture_values`**`(`*nested*, *prev_key*`)`

`keystoneauth1.fixture.hooks.`**`pre_record_hook`**`(`*interaction*, *cassette*`)`

Hook to mask saved data.

This hook will be triggered before saving the interaction, and will perform two tasks: - mask user, project and password in the saved data - set token expiration time to an inifinite time.

keystoneauth1.fixture.keystoneauth_betamax module

A fixture to wrap the session constructor for use with Betamax.

class `keystoneauth1.fixture.keystoneauth_betamax.`**`BetamaxFixture`**`(`*cassette_name*, *cassette_library_dir=None*, *serializer=None*, *record=False*, *pre_record_hook=<function pre_record_hook>*, *serializer_name=None*, *request_matchers=None*`)`

Bases: `Fixture`

`__annotations__ = {}`

`__doc__ = None`

`__init__`(*cassette_name*, *cassette_library_dir=None*, *serializer=None*, *record=False*, *pre_record_hook=<function pre_record_hook>*, *serializer_name=None*, *request_matchers=None*)

Configure Betamax for the test suite.

Parameters

- **`cassette_name`** (*`str`*) This is simply the name of the cassette without any file extension or containing directory. For example, to generate `keystoneauth1/tests/unit/data/example.yaml`, one would pass only `example`.
- **`cassette_library_dir`** (*`str`*) This is the directory that will contain all cassette files. In `keystoneauth1/tests/unit/data/example.yaml` you would pass `keystoneauth1/tests/unit/data/`.
- **`serializer`** A class that implements the Serializer API in Betamax. See also: https://betamax.readthedocs.io/en/latest/serializers.html
- **`record`** The Betamax record mode to use. If `False` (the default), then Betamax will not record anything. For more information about record modes, see: https://betamax.readthedocs.io/en/latest/record_modes.html
- **`pre_record_hook`** (*`callable`*) Function or callable to use to perform some handling of the request or response data prior to saving it to disk.
- **`serializer_name`** (*`str`*) The name of a serializer already registered with Betamax to use to handle cassettes. For example, if you want to use the default Betamax serializer, you would pass `'json'` to this parameter.
- **`request_matchers`** (*`list`*) The list of request matcher names to use with Betamax. Betamaxs default list is used if none are specified. See also: https://betamax.readthedocs.io/en/latest/matchers.html

`__module__ = 'keystoneauth1.fixture.keystoneauth_betamax'`

`property serializer_name`

Determine the name of the selected serializer.

If a class was specified, use the name attribute to generate this, otherwise, use the serializer_name parameter from `__init__`.

Returns

Name of the serializer

Return type

str

`setUp`()

Prepare the Fixture for use.

This should not be overridden. Concrete fixtures should implement _setUp. Overriding of setUp is still supported, just not recommended.

After setUp has completed, the fixture will have one or more attributes which can be used (these depend totally on the concrete subclass).

Raises
MultipleExceptions if _setUp fails. The last exception captured within the MultipleExceptions will be a SetupError exception.

Returns
None.

Changed in 1.3
The recommendation to override setUp has been reversed - before 1.3, setUp() should be overridden, now it should not be.

Changed in 1.3.1
BaseException is now caught, and only subclasses of Exception are wrapped in MultipleExceptions.

`keystoneauth1.fixture.keystoneauth_betamax.`**`_construct_session_with_betamax`**(*fixture*, *session_obj=None*)

keystoneauth1.fixture.plugin module

`class keystoneauth1.fixture.plugin.`**`LoadingFixture`**(*token=None*, *endpoint=None*, *user_id=None*, *project_id=None*)

Bases: `Fixture`

A fixture that will stub out all plugin loading calls.

When using keystoneauth plugins loaded from config, CLI or elsewhere it is often difficult to handle the plugin parts in tests because we dont have a reasonable default.

This fixture will create a `TestPlugin` that will be returned for all calls to plugin loading so you can simply bypass the authentication steps and return something well known.

Parameters

- **token** (*str*) The token to include in authenticated requests.
- **endpoint** (*str*) The endpoint to respond to service lookups with.
- **user_id** (*str*) The user_id to report for the authenticated user.
- **project_id** (*str*) The project_id to report for the authenticated user.

`MOCK_POINT = 'keystoneauth1.loading.base.get_plugin_loader'`

`__annotations__ = {}`

`__doc__ = "A fixture that will stub out all plugin loading calls.\n\n When using keystoneauth plugins loaded from config, CLI or elsewhere it is\n often difficult to handle the plugin parts in tests because we don't have a\n reasonable default.\n\n This fixture will create a :py:class:`TestPlugin` that will be\n returned for all calls to plugin loading so you can simply bypass the\n authentication steps and return something well known.\n\n :param str token: The token to include in authenticated requests.\n :param str endpoint: The endpoint to respond to service lookups with.\n :param str user_id: The user_id to report for the authenticated user.\n :param str project_id: The project_id to report for the authenticated user.\n "`

`__init__`(*token=None*, *endpoint=None*, *user_id=None*, *project_id=None*)

`__module__ = 'keystoneauth1.fixture.plugin'`

`create_plugin`()

`get_endpoint`(*path=None*, ***kwargs*)

Utility function to get the endpoint the plugin would return.

This function is provided as a convenience so you can do comparisons in your tests. Overriding it will not affect the endpoint returned by the plugin.

Parameters
`path` (*str*) The path to append to the plugin endpoint.

`get_plugin_loader`(*auth_type*)

`setUp`()

Prepare the Fixture for use.

This should not be overridden. Concrete fixtures should implement _setUp. Overriding of setUp is still supported, just not recommended.

After setUp has completed, the fixture will have one or more attributes which can be used (these depend totally on the concrete subclass).

Raises
MultipleExceptions if _setUp fails. The last exception captured within the MultipleExceptions will be a SetupError exception.

Returns
None.

Changed in 1.3
The recommendation to override setUp has been reversed - before 1.3, setUp() should be overridden, now it should not be.

Changed in 1.3.1
BaseException is now caught, and only subclasses of Exception are wrapped in MultipleExceptions.

class `keystoneauth1.fixture.plugin.TestPlugin`(*token=None*, *endpoint=None*, *user_id=None*, *project_id=None*)

Bases: `BaseAuthPlugin`

A simple plugin that returns what you gave it for testing.

When testing services that use authentication plugins you often want to stub out the authentication calls and focus on the important part of your service. This plugin acts like a real keystoneauth plugin and returns known standard values without having to stub out real keystone responses.

Note that this plugin is a BaseAuthPlugin and not a BaseIdentityPlugin. This means it implements the basic plugin interface that services should be using but does not implement get_auth_ref. get_auth_ref should not be relied upon by services because a user could always configure the service to use a non-keystone auth.

Parameters

- **token** (*str*) The token to include in authenticated requests.
- **endpoint** (*str*) The endpoint to respond to service lookups with.
- **user_id** (*str*) The user_id to report for the authenticated user.
- **project_id** (*str*) The project_id to report for the authenticated user.

`__annotations__ = {'_discovery_cache': 'dict[str, discover.Discover]'}`

`__doc__ = 'A simple plugin that returns what you gave it for testing.\n\n When testing services that use authentication plugins you often want to\n stub out the authentication calls and focus on the important part of your\n service. This plugin acts like a real keystoneauth plugin and returns known\n standard values without having to stub out real keystone responses.\n\n Note that this plugin is a BaseAuthPlugin and not a BaseIdentityPlugin.\n This means it implements the basic plugin interface that services should be\n using but does not implement get_auth_ref. get_auth_ref should not be\n relied upon by services because a user could always configure the service\n to use a non-keystone auth.\n\n :param str token: The token to include in authenticated requests.\n :param str endpoint: The endpoint to respond to service lookups with.\n :param str user_id: The user_id to report for the authenticated user.\n :param str project_id: The project_id to report for the authenticated user.\n '`

`__init__`(*token=None*, *endpoint=None*, *user_id=None*, *project_id=None*)

`__module__ = 'keystoneauth1.fixture.plugin'`

`_discovery_cache: dict[str, Discover]`

`auth_type = 'test_plugin'`

`get_endpoint`(*session*, ***kwargs*)

Return an endpoint for the client.

There are no required keyword arguments to `get_endpoint` as a plugin implementation should use best effort with the information available to determine the endpoint. However there are certain standard options that will be generated by the clients and should be used by plugins:

- `service_type`: what sort of service is required.
- `service_name`: the name of the service in the catalog.
- `interface`: what visibility the endpoint should have.
- `region_name`: the region the endpoint exists in.

Parameters

- `session` (*keystoneauth1.session.Session*) The session object that the auth_plugin belongs to.
- `kwargs` Ignored.

Returns
The base URL that will be used to talk to the required service or None if not available.

Return type
string

`get_project_id`(*session*)

Return the project id that we are authenticated to.

Wherever possible the project id should be inferred from the token however there are certain URLs and other places that require access to the currently authenticated project id.

Parameters
`session` (*keystoneauth1.session.Session*) A session object so the plugin can make HTTP calls.

Returns
A project identifier or None if one is not available.

Return type
str

`get_token`(*session*)

Obtain a token.

How the token is obtained is up to the plugin. If it is still valid it may be re-used, retrieved from cache or invoke an authentication request against a server.

Returning None will indicate that no token was able to be retrieved.

This function is misplaced as it should only be required for auth plugins that use the X-Auth-Token header. However due to the way plugins evolved this method is required and often called to trigger an authentication request on a new plugin.

When implementing a new plugin it is advised that you implement this method, however if you dont require the X-Auth-Token header override the *get_headers* method instead.

Parameters
`session` (*keystoneauth1.session.Session*) A session object so the plugin can make HTTP calls.

Returns
A token to use.

Return type
string

`get_user_id`(*session*)

Return a unique user identifier of the plugin.

Wherever possible the user id should be inferred from the token however there are certain URLs and other places that require access to the currently authenticated user id.

Parameters
`session` (*keystoneauth1.session.Session*) A session object so the plugin can make HTTP calls.

Returns
A user identifier or None if one is not available.

Return type
str

`invalidate()`

Invalidate the current authentication data.

This should result in fetching a new token on next call.

A plugin may be invalidated if an Unauthorized HTTP response is returned to indicate that the token may have been revoked or is otherwise now invalid.

Returns
True if there was something that the plugin did to invalidate. This means that it makes sense to try again. If nothing happens returns False to indicate give up.

Return type
bool

`class keystoneauth1.fixture.plugin._TestPluginLoader(plugin)`

Bases: `BaseLoader[TestPlugin]`

`__abstractmethods__ = frozenset({})`

`__doc__ = None`

`__init__(plugin)`

`__module__ = 'keystoneauth1.fixture.plugin'`

`__orig_bases__ = (keystoneauth1.loading.base.BaseLoader[keystoneauth1.fixture.plugin.TestPlugin],)`

`__parameters__ = ()`

`_abc_impl = <_abc._abc_data object>`

`create_plugin(**kwargs)`

Create a plugin from the options available for the loader.

Given the options that were specified by the loader create an appropriate plugin. You can override this function in your loader.

This used to be specified by providing the plugin_class property and this is still supported, however specifying a property didnt let you choose a plugin type based upon the options that were presented.

Override this function if you wish to return different plugins based on the options presented, otherwise you can simply provide the plugin_class property.

Added 2.9

get_options()

Return the list of parameters associated with the auth plugin.

This list may be used to generate CLI or config arguments.

Returns
A list of Param objects describing available plugin parameters.

Return type
list

keystoneauth1.fixture.plugin.**_format_endpoint**(*endpoint*, ***kwargs*)

keystoneauth1.fixture.serializer module

A serializer to emit YAML but with request body in nicely formatted JSON.

class keystoneauth1.fixture.serializer.**YamlJsonSerializer**

Bases: BaseSerializer

__doc__ = None

__module__ = 'keystoneauth1.fixture.serializer'

deserialize(*cassette_data*)

A method that must be implemented by the Serializer author.

The return value is extremely important. If it is not empty, the dictionary returned must have the following structure:

```
{
    'http_interactions': [{
        # Interaction
    },
    {
        # Interaction
    }],
    'recorded_with': 'name of recorder'
}
```

Params str cassette_data
The data serialized as a string which needs to be deserialized.

Returns
dictionary

static generate_cassette_name(*cassette_library_dir*, *cassette_name*)

name = 'yamljson'

serialize(*cassette_data*)

A method that must be implemented by the Serializer author.

Parameters
cassette_data (*dict*) A dictionary with two keys: http_interactions, recorded_with.

Returns
Serialized data as a string.

`keystoneauth1.fixture.serializer._indent_json(val)`

`keystoneauth1.fixture.serializer._is_json_body(interaction)`

`keystoneauth1.fixture.serializer._represent_scalar(self, tag, value, style=None)`

`keystoneauth1.fixture.serializer._should_use_block(value)`

`keystoneauth1.fixture.serializer._unicode_representer(dumper, uni)`

keystoneauth1.fixture.v2 module

`class keystoneauth1.fixture.v2.Token(token_id=None, expires=None, issued=None, tenant_id=None, tenant_name=None, user_id=None, user_name=None, trust_id=None, trustee_user_id=None, audit_id=None, audit_chain_id=None)`

Bases: `dict[str, Any]`

A V2 Keystone token that can be used for testing.

This object is designed to allow clients to generate a correct V2 token for use in there test code. It should prevent clients from having to know the correct token format and allow them to test the portions of token handling that matter to them and not copy and paste sample.

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.fixture.v2', '__doc__': 'A V2 Keystone token that can be used for testing.\n\n This object is designed to allow clients to generate a correct V2 token for\n use in there test code. It should prevent clients from having to know the\n correct token format and allow them to test the portions of token handling\n that matter to them and not copy and paste sample.\n ', '__init__': <function Token.__init__>, 'root': <property object>, '_token': <property object>, 'token_id': <property object>, 'expires_str': <property object>, 'expires': <property object>, 'issued_str': <property object>, 'issued': <property object>, '_user': <property object>, 'user_id': <property object>, 'user_name': <property object>, 'tenant_id': <property object>, 'tenant_name': <property object>, '_metadata': <property object>, 'trust_id': <property object>, 'trustee_user_id': <property object>, 'audit_id': <property object>, 'audit_chain_id': <property object>, 'validate': <function Token.validate>, 'add_role': <function Token.add_role>, 'add_service': <function Token.add_service>, 'remove_service': <function Token.remove_service>, 'set_scope': <function Token.set_scope>, 'set_trust': <function Token.set_trust>, 'set_bind': <function Token.set_bind>, '__orig_bases__': (dict[str, typing.Any],), '__dict__': <attribute '__dict__' of 'Token' objects>, '__weakref__': <attribute '__weakref__' of 'Token' objects>, '__annotations__': {}})`

`__doc__ = 'A V2 Keystone token that can be used for testing.\n\n This object is designed to allow clients to generate a correct V2 token for\n use in there test code. It should prevent clients from having to know the\n correct token format and allow them to test the portions of token handling\n that matter to them and not copy and paste sample.\n '`

`__init__`(*token_id=None*, *expires=None*, *issued=None*, *tenant_id=None*, *tenant_name=None*, *user_id=None*, *user_name=None*, *trust_id=None*, *trustee_user_id=None*, *audit_id=None*, *audit_chain_id=None*)

`__module__ = 'keystoneauth1.fixture.v2'`

`__orig_bases__ = (dict[str, typing.Any],)`

`__weakref__`

list of weak references to the object (if defined)

`property _metadata`

`property _token`

`property _user`

`add_role`(*name=None*, *id=None*)

`add_service`(*type*, *name=None*)

`property audit_chain_id`

`property audit_id`

`property expires`

`property expires_str`

`property issued`

`property issued_str`

`remove_service`(*type*)

`property root`

`set_bind`(*name*, *data*)

`set_scope`(*id=None*, *name=None*)

`set_trust`(*id=None*, *trustee_user_id=None*)

`property tenant_id`

`property tenant_name`

`property token_id`

`property trust_id`

property trustee_user_id

property user_id

property user_name

validate()

class keystoneauth1.fixture.v2.**_Service**

Bases: dict[str, Any]

__dict__ = mappingproxy({'__module__': 'keystoneauth1.fixture.v2', 'add_endpoint': <function _Service.add_endpoint>, '__orig_bases__': (dict[str, typing.Any],), '__dict__': <attribute '__dict__' of '_Service' objects>, '__weakref__': <attribute '__weakref__' of '_Service' objects>, '__doc__': None, '__annotations__': {}})

__doc__ = None

__module__ = 'keystoneauth1.fixture.v2'

__orig_bases__ = (dict[str, typing.Any],)

__weakref__

list of weak references to the object (if defined)

add_endpoint(*public*, *admin=None*, *internal=None*, *tenant_id=None*, *region=None*, *id=None*)

keystoneauth1.fixture.v3 module

class keystoneauth1.fixture.v3.**Token**(*expires=None*, *issued=None*, *user_id=None*, *user_name=None*, *user_domain_id=None*, *user_domain_name=None*, *methods=None*, *project_id=None*, *project_name=None*, *project_domain_id=None*, *project_domain_name=None*, *domain_id=None*, *domain_name=None*, *trust_id=None*, *trust_impersonation=None*, *trustee_user_id=None*, *trustor_user_id=None*, *application_credential_id=None*, *application_credential_access_rules=None*, *oauth_access_token_id=None*, *oauth_consumer_id=None*, *audit_id=None*, *audit_chain_id=None*, *is_admin_project=None*, *project_is_domain=None*, *oauth2_thumbprint=None*)

Bases: dict[str, Any]

A V3 Keystone token that can be used for testing.

This object is designed to allow clients to generate a correct V3 token for use in there test code. It should prevent clients from having to know the correct token format and allow them to test the portions of token handling that matter to them and not copy and paste sample.

__annotations__ = {}

```
__dict__ = mappingproxy({'__module__': 'keystoneauth1.fixture.v3',
'__doc__': 'A V3 Keystone token that can be used for testing.\n\n This
object is designed to allow clients to generate a correct V3 token for\n
use in there test code. It should prevent clients from having to know
the\n correct token format and allow them to test the portions of token
handling\n that matter to them and not copy and paste sample.\n ',
'__init__': <function Token.__init__>, 'root': <property object>,
'expires_str': <property object>, 'expires': <property object>,
'issued_str': <property object>, 'issued': <property object>, '_user':
<property object>, 'user_id': <property object>, 'user_name': <property
object>, '_user_domain': <property object>, 'user_domain_id': <property
object>, 'user_domain_name': <property object>, 'methods': <property
object>, 'project_id': <property object>, 'project_is_domain': <property
object>, 'project_name': <property object>, 'project_domain_id':
<property object>, 'project_domain_name': <property object>, 'domain_id':
<property object>, 'domain_name': <property object>, 'system': <property
object>, 'trust_id': <property object>, 'trust_impersonation': <property
object>, 'trustee_user_id': <property object>, 'trustor_user_id':
<property object>, 'application_credential_id': <property object>,
'application_credential_access_rules': <property object>,
'oauth_access_token_id': <property object>, 'oauth_consumer_id':
<property object>, 'audit_id': <property object>, 'audit_chain_id':
<property object>, 'role_ids': <property object>, 'role_names':
<property object>, 'is_admin_project': <property object>,
'oauth2_thumbprint': <property object>, 'oauth2_credential': <property
object>, 'validate': <function Token.validate>, 'add_role': <function
Token.add_role>, 'add_service': <function Token.add_service>,
'remove_service': <function Token.remove_service>, 'set_project_scope':
<function Token.set_project_scope>, 'set_domain_scope': <function
Token.set_domain_scope>, 'set_system_scope': <function
Token.set_system_scope>, 'set_trust_scope': <function
Token.set_trust_scope>, 'set_oauth': <function Token.set_oauth>,
'set_application_credential': <function
Token.set_application_credential>, 'service_providers': <property
object>, 'add_service_provider': <function Token.add_service_provider>,
'set_bind': <function Token.set_bind>, '__orig_bases__': (dict[str,
typing.Any],), '__dict__': <attribute '__dict__' of 'Token' objects>,
'__weakref__': <attribute '__weakref__' of 'Token' objects>,
'__annotations__': {}})
```

```
__doc__ = 'A V3 Keystone token that can be used for testing.\n\n This
object is designed to allow clients to generate a correct V3 token for\n
use in there test code. It should prevent clients from having to know
the\n correct token format and allow them to test the portions of token
handling\n that matter to them and not copy and paste sample.\n '
```

`__init__`(*expires=None*, *issued=None*, *user_id=None*, *user_name=None*, *user_domain_id=None*, *user_domain_name=None*, *methods=None*, *project_id=None*, *project_name=None*, *project_domain_id=None*, *project_domain_name=None*, *domain_id=None*, *domain_name=None*, *trust_id=None*, *trust_impersonation=None*, *trustee_user_id=None*, *trustor_user_id=None*, *application_credential_id=None*, *application_credential_access_rules=None*, *oauth_access_token_id=None*, *oauth_consumer_id=None*, *audit_id=None*, *audit_chain_id=None*, *is_admin_project=None*, *project_is_domain=None*, *oauth2_thumbprint=None*)

`__module__ = 'keystoneauth1.fixture.v3'`

`__orig_bases__ = (dict[str, typing.Any],)`

`__weakref__`

list of weak references to the object (if defined)

`property _user`

`property _user_domain`

`add_role`(*name=None*, *id=None*)

`add_service`(*type*, *name=None*, *id=None*)

`add_service_provider`(*sp_id*, *sp_auth_url*, *sp_url*)

`property application_credential_access_rules`

`property application_credential_id`

`property audit_chain_id`

`property audit_id`

`property domain_id`

`property domain_name`

`property expires`

`property expires_str`

`property is_admin_project`

`property issued`

`property issued_str`

`property methods`

`property oauth2_credential`

`property oauth2_thumbprint`

`property oauth_access_token_id`

property **oauth_consumer_id**

property **project_domain_id**

property **project_domain_name**

property **project_id**

property **project_is_domain**

property **project_name**

remove_service(*type*)

property **role_ids**

property **role_names**

property **root**

property **service_providers**

set_application_credential(*application_credential_id*, *access_rules=None*)

set_bind(*name*, *data*)

set_domain_scope(*id=None*, *name=None*)

set_oauth(*access_token_id=None*, *consumer_id=None*)

set_project_scope(*id=None*, *name=None*, *domain_id=None*, *domain_name=None*, *is_domain=None*)

set_system_scope()

set_trust_scope(*id=None*, *impersonation=False*, *trustee_user_id=None*, *trustor_user_id=None*)

property **system**

property **trust_id**

property **trust_impersonation**

property **trustee_user_id**

property **trustor_user_id**

property **user_domain_id**

property **user_domain_name**

property **user_id**

property **user_name**

validate()

class keystoneauth1.fixture.v3.**V3FederationToken**(*methods=None*, *identity_provider=None*, *protocol=None*, *groups=None*)

Bases: Token

A V3 Keystone Federation token that can be used for testing.

Similar to V3Token, this object is designed to allow clients to generate a correct V3 federation token for use in test code.

FEDERATED_DOMAIN_ID = 'Federated'

__annotations__ = {}

__doc__ = 'A V3 Keystone Federation token that can be used for testing.\n\n Similar to V3Token, this object is designed to allow clients to generate\n a correct V3 federation token for use in test code.\n '

__init__(*methods=None*, *identity_provider=None*, *protocol=None*, *groups=None*)

__module__ = 'keystoneauth1.fixture.v3'

add_federation_info_to_user(*identity_provider=None*, *protocol=None*, *groups=None*)

class keystoneauth1.fixture.v3.**_Service**

Bases: dict[str, Any]

One of the services that exist in the catalog.

You use this by adding a service to a token which returns an instance of this object and then you can add_endpoints to the service.

__dict__ = mappingproxy({'__module__': 'keystoneauth1.fixture.v3', '__doc__': 'One of the services that exist in the catalog.\n\n You use this by adding a service to a token which returns an instance of\n this object and then you can add_endpoints to the service.\n ', 'add_endpoint': <function _Service.add_endpoint>, 'add_standard_endpoints': <function _Service.add_standard_endpoints>, '__orig_bases__': (dict[str, typing.Any],), '__dict__': <attribute '__dict__' of '_Service' objects>, '__weakref__': <attribute '__weakref__' of '_Service' objects>, '__annotations__': {}})

__doc__ = 'One of the services that exist in the catalog.\n\n You use this by adding a service to a token which returns an instance of\n this object and then you can add_endpoints to the service.\n '

__module__ = 'keystoneauth1.fixture.v3'

__orig_bases__ = (dict[str, typing.Any],)

__weakref__

list of weak references to the object (if defined)

add_endpoint(*interface*, *url*, *region=None*, *id=None*)

add_standard_endpoints(*public=None*, *admin=None*, *internal=None*, *region=None*)

Module contents

Produce keystone compliant structures for use in testing.

They are part of the public API because they may be relied upon to generate test tokens for other clients. However they should never be imported into the main client (keystoneauth or other). Because of this there may be dependencies from this module on libraries that are only available in testing.

class keystoneauth1.fixture.**DiscoveryList**(*href=None*, *v2=True*, *v3=True*, *v2_id=None*, *v3_id=None*, *v2_status=None*, *v2_updated=None*, *v2_html=True*, *v2_pdf=True*, *v3_status=None*, *v3_updated=None*, *v3_json=True*, *v3_xml=True*)

Bases: dict[str, Any]

A List of version elements.

Creates a correctly structured list of identity service endpoints for use in testing with discovery.

Parameters

- **href** (*string*) The url that this should be based at.
- **v2** (*bool*) Add a v2 element.
- **v3** (*bool*) Add a v3 element.
- **v2_status** (*string*) The status to use for the v2 element.
- **v2_updated** (*DateTime*) The update time to use for the v2 element.
- **v2_html** (*bool*) True to add a html link to the v2 element.
- **v2_pdf** (*bool*) True to add a pdf link to the v2 element.
- **v3_status** (*string*) The status to use for the v3 element.
- **v3_updated** (*DateTime*) The update time to use for the v3 element.
- **v3_json** (*bool*) True to add a html link to the v2 element.
- **v3_xml** (*bool*) True to add a pdf link to the v2 element.

TEST_URL = 'http://keystone.host:5000/'

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.fixture.discovery', '__doc__': 'A List of version elements.\n\n Creates a correctly structured list of identity service endpoints for\n use in testing with discovery.\n\n :param string href: The url that this should be based at.\n :param bool v2: Add a v2 element.\n :param bool v3: Add a v3 element.\n :param string v2_status: The status to use for the v2 element.\n :param DateTime v2_updated: The update time to use for the v2 element.\n :param bool v2_html: True to add a html link to the v2 element.\n :param bool v2_pdf: True to add a pdf link to the v2 element.\n :param string v3_status: The status to use for the v3 element.\n :param DateTime v3_updated: The update time to use for the v3 element.\n :param bool v3_json: True to add a html link to the v2 element.\n :param bool v3_xml: True to add a pdf link to the v2 element.\n ', 'TEST_URL': 'http://keystone.host:5000/', '__init__': <function DiscoveryList.__init__>, 'versions': <property object>, 'add_version': <function DiscoveryList.add_version>, 'add_v2': <function DiscoveryList.add_v2>, 'add_v3': <function DiscoveryList.add_v3>, 'add_microversion': <function DiscoveryList.add_microversion>, 'add_nova_microversion': <function DiscoveryList.add_nova_microversion>, '__orig_bases__': (dict[str, typing.Any],), '__dict__': <attribute '__dict__' of 'DiscoveryList' objects>, '__weakref__': <attribute '__weakref__' of 'DiscoveryList' objects>, '__annotations__': {}})`

`__doc__ = 'A List of version elements.\n\n Creates a correctly structured list of identity service endpoints for\n use in testing with discovery.\n\n :param string href: The url that this should be based at.\n :param bool v2: Add a v2 element.\n :param bool v3: Add a v3 element.\n :param string v2_status: The status to use for the v2 element.\n :param DateTime v2_updated: The update time to use for the v2 element.\n :param bool v2_html: True to add a html link to the v2 element.\n :param bool v2_pdf: True to add a pdf link to the v2 element.\n :param string v3_status: The status to use for the v3 element.\n :param DateTime v3_updated: The update time to use for the v3 element.\n :param bool v3_json: True to add a html link to the v2 element.\n :param bool v3_xml: True to add a pdf link to the v2 element.\n '`

`__init__`(*href=None*, *v2=True*, *v3=True*, *v2_id=None*, *v3_id=None*, *v2_status=None*, *v2_updated=None*, *v2_html=True*, *v2_pdf=True*, *v3_status=None*, *v3_updated=None*, *v3_json=True*, *v3_xml=True*)

`__module__ = 'keystoneauth1.fixture.discovery'`

`__orig_bases__ = (dict[str, typing.Any],)`

`__weakref__`

list of weak references to the object (if defined)

`add_microversion`(*href*, *id*, ***kwargs*)

Add a microversion version to the list.

The parameters are the same as MicroversionDiscovery.

add_nova_microversion(*href*, *id*, ***kwargs*)

Add a nova microversion version to the list.

The parameters are the same as NovaMicroversionDiscovery.

add_v2(*href*, ***kwargs*)

Add a v2 version to the list.

The parameters are the same as V2Discovery.

add_v3(*href*, ***kwargs*)

Add a v3 version to the list.

The parameters are the same as V3Discovery.

add_version(*version*)

Add a new version structure to the list.

Parameters

version (*dict*) A new version structure to add to the list.

property versions

exception keystoneauth1.fixture.**FixtureValidationError**

Bases: `Exception`

The token you created is not legitimate.

The data contained in the token that was generated is not valid and would not have been returned from a keystone server. You should not do testing with this token.

__doc__ = 'The token you created is not legitimate.\n\n The data contained in the token that was generated is not valid and would\n not have been returned from a keystone server. You should not do testing\n with this token.\n '

__module__ = 'keystoneauth1.fixture.exception'

__weakref__

list of weak references to the object (if defined)

class keystoneauth1.fixture.**LoadingFixture**(*token=None*, *endpoint=None*, *user_id=None*, *project_id=None*)

Bases: `Fixture`

A fixture that will stub out all plugin loading calls.

When using keystoneauth plugins loaded from config, CLI or elsewhere it is often difficult to handle the plugin parts in tests because we dont have a reasonable default.

This fixture will create a `TestPlugin` that will be returned for all calls to plugin loading so you can simply bypass the authentication steps and return something well known.

Parameters

- **token** (*str*) The token to include in authenticated requests.
- **endpoint** (*str*) The endpoint to respond to service lookups with.
- **user_id** (*str*) The user_id to report for the authenticated user.

- `project_id` (*str*) The project_id to report for the authenticated user.

`MOCK_POINT = 'keystoneauth1.loading.base.get_plugin_loader'`

`__doc__ = "A fixture that will stub out all plugin loading calls.\n\n When using keystoneauth plugins loaded from config, CLI or elsewhere it is\n often difficult to handle the plugin parts in tests because we don't have a\n reasonable default.\n\n This fixture will create a :py:class:`TestPlugin` that will be\n returned for all calls to plugin loading so you can simply bypass the\n authentication steps and return something well known.\n\n :param str token: The token to include in authenticated requests.\n :param str endpoint: The endpoint to respond to service lookups with.\n :param str user_id: The user_id to report for the authenticated user.\n :param str project_id: The project_id to report for the authenticated user.\n "`

`__init__`(*token=None*, *endpoint=None*, *user_id=None*, *project_id=None*)

`__module__ = 'keystoneauth1.fixture.plugin'`

`create_plugin`()

`get_endpoint`(*path=None*, ***kwargs*)

Utility function to get the endpoint the plugin would return.

This function is provided as a convenience so you can do comparisons in your tests. Overriding it will not affect the endpoint returned by the plugin.

Parameters
`path` (*str*) The path to append to the plugin endpoint.

`get_plugin_loader`(*auth_type*)

`setUp`()

Prepare the Fixture for use.

This should not be overridden. Concrete fixtures should implement _setUp. Overriding of setUp is still supported, just not recommended.

After setUp has completed, the fixture will have one or more attributes which can be used (these depend totally on the concrete subclass).

Raises
MultipleExceptions if _setUp fails. The last exception captured within the MultipleExceptions will be a SetupError exception.

Returns
None.

Changed in 1.3
The recommendation to override setUp has been reversed - before 1.3, setUp() should be overridden, now it should not be.

Changed in 1.3.1
BaseException is now caught, and only subclasses of Exception are wrapped in MultipleExceptions.

class keystoneauth1.fixture.TestPlugin(*token=None*, *endpoint=None*, *user_id=None*, *project_id=None*)

Bases: `BaseAuthPlugin`

A simple plugin that returns what you gave it for testing.

When testing services that use authentication plugins you often want to stub out the authentication calls and focus on the important part of your service. This plugin acts like a real keystoneauth plugin and returns known standard values without having to stub out real keystone responses.

Note that this plugin is a BaseAuthPlugin and not a BaseIdentityPlugin. This means it implements the basic plugin interface that services should be using but does not implement get_auth_ref. get_auth_ref should not be relied upon by services because a user could always configure the service to use a non-keystone auth.

Parameters

- **token** (*str*) The token to include in authenticated requests.
- **endpoint** (*str*) The endpoint to respond to service lookups with.
- **user_id** (*str*) The user_id to report for the authenticated user.
- **project_id** (*str*) The project_id to report for the authenticated user.

__annotations__ = {}

__doc__ = 'A simple plugin that returns what you gave it for testing.\n\n When testing services that use authentication plugins you often want to\n stub out the authentication calls and focus on the important part of your\n service. This plugin acts like a real keystoneauth plugin and returns known\n standard values without having to stub out real keystone responses.\n\n Note that this plugin is a BaseAuthPlugin and not a BaseIdentityPlugin.\n This means it implements the basic plugin interface that services should be\n using but does not implement get_auth_ref. get_auth_ref should not be\n relied upon by services because a user could always configure the service\n to use a non-keystone auth.\n\n :param str token: The token to include in authenticated requests.\n :param str endpoint: The endpoint to respond to service lookups with.\n :param str user_id: The user_id to report for the authenticated user.\n :param str project_id: The project_id to report for the authenticated user.\n '

__init__(*token=None*, *endpoint=None*, *user_id=None*, *project_id=None*)

__module__ = 'keystoneauth1.fixture.plugin'

auth_type = 'test_plugin'

get_endpoint(*session*, ***kwargs*)

Return an endpoint for the client.

There are no required keyword arguments to `get_endpoint` as a plugin implementation should use best effort with the information available to determine the endpoint. However there are certain standard options that will be generated by the clients and should be used by plugins:

- `service_type`: what sort of service is required.

- `service_name`: the name of the service in the catalog.
- `interface`: what visibility the endpoint should have.
- `region_name`: the region the endpoint exists in.

Parameters

- **session** (*keystoneauth1.session.Session*) The session object that the auth_plugin belongs to.
- **kwargs** Ignored.

Returns
The base URL that will be used to talk to the required service or None if not available.

Return type
string

`get_project_id`(*session*)

Return the project id that we are authenticated to.

Wherever possible the project id should be inferred from the token however there are certain URLs and other places that require access to the currently authenticated project id.

Parameters
session (*keystoneauth1.session.Session*) A session object so the plugin can make HTTP calls.

Returns
A project identifier or None if one is not available.

Return type
str

`get_token`(*session*)

Obtain a token.

How the token is obtained is up to the plugin. If it is still valid it may be re-used, retrieved from cache or invoke an authentication request against a server.

Returning None will indicate that no token was able to be retrieved.

This function is misplaced as it should only be required for auth plugins that use the X-Auth-Token header. However due to the way plugins evolved this method is required and often called to trigger an authentication request on a new plugin.

When implementing a new plugin it is advised that you implement this method, however if you dont require the X-Auth-Token header override the *get_headers* method instead.

Parameters
session (*keystoneauth1.session.Session*) A session object so the plugin can make HTTP calls.

Returns
A token to use.

Return type
string

get_user_id(*session*)

Return a unique user identifier of the plugin.

Wherever possible the user id should be inferred from the token however there are certain URLs and other places that require access to the currently authenticated user id.

Parameters

session (*keystoneauth1.session.Session*) A session object so the plugin can make HTTP calls.

Returns

A user identifier or None if one is not available.

Return type

str

invalidate()

Invalidate the current authentication data.

This should result in fetching a new token on next call.

A plugin may be invalidated if an Unauthorized HTTP response is returned to indicate that the token may have been revoked or is otherwise now invalid.

Returns

True if there was something that the plugin did to invalidate. This means that it makes sense to try again. If nothing happens returns False to indicate give up.

Return type

bool

class keystoneauth1.fixture.**V2Discovery**(*href*, *id=None*, *html=True*, *pdf=True*, ***kwargs*)

Bases: DiscoveryBase

A Version element for a V2 identity service endpoint.

Provides some default values and helper methods for creating a v2.0 endpoint version structure. Clients should use this instead of creating their own structures.

Parameters

- **href** (*string*) The url that this entry should point to.
- **id** (*string*) The version id that should be reported. (optional) Defaults to v2.0.
- **html** (*bool*) Add HTML describedby links to the structure.
- **pdf** (*bool*) Add PDF describedby links to the structure.

_DESC_URL = 'https://developer.openstack.org/api-ref/identity/v2/'

__doc__ = "A Version element for a V2 identity service endpoint.\n\n Provides some default values and helper methods for creating a v2.0\n endpoint version structure. Clients should use this instead of creating\n their own structures.\n\n :param string href: The url that this entry should point to.\n :param string id: The version id that should be reported. (optional)\n Defaults to 'v2.0'.\n :param bool html: Add HTML describedby links to the structure.\n :param bool pdf: Add PDF describedby links to the structure.\n\n "

`__init__`(*href*, *id=None*, *html=True*, *pdf=True*, ***kwargs*)

`__module__ = 'keystoneauth1.fixture.discovery'`

`add_html_description`()

Add the HTML described by links.

The standard structure includes a link to a HTML document with the API specification. Add it to this entry.

`add_pdf_description`()

Add the PDF described by links.

The standard structure includes a link to a PDF document with the API specification. Add it to this entry.

`keystoneauth1.fixture.`**`V2Token`**

alias of `Token`

class `keystoneauth1.fixture.`**`V3Discovery`**(*href*, *id=None*, *json=True*, *xml=True*, ***kwargs*)

Bases: `DiscoveryBase`

A Version element for a V3 identity service endpoint.

Provides some default values and helper methods for creating a v3 endpoint version structure. Clients should use this instead of creating their own structures.

Parameters

- **href** The url that this entry should point to.
- **id** (*string*) The version id that should be reported. (optional) Defaults to v3.0.
- **json** (*bool*) Add JSON media-type elements to the structure.
- **xml** (*bool*) Add XML media-type elements to the structure.

`__annotations__ = {}`

`__doc__ = "A Version element for a V3 identity service endpoint.\n\n Provides some default values and helper methods for creating a v3\n endpoint version structure. Clients should use this instead of creating\n their own structures.\n\n :param href: The url that this entry should point to.\n :param string id: The version id that should be reported. (optional)\n Defaults to 'v3.0'.\n :param bool json: Add JSON media-type elements to the structure.\n :param bool xml: Add XML media-type elements to the structure.\n "`

`__init__`(*href*, *id=None*, *json=True*, *xml=True*, ***kwargs*)

`__module__ = 'keystoneauth1.fixture.discovery'`

`add_json_media_type`()

Add the JSON media-type links.

The standard structure includes a list of media-types that the endpoint supports. Add JSON to the list.

add_xml_media_type()

Add the XML media-type links.

The standard structure includes a list of media-types that the endpoint supports. Add XML to the list.

class keystoneauth1.fixture.**V3FederationToken**(*methods=None*, *identity_provider=None*, *protocol=None*, *groups=None*)

Bases: Token

A V3 Keystone Federation token that can be used for testing.

Similar to V3Token, this object is designed to allow clients to generate a correct V3 federation token for use in test code.

FEDERATED_DOMAIN_ID = 'Federated'

__doc__ = 'A V3 Keystone Federation token that can be used for testing.\n\n Similar to V3Token, this object is designed to allow clients to generate\n a correct V3 federation token for use in test code.\n '

__init__(*methods=None*, *identity_provider=None*, *protocol=None*, *groups=None*)

__module__ = 'keystoneauth1.fixture.v3'

add_federation_info_to_user(*identity_provider=None*, *protocol=None*, *groups=None*)

keystoneauth1.fixture.**V3Token**

alias of Token

class keystoneauth1.fixture.**VersionDiscovery**(*href*, *id*, ***kwargs*)

Bases: DiscoveryBase

A Version element for non-keystone services without microversions.

Provides some default values and helper methods for creating a microversion endpoint version structure. Clients should use this instead of creating their own structures.

Parameters

- **href** (*string*) The url that this entry should point to.
- **id** (*string*) The version id that should be reported.

__annotations__ = {}

__doc__ = 'A Version element for non-keystone services without microversions.\n\n Provides some default values and helper methods for creating a microversion\n endpoint version structure. Clients should use this instead of creating\n their own structures.\n\n :param string href: The url that this entry should point to.\n :param string id: The version id that should be reported.\n '

__init__(*href*, *id*, ***kwargs*)

__module__ = 'keystoneauth1.fixture.discovery'

keystoneauth1.identity package

Subpackages

keystoneauth1.identity.generic package

Submodules

keystoneauth1.identity.generic.base module

class keystoneauth1.identity.generic.base.**BaseGenericPlugin**(*auth_url: str | None = None*, ***, *tenant_id: str | None = None*, *tenant_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *system_scope: str | None = None*, *trust_id: str | None = None*, *default_domain_id: str | None = None*, *default_domain_name: str | None = None*, *reauthenticate: bool = True*)

Bases: `BaseIdentityPlugin`

An identity plugin that is not version dependent.

Internally we will construct a version dependent plugin with the resolved URL and then proxy all calls from the base plugin to the versioned one.

`__abstractmethods__ = frozenset({'create_plugin', 'get_cache_id_elements'})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', '_plugin': 'ty.Union[v2.Auth, v3.Auth, None]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': <class 'str'>, 'reauthenticate': 'bool'}`

`__doc__ = 'An identity plugin that is not version dependent.\n\n Internally we will construct a version dependent plugin with the resolved\n URL and then proxy all calls from the base plugin to the versioned one.\n '`

`__init__`(*auth_url: str | None = None*, ***, *tenant_id: str | None = None*, *tenant_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *system_scope: str | None = None*, *trust_id: str | None = None*, *default_domain_id: str | None = None*, *default_domain_name: str | None = None*, *reauthenticate: bool = True*)

`__module__` `= 'keystoneauth1.identity.generic.base'`

`_abc_impl` `= <_abc._abc_data object>`

`_discovery_cache``: dict[str, Discover]`

`_do_create_plugin`(*session: Session*) → Auth | Auth

`property` **`_has_domain_scope`**`: bool`

Are there domain parameters.

Domain parameters are v3 only so returns if any are set.

Returns

True if a domain parameter is set, false otherwise.

`_plugin``: Auth | Auth | None`

`auth_ref``: AccessInfo | None`

`auth_url``: str`

`abstract` **`create_plugin`**(*session: Session*, *version: tuple[int | float, ...]*, *url: str*, *raw_status: str | None = None*) → None | Auth | Auth

Create a plugin from the given parameters.

This function will be called multiple times with the version and url of a potential endpoint. If a plugin can be constructed that fits the params then it should return it. If not return None and then another call will be made with other available URLs.

Parameters

- **session** (`keystoneauth1.session.Session`) A session object.
- **version** (`tuple`) A tuple of the API version at the URL.
- **url** (`str`) The base URL for this version.
- **raw_status** (`str`) The status that was in the discovery field.

Returns

A plugin that can match the parameters or None if nothing.

`get_auth_ref`(*session: Session*) → AccessInfo

Obtain a token from an OpenStack Identity Service.

This method is overridden by the various token version plugins.

This function should not be called independently and is expected to be invoked via the do_authenticate function.

This function will be invoked if the AcessInfo object cached by the plugin is not valid. Thus plugins should always fetch a new AccessInfo when invoked. If you are looking to just retrieve the current auth data then you should use get_access.

Parameters
`session` (*keystoneauth1.session.Session*) A session object that can be used for communication.

Raises

- **keystoneauth1.exceptions.response.InvalidResponse** The response returned wasnt appropriate.
- **keystoneauth1.exceptions.http.HttpError** An error from an invalid HTTP response.

Returns
Token access information.

Return type
`keystoneauth1.access.AccessInfo`

abstract get_cache_id_elements() → dict[str, str | None]

Get the elements for this auth plugin that make it unique.

As part of the get_cache_id requirement we need to determine what aspects of this plugin and its values that make up the unique elements.

This should be overridden by plugins that wish to allow caching.

Returns
The unique attributes and values of this plugin.

Return type
A flat dict with a str key and str or None value. This is required as we feed these values into a hash. Pairs where the value is None are ignored in the hashed id.

property project_domain_id: str | None

property project_domain_name: str | None

reauthenticate: bool

keystoneauth1.identity.generic.password module

class keystoneauth1.identity.generic.password.**Password**(*auth_url: str, username: str | None = None, user_id: str | None = None, password: str | None = None, user_domain_id: str | None = None, user_domain_name: str | None = None, *, tenant_id: str | None = None, tenant_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, domain_id: str | None = None, domain_name: str | None = None, system_scope: str | None = None, trust_id: str | None = None, default_domain_id: str | None = None, default_domain_name: str | None = None, reauthenticate: bool = True*)

Bases: `BaseGenericPlugin`

A common user/password authentication plugin.

Parameters

- **username** (*string*) Username for authentication.
- **user_id** (*string*) User ID for authentication.
- **password** (*string*) Password for authentication.
- **user_domain_id** (*string*) Users domain ID for authentication.
- **user_domain_name** (*string*) Users domain name for authentication.

__abstractmethods__ = frozenset({})

__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', '_plugin': 'ty.Union[v2.Auth, v3.Auth, None]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}

__doc__ = "A common user/password authentication plugin.\n\n :param string username: Username for authentication.\n :param string user_id: User ID for authentication.\n :param string password: Password for authentication.\n :param string user_domain_id: User's domain ID for authentication.\n :param string user_domain_name: User's domain name for authentication.\n "

__init__(*auth_url: str, username: str | None = None, user_id: str | None = None, password: str | None = None, user_domain_id: str | None = None, user_domain_name: str | None = None, *, tenant_id: str | None = None, tenant_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, domain_id: str | None = None, domain_name: str | None = None, system_scope: str | None = None, trust_id: str | None = None, default_domain_id: str | None = None, default_domain_name: str | None = None, reauthenticate: bool = True*)

__module__ = 'keystoneauth1.identity.generic.password'

_abc_impl = <_abc._abc_data object>

_discovery_cache: dict[str, discover.Discover]

_plugin: ty.Union[v2.Auth, v3.Auth, None]

auth_ref: ty.Optional[access.AccessInfo]

auth_url: str

create_plugin(*session: Session, version: tuple[int | float, ...], url: str, raw_status: str | None = None*) → None | Password | Password

Create a plugin from the given parameters.

This function will be called multiple times with the version and url of a potential endpoint. If a plugin can be constructed that fits the params then it should return it. If not return None and then another call will be made with other available URLs.

Parameters

- **session** (*keystoneauth1.session.Session*) A session object.
- **version** (*tuple*) A tuple of the API version at the URL.
- **url** (*str*) The base URL for this version.
- **raw_status** (*str*) The status that was in the discovery field.

Returns

A plugin that can match the parameters or None if nothing.

get_cache_id_elements() → dict[str, str | None]

Get the elements for this auth plugin that make it unique.

As part of the get_cache_id requirement we need to determine what aspects of this plugin and its values that make up the unique elements.

This should be overridden by plugins that wish to allow caching.

Returns

The unique attributes and values of this plugin.

Return type

A flat dict with a str key and str or None value. This is required as we feed these values into a hash. Pairs where the value is None are ignored in the hashed id.

reauthenticate: bool

property user_domain_id: str | None

property user_domain_name: str | None

keystoneauth1.identity.generic.token module

class keystoneauth1.identity.generic.token.**Token**(*auth_url: str, token: str, *, tenant_id: str | None = None, tenant_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, domain_id: str | None = None, domain_name: str | None = None, system_scope: str | None = None, trust_id: str | None = None, default_domain_id: str | None = None, default_domain_name: str | None = None, reauthenticate: bool = True*)

Bases: BaseGenericPlugin

Generic token auth plugin.

Parameters
token (*string*) Token for authentication.

__abstractmethods__ = frozenset({})

__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', '_plugin': 'ty.Union[v2.Auth, v3.Auth, None]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}

__doc__ = 'Generic token auth plugin.\n\n :param string token: Token for authentication.\n '

__init__(*auth_url: str, token: str, *, tenant_id: str | None = None, tenant_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, domain_id: str | None = None, domain_name: str | None = None, system_scope: str | None = None, trust_id: str | None = None, default_domain_id: str | None = None, default_domain_name: str | None = None, reauthenticate: bool = True*)

__module__ = 'keystoneauth1.identity.generic.token'

_abc_impl = <_abc._abc_data object>

_discovery_cache: dict[str, discover.Discover]

_plugin: ty.Union[v2.Auth, v3.Auth, None]

auth_ref: ty.Optional[access.AccessInfo]

auth_url: str

create_plugin(*session: Session*, *version: tuple[int | float, ...]*, *url: str*, *raw_status: str | None = None*) → None | Token | Token

Create a plugin from the given parameters.

This function will be called multiple times with the version and url of a potential endpoint. If a plugin can be constructed that fits the params then it should return it. If not return None and then another call will be made with other available URLs.

Parameters

- **session** (*keystoneauth1.session.Session*) A session object.
- **version** (*tuple*) A tuple of the API version at the URL.
- **url** (*str*) The base URL for this version.
- **raw_status** (*str*) The status that was in the discovery field.

Returns

A plugin that can match the parameters or None if nothing.

get_cache_id_elements() → dict[str, str | None]

Get the elements for this auth plugin that make it unique.

As part of the get_cache_id requirement we need to determine what aspects of this plugin and its values that make up the unique elements.

This should be overridden by plugins that wish to allow caching.

Returns

The unique attributes and values of this plugin.

Return type

A flat dict with a str key and str or None value. This is required as we feed these values into a hash. Pairs where the value is None are ignored in the hashed id.

reauthenticate: bool

Module contents

class keystoneauth1.identity.generic.**BaseGenericPlugin**(*auth_url: str | None = None*, ***, *tenant_id: str | None = None*, *tenant_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *system_scope: str | None = None*, *trust_id: str | None = None*, *default_domain_id: str | None = None*, *default_domain_name: str | None = None*, *reauthenticate: bool = True*)

Bases: `BaseIdentityPlugin`

An identity plugin that is not version dependent.

Internally we will construct a version dependent plugin with the resolved URL and then proxy all calls from the base plugin to the versioned one.

`__abstractmethods__ = frozenset({'create_plugin', 'get_cache_id_elements'})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', '_plugin': 'ty.Union[v2.Auth, v3.Auth, None]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': <class 'str'>, 'reauthenticate': 'bool'}`

`__doc__ = 'An identity plugin that is not version dependent.\n\n Internally we will construct a version dependent plugin with the resolved\n URL and then proxy all calls from the base plugin to the versioned one.\n '`

`__init__`(*auth_url: str | None = None*, ***, *tenant_id: str | None = None*, *tenant_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *system_scope: str | None = None*, *trust_id: str | None = None*, *default_domain_id: str | None = None*, *default_domain_name: str | None = None*, *reauthenticate: bool = True*)

`__module__ = 'keystoneauth1.identity.generic.base'`

`_abc_impl = <_abc._abc_data object>`

`_do_create_plugin`(*session: Session*) → Auth | Auth

property `_has_domain_scope`: `bool`

Are there domain parameters.

Domain parameters are v3 only so returns if any are set.

Returns
True if a domain parameter is set, false otherwise.

auth_url: str

abstract create_plugin(*session: Session*, *version: tuple[int | float, ...]*, *url: str*, *raw_status: str | None = None*) → None | Auth | Auth

Create a plugin from the given parameters.

This function will be called multiple times with the version and url of a potential endpoint. If a plugin can be constructed that fits the params then it should return it. If not return None and then another call will be made with other available URLs.

Parameters

- **session** (*keystoneauth1.session.Session*) A session object.
- **version** (*tuple*) A tuple of the API version at the URL.
- **url** (*str*) The base URL for this version.
- **raw_status** (*str*) The status that was in the discovery field.

Returns
A plugin that can match the parameters or None if nothing.

get_auth_ref(*session: Session*) → AccessInfo

Obtain a token from an OpenStack Identity Service.

This method is overridden by the various token version plugins.

This function should not be called independently and is expected to be invoked via the do_authenticate function.

This function will be invoked if the AcessInfo object cached by the plugin is not valid. Thus plugins should always fetch a new AccessInfo when invoked. If you are looking to just retrieve the current auth data then you should use get_access.

Parameters
session (*keystoneauth1.session.Session*) A session object that can be used for communication.

Raises

- **keystoneauth1.exceptions.response.InvalidResponse** The response returned wasnt appropriate.
- **keystoneauth1.exceptions.http.HttpError** An error from an invalid HTTP response.

Returns
Token access information.

Return type
`keystoneauth1.access.AccessInfo`

abstract get_cache_id_elements() → dict[str, str | None]

Get the elements for this auth plugin that make it unique.

As part of the get_cache_id requirement we need to determine what aspects of this plugin and its values that make up the unique elements.

This should be overridden by plugins that wish to allow caching.

Returns

The unique attributes and values of this plugin.

Return type

A flat dict with a str key and str or None value. This is required as we feed these values into a hash. Pairs where the value is None are ignored in the hashed id.

property project_domain_id: str | None

property project_domain_name: str | None

class keystoneauth1.identity.generic.**Password**(*auth_url: str*, *username: str | None = None*, *user_id: str | None = None*, *password: str | None = None*, *user_domain_id: str | None = None*, *user_domain_name: str | None = None*, ***, *tenant_id: str | None = None*, *tenant_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *system_scope: str | None = None*, *trust_id: str | None = None*, *default_domain_id: str | None = None*, *default_domain_name: str | None = None*, *reauthenticate: bool = True*)

Bases: `BaseGenericPlugin`

A common user/password authentication plugin.

Parameters

- **username** (*string*) Username for authentication.
- **user_id** (*string*) User ID for authentication.
- **password** (*string*) Password for authentication.
- **user_domain_id** (*string*) Users domain ID for authentication.
- **user_domain_name** (*string*) Users domain name for authentication.

__abstractmethods__ = frozenset({})

__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', '_plugin': 'ty.Union[v2.Auth, v3.Auth, None]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}

`__doc__ = "A common user/password authentication plugin.\n\n :param string username: Username for authentication.\n :param string user_id: User ID for authentication.\n :param string password: Password for authentication.\n :param string user_domain_id: User's domain ID for authentication.\n :param string user_domain_name: User's domain name for authentication.\n "`

`__init__`(*auth_url: str, username: str | None = None, user_id: str | None = None, password: str | None = None, user_domain_id: str | None = None, user_domain_name: str | None = None, *, tenant_id: str | None = None, tenant_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, domain_id: str | None = None, domain_name: str | None = None, system_scope: str | None = None, trust_id: str | None = None, default_domain_id: str | None = None, default_domain_name: str | None = None, reauthenticate: bool = True*)

`__module__ = 'keystoneauth1.identity.generic.password'`

`_abc_impl = <_abc._abc_data object>`

`_discovery_cache: dict[str, discover.Discover]`

`_plugin: ty.Union[v2.Auth, v3.Auth, None]`

`auth_ref: ty.Optional[access.AccessInfo]`

`auth_url: str`

`create_plugin`(*session: Session, version: tuple[int | float, ...], url: str, raw_status: str | None = None*) → None | Password | Password

Create a plugin from the given parameters.

This function will be called multiple times with the version and url of a potential endpoint. If a plugin can be constructed that fits the params then it should return it. If not return None and then another call will be made with other available URLs.

Parameters

- **session** (*keystoneauth1.session.Session*) A session object.
- **version** (*tuple*) A tuple of the API version at the URL.
- **url** (*str*) The base URL for this version.
- **raw_status** (*str*) The status that was in the discovery field.

Returns

A plugin that can match the parameters or None if nothing.

`get_cache_id_elements`() → dict[str, str | None]

Get the elements for this auth plugin that make it unique.

As part of the get_cache_id requirement we need to determine what aspects of this plugin and its values that make up the unique elements.

This should be overridden by plugins that wish to allow caching.

Returns
The unique attributes and values of this plugin.

Return type
A flat dict with a str key and str or None value. This is required as we feed these values into a hash. Pairs where the value is None are ignored in the hashed id.

reauthenticate: bool

property user_domain_id: str | None

property user_domain_name: str | None

class keystoneauth1.identity.generic.Token(*auth_url: str*, *token: str*, ***, *tenant_id: str | None = None*, *tenant_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *system_scope: str | None = None*, *trust_id: str | None = None*, *default_domain_id: str | None = None*, *default_domain_name: str | None = None*, *reauthenticate: bool = True*)

Bases: `BaseGenericPlugin`

Generic token auth plugin.

Parameters
token (*string*) Token for authentication.

__abstractmethods__ = frozenset({})

__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', '_plugin': 'ty.Union[v2.Auth, v3.Auth, None]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}

__doc__ = 'Generic token auth plugin.\n\n :param string token: Token for authentication.\n '

__init__(*auth_url: str*, *token: str*, ***, *tenant_id: str | None = None*, *tenant_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *system_scope: str | None = None*, *trust_id: str | None = None*, *default_domain_id: str | None = None*, *default_domain_name: str | None = None*, *reauthenticate: bool = True*)

__module__ = 'keystoneauth1.identity.generic.token'

_abc_impl = <_abc._abc_data object>

_discovery_cache: dict[str, discover.Discover]

`_plugin: ty.Union[v2.Auth, v3.Auth, None]`

`auth_ref: ty.Optional[access.AccessInfo]`

`auth_url: str`

`create_plugin`(*session: Session*, *version: tuple[int | float, ...]*, *url: str*, *raw_status: str | None = None*) → None | Token | Token

Create a plugin from the given parameters.

This function will be called multiple times with the version and url of a potential endpoint. If a plugin can be constructed that fits the params then it should return it. If not return None and then another call will be made with other available URLs.

Parameters

- **session** (*keystoneauth1.session.Session*) A session object.
- **version** (*tuple*) A tuple of the API version at the URL.
- **url** (*str*) The base URL for this version.
- **raw_status** (*str*) The status that was in the discovery field.

Returns

A plugin that can match the parameters or None if nothing.

`get_cache_id_elements`() → dict[str, str | None]

Get the elements for this auth plugin that make it unique.

As part of the get_cache_id requirement we need to determine what aspects of this plugin and its values that make up the unique elements.

This should be overridden by plugins that wish to allow caching.

Returns

The unique attributes and values of this plugin.

Return type

A flat dict with a str key and str or None value. This is required as we feed these values into a hash. Pairs where the value is None are ignored in the hashed id.

`reauthenticate: bool`

keystoneauth1.identity.v3 package

Submodules

keystoneauth1.identity.v3.application_credential module

class keystoneauth1.identity.v3.application_credential.**ApplicationCredential**(*auth_url: str, application_credential_se… str, application_credential_id… str | None = None, application_credential_na… str | None = None, user_id: str | None = None, username: str | None = None, user_domain_id: str | None = None, user_domain_nam… str | None = None, *, unscoped: bool = False, trust_id:*

Bases: `Auth`

A plugin for authenticating with an application credential.

Parameters

- **auth_url** (*string*) Identity service endpoint for authentication.
- **application_credential_secret** (*string*) Application credential secret.
- **application_credential_id** (*string*) Application credential ID.
- **application_credential_name** (*string*) Application credential name.
- **username** (*string*) Username for authentication.
- **user_id** (*string*) User ID for authentication.
- **user_domain_id** (*string*) Users domain ID for authentication.
- **user_domain_name** (*string*) Users domain name for authentication.
- **reauthenticate** (*bool*) Allow fetching a new token if the current one is going to expire. (optional) default True

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}`

`__doc__ = "A plugin for authenticating with an application credential.\n\n :param string auth_url: Identity service endpoint for authentication.\n :param string application_credential_secret: Application credential secret.\n :param string application_credential_id: Application credential ID.\n :param string application_credential_name: Application credential name.\n :param string username: Username for authentication.\n :param string user_id: User ID for authentication.\n :param string user_domain_id: User's domain ID for authentication.\n :param string user_domain_name: User's domain name for authentication.\n :param bool reauthenticate: Allow fetching a new token if the current one\n is going to expire. (optional) default True\n "`

`__init__`(*auth_url: str*, *application_credential_secret: str*, *application_credential_id: str | None = None*, *application_credential_name: str | None = None*, *user_id: str | None = None*, *username: str | None = None*, *user_domain_id: str | None = None*, *user_domain_name: str | None = None*, ***, *unscoped: bool = False*, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*) → None

`__module__ = 'keystoneauth1.identity.v3.application_credential'`

`_abc_impl = <_abc._abc_data object>`

_auth_method_class

alias of `ApplicationCredentialMethod`

_discovery_cache: `dict[str, discover.Discover]`

auth_ref: `ty.Optional[access.AccessInfo]`

auth_url: `str`

reauthenticate: `bool`

class keystoneauth1.identity.v3.application_credential.**ApplicationCredentialMethod**(**, application_creden str, application_creden str | None = None, application_creden str | None = None, user_id: str | None = None, username: str | None = None, user_doma str | None = None, user_doma str | None = None*)

Bases: `AuthMethod`

Construct a User/Passcode based authentication method.

Parameters

- **application_credential_secret** (*string*) Application credential secret.
- **application_credential_id** (*string*) Application credential id.
- **application_credential_name** (*string*) The name of the application credential, if an ID is not provided.
- **username** (*string*) Username for authentication, if an application credential ID is not provided.
- **user_id** (*string*) User ID for authentication, if an application credential ID is not provided.
- **user_domain_id** (*string*) Users domain ID for authentication, if an application credential ID is not provided.
- **user_domain_name** (*string*) Users domain name for authentication, if an application credential ID is not provided.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'_method_parameters': 'ty.Optional[list[str]]', 'application_credential_id': typing.Optional[str], 'application_credential_name': typing.Optional[str], 'application_credential_secret': <class 'str'>, 'user_domain_id': typing.Optional[str], 'user_domain_name': typing.Optional[str], 'user_id': typing.Optional[str], 'username': typing.Optional[str]}`

`__doc__ = "Construct a User/Passcode based authentication method.\n\n :param string application_credential_secret: Application credential secret.\n :param string application_credential_id: Application credential id.\n :param string application_credential_name: The name of the application\n credential, if an ID is not\n provided.\n :param string username: Username for authentication, if an application\n credential ID is not provided.\n :param string user_id: User ID for authentication, if an application\n credential ID is not provided.\n :param string user_domain_id: User's domain ID for authentication, if an\n application credential ID is not provided.\n :param string user_domain_name: User's domain name for authentication, if\n an application credential ID is not\n provided.\n "`

`__init__`(**, application_credential_secret: str, application_credential_id: str | None = None, application_credential_name: str | None = None, user_id: str | None = None, username: str | None = None, user_domain_id: str | None = None, user_domain_name: str | None = None*) → None

`__module__ = 'keystoneauth1.identity.v3.application_credential'`

`_abc_impl = <_abc._abc_data object>`

`application_credential_id: str | None = None`

`application_credential_name: str | None = None`

`application_credential_secret: str`

get_auth_data(*session: Session*, *auth: Auth*, *headers: dict[str, str]*, *request_kwargs: dict[str, object]*) → tuple[None, None] | tuple[str, Mapping[str, object]]

Return the authentication section of an auth plugin.

Parameters

- **session** (*keystoneauth1.session.Session*) The communication session.
- **auth** (*base.Auth*) The auth plugin calling the method.
- **headers** (*dict*) The headers that will be sent with the auth request if a plugin needs to add to them.

Returns

The identifier of this plugin and a dict of authentication data for the auth type.

Return type

tuple(string, dict)

get_cache_id_elements() → dict[str, str | None]

Get the elements for this auth method that make it unique.

These elements will be used as part of the `keystoneauth1.plugin.BaseIdentityPlugin.get_cache_id()` to allow caching of the auth plugin.

Plugins should override this if they want to allow caching of their state.

To avoid collision or overrides the keys of the returned dictionary should be prefixed with the plugin identifier. For example the password plugin returns its username value as password_username.

`user_domain_id: str | None = None`

`user_domain_name: str | None = None`

`user_id: str | None = None`

`username: str | None = None`

keystoneauth1.identity.v3.base module

class keystoneauth1.identity.v3.base.**Auth**(*auth_url: str*, *auth_methods: list[AuthMethod]*, **, *unscoped: bool = False*, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*)

Bases: `BaseAuth`

Identity V3 Authentication Plugin.

Parameters

- **auth_url** (*string*) Identity service endpoint for authentication.
- **auth_methods** (*list*) A collection of methods to authenticate with.
- **trust_id** (*string*) Trust ID for trust scoping.
- **domain_id** (*string*) Domain ID for domain scoping.
- **domain_name** (*string*) Domain name for domain scoping.
- **project_id** (*string*) Project ID for project scoping.
- **project_name** (*string*) Project name for project scoping.
- **project_domain_id** (*string*) Projects domain ID for project.
- **project_domain_name** (*string*) Projects domain name for project.
- **reauthenticate** (*bool*) Allow fetching a new token if the current one is going to expire. (optional) default True
- **include_catalog** (*bool*) Include the service catalog in the returned token. (optional) default True.
- **unscoped** (*bool*) Force the return of an unscoped token. This will make the keystone server return an unscoped token even if a default_project_id is set for this user.

__abstractmethods__ = frozenset({})

__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}

__doc__ = "Identity V3 Authentication Plugin.\n\n :param string auth_url: Identity service endpoint for authentication.\n :param list auth_methods: A collection of methods to authenticate with.\n :param string trust_id: Trust ID for trust scoping.\n :param string domain_id: Domain ID for domain scoping.\n :param string domain_name: Domain name for domain scoping.\n :param string project_id: Project ID for project scoping.\n :param string project_name: Project name for project scoping.\n :param string project_domain_id: Project's domain ID for project.\n :param string project_domain_name: Project's domain name for project.\n :param bool reauthenticate: Allow fetching a new token if the current one\n is going to expire. (optional) default True\n :param bool include_catalog: Include the service catalog in the returned\n token. (optional) default True.\n :param bool unscoped: Force the return of an unscoped token. This will make\n the keystone server return an unscoped token even if\n a default_project_id is set for this user.\n "

`__init__`(*auth_url: str*, *auth_methods: list[AuthMethod]*, ***, *unscoped: bool = False*, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*)

`__module__ = 'keystoneauth1.identity.v3.base'`

`_abc_impl = <_abc._abc_data object>`

`_discovery_cache: dict[str, discover.Discover]`

`add_method`(*method: AuthMethod*) → None

Add an additional initialized AuthMethod instance.

`auth_ref: ty.Optional[access.AccessInfo]`

`auth_url: str`

`get_auth_ref`(*session: Session*) → AccessInfoV3

Obtain a token from an OpenStack Identity Service.

This method is overridden by the various token version plugins.

This function should not be called independently and is expected to be invoked via the do_authenticate function.

This function will be invoked if the AcessInfo object cached by the plugin is not valid. Thus plugins should always fetch a new AccessInfo when invoked. If you are looking to just retrieve the current auth data then you should use get_access.

Parameters

session (*keystoneauth1.session.Session*) A session object that can be used for communication.

Raises

- **keystoneauth1.exceptions.response.InvalidResponse** The response returned wasnt appropriate.
- **keystoneauth1.exceptions.http.HttpError** An error from an invalid HTTP response.

Returns

Token access information.

Return type

`keystoneauth1.access.AccessInfo`

`get_cache_id_elements`() → dict[str, str | None]

Get the elements for this auth plugin that make it unique.

As part of the get_cache_id requirement we need to determine what aspects of this plugin and its values that make up the unique elements.

This should be overridden by plugins that wish to allow caching.

Returns

The unique attributes and values of this plugin.

Return type
A flat dict with a str key and str or None value. This is required as we feed these values into a hash. Pairs where the value is None are ignored in the hashed id.

reauthenticate: bool

class keystoneauth1.identity.v3.base.AuthConstructor(*auth_url: str*, **args: Any*, *unscoped: bool = False*, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*, ***kwargs: Any*)

Bases: `Auth`

Abstract base class for creating an Auth Plugin.

The Auth Plugin created contains only one authentication method. This is generally the required usage.

An AuthConstructor creates an AuthMethod based on the methods arguments and the auth_method_class defined by the plugin. It then creates the auth plugin with only that authentication method.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_auth_method_class': typing.ClassVar[typing.Type[keystoneauth1.identity.v3.base.AuthMethod]], '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}`

`__doc__ = "Abstract base class for creating an Auth Plugin.\n\n The Auth Plugin created contains only one authentication method. This\n is generally the required usage.\n\n An AuthConstructor creates an AuthMethod based on the method's\n arguments and the auth_method_class defined by the plugin. It then\n creates the auth plugin with only that authentication method.\n "`

`__init__`(*auth_url: str*, **args: Any*, *unscoped: bool = False*, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*, ***kwargs: Any*)

`__module__ = 'keystoneauth1.identity.v3.base'`

`_abc_impl = <_abc._abc_data object>`

_auth_method_class: ClassVar[Type[AuthMethod]]

_discovery_cache: dict[str, discover.Discover]

auth_ref: ty.Optional[access.AccessInfo]

auth_url: str

reauthenticate: bool

class keystoneauth1.identity.v3.base.**AuthMethod**(***kwargs: object*)

Bases: object

One part of a V3 Authentication strategy.

The v3 /tokens API allow multiple methods to be presented when authentication against the server. Each one of these methods is implemented by an AuthMethod.

Note: When implementing an AuthMethod use keyword arguments to ensure they are supported by the MultiFactor auth plugin.

__abstractmethods__ = frozenset({'get_auth_data'})

__annotations__ = {'_method_parameters': typing.Optional[list[str]]}

__dict__ = mappingproxy({'__module__': 'keystoneauth1.identity.v3.base', '__annotations__': {'_method_parameters': typing.Optional[list[str]]}, '__doc__': "One part of a V3 Authentication strategy.\n\n The v3 '/tokens' API allow multiple methods to be presented when\n authentication against the server. Each one of these methods is implemented\n by an AuthMethod.\n\n Note: When implementing an AuthMethod use keyword arguments to ensure they\n are supported by the MultiFactor auth plugin.\n ", '_method_parameters': None, '__init__': <function AuthMethod.__init__>, '_extract_kwargs': <classmethod(<function AuthMethod._extract_kwargs>)>, 'get_auth_data': <function AuthMethod.get_auth_data>, 'get_cache_id_elements': <function AuthMethod.get_cache_id_elements>, '__dict__': <attribute '__dict__' of 'AuthMethod' objects>, '__weakref__': <attribute '__weakref__' of 'AuthMethod' objects>, '__abstractmethods__': frozenset({'get_auth_data'}), '_abc_impl': <_abc._abc_data object>})

__doc__ = "One part of a V3 Authentication strategy.\n\n The v3 '/tokens' API allow multiple methods to be presented when\n authentication against the server. Each one of these methods is implemented\n by an AuthMethod.\n\n Note: When implementing an AuthMethod use keyword arguments to ensure they\n are supported by the MultiFactor auth plugin.\n "

__init__(***kwargs: object*)

__module__ = 'keystoneauth1.identity.v3.base'

__weakref__

list of weak references to the object (if defined)

_abc_impl = <_abc._abc_data object>

classmethod _extract_kwargs(*kwargs: dict[str, object]*) → dict[str, object]

Remove parameters related to this method from other kwargs.

_method_parameters: list[str] | None = None

Deprecated parameter for defining the parameters supported by the plugin. These should now be defined by typed class attributes.

abstract get_auth_data(*session: Session, auth: Auth, headers: dict[str, str], request_kwargs: dict[str, object]*) → tuple[None, None] | tuple[str, Mapping[str, object]]

Return the authentication section of an auth plugin.

Parameters

- **session** (*keystoneauth1.session.Session*) The communication session.
- **auth** (*base.Auth*) The auth plugin calling the method.
- **headers** (*dict*) The headers that will be sent with the auth request if a plugin needs to add to them.

Returns

The identifier of this plugin and a dict of authentication data for the auth type.

Return type

tuple(string, dict)

get_cache_id_elements() → dict[str, str | None]

Get the elements for this auth method that make it unique.

These elements will be used as part of the `keystoneauth1.plugin.BaseIdentityPlugin.get_cache_id()` to allow caching of the auth plugin.

Plugins should override this if they want to allow caching of their state.

To avoid collision or overrides the keys of the returned dictionary should be prefixed with the plugin identifier. For example the password plugin returns its username value as password_username.

class keystoneauth1.identity.v3.base.**BaseAuth**(*auth_url: str, *, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

Bases: `BaseIdentityPlugin`

Identity V3 Authentication Plugin.

Parameters

- **auth_url** (*string*) Identity service endpoint for authentication.
- **trust_id** (*string*) Trust ID for trust scoping.

- **system_scope** (*string*) System information to scope to.
- **domain_id** (*string*) Domain ID for domain scoping.
- **domain_name** (*string*) Domain name for domain scoping.
- **project_id** (*string*) Project ID for project scoping.
- **project_name** (*string*) Project name for project scoping.
- **project_domain_id** (*string*) Projects domain ID for project.
- **project_domain_name** (*string*) Projects domain name for project.
- **reauthenticate** (*bool*) Allow fetching a new token if the current one is going to expire. (optional) default True
- **include_catalog** (*bool*) Include the service catalog in the returned token. (optional) default True.

__abstractmethods__ = frozenset({'get_auth_ref'})

__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': <class 'str'>, 'reauthenticate': 'bool'}

__doc__ = "Identity V3 Authentication Plugin.\n\n :param string auth_url: Identity service endpoint for authentication.\n :param string trust_id: Trust ID for trust scoping.\n :param string system_scope: System information to scope to.\n :param string domain_id: Domain ID for domain scoping.\n :param string domain_name: Domain name for domain scoping.\n :param string project_id: Project ID for project scoping.\n :param string project_name: Project name for project scoping.\n :param string project_domain_id: Project's domain ID for project.\n :param string project_domain_name: Project's domain name for project.\n :param bool reauthenticate: Allow fetching a new token if the current one\n is going to expire. (optional) default True\n :param bool include_catalog: Include the service catalog in the returned\n token. (optional) default True.\n "

__init__(*auth_url: str*, ***, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*)

__module__ = 'keystoneauth1.identity.v3.base'

_abc_impl = <_abc._abc_data object>

_discovery_cache: dict[str, discover.Discover]

auth_ref: ty.Optional[access.AccessInfo]

auth_url: str

property has_scope_parameters: bool

Return true if parameters can be used to create a scoped token.

reauthenticate: bool

property token_url: str

The full URL where we will send authentication data.

class keystoneauth1.identity.v3.base.**SupportsMultiFactor**(**args*, ***kwargs*)

Bases: Protocol

__abstractmethods__ = frozenset({})

__annotations__ = {'_auth_method_class': typing.ClassVar[typing.Type[keystoneauth1.identity.v3.base.AuthMethod]]}

__dict__ = mappingproxy({'__module__': 'keystoneauth1.identity.v3.base', '__annotations__': {'_auth_method_class': typing.ClassVar[typing.Type[keystoneauth1.identity.v3.base.AuthMethod]]}, '__dict__': <attribute '__dict__' of 'SupportsMultiFactor' objects>, '__weakref__': <attribute '__weakref__' of 'SupportsMultiFactor' objects>, '__doc__': None, '__parameters__': (), '_is_protocol': True, '__subclasshook__': <function Protocol.__init_subclass__.<locals>._proto_hook>, '__init__': <function _no_init_or_replace_init>, '__abstractmethods__': frozenset(), '_abc_impl': <_abc._abc_data object>, '_is_runtime_protocol': True})

__doc__ = None

__init__(**args*, ***kwargs*)

__module__ = 'keystoneauth1.identity.v3.base'

__parameters__ = ()

__subclasshook__()

Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__

list of weak references to the object (if defined)

_abc_impl = <_abc._abc_data object>

_auth_method_class: ClassVar[Type[AuthMethod]]

_is_protocol = True

_is_runtime_protocol = True

class keystoneauth1.identity.v3.base.**_AuthBody**

Bases: TypedDict

__annotations__ = {'auth': <class 'keystoneauth1.identity.v3.base._AuthIdentity'>}

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.identity.v3.base', '__annotations__': {'auth': <class 'keystoneauth1.identity.v3.base._AuthIdentity'>}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of '_AuthBody' objects>, '__weakref__': <attribute '__weakref__' of '_AuthBody' objects>, '__doc__': None, '__required_keys__': frozenset({'auth'}), '__optional_keys__': frozenset(), '__total__': True})`

`__doc__ = None`

`__module__ = 'keystoneauth1.identity.v3.base'`

`__optional_keys__ = frozenset({})`

`__orig_bases__ = (<function TypedDict>,)`

`__required_keys__ = frozenset({'auth'})`

`__total__ = True`

`__weakref__`

list of weak references to the object (if defined)

`auth: _AuthIdentity`

class keystoneauth1.identity.v3.base.**_AuthIdentity**

Bases: TypedDict

`__annotations__ = {'identity': dict[str, typing.Any], 'scope': typing_extensions.NotRequired[typing.Union[dict[str, typing.Any], str]]}`

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.identity.v3.base', '__annotations__': {'identity': dict[str, typing.Any], 'scope': typing_extensions.NotRequired[typing.Union[dict[str, typing.Any], str]]}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of '_AuthIdentity' objects>, '__weakref__': <attribute '__weakref__' of '_AuthIdentity' objects>, '__doc__': None, '__required_keys__': frozenset({'scope', 'identity'}), '__optional_keys__': frozenset(), '__total__': True})`

`__doc__ = None`

`__module__ = 'keystoneauth1.identity.v3.base'`

`__optional_keys__ = frozenset({})`

`__orig_bases__ = (<function TypedDict>,)`

`__required_keys__ = frozenset({'identity', 'scope'})`

`__total__ = True`

`__weakref__`

list of weak references to the object (if defined)

`identity: dict[str, Any]`

`scope: NotRequired[dict[str, Any] | str]`

keystoneauth1.identity.v3.federation module

class keystoneauth1.identity.v3.federation.**FederationBaseAuth**(*auth_url: str*, *identity_provider: str*, *protocol: str*, ***, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*)

Bases: `_Rescoped`

Federation authentication plugin.

Parameters

- **auth_url** (*string*) URL of the Identity Service
- **identity_provider** (*string*) name of the Identity Provider the client will authenticate against. This parameter will be used to build a dynamic URL used to obtain unscoped OpenStack token.
- **protocol** (*string*) name of the protocol the client will authenticate against.

__abstractmethods__ = frozenset({'get_unscoped_auth_ref'})

__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}

__doc__ = 'Federation authentication plugin.\n\n :param auth_url: URL of the Identity Service\n :type auth_url: string\n :param identity_provider: name of the Identity Provider the client\n will authenticate against. This parameter\n will be used to build a dynamic URL used to\n obtain unscoped OpenStack token.\n :type identity_provider: string\n :param protocol: name of the protocol the client will authenticate\n against.\n :type protocol: string\n\n '

`__init__`(*auth_url: str, identity_provider: str, protocol: str, *, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

`__module__ = 'keystoneauth1.identity.v3.federation'`

`_abc_impl = <_abc._abc_data object>`

`_discovery_cache: dict[str, discover.Discover]`

`auth_ref: ty.Optional[access.AccessInfo]`

`auth_url: str`

`property federated_token_url: str`

Full URL where authorization data is sent.

`reauthenticate: bool`

class `keystoneauth1.identity.v3.federation.`**`_Rescoped`**(*auth_url: str, *, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

Bases: `BaseAuth`

A plugin that is always going to go through a rescope process.

The original keystone plugins could simply pass a project or domain to along with the credentials and get a scoped token. For federation, K2K and newer mechanisms we always get an unscoped token first and then rescope.

This is currently not public as its generally an abstraction of a flow used by plugins within keystoneauth1.

It also cannot go in base as it depends on token.Token for rescoping which would create a circular dependency.

`__abstractmethods__ = frozenset({'get_unscoped_auth_ref'})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}`

`__doc__ = "A plugin that is always going to go through a rescope process.\n\n The original keystone plugins could simply pass a project or domain to\n along with the credentials and get a scoped token. For federation, K2K and\n newer mechanisms we always get an unscoped token first and then rescope.\n\n This is currently not public as it's generally an abstraction of a flow\n used by plugins within keystoneauth1.\n\n It also cannot go in base as it depends on token.Token for rescoping which\n would create a circular dependency.\n "`

`__module__ = 'keystoneauth1.identity.v3.federation'`

`_abc_impl = <_abc._abc_data object>`

get_auth_ref(*session: Session*) → AccessInfoV3

Authenticate retrieve token information.

This is a multi-step process where a client does federated authn receives an unscoped token.

If an unscoped token is successfully received and scoping information is present then the token is rescoped to that target.

Parameters

session (*keystoneauth1.session.Session*) a session object to send out HTTP requests.

Returns

a token data representation

Return type

`keystoneauth1.access.AccessInfo`

abstract **get_unscoped_auth_ref**(*session: Session*) → AccessInfoV3

Fetch unscoped federated token.

rescoping_plugin

alias of `Token`

keystoneauth1.identity.v3.k2k module

class keystoneauth1.identity.v3.k2k.**Keystone2Keystone**(*base_plugin: BaseIdentityPlugin*, *service_provider: str*, ***, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*)

Bases: `_Rescoped`

Plugin to execute the Keystone to Keyestone authentication flow.

In this plugin, an ECP wrapped SAML assertion provided by a keystone Identity Provider (IdP) is used to request an OpenStack unscoped token from a keystone Service Provider (SP).

Parameters

- **base_plugin** (*keystoneauth1.identity.v3.base.BaseAuth*) Auth plugin already authenticated against the keystone IdP.
- **service_provider** (*str*) The Service Provider ID as returned by ServiceProviderManager.list()

HTTP_MOVED_TEMPORARILY = 302

HTTP_SEE_OTHER = 303

REQUEST_ECP_URL = '/auth/OS-FEDERATION/saml2/ecp'

Path where the ECP wrapped SAML assertion should be presented to the Keystone Service Provider.

__abstractmethods__ = frozenset({})

__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}

__doc__ = 'Plugin to execute the Keystone to Keyestone authentication flow.\n\n In this plugin, an ECP wrapped SAML assertion provided by a keystone\n Identity Provider (IdP) is used to request an OpenStack unscoped token\n from a keystone Service Provider (SP).\n\n :param base_plugin: Auth plugin already authenticated against the keystone\n IdP.\n :type base_plugin: keystoneauth1.identity.v3.base.BaseAuth\n\n :param service_provider: The Service Provider ID as returned by\n ServiceProviderManager.list()\n :type service_provider: str\n\n '

__init__(*base_plugin: BaseIdentityPlugin*, *service_provider: str*, ***, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*)

__module__ = 'keystoneauth1.identity.v3.k2k'

_abc_impl = <_abc._abc_data object>

_discovery_cache: dict[str, discover.Discover]

_get_ecp_assertion(*session: Session*) → str

classmethod _remote_auth_url(*auth_url: str*) → str

Return auth_url of the remote Keystone Service Provider.

Remote clouds auth_url is an endpoint for getting federated unscoped token, typically that would be `https://remote.example.com:5000/v3/OS-FEDERATION/identity_providers/ <idp>/protocols/<protocol_id>/auth`. However we need to generate a real auth_url, used for token scoping. This function assumes there are static values today in the remote auth_url stored in the Service Provider attribute and those can be used as a delimiter. If the sp_auth_url doesnt comply with standard federation auth url the function will simply return whole string.

Parameters
auth_url (*str*) auth_url of the remote cloud

Returns
auth_url of remote cloud where a token can be validated or scoped.

Return type
str

_send_service_provider_ecp_authn_response(*session: Session*, *sp_url: str*, *sp_auth_url: str*) → Response

Present ECP wrapped SAML assertion to the keystone SP.

The assertion is issued by the keystone IdP and it is targeted to the keystone that will serve as Service Provider.

Parameters

- **session** a session object to send out HTTP requests.
- **sp_url** (*str*) URL where the ECP wrapped SAML assertion will be presented to the keystone SP. Usually, something like: https://sp.com/Shibboleth.sso/SAML2/ECP
- **sp_auth_url** (*str*) Federated authentication URL of the keystone SP. It is specified by IdP, for example: https://sp.com/v3/OS-FEDERATION/identity_providers/ idp_id/protocols/protocol_id/auth

auth_ref: ty.Optional[access.AccessInfo]

auth_url: str

get_unscoped_auth_ref(*session: Session*) → AccessInfoV3

Fetch unscoped federated token.

reauthenticate: bool

keystoneauth1.identity.v3.multi_factor module

class keystoneauth1.identity.v3.multi_factor.**MultiFactor**(*auth_url: str, auth_methods: list[str], *, unscoped: bool = False, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True, **kwargs: Any*)

Bases: `Auth`

A plugin for authenticating with multiple auth methods.

Parameters

- **auth_url** (*string*) Identity service endpoint for authentication.
- **auth_methods** (*string*) names of the methods to authenticate with.
- **trust_id** (*string*) Trust ID for trust scoping.
- **system_scope** (*string*) System information to scope to.
- **domain_id** (*string*) Domain ID for domain scoping.
- **domain_name** (*string*) Domain name for domain scoping.
- **project_id** (*string*) Project ID for project scoping.
- **project_name** (*string*) Project name for project scoping.
- **project_domain_id** (*string*) Projects domain ID for project.
- **project_domain_name** (*string*) Projects domain name for project.
- **reauthenticate** (*bool*) Allow fetching a new token if the current one is going to expire. (optional) default True

Also accepts various keyword args based on which methods are specified.

__abstractmethods__ = frozenset({})

__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}

`__doc__ = "A plugin for authenticating with multiple auth methods.\n\n :param string auth_url: Identity service endpoint for authentication.\n :param string auth_methods: names of the methods to authenticate with.\n :param string trust_id: Trust ID for trust scoping.\n :param string system_scope: System information to scope to.\n :param string domain_id: Domain ID for domain scoping.\n :param string domain_name: Domain name for domain scoping.\n :param string project_id: Project ID for project scoping.\n :param string project_name: Project name for project scoping.\n :param string project_domain_id: Project's domain ID for project.\n :param string project_domain_name: Project's domain name for project.\n :param bool reauthenticate: Allow fetching a new token if the current one\n is going to expire. (optional) default True\n\n Also accepts various keyword args based on which methods are specified.\n "`

__init__(*auth_url: str, auth_methods: list[str], *, unscoped: bool = False, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True, **kwargs: Any*)

`__module__ = 'keystoneauth1.identity.v3.multi_factor'`

`_abc_impl = <_abc._abc_data object>`

`_discovery_cache: dict[str, discover.Discover]`

`auth_ref: ty.Optional[access.AccessInfo]`

`auth_url: str`

`reauthenticate: bool`

keystoneauth1.identity.v3.oauth2_client_credential module

exception `keystoneauth1.identity.v3.oauth2_client_credential.`**ClientException**(*message: str | None = None*)

Bases: `Exception`

The base exception for everything to do with clients.

`__annotations__ = {}`

`__doc__ = 'The base exception for everything to do with clients.'`

__init__(*message: str | None = None*)

`__module__ = 'keystoneauth1.exceptions.base'`

__weakref__

list of weak references to the object (if defined)

message = 'ClientException'

class keystoneauth1.identity.v3.oauth2_client_credential.**OAuth2ClientCredential**(*auth_url: str, oauth2_endpoi str, oauth2_client_ str, oauth2_client_ str, *, trust_id: str | None = None, sys-tem_scope: str | None = None, do-main_id: str | None = None, do-main_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domai str | None = None, project_domai str |*

Bases: `Auth`

A plugin for authenticating via an OAuth2.0 client credential.

Parameters

- **auth_url** (*string*) Identity service endpoint for authentication.
- **oauth2_endpoint** (*string*) OAuth2.0 endpoint.
- **oauth2_client_id** (*string*) OAuth2.0 client credential id.
- **oauth2_client_secret** (*string*) OAuth2.0 client credential secret.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}`

`__doc__ = 'A plugin for authenticating via an OAuth2.0 client credential.\n\n :param string auth_url: Identity service endpoint for authentication.\n :param string oauth2_endpoint: OAuth2.0 endpoint.\n :param string oauth2_client_id: OAuth2.0 client credential id.\n :param string oauth2_client_secret: OAuth2.0 client credential secret.\n '`

`__init__`(*auth_url: str*, *oauth2_endpoint: str*, *oauth2_client_id: str*, *oauth2_client_secret: str*, **, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*) → None

`__module__ = 'keystoneauth1.identity.v3.oauth2_client_credential'`

`_abc_impl = <_abc._abc_data object>`

`_auth_method_class`

alias of `OAuth2ClientCredentialMethod`

`_discovery_cache: dict[str, discover.Discover]`

`auth_ref: ty.Optional[access.AccessInfo]`

`auth_url: str`

`get_headers`(*session: Session*) → dict[str, str] | None

Fetch authentication headers for message.

Parameters

session (*keystoneauth1.session.Session*) The session object that the auth_plugin belongs to.

Returns

Headers that are set to authenticate a message or None for failure. Note that when checking this value that the empty dict is a valid, non-failure response.

Return type
dict

reauthenticate: bool

class keystoneauth1.identity.v3.oauth2_client_credential.**OAuth2ClientCredentialMethod**(*, *oauth2_ str*, *oauth2_ str*, *oauth2_ str*)

Bases: AuthMethod

An auth method to fetch a token via an OAuth2.0 client credential.

Parameters

- **oauth2_endpoint** (*string*) OAuth2.0 endpoint.
- **oauth2_client_id** (*string*) OAuth2.0 client credential id.
- **oauth2_client_secret** (*string*) OAuth2.0 client credential secret.

__abstractmethods__ = frozenset({})

__annotations__ = {'_method_parameters': 'ty.Optional[list[str]]', 'oauth2_client_id': <class 'str'>, 'oauth2_client_secret': <class 'str'>, 'oauth2_endpoint': <class 'str'>}

__doc__ = 'An auth method to fetch a token via an OAuth2.0 client credential.\n\n :param string oauth2_endpoint: OAuth2.0 endpoint.\n :param string oauth2_client_id: OAuth2.0 client credential id.\n :param string oauth2_client_secret: OAuth2.0 client credential secret.\n '

__init__(*, *oauth2_endpoint: str*, *oauth2_client_id: str*, *oauth2_client_secret: str*) → None

__module__ = 'keystoneauth1.identity.v3.oauth2_client_credential'

_abc_impl = <_abc._abc_data object>

get_auth_data(*session: Session*, *auth: Auth*, *headers: dict[str, str]*, *request_kwargs: dict[str, object]*) → tuple[None, None] | tuple[str, Mapping[str, object]]

Return the authentication section of an auth plugin.

Parameters

- **session** (*keystoneauth1.session.Session*) The communication session.
- **auth** (*base.Auth*) The auth plugin calling the method.
- **headers** (*dict*) The headers that will be sent with the auth request if a plugin needs to add to them.

Returns
The identifier of this plugin and a dict of authentication data for the auth type.

Return type
tuple(string, dict)

`get_cache_id_elements()` → dict[str, str | None]

Get the elements for this auth method that make it unique.

These elements will be used as part of the `keystoneauth1.plugin.BaseIdentityPlugin.get_cache_id()` to allow caching of the auth plugin.

Plugins should override this if they want to allow caching of their state.

To avoid collision or overrides the keys of the returned dictionary should be prefixed with the plugin identifier. For example the password plugin returns its username value as password_username.

`oauth2_client_id:` `str`

`oauth2_client_secret:` `str`

`oauth2_endpoint:` `str`

keystoneauth1.identity.v3.oauth2_mtls_client_credential module

class keystoneauth1.identity.v3.oauth2_mtls_client_credential.**OAuth2mTlsClientCredential**(*aut*
str,
oau
str,
oau
str,
*,
tru
str
|
No
=
No
sys
tem
str
|
No
=
No
do-
ma
str
|
No
=
No
do-
ma
str
|
No
=
No
pro
str
|
No
=
No
pro
str
|
No
=
No
pro
str
|
No
=
No
pro
str
|
No
=

Bases: `BaseAuth`

A plugin for authenticating via an OAuth2.0 mTLS client credential.

Parameters

- **auth_url** (*string*) keystone authorization endpoint.
- **oauth2_endpoint** (*string*) OAuth2.0 endpoint.
- **oauth2_client_id** (*string*) OAuth2.0 client credential id.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}`

`__doc__ = 'A plugin for authenticating via an OAuth2.0 mTLS client credential.\n\n :param string auth_url: keystone authorization endpoint.\n :param string oauth2_endpoint: OAuth2.0 endpoint.\n :param string oauth2_client_id: OAuth2.0 client credential id.\n '`

`__init__`(*auth_url: str*, *oauth2_endpoint: str*, *oauth2_client_id: str*, ***, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*)

`__module__ = 'keystoneauth1.identity.v3.oauth2_mtls_client_credential'`

`_abc_impl = <_abc._abc_data object>`

`_discovery_cache: dict[str, discover.Discover]`

`auth_ref: ty.Optional[access.AccessInfo]`

`auth_url: str`

`get_auth_ref`(*session: Session*) → AccessInfoV3

Obtain a token from an OpenStack Identity Service.

This method is overridden by the various token version plugins.

This function should not be called independently and is expected to be invoked via the do_authenticate function.

This function will be invoked if the AcessInfo object cached by the plugin is not valid. Thus plugins should always fetch a new AccessInfo when invoked. If you are looking to just retrieve the current auth data then you should use get_access.

Parameters

session (*keystoneauth1.session.Session*) A session object that can be used for communication.

Raises

- **keystoneauth1.exceptions.response.InvalidResponse** The response returned wasnt appropriate.
- **keystoneauth1.exceptions.http.HttpError** An error from an invalid HTTP response.
- **keystoneauth1.exceptions.ClientException** An error from getting OAuth2.0 access token.

Returns
Token access information.

Return type
`keystoneauth1.access.AccessInfo`

get_headers(*session: Session*) → dict[str, str] | None

Fetch authentication headers for message.

Parameters
session (*keystoneauth1.session.Session*) The session object that the auth_plugin belongs to.

Returns
Headers that are set to authenticate a message or None for failure. Note that when checking this value that the empty dict is a valid, non-failure response.

Return type
dict

reauthenticate: bool

keystoneauth1.identity.v3.oidc module

class keystoneauth1.identity.v3.oidc.**OidcAccessToken**(*auth_url: str, identity_provider: str, protocol: str, access_token_type: str = 'access_token', scope: str = 'openid profile', access_token_endpoint: str | None = None, discovery_endpoint: str | None = None, access_token: str | None = None, *, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

Bases: `_OidcBase`

Implementation for OpenID Connect access token reuse.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', '_discovery_document': 'dict[str, object]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'grant_type': 'ty.ClassVar[str]', 'reauthenticate': 'bool'}`

`__doc__ = 'Implementation for OpenID Connect access token reuse.'`

`__init__`(*auth_url: str, identity_provider: str, protocol: str, access_token_type: str = 'access_token', scope: str = 'openid profile', access_token_endpoint: str | None = None, discovery_endpoint: str | None = None, access_token: str | None = None, *, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

The OpenID Connect plugin based on the Access Token.

It expects the following:

Parameters

- **`auth_url`** (*string*) URL of the Identity Service
- **`identity_provider`** (*string*) Name of the Identity Provider the client will authenticate against
- **`protocol`** (*string*) Protocol name as configured in keystone
- **`access_token`** (*string*) OpenID Connect Access token

`__module__ = 'keystoneauth1.identity.v3.oidc'`

`_abc_impl = <_abc._abc_data object>`

`_discovery_cache: dict[str, discover.Discover]`

`_discovery_document: dict[str, object]`

`auth_ref: ty.Optional[access.AccessInfo]`

`auth_url: str`

`get_payload`(*session: Session*) → dict[str, str | None]

OidcAccessToken does not require a payload.

`get_unscoped_auth_ref`(*session: Session*) → AccessInfoV3

Authenticate with OpenID Connect and get back claims.

We exchange the access token upon accessing the protected Keystone endpoint (federated auth URL). This will trigger the OpenID Connect Provider to perform a user introspection and retrieve information (specified in the scope) about the user in the form of an OpenID Connect Claim. These claims will be sent to Keystone in the form of environment variables.

Parameters
`session` (*keystoneauth1.session.Session*) a session object to send out HTTP requests.

Returns
a token data representation

Return type
`keystoneauth1.access.AccessInfoV3`

grant_type: ty.ClassVar[str]

reauthenticate: bool

class keystoneauth1.identity.v3.oidc.**OidcAuthorizationCode**(*auth_url: str, identity_provider: str, protocol: str, client_id: str, client_secret: str, access_token_type: str = 'access_token', scope: str = 'openid profile', access_token_endpoint: str | None = None, discovery_endpoint: str | None = None, code: str | None = None, *, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True, redirect_uri: str | None = None*)

Bases: `_OidcBase`

Implementation for OpenID Connect Authorization Code.

__abstractmethods__ = frozenset({})

__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', '_discovery_document': 'dict[str, object]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'grant_type': 'ty.ClassVar[str]', 'reauthenticate': 'bool'}

`__doc__ = 'Implementation for OpenID Connect Authorization Code.'`

`__init__`(*auth_url: str*, *identity_provider: str*, *protocol: str*, *client_id: str*, *client_secret: str*, *access_token_type: str = 'access_token'*, *scope: str = 'openid profile'*, *access_token_endpoint: str | None = None*, *discovery_endpoint: str | None = None*, *code: str | None = None*, ***, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*, *redirect_uri: str | None = None*)

The OpenID Authorization Code plugin expects the following.

Parameters

- **redirect_uri** (*string*) OpenID Connect Client Redirect URL
- **code** (*string*) OAuth 2.0 Authorization Code

`__module__ = 'keystoneauth1.identity.v3.oidc'`

`_abc_impl = <_abc._abc_data object>`

`_discovery_cache: dict[str, discover.Discover]`

`_discovery_document: dict[str, object]`

`auth_ref: ty.Optional[access.AccessInfo]`

`auth_url: str`

`get_payload`(*session: Session*) → dict[str, str | None]

Get an authorization grant for the authorization_code grant type.

Parameters

session (*keystoneauth1.session.Session*) a session object to send out HTTP requests.

Returns

a python dictionary containing the payload to be exchanged

Return type

dict

`grant_type: ty.ClassVar[str] = 'authorization_code'`

`reauthenticate: bool`

class keystoneauth1.identity.v3.oidc.**OidcClientCredentials**(*auth_url: str, identity_provider: str, protocol: str, client_id: str, client_secret: str, access_token_type: str = 'access_token', scope: str = 'openid profile', access_token_endpoint: str | None = None, discovery_endpoint: str | None = None, *, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

Bases: `_OidcBase`

Implementation for OpenID Connect Client Credentials.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', '_discovery_document': 'dict[str, object]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'grant_type': 'ty.ClassVar[str]', 'reauthenticate': 'bool'}`

`__doc__ = 'Implementation for OpenID Connect Client Credentials.'`

`__init__`(*auth_url: str, identity_provider: str, protocol: str, client_id: str, client_secret: str, access_token_type: str = 'access_token', scope: str = 'openid profile', access_token_endpoint: str | None = None, discovery_endpoint: str | None = None, *, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

The OpenID Client Credentials expects the following.

Parameters

- **`client_id`** Client ID used to authenticate

- `client_secret` Client Secret used to authenticate

`__module__ = 'keystoneauth1.identity.v3.oidc'`

`_abc_impl = <_abc._abc_data object>`

`_discovery_cache: dict[str, discover.Discover]`

`_discovery_document: dict[str, object]`

`auth_ref: ty.Optional[access.AccessInfo]`

`auth_url: str`

`get_payload`(*session: Session*) → dict[str, str | None]

Get an authorization grant for the client credentials grant type.

Parameters
`session` (*keystoneauth1.session.Session*) a session object to send out HTTP requests.

Returns
a python dictionary containing the payload to be exchanged

Return type
dict

`grant_type: ty.ClassVar[str] = 'client_credentials'`

`reauthenticate: bool`

class keystoneauth1.identity.v3.oidc.**OidcDeviceAuthorization**(*auth_url: str, identity_provider: str, protocol: str, client_id: str, client_secret: str | None, scope: str = 'openid profile', access_token_endpoint: str | None = None, discovery_endpoint: str | None = None, device_authorization_endpoint: str | None = None, code_challenge: str | None = None, code_challenge_method: str | None = None, *, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

Bases: `_OidcBase`

Implementation for OAuth 2.0 Device Authorization Grant.

HEADER_X_FORM = {'Content-Type': 'application/x-www-form-urlencoded'}

__abstractmethods__ = frozenset({})

__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', '_discovery_document': 'dict[str, object]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'grant_type': 'ty.ClassVar[str]', 'reauthenticate': 'bool'}

__doc__ = 'Implementation for OAuth 2.0 Device Authorization Grant.'

`__init__`(*auth_url: str, identity_provider: str, protocol: str, client_id: str, client_secret: str | None, scope: str = 'openid profile', access_token_endpoint: str | None = None, discovery_endpoint: str | None = None, device_authorization_endpoint: str | None = None, code_challenge: str | None = None, code_challenge_method: str | None = None, *, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

The OAuth 2.0 Device Authorization plugin expects the following.

Parameters

- **device_authorization_endpoint** (*string*) OAuth 2.0 Device Authorization Endpoint, for example: https://localhost:8020/oidc/authorize/device Note that if a discovery document is provided this value will override the discovered one.
- **code_challenge_method** (*string*) PKCE Challenge Method (RFC 7636).

`__module__ = 'keystoneauth1.identity.v3.oidc'`

`_abc_impl = <_abc._abc_data object>`

`_generate_pkce_challenge`() → str | None

Generate PKCE challenge string as defined in RFC 7636.

`_generate_pkce_verifier`() → str

Generate PKCE verifier string as defined in RFC 7636.

`_get_access_token`(*session: Session, payload: dict[str, str | None]*) → str

Poll token endpoint for an access token.

Parameters

- **session** (*keystoneauth1.session.Session*) a session object to send out HTTP requests.
- **payload** (*dict*) a dict containing various OpenID Connect values, for example:

```
{
    'grant_type': 'urn:ietf:params:oauth:grant-
↪type:device_code',
    'device_code': self.device_code,
}
```

`_get_device_authorization_endpoint`(*session: Session*) → str | None

Get the endpoint for the OAuth 2.0 Device Authorization flow.

This method will return the correct device authorization endpoint to be used. If the user has explicitly passed an device_authorization_endpoint to the constructor that will be returned. If there is no explicit endpoint and a discovery url is provided, it will try to get it from the discovery document. If nothing is found, an exception will be raised.

Parameters
: **session** (*keystoneauth1.session.Session*) a session object to send out HTTP requests.

Returns
: the endpoint to use

Return type
: string or None if no endpoint is found

get_payload(*session: Session*) → dict[str, str | None]

Get an authorization grant for the device_code grant type.

Parameters
: **session** (*keystoneauth1.session.Session*) a session object to send out HTTP requests.

Returns
: a python dictionary containing the payload to be exchanged

Return type
: dict

grant_type: ty.ClassVar[str] = 'urn:ietf:params:oauth:grant-type:device_code'

class keystoneauth1.identity.v3.oidc.**OidcPassword**(*auth_url: str, identity_provider: str, protocol: str, client_id: str, client_secret: str, access_token_type: str = 'access_token', scope: str = 'openid profile', access_token_endpoint: str | None = None, discovery_endpoint: str | None = None, username: str | None = None, password: str | None = None, idp_otp_key: str | None = None, *, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

Bases: _OidcBase

Implementation for OpenID Connect Resource Owner Password Credential.

__abstractmethods__ = frozenset({})

__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', '_discovery_document': 'dict[str, object]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'grant_type': 'ty.ClassVar[str]', 'reauthenticate': 'bool'}

`__doc__ = 'Implementation for OpenID Connect Resource Owner Password Credential.'`

`__init__`(*auth_url: str, identity_provider: str, protocol: str, client_id: str, client_secret: str, access_token_type: str = 'access_token', scope: str = 'openid profile', access_token_endpoint: str | None = None, discovery_endpoint: str | None = None, username: str | None = None, password: str | None = None, idp_otp_key: str | None = None, *, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

The OpenID Password plugin expects the following.

Parameters

- **`username`** (*`string`*) Username used to authenticate
- **`password`** (*`string`*) Password used to authenticate

`__module__ = 'keystoneauth1.identity.v3.oidc'`

`_abc_impl = <_abc._abc_data object>`

`_discovery_cache: dict[str, discover.Discover]`

`_discovery_document: dict[str, object]`

`auth_ref: ty.Optional[access.AccessInfo]`

`auth_url: str`

`get_payload`(*session: Session*) → dict[str, str | None]

Get an authorization grant for the password grant type.

Parameters

`session` (*`keystoneauth1.session.Session`*) a session object to send out HTTP requests.

Returns

a python dictionary containing the payload to be exchanged

Return type

dict

`grant_type: ty.ClassVar[str] = 'password'`

`manage_otp_from_session_or_request_to_the_user`(*payload: dict[str, str | None], session: Session*) → None

Get the OTP code from the session or else request to the user.

When the OS_IDP_OTP_KEY environment variable is set, this method will verify if there is an OTP value in the current session, if it exists, we use it (the OTP from session) to send to the Identity Provider when retrieving the access token. If there is no OTP in the current session, we ask the user to enter it (the OTP), and we add it to the session to execute the authentication flow.

The OTP is being stored in the session because in some flows, the CLI is doing the authentication process two times, so saving the OTP in the session, allow us to use the same OTP in a short time interval, avoiding to request it to the user twice in a row.

Parameters

- `payload`
- `session`

Returns

reauthenticate: `bool`

class keystoneauth1.identity.v3.oidc._OidcBase(*auth_url: str, identity_provider: str, protocol: str, client_id: str, client_secret: str | None, access_token_type: str, scope: str, access_token_endpoint: str | None, discovery_endpoint: str | None, *, trust_id: str | None, system_scope: str | None, domain_id: str | None, domain_name: str | None, project_id: str | None, project_name: str | None, project_domain_id: str | None, project_domain_name: str | None, reauthenticate: bool, include_catalog: bool*)

Bases: `FederationBaseAuth`

Base class for different OpenID Connect based flows.

The OpenID Connect specification can be found at:: `http://openid.net/specs/openid-connect-core-1_0.html`

`__abstractmethods__ = frozenset({'get_payload'})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', '_discovery_document': 'dict[str, object]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'grant_type': typing.ClassVar[str], 'reauthenticate': 'bool'}`

`__doc__ = 'Base class for different OpenID Connect based flows.\n\n The OpenID Connect specification can be found at::\n ``http://openid.net/specs/openid-connect-core-1_0.html``\n '`

`__init__`(*auth_url: str, identity_provider: str, protocol: str, client_id: str, client_secret: str | None, access_token_type: str, scope: str, access_token_endpoint: str | None, discovery_endpoint: str | None, *, trust_id: str | None, system_scope: str | None, domain_id: str | None, domain_name: str | None, project_id: str | None, project_name: str | None, project_domain_id: str | None, project_domain_name: str | None, reauthenticate: bool, include_catalog: bool*)

The OpenID Connect plugin expects the following.

Parameters

- **auth_url** (*string*) URL of the Identity Service

- **identity_provider** (*string*) Name of the Identity Provider the client will authenticate against
- **protocol** (*string*) Protocol name as configured in keystone
- **client_id** (*string*) OAuth 2.0 Client ID
- **client_secret** (*string*) OAuth 2.0 Client Secret
- **access_token_type** (*string*) OAuth 2.0 Authorization Server Introspection token type, it is used to decide which type of token will be used when processing token introspection. Valid values are: access_token or id_token
- **access_token_endpoint** (*string*) OpenID Connect Provider Token Endpoint, for example: https://localhost:8020/oidc/OP/token Note that if a discovery document is provided this value will override the discovered one.
- **discovery_endpoint** OpenID Connect Discovery Document URL, for example: https://localhost:8020/oidc/.well-known/openid-configuration
- **scope** (*string*) OpenID Connect scope that is requested from OP, for example: openid profile email, defaults to openid profile. Note that OpenID Connect specification states that openid must be always specified.

__module__ = 'keystoneauth1.identity.v3.oidc'

_abc_impl = <_abc._abc_data object>

_get_access_token(*session: Session*, *payload: dict[str, str | None]*) → str

Exchange a variety of user supplied values for an access token.

Parameters

- **session** (*keystoneauth1.session.Session*) a session object to send out HTTP requests.
- **payload** (*dict*) a dict containing various OpenID Connect values, for example:

```
{
    'grant_type': 'password',
    'username': self.username,
    'password': self.password,
    'scope': self.scope,
}
```

_get_access_token_endpoint(*session: Session*) → str

Get the token_endpoint for the OpenID Connect flow.

This method will return the correct access token endpoint to be used. If the user has explicitly passed an access_token_endpoint to the constructor that will be returned. If there is no explicit endpoint and a discovery url is provided, it will try to get it from the discovery document. If nothing is found, an exception will be raised.

Parameters

session (*keystoneauth1.session.Session*) a session object to send out HTTP requests.

Returns
the endpoint to use

Return type
string

`_get_discovery_document`(*session: Session*) → dict[str, object]

Get the contents of the OpenID Connect Discovery Document.

This method grabs the contents of the OpenID Connect Discovery Document if a discovery_endpoint was passed to the constructor and returns it as a dict, otherwise returns an empty dict. Note that it will fetch the discovery document only once, so subsequent calls to this method will return the cached result, if any.

Parameters
session (*keystoneauth1.session.Session*) a session object to send out HTTP requests.

Returns
a python dictionary containing the discovery document if any, otherwise it will return an empty dict.

Return type
dict

`_get_keystone_token`(*session: Session*, *access_token: str*) → Response

Exchange an access token for a keystone token.

By Sending the access token in an *Authorization: Bearer* header, to an OpenID Connect protected endpoint (Federated Token URL). The OpenID Connect server will use the access token to look up information about the authenticated user (this technique is called instrospection). The output of the instrospection will be an OpenID Connect Claim, that will be used against the mapping engine. Should the mapping engine succeed, a Keystone token will be presented to the user.

Parameters

- **session** (*keystoneauth1.session.Session*) a session object to send out HTTP requests.
- **access_token** (*str*) The OpenID Connect access token.

`_sanitize`(*data: dict[str, str | None]*) → dict[str, str | None]

`abstract` **`get_payload`**(*session: Session*) → dict[str, str | None]

Get the plugin specific payload for obtainin an access token.

OpenID Connect supports different grant types. This method should prepare the payload that needs to be exchanged with the server in order to get an access token for the particular grant type that the plugin is implementing.

Parameters
session (*keystoneauth1.session.Session*) a session object to send out HTTP requests.

Returns
a python dictionary containing the payload to be exchanged

Return type
dict

get_unscoped_auth_ref(*session: Session*) → AccessInfoV3

Authenticate with OpenID Connect and get back claims.

This is a multi-step process:

1.- An access token must be retrieved from the server. In order to do
so, we need to exchange an authorization grant or refresh token with the token endpoint in order to obtain an access token. The authorization grant varies from plugin to plugin.

2.- We then exchange the access token upon accessing the protected
Keystone endpoint (federated auth URL). This will trigger the OpenID Connect Provider to perform a user introspection and retrieve information (specified in the scope) about the user in the form of an OpenID Connect Claim. These claims will be sent to Keystone in the form of environment variables.

Parameters
session (*keystoneauth1.session.Session*) a session object to send out HTTP requests.

Returns
a token data representation

Return type
`keystoneauth1.access.AccessInfoV3`

grant_type: ClassVar[str]

keystoneauth1.identity.v3.password module

class keystoneauth1.identity.v3.password.**Password**(*auth_url: str, password: str, user_id: str | None = None, username: str | None = None, user_domain_id: str | None = None, user_domain_name: str | None = None, *, unscoped: bool = False, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

Bases: Auth

A plugin for authenticating with a username and password.

Parameters

- **auth_url** (*string*) Identity service endpoint for authentication.
- **password** (*string*) Password for authentication.

- **user_id** (*string*) User ID for authentication.
- **username** (*string*) Username for authentication.
- **user_domain_id** (*string*) Users domain ID for authentication.
- **user_domain_name** (*string*) Users domain name for authentication.
- **trust_id** (*string*) Trust ID for trust scoping.
- **system_scope** (*string*) System information to scope to.
- **domain_id** (*string*) Domain ID for domain scoping.
- **domain_name** (*string*) Domain name for domain scoping.
- **project_id** (*string*) Project ID for project scoping.
- **project_name** (*string*) Project name for project scoping.
- **project_domain_id** (*string*) Projects domain ID for project.
- **project_domain_name** (*string*) Projects domain name for project.
- **reauthenticate** (*bool*) Allow fetching a new token if the current one is going to expire. (optional) default True

__abstractmethods__ = frozenset({})

__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}

__doc__ = "A plugin for authenticating with a username and password.\n\n :param string auth_url: Identity service endpoint for authentication.\n :param string password: Password for authentication.\n :param string user_id: User ID for authentication.\n :param string username: Username for authentication.\n :param string user_domain_id: User's domain ID for authentication.\n :param string user_domain_name: User's domain name for authentication.\n :param string trust_id: Trust ID for trust scoping.\n :param string system_scope: System information to scope to.\n :param string domain_id: Domain ID for domain scoping.\n :param string domain_name: Domain name for domain scoping.\n :param string project_id: Project ID for project scoping.\n :param string project_name: Project name for project scoping.\n :param string project_domain_id: Project's domain ID for project.\n :param string project_domain_name: Project's domain name for project.\n :param bool reauthenticate: Allow fetching a new token if the current one\n is going to expire. (optional) default True\n "

__init__(*auth_url: str*, *password: str*, *user_id: str | None = None*, *username: str | None = None*, *user_domain_id: str | None = None*, *user_domain_name: str | None = None*, ***, *unscoped: bool = False*, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*) → None

__module__ = 'keystoneauth1.identity.v3.password'

_abc_impl = <_abc._abc_data object>

_auth_method_class

alias of `PasswordMethod`

_discovery_cache: dict[str, discover.Discover]

auth_ref: ty.Optional[access.AccessInfo]

auth_url: str

reauthenticate: bool

class `keystoneauth1.identity.v3.password.`**PasswordMethod**(**, password: str, user_id: str | None = None, username: str | None = None, user_domain_id: str | None = None, user_domain_name: str | None = None*)

Bases: `AuthMethod`

Construct a User/Password based authentication method.

Parameters

- **password** (*string*) Password for authentication.
- **username** (*string*) Username for authentication.
- **user_id** (*string*) User ID for authentication.
- **user_domain_id** (*string*) Users domain ID for authentication.
- **user_domain_name** (*string*) Users domain name for authentication.

__abstractmethods__ = frozenset({})

__annotations__ = {'_method_parameters': 'ty.Optional[list[str]]', 'password': <class 'str'>, 'user_domain_id': typing.Optional[str], 'user_domain_name': typing.Optional[str], 'user_id': typing.Optional[str], 'username': typing.Optional[str]}

__doc__ = "Construct a User/Password based authentication method.\n\n :param string password: Password for authentication.\n :param string username: Username for authentication.\n :param string user_id: User ID for authentication.\n :param string user_domain_id: User's domain ID for authentication.\n :param string user_domain_name: User's domain name for authentication.\n "

__init__(**, password: str, user_id: str | None = None, username: str | None = None, user_domain_id: str | None = None, user_domain_name: str | None = None*) → None

__module__ = 'keystoneauth1.identity.v3.password'

_abc_impl = <_abc._abc_data object>

get_auth_data(*session: Session*, *auth: Auth*, *headers: dict[str, str]*, *request_kwargs: dict[str, object]*) → tuple[None, None] | tuple[str, Mapping[str, object]]

Return the authentication section of an auth plugin.

Parameters

- **session** (*keystoneauth1.session.Session*) The communication session.
- **auth** (*base.Auth*) The auth plugin calling the method.
- **headers** (*dict*) The headers that will be sent with the auth request if a plugin needs to add to them.

Returns

The identifier of this plugin and a dict of authentication data for the auth type.

Return type

tuple(string, dict)

get_cache_id_elements() → dict[str, str | None]

Get the elements for this auth method that make it unique.

These elements will be used as part of the `keystoneauth1.plugin.BaseIdentityPlugin.get_cache_id()` to allow caching of the auth plugin.

Plugins should override this if they want to allow caching of their state.

To avoid collision or overrides the keys of the returned dictionary should be prefixed with the plugin identifier. For example the password plugin returns its username value as password_username.

password: str

user_domain_id: str | None = None

user_domain_name: str | None = None

user_id: str | None = None

username: str | None = None

keystoneauth1.identity.v3.receipt module

class keystoneauth1.identity.v3.receipt.**ReceiptMethod**(**, receipt: str*)

Bases: `AuthMethod`

Construct an Auth plugin to continue authentication with a receipt.

Parameters

receipt (*string*) Receipt for authentication.

__abstractmethods__ = frozenset({})

__annotations__ = {'_method_parameters': 'ty.Optional[list[str]]', 'receipt': <class 'str'>}

`__doc__ = 'Construct an Auth plugin to continue authentication with a receipt.\n\n :param string receipt:  Receipt for authentication.\n '`

`__init__`(*, *receipt: str*) → None

`__module__ = 'keystoneauth1.identity.v3.receipt'`

`_abc_impl = <_abc._abc_data object>`

`get_auth_data`(*session: Session*, *auth: Auth*, *headers: dict[str, str]*, *request_kwargs: dict[str, object]*) → tuple[None, None] | tuple[str, Mapping[str, object]]

Add the auth receipt to the headers.

We explicitly return None to avoid being added to the request methods, or body.

`get_cache_id_elements`() → dict[str, str | None]

Get the elements for this auth method that make it unique.

These elements will be used as part of the `keystoneauth1.plugin.BaseIdentityPlugin.get_cache_id()` to allow caching of the auth plugin.

Plugins should override this if they want to allow caching of their state.

To avoid collision or overrides the keys of the returned dictionary should be prefixed with the plugin identifier. For example the password plugin returns its username value as password_username.

`receipt:` `str`

keystoneauth1.identity.v3.token module

class `keystoneauth1.identity.v3.token.`**`Token`**(*auth_url: str*, *token: str*, *, *unscoped: bool = False*, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*)

Bases: `Auth`

A plugin for authenticating with an existing Token.

Parameters

- **auth_url** (*string*) Identity service endpoint for authentication.
- **token** (*string*) Token for authentication.
- **trust_id** (*string*) Trust ID for trust scoping.
- **domain_id** (*string*) Domain ID for domain scoping.
- **domain_name** (*string*) Domain name for domain scoping.
- **project_id** (*string*) Project ID for project scoping.

- **project_name** (*string*) Project name for project scoping.
- **project_domain_id** (*string*) Projects domain ID for project.
- **project_domain_name** (*string*) Projects domain name for project.
- **reauthenticate** (*bool*) Allow fetching a new token if the current one is going to expire. (optional) default True

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}`

`__doc__ = "A plugin for authenticating with an existing Token.\n\n :param string auth_url: Identity service endpoint for authentication.\n :param string token: Token for authentication.\n :param string trust_id: Trust ID for trust scoping.\n :param string domain_id: Domain ID for domain scoping.\n :param string domain_name: Domain name for domain scoping.\n :param string project_id: Project ID for project scoping.\n :param string project_name: Project name for project scoping.\n :param string project_domain_id: Project's domain ID for project.\n :param string project_domain_name: Project's domain name for project.\n :param bool reauthenticate: Allow fetching a new token if the current one\n is going to expire. (optional) default True\n "`

`__init__`(*auth_url: str*, *token: str*, ***, *unscoped: bool = False*, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*) → None

`__module__ = 'keystoneauth1.identity.v3.token'`

`_abc_impl = <_abc._abc_data object>`

`_auth_method_class`

alias of `TokenMethod`

`_discovery_cache: dict[str, discover.Discover]`

`auth_ref: ty.Optional[access.AccessInfo]`

`auth_url: str`

`reauthenticate: bool`

class `keystoneauth1.identity.v3.token.`**`TokenMethod`**(***, *token: str*)

Bases: `AuthMethod`

Construct an Auth plugin to fetch a token from a token.

Parameters

token (*string*) Token for authentication.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'_method_parameters': 'ty.Optional[list[str]]', 'token': <class 'str'>}`

`__doc__ = 'Construct an Auth plugin to fetch a token from a token.\n\n :param string token: Token for authentication.\n '`

`__init__`(**, token: str*) → None

`__module__ = 'keystoneauth1.identity.v3.token'`

`_abc_impl = <_abc._abc_data object>`

`get_auth_data`(*session: Session, auth: Auth, headers: dict[str, str], request_kwargs: dict[str, object]*) → tuple[None, None] | tuple[str, Mapping[str, object]]

Return the authentication section of an auth plugin.

Parameters

- **session** (*keystoneauth1.session.Session*) The communication session.
- **auth** (*base.Auth*) The auth plugin calling the method.
- **headers** (*dict*) The headers that will be sent with the auth request if a plugin needs to add to them.

Returns

The identifier of this plugin and a dict of authentication data for the auth type.

Return type

tuple(string, dict)

`get_cache_id_elements`() → dict[str, str | None]

Get the elements for this auth method that make it unique.

These elements will be used as part of the `keystoneauth1.plugin.BaseIdentityPlugin.get_cache_id()` to allow caching of the auth plugin.

Plugins should override this if they want to allow caching of their state.

To avoid collision or overrides the keys of the returned dictionary should be prefixed with the plugin identifier. For example the password plugin returns its username value as password_username.

`token: str`

keystoneauth1.identity.v3.tokenless_auth module

class keystoneauth1.identity.v3.tokenless_auth.**TokenlessAuth**(*auth_url: str, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None*)

Bases: `BaseAuthPlugin`

A plugin for authenticating with Tokenless Auth.

This is for Tokenless Authentication. Scoped information like domain name and project ID will be passed in the headers and token validation request will be authenticated based on the provided HTTPS certificate along with the scope information.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'_discovery_cache': 'dict[str, discover.Discover]'}`

`__doc__ = 'A plugin for authenticating with Tokenless Auth.\n\n This is for Tokenless Authentication. Scoped information\n like domain name and project ID will be passed in the headers and\n token validation request will be authenticated based on\n the provided HTTPS certificate along with the scope information.\n '`

`__init__`(*auth_url: str, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None*)

A init method for TokenlessAuth.

Parameters

- **auth_url** (*string*) Identity service endpoint for authentication. The URL must include a version or any request will result in a 404 NotFound error.
- **domain_id** (*string*) Domain ID for domain scoping.
- **domain_name** (*string*) Domain name for domain scoping.
- **project_id** (*string*) Project ID for project scoping.
- **project_name** (*string*) Project name for project scoping.
- **project_domain_id** (*string*) Projects domain ID for project.
- **project_domain_name** (*string*) Projects domain name for project.

`__module__ = 'keystoneauth1.identity.v3.tokenless_auth'`

`_abc_impl = <_abc._abc_data object>`

`_discovery_cache: dict[str, discover.Discover]`

`get_endpoint`(*session: ks_session.Session*, *service_type: str | None = None*, ***kwargs: Any*) → str | None

Return a valid endpoint for a service.

Parameters

- **session** (*keystoneauth1.session.Session*) A session object that can be used for communication.
- **service_type** (*string*) The type of service to lookup the endpoint for. This plugin will return None (failure) if service_type is not provided.

Returns
A valid endpoint URL or None if not available.

Return type
string or None

`get_headers`(*session: ks_session.Session*) → dict[str, str] | None

Fetch authentication headers for message.

This is to override the default get_headers method to provide tokenless auth scope headers if token is not provided in the session.

Parameters
session (*keystoneauth1.session.Session*) The session object that the auth_plugin belongs to.

Returns
Headers that are set to authenticate a message or None for failure. Note that when checking this value that the empty dict is a valid, non-failure response.

Return type
dict

keystoneauth1.identity.v3.totp module

class `keystoneauth1.identity.v3.totp.TOTP`(*auth_url: str*, *passcode: str*, *user_id: str | None = None*, *username: str | None = None*, *user_domain_id: str | None = None*, *user_domain_name: str | None = None*, ***, *unscoped: bool = False*, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*)

Bases: `Auth`

A plugin for authenticating with a username and TOTP passcode.

Parameters

- **auth_url** (*string*) Identity service endpoint for authentication.

- **passcode** (*string*) TOTP passcode for authentication.
- **user_id** (*string*) User ID for authentication.
- **username** (*string*) Username for authentication.
- **user_domain_id** (*string*) Users domain ID for authentication.
- **user_domain_name** (*string*) Users domain name for authentication.
- **trust_id** (*string*) Trust ID for trust scoping.
- **domain_id** (*string*) Domain ID for domain scoping.
- **domain_name** (*string*) Domain name for domain scoping.
- **project_id** (*string*) Project ID for project scoping.
- **project_name** (*string*) Project name for project scoping.
- **project_domain_id** (*string*) Projects domain ID for project.
- **project_domain_name** (*string*) Projects domain name for project.
- **reauthenticate** (*bool*) Allow fetching a new token if the current one is going to expire. (optional) default True

__abstractmethods__ = frozenset({})

__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}

__doc__ = "A plugin for authenticating with a username and TOTP passcode.\n\n :param string auth_url: Identity service endpoint for authentication.\n :param string passcode: TOTP passcode for authentication.\n :param string user_id: User ID for authentication.\n :param string username: Username for authentication.\n :param string user_domain_id: User's domain ID for authentication.\n :param string user_domain_name: User's domain name for authentication.\n :param string trust_id: Trust ID for trust scoping.\n :param string domain_id: Domain ID for domain scoping.\n :param string domain_name: Domain name for domain scoping.\n :param string project_id: Project ID for project scoping.\n :param string project_name: Project name for project scoping.\n :param string project_domain_id: Project's domain ID for project.\n :param string project_domain_name: Project's domain name for project.\n :param bool reauthenticate: Allow fetching a new token if the current one\n is going to expire. (optional) default True\n "

__init__(*auth_url: str*, *passcode: str*, *user_id: str | None = None*, *username: str | None = None*, *user_domain_id: str | None = None*, *user_domain_name: str | None = None*, ***, *unscoped: bool = False*, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*) → None

`__module__ = 'keystoneauth1.identity.v3.totp'`

`_abc_impl = <_abc._abc_data object>`

`_auth_method_class`

alias of `TOTPMethod`

`_discovery_cache: dict[str, discover.Discover]`

`auth_ref: ty.Optional[access.AccessInfo]`

`auth_url: str`

`reauthenticate: bool`

class `keystoneauth1.identity.v3.totp.`**`TOTPMethod`**(**, passcode: str, user_id: str | None = None, username: str | None = None, user_domain_id: str | None = None, user_domain_name: str | None = None*)

Bases: `AuthMethod`

Construct a User/Passcode based authentication method.

Parameters

- **passcode** (*string*) TOTP passcode for authentication.
- **username** (*string*) Username for authentication.
- **user_id** (*string*) User ID for authentication.
- **user_domain_id** (*string*) Users domain ID for authentication.
- **user_domain_name** (*string*) Users domain name for authentication.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'_method_parameters': 'ty.Optional[list[str]]', 'passcode': <class 'str'>, 'user_domain_id': typing.Optional[str], 'user_domain_name': typing.Optional[str], 'user_id': typing.Optional[str], 'username': typing.Optional[str]}`

`__doc__ = "Construct a User/Passcode based authentication method.\n\n :param string passcode: TOTP passcode for authentication.\n :param string username: Username for authentication.\n :param string user_id: User ID for authentication.\n :param string user_domain_id: User's domain ID for authentication.\n :param string user_domain_name: User's domain name for authentication.\n "`

`__init__`(**, passcode: str, user_id: str | None = None, username: str | None = None, user_domain_id: str | None = None, user_domain_name: str | None = None*) → None

`__module__ = 'keystoneauth1.identity.v3.totp'`

`_abc_impl = <_abc._abc_data object>`

get_auth_data(*session: Session*, *auth: Auth*, *headers: dict[str, str]*, *request_kwargs: dict[str, object]*) → tuple[None, None] | tuple[str, Mapping[str, object]]

Return the authentication section of an auth plugin.

Parameters

- **session** (*keystoneauth1.session.Session*) The communication session.
- **auth** (*base.Auth*) The auth plugin calling the method.
- **headers** (*dict*) The headers that will be sent with the auth request if a plugin needs to add to them.

Returns
The identifier of this plugin and a dict of authentication data for the auth type.

Return type
tuple(string, dict)

get_cache_id_elements() → dict[str, str | None]

Get the elements for this auth method that make it unique.

These elements will be used as part of the `keystoneauth1.plugin.BaseIdentityPlugin.get_cache_id()` to allow caching of the auth plugin.

Plugins should override this if they want to allow caching of their state.

To avoid collision or overrides the keys of the returned dictionary should be prefixed with the plugin identifier. For example the password plugin returns its username value as password_username.

passcode: str

user_domain_id: str | None = None

user_domain_name: str | None = None

user_id: str | None = None

username: str | None = None

Module contents

class keystoneauth1.identity.v3.**ApplicationCredential**(*auth_url: str, application_credential_secret: str, application_credential_id: str | None = None, application_credential_name: str | None = None, user_id: str | None = None, username: str | None = None, user_domain_id: str | None = None, user_domain_name: str | None = None, *, unscoped: bool = False, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

Bases: `Auth`

A plugin for authenticating with an application credential.

Parameters

- **auth_url** (*string*) Identity service endpoint for authentication.
- **application_credential_secret** (*string*) Application credential secret.
- **application_credential_id** (*string*) Application credential ID.
- **application_credential_name** (*string*) Application credential name.
- **username** (*string*) Username for authentication.
- **user_id** (*string*) User ID for authentication.
- **user_domain_id** (*string*) Users domain ID for authentication.
- **user_domain_name** (*string*) Users domain name for authentication.
- **reauthenticate** (*bool*) Allow fetching a new token if the current one is going to expire. (optional) default True

__abstractmethods__ = frozenset({})

__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}

`__doc__ = "A plugin for authenticating with an application credential.\n\n :param string auth_url: Identity service endpoint for authentication.\n :param string application_credential_secret: Application credential secret.\n :param string application_credential_id: Application credential ID.\n :param string application_credential_name: Application credential name.\n :param string username: Username for authentication.\n :param string user_id: User ID for authentication.\n :param string user_domain_id: User's domain ID for authentication.\n :param string user_domain_name: User's domain name for authentication.\n :param bool reauthenticate: Allow fetching a new token if the current one\n is going to expire. (optional) default True\n "`

`__init__`(*auth_url: str, application_credential_secret: str, application_credential_id: str | None = None, application_credential_name: str | None = None, user_id: str | None = None, username: str | None = None, user_domain_id: str | None = None, user_domain_name: str | None = None, *, unscoped: bool = False, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*) → None

`__module__ = 'keystoneauth1.identity.v3.application_credential'`

`_abc_impl = <_abc._abc_data object>`

`_auth_method_class`

alias of `ApplicationCredentialMethod`

class `keystoneauth1.identity.v3.`**`ApplicationCredentialMethod`**(**, application_credential_secret: str, application_credential_id: str | None = None, application_credential_name: str | None = None, user_id: str | None = None, username: str | None = None, user_domain_id: str | None = None, user_domain_name: str | None = None*)

Bases: `AuthMethod`

Construct a User/Passcode based authentication method.

Parameters

- **`application_credential_secret`** (*`string`*) Application credential secret.
- **`application_credential_id`** (*`string`*) Application credential id.
- **`application_credential_name`** (*`string`*) The name of the application credential, if an ID is not provided.

- **username** (*string*) Username for authentication, if an application credential ID is not provided.
- **user_id** (*string*) User ID for authentication, if an application credential ID is not provided.
- **user_domain_id** (*string*) Users domain ID for authentication, if an application credential ID is not provided.
- **user_domain_name** (*string*) Users domain name for authentication, if an application credential ID is not provided.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'_method_parameters': 'ty.Optional[list[str]]', 'application_credential_id': typing.Optional[str], 'application_credential_name': typing.Optional[str], 'application_credential_secret': <class 'str'>, 'user_domain_id': typing.Optional[str], 'user_domain_name': typing.Optional[str], 'user_id': typing.Optional[str], 'username': typing.Optional[str]}`

`__doc__ = "Construct a User/Passcode based authentication method.\n\n :param string application_credential_secret: Application credential secret.\n :param string application_credential_id: Application credential id.\n :param string application_credential_name: The name of the application\n credential, if an ID is not\n provided.\n :param string username: Username for authentication, if an application\n credential ID is not provided.\n :param string user_id: User ID for authentication, if an application\n credential ID is not provided.\n :param string user_domain_id: User's domain ID for authentication, if an\n application credential ID is not provided.\n :param string user_domain_name: User's domain name for authentication, if\n an application credential ID is not\n provided.\n "`

`__init__`(**, application_credential_secret: str, application_credential_id: str | None = None, application_credential_name: str | None = None, user_id: str | None = None, username: str | None = None, user_domain_id: str | None = None, user_domain_name: str | None = None*) → None

`__module__ = 'keystoneauth1.identity.v3.application_credential'`

`_abc_impl = <_abc._abc_data object>`

`application_credential_id: str | None = None`

`application_credential_name: str | None = None`

`application_credential_secret: str`

`get_auth_data`(*session: Session, auth: Auth, headers: dict[str, str], request_kwargs: dict[str, object]*) → tuple[None, None] | tuple[str, Mapping[str, object]]

Return the authentication section of an auth plugin.

Parameters

- **session** (*keystoneauth1.session.Session*) The communication session.

- **auth** (*base.Auth*) The auth plugin calling the method.
- **headers** (*dict*) The headers that will be sent with the auth request if a plugin needs to add to them.

Returns
The identifier of this plugin and a dict of authentication data for the auth type.

Return type
tuple(string, dict)

get_cache_id_elements() → dict[str, str | None]

Get the elements for this auth method that make it unique.

These elements will be used as part of the `keystoneauth1.plugin.BaseIdentityPlugin.get_cache_id()` to allow caching of the auth plugin.

Plugins should override this if they want to allow caching of their state.

To avoid collision or overrides the keys of the returned dictionary should be prefixed with the plugin identifier. For example the password plugin returns its username value as password_username.

user_domain_id: str | None = None

user_domain_name: str | None = None

user_id: str | None = None

username: str | None = None

class keystoneauth1.identity.v3.**Auth**(*auth_url: str, auth_methods: list[AuthMethod], *, unscoped: bool = False, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

Bases: `BaseAuth`

Identity V3 Authentication Plugin.

Parameters

- **auth_url** (*string*) Identity service endpoint for authentication.
- **auth_methods** (*list*) A collection of methods to authenticate with.
- **trust_id** (*string*) Trust ID for trust scoping.
- **domain_id** (*string*) Domain ID for domain scoping.
- **domain_name** (*string*) Domain name for domain scoping.
- **project_id** (*string*) Project ID for project scoping.
- **project_name** (*string*) Project name for project scoping.
- **project_domain_id** (*string*) Projects domain ID for project.

- **project_domain_name** (*string*) Projects domain name for project.
- **reauthenticate** (*bool*) Allow fetching a new token if the current one is going to expire. (optional) default True
- **include_catalog** (*bool*) Include the service catalog in the returned token. (optional) default True.
- **unscoped** (*bool*) Force the return of an unscoped token. This will make the keystone server return an unscoped token even if a default_project_id is set for this user.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}`

`__doc__ = "Identity V3 Authentication Plugin.\n\n :param string auth_url: Identity service endpoint for authentication.\n :param list auth_methods: A collection of methods to authenticate with.\n :param string trust_id: Trust ID for trust scoping.\n :param string domain_id: Domain ID for domain scoping.\n :param string domain_name: Domain name for domain scoping.\n :param string project_id: Project ID for project scoping.\n :param string project_name: Project name for project scoping.\n :param string project_domain_id: Project's domain ID for project.\n :param string project_domain_name: Project's domain name for project.\n :param bool reauthenticate: Allow fetching a new token if the current one\n is going to expire. (optional) default True\n :param bool include_catalog: Include the service catalog in the returned\n token. (optional) default True.\n :param bool unscoped: Force the return of an unscoped token. This will make\n the keystone server return an unscoped token even if\n a default_project_id is set for this user.\n "`

`__init__`(*auth_url: str*, *auth_methods: list[AuthMethod]*, ***, *unscoped: bool = False*, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*)

`__module__ = 'keystoneauth1.identity.v3.base'`

`_abc_impl = <_abc._abc_data object>`

`add_method`(*method: AuthMethod*) → None

Add an additional initialized AuthMethod instance.

`get_auth_ref`(*session: Session*) → AccessInfoV3

Obtain a token from an OpenStack Identity Service.

This method is overridden by the various token version plugins.

This function should not be called independently and is expected to be invoked via the do_authenticate function.

This function will be invoked if the AcessInfo object cached by the plugin is not valid. Thus plugins should always fetch a new AccessInfo when invoked. If you are looking to just retrieve the current auth data then you should use get_access.

Parameters

`session` (*keystoneauth1.session.Session*) A session object that can be used for communication.

Raises

- `keystoneauth1.exceptions.response.InvalidResponse` The response returned wasnt appropriate.
- `keystoneauth1.exceptions.http.HttpError` An error from an invalid HTTP response.

Returns

Token access information.

Return type

`keystoneauth1.access.AccessInfo`

get_cache_id_elements() → dict[str, str | None]

Get the elements for this auth plugin that make it unique.

As part of the get_cache_id requirement we need to determine what aspects of this plugin and its values that make up the unique elements.

This should be overridden by plugins that wish to allow caching.

Returns

The unique attributes and values of this plugin.

Return type

A flat dict with a str key and str or None value. This is required as we feed these values into a hash. Pairs where the value is None are ignored in the hashed id.

class keystoneauth1.identity.v3.**AuthConstructor**(*auth_url: str*, **args: Any*, *unscoped: bool = False*, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*, ***kwargs: Any*)

Bases: `Auth`

Abstract base class for creating an Auth Plugin.

The Auth Plugin created contains only one authentication method. This is generally the required usage.

An AuthConstructor creates an AuthMethod based on the methods arguments and the auth_method_class defined by the plugin. It then creates the auth plugin with only that authentication method.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_auth_method_class': typing.ClassVar[typing.Type[keystoneauth1.identity.v3.base.AuthMethod]], '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}`

`__doc__ = "Abstract base class for creating an Auth Plugin.\n\n The Auth Plugin created contains only one authentication method. This\n is generally the required usage.\n\n An AuthConstructor creates an AuthMethod based on the method's\n arguments and the auth_method_class defined by the plugin. It then\n creates the auth plugin with only that authentication method.\n "`

`__init__`(*auth_url: str*, **args: Any*, *unscoped: bool = False*, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*, ***kwargs: Any*)

`__module__ = 'keystoneauth1.identity.v3.base'`

`_abc_impl = <_abc._abc_data object>`

`_auth_method_class: ClassVar[Type[AuthMethod]]`

class `keystoneauth1.identity.v3.`**`AuthMethod`**(***kwargs: object*)

Bases: `object`

One part of a V3 Authentication strategy.

The v3 /tokens API allow multiple methods to be presented when authentication against the server. Each one of these methods is implemented by an AuthMethod.

Note: When implementing an AuthMethod use keyword arguments to ensure they are supported by the MultiFactor auth plugin.

`__abstractmethods__ = frozenset({'get_auth_data'})`

`__annotations__ = {'_method_parameters': typing.Optional[list[str]]}`

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.identity.v3.base', '__annotations__': {'_method_parameters': typing.Optional[list[str]]}, '__doc__': "One part of a V3 Authentication strategy.\n\n The v3 '/tokens' API allow multiple methods to be presented when\n authentication against the server. Each one of these methods is implemented\n by an AuthMethod.\n\n Note: When implementing an AuthMethod use keyword arguments to ensure they\n are supported by the MultiFactor auth plugin.\n ", '_method_parameters': None, '__init__': <function AuthMethod.__init__>, '_extract_kwargs': <classmethod(<function AuthMethod._extract_kwargs>)>, 'get_auth_data': <function AuthMethod.get_auth_data>, 'get_cache_id_elements': <function AuthMethod.get_cache_id_elements>, '__dict__': <attribute '__dict__' of 'AuthMethod' objects>, '__weakref__': <attribute '__weakref__' of 'AuthMethod' objects>, '__abstractmethods__': frozenset({'get_auth_data'}), '_abc_impl': <_abc._abc_data object>})`

`__doc__ = "One part of a V3 Authentication strategy.\n\n The v3 '/tokens' API allow multiple methods to be presented when\n authentication against the server. Each one of these methods is implemented\n by an AuthMethod.\n\n Note: When implementing an AuthMethod use keyword arguments to ensure they\n are supported by the MultiFactor auth plugin.\n "`

`__init__`(***kwargs: object*)

`__module__ = 'keystoneauth1.identity.v3.base'`

`__weakref__`

list of weak references to the object (if defined)

`_abc_impl = <_abc._abc_data object>`

`classmethod _extract_kwargs`(*kwargs: dict[str, object]*) → dict[str, object]

Remove parameters related to this method from other kwargs.

`_method_parameters: list[str] | None = None`

Deprecated parameter for defining the parameters supported by the plugin. These should now be defined by typed class attributes.

`abstract get_auth_data`(*session: Session, auth: Auth, headers: dict[str, str], request_kwargs: dict[str, object]*) → tuple[None, None] | tuple[str, Mapping[str, object]]

Return the authentication section of an auth plugin.

Parameters

- **session** (*keystoneauth1.session.Session*) The communication session.
- **auth** (*base.Auth*) The auth plugin calling the method.
- **headers** (*dict*) The headers that will be sent with the auth request if a plugin needs to add to them.

Returns

The identifier of this plugin and a dict of authentication data for the auth type.

Return type
tuple(string, dict)

get_cache_id_elements() → dict[str, str | None]

Get the elements for this auth method that make it unique.

These elements will be used as part of the `keystoneauth1.plugin.BaseIdentityPlugin.get_cache_id()` to allow caching of the auth plugin.

Plugins should override this if they want to allow caching of their state.

To avoid collision or overrides the keys of the returned dictionary should be prefixed with the plugin identifier. For example the password plugin returns its username value as password_username.

class keystoneauth1.identity.v3.**BaseAuth**(*auth_url: str*, **, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*)

Bases: `BaseIdentityPlugin`

Identity V3 Authentication Plugin.

Parameters

- **auth_url** (*string*) Identity service endpoint for authentication.
- **trust_id** (*string*) Trust ID for trust scoping.
- **system_scope** (*string*) System information to scope to.
- **domain_id** (*string*) Domain ID for domain scoping.
- **domain_name** (*string*) Domain name for domain scoping.
- **project_id** (*string*) Project ID for project scoping.
- **project_name** (*string*) Project name for project scoping.
- **project_domain_id** (*string*) Projects domain ID for project.
- **project_domain_name** (*string*) Projects domain name for project.
- **reauthenticate** (*bool*) Allow fetching a new token if the current one is going to expire. (optional) default True
- **include_catalog** (*bool*) Include the service catalog in the returned token. (optional) default True.

__abstractmethods__ = frozenset({'get_auth_ref'})

__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': <class 'str'>, 'reauthenticate': 'bool'}

`__doc__ = "Identity V3 Authentication Plugin.\n\n :param string auth_url: Identity service endpoint for authentication.\n :param string trust_id: Trust ID for trust scoping.\n :param string system_scope: System information to scope to.\n :param string domain_id: Domain ID for domain scoping.\n :param string domain_name: Domain name for domain scoping.\n :param string project_id: Project ID for project scoping.\n :param string project_name: Project name for project scoping.\n :param string project_domain_id: Project's domain ID for project.\n :param string project_domain_name: Project's domain name for project.\n :param bool reauthenticate: Allow fetching a new token if the current one\n is going to expire. (optional) default True\n :param bool include_catalog: Include the service catalog in the returned\n token. (optional) default True.\n "`

`__init__`(*auth_url: str, *, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

`__module__ = 'keystoneauth1.identity.v3.base'`

`_abc_impl = <_abc._abc_data object>`

`auth_url:` `str`

`property has_scope_parameters:` `bool`

Return true if parameters can be used to create a scoped token.

`property token_url:` `str`

The full URL where we will send authentication data.

`class keystoneauth1.identity.v3.FederationBaseAuth`(*auth_url: str, identity_provider: str, protocol: str, *, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

Bases: `_Rescoped`

Federation authentication plugin.

Parameters

- **auth_url** (*string*) URL of the Identity Service
- **identity_provider** (*string*) name of the Identity Provider the client will authenticate against. This parameter will be used to build a dynamic URL used to obtain unscoped OpenStack token.
- **protocol** (*string*) name of the protocol the client will authenticate against.

`__abstractmethods__ = frozenset({'get_unscoped_auth_ref'})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}`

`__doc__ = 'Federation authentication plugin.\n\n :param auth_url: URL of the Identity Service\n :type auth_url: string\n :param identity_provider: name of the Identity Provider the client\n will authenticate against. This parameter\n will be used to build a dynamic URL used to\n obtain unscoped OpenStack token.\n :type identity_provider: string\n :param protocol: name of the protocol the client will authenticate\n against.\n :type protocol: string\n\n '`

`__init__`(*auth_url: str, identity_provider: str, protocol: str, *, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

`__module__ = 'keystoneauth1.identity.v3.federation'`

`_abc_impl = <_abc._abc_data object>`

`property federated_token_url: str`

Full URL where authorization data is sent.

`class keystoneauth1.identity.v3.`**`Keystone2Keystone`**(*base_plugin: BaseIdentityPlugin, service_provider: str, *, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

Bases: `_Rescoped`

Plugin to execute the Keystone to Keyestone authentication flow.

In this plugin, an ECP wrapped SAML assertion provided by a keystone Identity Provider (IdP) is used to request an OpenStack unscoped token from a keystone Service Provider (SP).

Parameters

- **`base_plugin`** (*`keystoneauth1.identity.v3.base.BaseAuth`*) Auth plugin already authenticated against the keystone IdP.
- **`service_provider`** (*`str`*) The Service Provider ID as returned by ServiceProviderManager.list()

`HTTP_MOVED_TEMPORARILY = 302`

`HTTP_SEE_OTHER = 303`

`REQUEST_ECP_URL = '/auth/OS-FEDERATION/saml2/ecp'`

Path where the ECP wrapped SAML assertion should be presented to the Keystone Service Provider.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}`

`__doc__ = 'Plugin to execute the Keystone to Keyestone authentication flow.\n\n In this plugin, an ECP wrapped SAML assertion provided by a keystone\n Identity Provider (IdP) is used to request an OpenStack unscoped token\n from a keystone Service Provider (SP).\n\n :param base_plugin: Auth plugin already authenticated against the keystone\n IdP.\n :type base_plugin: keystoneauth1.identity.v3.base.BaseAuth\n\n :param service_provider: The Service Provider ID as returned by\n ServiceProviderManager.list()\n :type service_provider: str\n\n '`

`__init__`(*base_plugin: BaseIdentityPlugin*, *service_provider: str*, ***, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*)

`__module__ = 'keystoneauth1.identity.v3.k2k'`

`_abc_impl = <_abc._abc_data object>`

`_get_ecp_assertion`(*session: Session*) → str

`classmethod _remote_auth_url`(*auth_url: str*) → str

Return auth_url of the remote Keystone Service Provider.

Remote clouds auth_url is an endpoint for getting federated unscoped token, typically that would be `https://remote.example.com:5000/v3/OS-FEDERATION/identity_providers/ <idp>/protocols/<protocol_id>/auth`. However we need to generate a real auth_url, used for token scoping. This function assumes there are static values today in the remote auth_url stored in the Service Provider attribute and those can be used as a delimiter. If the sp_auth_url doesnt comply with standard federation auth url the function will simply return whole string.

Parameters
auth_url (*str*) auth_url of the remote cloud

Returns
auth_url of remote cloud where a token can be validated or scoped.

Return type
str

`_send_service_provider_ecp_authn_response`(*session: Session*, *sp_url: str*, *sp_auth_url: str*) → Response

Present ECP wrapped SAML assertion to the keystone SP.

The assertion is issued by the keystone IdP and it is targeted to the keystone that will serve as Service Provider.

Parameters

- **session** a session object to send out HTTP requests.
- **sp_url** (*str*) URL where the ECP wrapped SAML assertion will be presented to the keystone SP. Usually, something like: https://sp.com/Shibboleth.sso/SAML2/ECP
- **sp_auth_url** (*str*) Federated authentication URL of the keystone SP. It is specified by IdP, for example: https://sp.com/v3/OS-FEDERATION/identity_providers/ idp_id/protocols/protocol_id/auth

get_unscoped_auth_ref(*session: Session*) → AccessInfoV3

Fetch unscoped federated token.

class keystoneauth1.identity.v3.**MultiFactor**(*auth_url: str*, *auth_methods: list[str]*, ***, *unscoped: bool = False*, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*, ***kwargs: Any*)

Bases: Auth

A plugin for authenticating with multiple auth methods.

Parameters

- **auth_url** (*string*) Identity service endpoint for authentication.
- **auth_methods** (*string*) names of the methods to authenticate with.
- **trust_id** (*string*) Trust ID for trust scoping.
- **system_scope** (*string*) System information to scope to.
- **domain_id** (*string*) Domain ID for domain scoping.
- **domain_name** (*string*) Domain name for domain scoping.
- **project_id** (*string*) Project ID for project scoping.
- **project_name** (*string*) Project name for project scoping.
- **project_domain_id** (*string*) Projects domain ID for project.
- **project_domain_name** (*string*) Projects domain name for project.
- **reauthenticate** (*bool*) Allow fetching a new token if the current one is going to expire. (optional) default True

Also accepts various keyword args based on which methods are specified.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}`

`__doc__ = "A plugin for authenticating with multiple auth methods.\n\n :param string auth_url: Identity service endpoint for authentication.\n :param string auth_methods: names of the methods to authenticate with.\n :param string trust_id: Trust ID for trust scoping.\n :param string system_scope: System information to scope to.\n :param string domain_id: Domain ID for domain scoping.\n :param string domain_name: Domain name for domain scoping.\n :param string project_id: Project ID for project scoping.\n :param string project_name: Project name for project scoping.\n :param string project_domain_id: Project's domain ID for project.\n :param string project_domain_name: Project's domain name for project.\n :param bool reauthenticate: Allow fetching a new token if the current one\n is going to expire. (optional) default True\n\n Also accepts various keyword args based on which methods are specified.\n "`

`__init__`(*auth_url: str*, *auth_methods: list[str]*, ***, *unscoped: bool = False*, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*, ***kwargs: Any*)

`__module__ = 'keystoneauth1.identity.v3.multi_factor'`

`_abc_impl = <_abc._abc_data object>`

class `keystoneauth1.identity.v3.`**`OAuth2ClientCredential`**(*auth_url: str*, *oauth2_endpoint: str*, *oauth2_client_id: str*, *oauth2_client_secret: str*, ***, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*)

Bases: `Auth`

A plugin for authenticating via an OAuth2.0 client credential.

Parameters

- **auth_url** (*string*) Identity service endpoint for authentication.
- **oauth2_endpoint** (*string*) OAuth2.0 endpoint.
- **oauth2_client_id** (*string*) OAuth2.0 client credential id.
- **oauth2_client_secret** (*string*) OAuth2.0 client credential secret.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}`

`__doc__ = 'A plugin for authenticating via an OAuth2.0 client credential.\n\n :param string auth_url: Identity service endpoint for authentication.\n :param string oauth2_endpoint: OAuth2.0 endpoint.\n :param string oauth2_client_id: OAuth2.0 client credential id.\n :param string oauth2_client_secret: OAuth2.0 client credential secret.\n '`

`__init__`(*auth_url: str*, *oauth2_endpoint: str*, *oauth2_client_id: str*, *oauth2_client_secret: str*, ***, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*) → None

`__module__ = 'keystoneauth1.identity.v3.oauth2_client_credential'`

`_abc_impl = <_abc._abc_data object>`

`_auth_method_class`

alias of `OAuth2ClientCredentialMethod`

`get_headers`(*session: Session*) → dict[str, str] | None

Fetch authentication headers for message.

Parameters

session (*keystoneauth1.session.Session*) The session object that the auth_plugin belongs to.

Returns

Headers that are set to authenticate a message or None for failure. Note that when checking this value that the empty dict is a valid, non-failure response.

Return type

dict

class `keystoneauth1.identity.v3.`**`OAuth2ClientCredentialMethod`**(***, *oauth2_endpoint: str*, *oauth2_client_id: str*, *oauth2_client_secret: str*)

Bases: `AuthMethod`

An auth method to fetch a token via an OAuth2.0 client credential.

Parameters

- **oauth2_endpoint** (*string*) OAuth2.0 endpoint.
- **oauth2_client_id** (*string*) OAuth2.0 client credential id.
- **oauth2_client_secret** (*string*) OAuth2.0 client credential secret.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'_method_parameters': 'ty.Optional[list[str]]', 'oauth2_client_id': <class 'str'>, 'oauth2_client_secret': <class 'str'>, 'oauth2_endpoint': <class 'str'>}`

`__doc__ = 'An auth method to fetch a token via an OAuth2.0 client credential.\n\n :param string oauth2_endpoint: OAuth2.0 endpoint.\n :param string oauth2_client_id: OAuth2.0 client credential id.\n :param string oauth2_client_secret: OAuth2.0 client credential secret.\n '`

`__init__`(*, *oauth2_endpoint: str*, *oauth2_client_id: str*, *oauth2_client_secret: str*) → None

`__module__ = 'keystoneauth1.identity.v3.oauth2_client_credential'`

`_abc_impl = <_abc._abc_data object>`

`get_auth_data`(*session: Session*, *auth: Auth*, *headers: dict[str, str]*, *request_kwargs: dict[str, object]*) → tuple[None, None] | tuple[str, Mapping[str, object]]

Return the authentication section of an auth plugin.

Parameters

- **session** (*keystoneauth1.session.Session*) The communication session.
- **auth** (*base.Auth*) The auth plugin calling the method.
- **headers** (*dict*) The headers that will be sent with the auth request if a plugin needs to add to them.

Returns

The identifier of this plugin and a dict of authentication data for the auth type.

Return type

tuple(string, dict)

`get_cache_id_elements`() → dict[str, str | None]

Get the elements for this auth method that make it unique.

These elements will be used as part of the `keystoneauth1.plugin.BaseIdentityPlugin.get_cache_id()` to allow caching of the auth plugin.

Plugins should override this if they want to allow caching of their state.

To avoid collision or overrides the keys of the returned dictionary should be prefixed with the plugin identifier. For example the password plugin returns its username value as password_username.

oauth2_client_id: str

oauth2_client_secret: str

oauth2_endpoint: str

class keystoneauth1.identity.v3.**OAuth2mTlsClientCredential**(*auth_url: str, oauth2_endpoint: str, oauth2_client_id: str, *, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

Bases: BaseAuth

A plugin for authenticating via an OAuth2.0 mTLS client credential.

Parameters

- **auth_url** (*string*) keystone authorization endpoint.
- **oauth2_endpoint** (*string*) OAuth2.0 endpoint.
- **oauth2_client_id** (*string*) OAuth2.0 client credential id.

__abstractmethods__ = frozenset({})

__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}

__doc__ = 'A plugin for authenticating via an OAuth2.0 mTLS client credential.\n\n :param string auth_url: keystone authorization endpoint.\n :param string oauth2_endpoint: OAuth2.0 endpoint.\n :param string oauth2_client_id: OAuth2.0 client credential id.\n '

__init__(*auth_url: str, oauth2_endpoint: str, oauth2_client_id: str, *, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*) `__module__ = 'keystoneauth1.identity.v3.oauth2_mtls_client_credential'`

`_abc_impl = <_abc._abc_data object>`

`get_auth_ref`(*session: Session*) → AccessInfoV3

Obtain a token from an OpenStack Identity Service.

This method is overridden by the various token version plugins.

This function should not be called independently and is expected to be invoked via the do_authenticate function.

This function will be invoked if the AcessInfo object cached by the plugin is not valid. Thus plugins should always fetch a new AccessInfo when invoked. If you are looking to just retrieve the current auth data then you should use get_access.

Parameters
`session` (*keystoneauth1.session.Session*) A session object that can be used for communication.

Raises

- `keystoneauth1.exceptions.response.InvalidResponse` The response returned wasnt appropriate.
- `keystoneauth1.exceptions.http.HttpError` An error from an invalid HTTP response.
- `keystoneauth1.exceptions.ClientException` An error from getting OAuth2.0 access token.

Returns
Token access information.

Return type
`keystoneauth1.access.AccessInfo`

`get_headers`(*session: Session*) → dict[str, str] | None

Fetch authentication headers for message.

Parameters
`session` (*keystoneauth1.session.Session*) The session object that the auth_plugin belongs to.

Returns
Headers that are set to authenticate a message or None for failure. Note that when checking this value that the empty dict is a valid, non-failure response.

Return type
dict

class keystoneauth1.identity.v3.**OidcAccessToken**(*auth_url: str, identity_provider: str, protocol: str, access_token_type: str = 'access_token', scope: str = 'openid profile', access_token_endpoint: str | None = None, discovery_endpoint: str | None = None, access_token: str | None = None, *, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

Bases: `_OidcBase`

Implementation for OpenID Connect access token reuse.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', '_discovery_document': 'dict[str, object]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'grant_type': 'ty.ClassVar[str]', 'reauthenticate': 'bool'}`

`__doc__ = 'Implementation for OpenID Connect access token reuse.'`

`__init__`(*auth_url: str, identity_provider: str, protocol: str, access_token_type: str = 'access_token', scope: str = 'openid profile', access_token_endpoint: str | None = None, discovery_endpoint: str | None = None, access_token: str | None = None, *, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

The OpenID Connect plugin based on the Access Token.

It expects the following:

Parameters

- **auth_url** (*string*) URL of the Identity Service
- **identity_provider** (*string*) Name of the Identity Provider the client will authenticate against
- **protocol** (*string*) Protocol name as configured in keystone
- **access_token** (*string*) OpenID Connect Access token

`__module__ = 'keystoneauth1.identity.v3.oidc'`

`_abc_impl = <_abc._abc_data object>`

get_payload(*session: Session*) → dict[str, str | None]

OidcAccessToken does not require a payload.

get_unscoped_auth_ref(*session: Session*) → AccessInfoV3

Authenticate with OpenID Connect and get back claims.

We exchange the access token upon accessing the protected Keystone endpoint (federated auth URL). This will trigger the OpenID Connect Provider to perform a user introspection and retrieve information (specified in the scope) about the user in the form of an OpenID Connect Claim. These claims will be sent to Keystone in the form of environment variables.

Parameters

`session` (*keystoneauth1.session.Session*) a session object to send out HTTP requests.

Returns

a token data representation

Return type

`keystoneauth1.access.AccessInfoV3`

class keystoneauth1.identity.v3.**OidcAuthorizationCode**(*auth_url: str, identity_provider: str, protocol: str, client_id: str, client_secret: str, access_token_type: str = 'access_token', scope: str = 'openid profile', access_token_endpoint: str | None = None, discovery_endpoint: str | None = None, code: str | None = None, *, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True, redirect_uri: str | None = None*)

Bases: `_OidcBase`

Implementation for OpenID Connect Authorization Code.

__abstractmethods__ = frozenset({})

__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', '_discovery_document': 'dict[str, object]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'grant_type': 'ty.ClassVar[str]', 'reauthenticate': 'bool'}

`__doc__ = 'Implementation for OpenID Connect Authorization Code.'`

`__init__`(*auth_url: str*, *identity_provider: str*, *protocol: str*, *client_id: str*, *client_secret: str*, *access_token_type: str = 'access_token'*, *scope: str = 'openid profile'*, *access_token_endpoint: str | None = None*, *discovery_endpoint: str | None = None*, *code: str | None = None*, ***, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*, *redirect_uri: str | None = None*)

The OpenID Authorization Code plugin expects the following.

Parameters

- **`redirect_uri`** (*`string`*) OpenID Connect Client Redirect URL
- **`code`** (*`string`*) OAuth 2.0 Authorization Code

`__module__ = 'keystoneauth1.identity.v3.oidc'`

`_abc_impl = <_abc._abc_data object>`

`get_payload`(*session: Session*) → dict[str, str | None]

Get an authorization grant for the authorization_code grant type.

Parameters
`session` (*`keystoneauth1.session.Session`*) a session object to send out HTTP requests.

Returns
a python dictionary containing the payload to be exchanged

Return type
dict

`grant_type: ty.ClassVar[str] = 'authorization_code'`

class keystoneauth1.identity.v3.**OidcClientCredentials**(*auth_url: str, identity_provider: str, protocol: str, client_id: str, client_secret: str, access_token_type: str = 'access_token', scope: str = 'openid profile', access_token_endpoint: str | None = None, discovery_endpoint: str | None = None, *, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

Bases: `_OidcBase`

Implementation for OpenID Connect Client Credentials.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', '_discovery_document': 'dict[str, object]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'grant_type': 'ty.ClassVar[str]', 'reauthenticate': 'bool'}`

`__doc__ = 'Implementation for OpenID Connect Client Credentials.'`

`__init__`(*auth_url: str, identity_provider: str, protocol: str, client_id: str, client_secret: str, access_token_type: str = 'access_token', scope: str = 'openid profile', access_token_endpoint: str | None = None, discovery_endpoint: str | None = None, *, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

The OpenID Client Credentials expects the following.

Parameters

- **`client_id`** Client ID used to authenticate
- **`client_secret`** Client Secret used to authenticate

`__module__ = 'keystoneauth1.identity.v3.oidc'`

`_abc_impl = <_abc._abc_data object>`

get_payload(*session: Session*) → dict[str, str | None]

Get an authorization grant for the client credentials grant type.

Parameters

session (*keystoneauth1.session.Session*) a session object to send out HTTP requests.

Returns

a python dictionary containing the payload to be exchanged

Return type

dict

grant_type: ty.ClassVar[str] = 'client_credentials'

class keystoneauth1.identity.v3.**OidcPassword**(*auth_url: str, identity_provider: str, protocol: str, client_id: str, client_secret: str, access_token_type: str = 'access_token', scope: str = 'openid profile', access_token_endpoint: str | None = None, discovery_endpoint: str | None = None, username: str | None = None, password: str | None = None, idp_otp_key: str | None = None, *, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

Bases: _OidcBase

Implementation for OpenID Connect Resource Owner Password Credential.

__abstractmethods__ = frozenset({})

__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', '_discovery_document': 'dict[str, object]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'grant_type': 'ty.ClassVar[str]', 'reauthenticate': 'bool'}

__doc__ = 'Implementation for OpenID Connect Resource Owner Password Credential.'

__init__(*auth_url: str, identity_provider: str, protocol: str, client_id: str, client_secret: str, access_token_type: str = 'access_token', scope: str = 'openid profile', access_token_endpoint: str | None = None, discovery_endpoint: str | None = None, username: str | None = None, password: str | None = None, idp_otp_key: str | None = None, *, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

The OpenID Password plugin expects the following.

Parameters

- **username** (*string*) Username used to authenticate
- **password** (*string*) Password used to authenticate

`__module__ = 'keystoneauth1.identity.v3.oidc'`

`_abc_impl = <_abc._abc_data object>`

get_payload(*session: Session*) → dict[str, str | None]

Get an authorization grant for the password grant type.

Parameters

session (*keystoneauth1.session.Session*) a session object to send out HTTP requests.

Returns

a python dictionary containing the payload to be exchanged

Return type

dict

`grant_type: ty.ClassVar[str] = 'password'`

manage_otp_from_session_or_request_to_the_user(*payload: dict[str, str | None], session: Session*) → None

Get the OTP code from the session or else request to the user.

When the OS_IDP_OTP_KEY environment variable is set, this method will verify if there is an OTP value in the current session, if it exists, we use it (the OTP from session) to send to the Identity Provider when retrieving the access token. If there is no OTP in the current session, we ask the user to enter it (the OTP), and we add it to the session to execute the authentication flow.

The OTP is being stored in the session because in some flows, the CLI is doing the authentication process two times, so saving the OTP in the session, allow us to use the same OTP in a short time interval, avoiding to request it to the user twice in a row.

Parameters

- **payload**
- **session**

Returns

class keystoneauth1.identity.v3.**Password**(*auth_url: str, password: str, user_id: str | None = None, username: str | None = None, user_domain_id: str | None = None, user_domain_name: str | None = None, *, unscoped: bool = False, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*)

Bases: `Auth`

A plugin for authenticating with a username and password.

Parameters

- **auth_url** (*string*) Identity service endpoint for authentication.
- **password** (*string*) Password for authentication.
- **user_id** (*string*) User ID for authentication.
- **username** (*string*) Username for authentication.
- **user_domain_id** (*string*) Users domain ID for authentication.
- **user_domain_name** (*string*) Users domain name for authentication.
- **trust_id** (*string*) Trust ID for trust scoping.
- **system_scope** (*string*) System information to scope to.
- **domain_id** (*string*) Domain ID for domain scoping.
- **domain_name** (*string*) Domain name for domain scoping.
- **project_id** (*string*) Project ID for project scoping.
- **project_name** (*string*) Project name for project scoping.
- **project_domain_id** (*string*) Projects domain ID for project.
- **project_domain_name** (*string*) Projects domain name for project.
- **reauthenticate** (*bool*) Allow fetching a new token if the current one is going to expire. (optional) default True

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}`

__doc__ = "A plugin for authenticating with a username and password.\n\n :param string auth_url: Identity service endpoint for authentication.\n :param string password: Password for authentication.\n :param string user_id: User ID for authentication.\n :param string username: Username for authentication.\n :param string user_domain_id: User's domain ID for authentication.\n :param string user_domain_name: User's domain name for authentication.\n :param string trust_id: Trust ID for trust scoping.\n :param string system_scope: System information to scope to.\n :param string domain_id: Domain ID for domain scoping.\n :param string domain_name: Domain name for domain scoping.\n :param string project_id: Project ID for project scoping.\n :param string project_name: Project name for project scoping.\n :param string project_domain_id: Project's domain ID for project.\n :param string project_domain_name: Project's domain name for project.\n :param bool reauthenticate: Allow fetching a new token if the current one\n is going to expire. (optional) default True\n "

__init__(*auth_url: str, password: str, user_id: str | None = None, username: str | None = None, user_domain_id: str | None = None, user_domain_name: str | None = None, *, unscoped: bool = False, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True*) → None

__module__ = 'keystoneauth1.identity.v3.password'

_abc_impl = <_abc._abc_data object>

_auth_method_class

alias of `PasswordMethod`

class keystoneauth1.identity.v3.**PasswordMethod**(**, password: str, user_id: str | None = None, username: str | None = None, user_domain_id: str | None = None, user_domain_name: str | None = None*)

Bases: `AuthMethod`

Construct a User/Password based authentication method.

Parameters

- **password** (*string*) Password for authentication.
- **username** (*string*) Username for authentication.
- **user_id** (*string*) User ID for authentication.
- **user_domain_id** (*string*) Users domain ID for authentication.
- **user_domain_name** (*string*) Users domain name for authentication.

__abstractmethods__ = frozenset({})

__annotations__ = {'_method_parameters': 'ty.Optional[list[str]]', 'password': <class 'str'>, 'user_domain_id': typing.Optional[str], 'user_domain_name': typing.Optional[str], 'user_id': typing.Optional[str], 'username': typing.Optional[str]}

`__doc__ = "Construct a User/Password based authentication method.\n\n :param string password: Password for authentication.\n :param string username: Username for authentication.\n :param string user_id: User ID for authentication.\n :param string user_domain_id: User's domain ID for authentication.\n :param string user_domain_name: User's domain name for authentication.\n "`

`__init__`(**, password: str | None = None, user_id: str | None = None, username: str | None = None, user_domain_id: str | None = None, user_domain_name: str | None = None*) → None

`__module__ = 'keystoneauth1.identity.v3.password'`

`_abc_impl = <_abc._abc_data object>`

`get_auth_data`(*session: Session, auth: Auth, headers: dict[str, str], request_kwargs: dict[str, object]*) → tuple[None, None] | tuple[str, Mapping[str, object]]

Return the authentication section of an auth plugin.

Parameters

- **session** (*`keystoneauth1.session.Session`*) The communication session.
- **auth** (*`base.Auth`*) The auth plugin calling the method.
- **headers** (*`dict`*) The headers that will be sent with the auth request if a plugin needs to add to them.

Returns

The identifier of this plugin and a dict of authentication data for the auth type.

Return type

tuple(string, dict)

`get_cache_id_elements`() → dict[str, str | None]

Get the elements for this auth method that make it unique.

These elements will be used as part of the `keystoneauth1.plugin.BaseIdentityPlugin.get_cache_id()` to allow caching of the auth plugin.

Plugins should override this if they want to allow caching of their state.

To avoid collision or overrides the keys of the returned dictionary should be prefixed with the plugin identifier. For example the password plugin returns its username value as password_username.

`password: str`

`user_domain_id: str | None = None`

`user_domain_name: str | None = None`

`user_id: str | None = None`

`username: str | None = None`

class `keystoneauth1.identity.v3.`**`ReceiptMethod`**`(*, receipt: str)`

Bases: `AuthMethod`

Construct an Auth plugin to continue authentication with a receipt.

Parameters
`receipt` (*string*) Receipt for authentication.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'_method_parameters': 'ty.Optional[list[str]]', 'receipt': <class 'str'>}`

`__doc__ = 'Construct an Auth plugin to continue authentication with a receipt.\n\n :param string receipt: Receipt for authentication.\n '`

`__init__``(*, receipt: str) → None`

`__module__ = 'keystoneauth1.identity.v3.receipt'`

`_abc_impl = <_abc._abc_data object>`

`get_auth_data``(session: Session, auth: Auth, headers: dict[str, str], request_kwargs: dict[str, object]) → tuple[None, None] | tuple[str, Mapping[str, object]]`

Add the auth receipt to the headers.

We explicitly return None to avoid being added to the request methods, or body.

`get_cache_id_elements``() → dict[str, str | None]`

Get the elements for this auth method that make it unique.

These elements will be used as part of the `keystoneauth1.plugin.BaseIdentityPlugin.get_cache_id()` to allow caching of the auth plugin.

Plugins should override this if they want to allow caching of their state.

To avoid collision or overrides the keys of the returned dictionary should be prefixed with the plugin identifier. For example the password plugin returns its username value as password_username.

`receipt: str`

class `keystoneauth1.identity.v3.`**`TOTP`**`(auth_url: str, passcode: str, user_id: str | None = None, username: str | None = None, user_domain_id: str | None = None, user_domain_name: str | None = None, *, unscoped: bool = False, trust_id: str | None = None, system_scope: str | None = None, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, reauthenticate: bool = True, include_catalog: bool = True)`

Bases: `Auth`

A plugin for authenticating with a username and TOTP passcode.

Parameters

- **auth_url** (*string*) Identity service endpoint for authentication.
- **passcode** (*string*) TOTP passcode for authentication.
- **user_id** (*string*) User ID for authentication.
- **username** (*string*) Username for authentication.
- **user_domain_id** (*string*) Users domain ID for authentication.
- **user_domain_name** (*string*) Users domain name for authentication.
- **trust_id** (*string*) Trust ID for trust scoping.
- **domain_id** (*string*) Domain ID for domain scoping.
- **domain_name** (*string*) Domain name for domain scoping.
- **project_id** (*string*) Project ID for project scoping.
- **project_name** (*string*) Project name for project scoping.
- **project_domain_id** (*string*) Projects domain ID for project.
- **project_domain_name** (*string*) Projects domain name for project.
- **reauthenticate** (*bool*) Allow fetching a new token if the current one is going to expire. (optional) default True

__abstractmethods__ = frozenset({})

__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}

__doc__ = "A plugin for authenticating with a username and TOTP passcode.\n\n :param string auth_url: Identity service endpoint for authentication.\n :param string passcode: TOTP passcode for authentication.\n :param string user_id: User ID for authentication.\n :param string username: Username for authentication.\n :param string user_domain_id: User's domain ID for authentication.\n :param string user_domain_name: User's domain name for authentication.\n :param string trust_id: Trust ID for trust scoping.\n :param string domain_id: Domain ID for domain scoping.\n :param string domain_name: Domain name for domain scoping.\n :param string project_id: Project ID for project scoping.\n :param string project_name: Project name for project scoping.\n :param string project_domain_id: Project's domain ID for project.\n :param string project_domain_name: Project's domain name for project.\n :param bool reauthenticate: Allow fetching a new token if the current one\n is going to expire. (optional) default True\n "

__init__(*auth_url: str*, *passcode: str*, *user_id: str | None = None*, *username: str | None = None*, *user_domain_id: str | None = None*, *user_domain_name: str | None = None*, ***, *unscoped: bool = False*, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*) → None

__module__ = 'keystoneauth1.identity.v3.totp'

_abc_impl = <_abc._abc_data object>

_auth_method_class

alias of `TOTPMethod`

class `keystoneauth1.identity.v3.`**`TOTPMethod`**(**, passcode: str, user_id: str | None = None, username: str | None = None, user_domain_id: str | None = None, user_domain_name: str | None = None*)

Bases: `AuthMethod`

Construct a User/Passcode based authentication method.

Parameters

- **passcode** (*string*) TOTP passcode for authentication.
- **username** (*string*) Username for authentication.
- **user_id** (*string*) User ID for authentication.
- **user_domain_id** (*string*) Users domain ID for authentication.
- **user_domain_name** (*string*) Users domain name for authentication.

__abstractmethods__ = frozenset({})

__annotations__ = {'_method_parameters': 'ty.Optional[list[str]]', 'passcode': <class 'str'>, 'user_domain_id': typing.Optional[str], 'user_domain_name': typing.Optional[str], 'user_id': typing.Optional[str], 'username': typing.Optional[str]}

__doc__ = "Construct a User/Passcode based authentication method.\n\n :param string passcode: TOTP passcode for authentication.\n :param string username: Username for authentication.\n :param string user_id: User ID for authentication.\n :param string user_domain_id: User's domain ID for authentication.\n :param string user_domain_name: User's domain name for authentication.\n "

__init__(**, passcode: str, user_id: str | None = None, username: str | None = None, user_domain_id: str | None = None, user_domain_name: str | None = None*) → None

__module__ = 'keystoneauth1.identity.v3.totp'

_abc_impl = <_abc._abc_data object>

get_auth_data(*session: Session, auth: Auth, headers: dict[str, str], request_kwargs: dict[str, object]*) → tuple[None, None] | tuple[str, Mapping[str, object]]

Return the authentication section of an auth plugin.

Parameters

- **session** (*keystoneauth1.session.Session*) The communication session.
- **auth** (*base.Auth*) The auth plugin calling the method.

- **headers** (*dict*) The headers that will be sent with the auth request if a plugin needs to add to them.

Returns
The identifier of this plugin and a dict of authentication data for the auth type.

Return type
tuple(string, dict)

get_cache_id_elements() → dict[str, str | None]

Get the elements for this auth method that make it unique.

These elements will be used as part of the `keystoneauth1.plugin.BaseIdentityPlugin.get_cache_id()` to allow caching of the auth plugin.

Plugins should override this if they want to allow caching of their state.

To avoid collision or overrides the keys of the returned dictionary should be prefixed with the plugin identifier. For example the password plugin returns its username value as password_username.

passcode: str

user_domain_id: str | None = None

user_domain_name: str | None = None

user_id: str | None = None

username: str | None = None

class keystoneauth1.identity.v3.**Token**(*auth_url: str*, *token: str*, ***, *unscoped: bool = False*, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*)

Bases: `Auth`

A plugin for authenticating with an existing Token.

Parameters

- **auth_url** (*string*) Identity service endpoint for authentication.
- **token** (*string*) Token for authentication.
- **trust_id** (*string*) Trust ID for trust scoping.
- **domain_id** (*string*) Domain ID for domain scoping.
- **domain_name** (*string*) Domain name for domain scoping.
- **project_id** (*string*) Project ID for project scoping.
- **project_name** (*string*) Project name for project scoping.
- **project_domain_id** (*string*) Projects domain ID for project.

- **project_domain_name** (*string*) Projects domain name for project.
- **reauthenticate** (*bool*) Allow fetching a new token if the current one is going to expire. (optional) default True

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}`

`__doc__ = "A plugin for authenticating with an existing Token.\n\n :param string auth_url: Identity service endpoint for authentication.\n :param string token: Token for authentication.\n :param string trust_id: Trust ID for trust scoping.\n :param string domain_id: Domain ID for domain scoping.\n :param string domain_name: Domain name for domain scoping.\n :param string project_id: Project ID for project scoping.\n :param string project_name: Project name for project scoping.\n :param string project_domain_id: Project's domain ID for project.\n :param string project_domain_name: Project's domain name for project.\n :param bool reauthenticate: Allow fetching a new token if the current one\n is going to expire. (optional) default True\n "`

`__init__`(*auth_url: str*, *token: str*, ***, *unscoped: bool = False*, *trust_id: str | None = None*, *system_scope: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *reauthenticate: bool = True*, *include_catalog: bool = True*) → None

`__module__ = 'keystoneauth1.identity.v3.token'`

`_abc_impl = <_abc._abc_data object>`

`_auth_method_class`

alias of `TokenMethod`

class `keystoneauth1.identity.v3.`**`TokenMethod`**(***, *token: str*)

Bases: `AuthMethod`

Construct an Auth plugin to fetch a token from a token.

Parameters

token (*string*) Token for authentication.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'_method_parameters': 'ty.Optional[list[str]]', 'token': <class 'str'>}`

`__doc__ = 'Construct an Auth plugin to fetch a token from a token.\n\n :param string token: Token for authentication.\n '`

`__init__`(***, *token: str*) → None

`__module__ = 'keystoneauth1.identity.v3.token'`

_abc_impl = <_abc._abc_data object>

get_auth_data(*session: Session*, *auth: Auth*, *headers: dict[str, str]*, *request_kwargs: dict[str, object]*) → tuple[None, None] | tuple[str, Mapping[str, object]]

Return the authentication section of an auth plugin.

Parameters

- **session** (*keystoneauth1.session.Session*) The communication session.
- **auth** (*base.Auth*) The auth plugin calling the method.
- **headers** (*dict*) The headers that will be sent with the auth request if a plugin needs to add to them.

Returns
The identifier of this plugin and a dict of authentication data for the auth type.

Return type
tuple(string, dict)

get_cache_id_elements() → dict[str, str | None]

Get the elements for this auth method that make it unique.

These elements will be used as part of the `keystoneauth1.plugin.BaseIdentityPlugin.get_cache_id()` to allow caching of the auth plugin.

Plugins should override this if they want to allow caching of their state.

To avoid collision or overrides the keys of the returned dictionary should be prefixed with the plugin identifier. For example the password plugin returns its username value as password_username.

token: str

class keystoneauth1.identity.v3.**TokenlessAuth**(*auth_url: str*, *domain_id: str | None = None*, *domain_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*)

Bases: `BaseAuthPlugin`

A plugin for authenticating with Tokenless Auth.

This is for Tokenless Authentication. Scoped information like domain name and project ID will be passed in the headers and token validation request will be authenticated based on the provided HTTPS certificate along with the scope information.

__abstractmethods__ = frozenset({})

__annotations__ = {'_discovery_cache': 'dict[str, discover.Discover]'}

__doc__ = 'A plugin for authenticating with Tokenless Auth.\n\n This is for Tokenless Authentication. Scoped information\n like domain name and project ID will be passed in the headers and\n token validation request will be authenticated based on\n the provided HTTPS certificate along with the scope information.\n '

`__init__`(*auth_url: str, domain_id: str | None = None, domain_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None*)

A init method for TokenlessAuth.

Parameters

- **`auth_url`** (*`string`*) Identity service endpoint for authentication. The URL must include a version or any request will result in a 404 NotFound error.
- **`domain_id`** (*`string`*) Domain ID for domain scoping.
- **`domain_name`** (*`string`*) Domain name for domain scoping.
- **`project_id`** (*`string`*) Project ID for project scoping.
- **`project_name`** (*`string`*) Project name for project scoping.
- **`project_domain_id`** (*`string`*) Projects domain ID for project.
- **`project_domain_name`** (*`string`*) Projects domain name for project.

`__module__ = 'keystoneauth1.identity.v3.tokenless_auth'`

`_abc_impl = <_abc._abc_data object>`

`get_endpoint`(*session: ks_session.Session, service_type: str | None = None, **kwargs: Any*) → str | None

Return a valid endpoint for a service.

Parameters

- **`session`** (*`keystoneauth1.session.Session`*) A session object that can be used for communication.
- **`service_type`** (*`string`*) The type of service to lookup the endpoint for. This plugin will return None (failure) if service_type is not provided.

Returns
A valid endpoint URL or None if not available.

Return type
string or None

`get_headers`(*session: ks_session.Session*) → dict[str, str] | None

Fetch authentication headers for message.

This is to override the default get_headers method to provide tokenless auth scope headers if token is not provided in the session.

Parameters
`session` (*`keystoneauth1.session.Session`*) The session object that the auth_plugin belongs to.

Returns
Headers that are set to authenticate a message or None for failure. Note that when checking this value that the empty dict is a valid, non-failure response.

Return type
dict

Submodules

keystoneauth1.identity.access module

class keystoneauth1.identity.access.**AccessInfoPlugin**(*auth_ref: AccessInfo*, *auth_url: str | None = None*)

Bases: `BaseIdentityPlugin`

A plugin that turns an existing AccessInfo object into a usable plugin.

There are cases where reuse of an auth_ref or AccessInfo object is warranted such as from a cache, from auth_token middleware, or another source.

Turn the existing access info object into an identity plugin. This plugin cannot be refreshed as the AccessInfo object does not contain any authorizing information.

Parameters

- **auth_ref** (*keystoneauth1.access.AccessInfo*) the existing AccessInfo object.
- **auth_url** the url where this AccessInfo was retrieved from. Required if using the AUTH_INTERFACE with get_endpoint. (optional)

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': <class 'keystoneauth1.access.access.AccessInfo'>, 'auth_url': 'ty.Optional[str]', 'reauthenticate': 'bool'}`

`__doc__ = 'A plugin that turns an existing AccessInfo object into a usable plugin.\n\n There are cases where reuse of an auth_ref or AccessInfo object is\n warranted such as from a cache, from auth_token middleware, or another\n source.\n\n Turn the existing access info object into an identity plugin. This plugin\n cannot be refreshed as the AccessInfo object does not contain any\n authorizing information.\n\n :param auth_ref: the existing AccessInfo object.\n :type auth_ref: keystoneauth1.access.AccessInfo\n :param auth_url: the url where this AccessInfo was retrieved from. Required\n if using the AUTH_INTERFACE with get_endpoint. (optional)\n '`

`__init__`(*auth_ref: AccessInfo*, *auth_url: str | None = None*)

`__module__ = 'keystoneauth1.identity.access'`

`_abc_impl = <_abc._abc_data object>`

`auth_ref: AccessInfo`

`get_auth_ref`(*session: Session*) → AccessInfo

Obtain a token from an OpenStack Identity Service.

This method is overridden by the various token version plugins.

This function should not be called independently and is expected to be invoked via the do_authenticate function.

This function will be invoked if the AcessInfo object cached by the plugin is not valid. Thus plugins should always fetch a new AccessInfo when invoked. If you are looking to just retrieve the current auth data then you should use get_access.

Parameters
`session` (*`keystoneauth1.session.Session`*) A session object that can be used for communication.

Raises

- **`keystoneauth1.exceptions.response.InvalidResponse`** The response returned wasnt appropriate.
- **`keystoneauth1.exceptions.http.HttpError`** An error from an invalid HTTP response.

Returns
Token access information.

Return type
`keystoneauth1.access.AccessInfo`

`invalidate``()` → bool

Invalidate the current authentication data.

This should result in fetching a new token on next call.

A plugin may be invalidated if an Unauthorized HTTP response is returned to indicate that the token may have been revoked or is otherwise now invalid.

Returns
True if there was something that the plugin did to invalidate. This means that it makes sense to try again. If nothing happens returns False to indicate give up.

Return type
bool

keystoneauth1.identity.base module

class `keystoneauth1.identity.base.`**`BaseIdentityPlugin`**(*auth_url: str | None = None, reauthenticate: bool = True*)

Bases: `BaseAuthPlugin`

`MIN_TOKEN_LIFE_SECONDS: int = 120`

`__abstractmethods__ = frozenset({'get_auth_ref'})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': <class 'int'>, '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': typing.Optional[keystoneauth1.access.access.AccessInfo], 'auth_url': typing.Optional[str], 'reauthenticate': <class 'bool'>}`

`__doc__ = None`

`__init__`(*auth_url: str | None = None, reauthenticate: bool = True*) `__module__ = 'keystoneauth1.identity.base'`

`_abc_impl = <_abc._abc_data object>`

`_discovery_cache: dict[str, Discover]`

`_needs_reauthenticate()` → bool

Return if the existing token needs to be re-authenticated.

The token should be refreshed if it is about to expire.

Returns
True if the plugin should fetch a new token. False otherwise.

`auth_ref: AccessInfo | None`

`auth_url: str | None`

`get_access`(*session: Session*, ***kwargs: Any*) → AccessInfo

Fetch or return a current AccessInfo object.

If a valid AccessInfo is present then it is returned otherwise a new one will be fetched.

Parameters
`session` (*keystoneauth1.session.Session*) A session object that can be used for communication.

Raises
`keystoneauth1.exceptions.http.HttpError` An error from an invalid HTTP response.

Returns
Valid AccessInfo

Return type
`keystoneauth1.access.AccessInfo`

`get_all_version_data`(*session: Session*, *interface: str = 'public'*, *region_name: str | None = None*, *service_type: str | None = None*) → dict[str, dict[str, dict[str, list[VersionData]]]]

Get version data for all services in the catalog.

Parameters

- **session** (*keystoneauth1.session.Session*) A session object that can be used for communication.
- **interface** Type of endpoint to get version data for. Can be a single value or a list of values. A value of None indicates that all interfaces should be queried. (optional, defaults to public)
- **region_name** (*string*) Region of endpoints to get version data for. A valueof None indicates that all regions should be queried. (optional, defaults to None)
- **service_type** (*string*) Limit the version data to a single service. (optional, defaults to None)

Returns

A dictionary keyed by region_name with values containing dictionaries keyed by interface with values being a list of `VersionData`.

get_api_major_version(*session: Session*, *, *endpoint_override: str | None = None*, *service_type: str | None = None*, *interface: str | None = None*, *region_name: str | None = None*, *service_name: str | None = None*, *version: str | None = None*, *allow: dict[str, Any] | None = None*, *allow_version_hack: bool = True*, *skip_discovery: bool = False*, *discover_versions: bool = False*, *min_version: str | int | float | Iterable[str | int | float] | None = None*, *max_version: str | int | float | Iterable[str | int | float] | None = None*, ***kwargs: Any*) → tuple[int | float, ...] | None

Return the major API version for a service.

If a valid token is not present then a new one will be fetched using the session and kwargs.

version, min_version and max_version can all be given either as a string or a tuple.

Valid interface types: *public* or *publicURL*,
internal or *internalURL*, *admin* or adminURL'

Parameters

- **session** (*keystoneauth1.session.Session*) A session object that can be used for communication.
- **endpoint_override** (*str*) URL to use for version discovery.
- **service_type** (*string*) The type of service to lookup the endpoint for. This plugin will return None (failure) if service_type is not provided.
- **interface** Type of endpoint. Can be a single value or a list of values. If its a list of values, they will be looked for in order of preference. Can also be *keystoneauth1.plugin.AUTH_INTERFACE* to indicate that the auth_url should be used instead of the value in the catalog. (optional, defaults to public)
- **region_name** (*string*) The region the endpoint should exist in. (optional)
- **service_name** (*string*) The name of the service in the catalog. (optional)
- **version** The minimum version number required for this endpoint. (optional)
- **allow** (*dict*) Extra filters to pass when discovering API versions. (optional)
- **allow_version_hack** (*bool*) Allow keystoneauth to hack up catalog URLS to support older schemes. (optional, default True)
- **skip_discovery** (*bool*) Whether to skip version discovery even if a version has been given. This is useful if endpoint_override or similar has been given and grabbing additional information about the endpoint is not useful.
- **discover_versions** (*bool*) Whether to get version metadata from the version discovery document even if its not neccessary to fulfill the major version request. Defaults to False because get_endpoint doesnt need metadata. (optional, defaults to False)

- **min_version** The minimum version that is acceptable. Mutually exclusive with version. If min_version is given with no max_version it is as if max version is latest. (optional)
- **max_version** The maximum version that is acceptable. Mutually exclusive with version. If min_version is given with no max_version it is as if max version is latest. (optional)

Raises
`keystoneauth1.exceptions.http.HttpError` An error from an invalid HTTP response.

Returns
The major version of the API of the service discovered.

Return type
tuple or None

Note

Implementation notes follow. Users should not need to wrap their head around these implementation notes. *get_api_major_version* should do what is expected with the least possible cost while still consistently returning a value if possible.

There are many cases when major version can be satisfied without actually calling the discovery endpoint (like when the version is in the url). If the user has a cloud with the versioned endpoint `https://volume.example.com/v3` in the catalog for the `block-storage` service and they do:

```
client = adapter.Adapter(
    session,
    service_type='block-storage',
    min_version=2,
    max_version=3,
)
volume_version = client.get_api_major_version()
```

The version actually be returned with no api calls other than getting the token. For that reason, `get_api_major_version()` first calls `get_endpoint_data()` with `discover_versions=False`.

If their catalog has an unversioned endpoint `https://volume.example.com` for the `block-storage` service and they do this:

```
client = adapter.Adapter(session, service_type='block-storage')
```

client is now set up to use whatever is in the catalog. Since the url doesnt have a version, `get_endpoint_data()` with `discover_versions=False` will result in `api_version=None`. (No version was requested so it didnt need to do the round trip)

In order to find out what version the endpoint actually is, we must make a round trip. Therefore, if `api_version` is `None` after the first call, `get_api_major_version()` will make a second call to `get_endpoint_data()` with `discover_versions=True`.

abstract get_auth_ref(*session: Session*) → AccessInfo

Obtain a token from an OpenStack Identity Service.

This method is overridden by the various token version plugins.

This function should not be called independently and is expected to be invoked via the do_authenticate function.

This function will be invoked if the AcessInfo object cached by the plugin is not valid. Thus plugins should always fetch a new AccessInfo when invoked. If you are looking to just retrieve the current auth data then you should use get_access.

Parameters

session (*keystoneauth1.session.Session*) A session object that can be used for communication.

Raises

- **keystoneauth1.exceptions.response.InvalidResponse** The response returned wasnt appropriate.
- **keystoneauth1.exceptions.http.HttpError** An error from an invalid HTTP response.

Returns

Token access information.

Return type

`keystoneauth1.access.AccessInfo`

get_auth_state() → str | None

Retrieve the current authentication state for the plugin.

Retrieve any internal state that represents the authenticated plugin.

This should not fetch any new data if it is not present.

Returns

a string that can be stored or None if there is no auth state present in the plugin. This string can be reloaded with set_auth_state to set the same authentication.

Return type

str or None if no auth present.

get_cache_id() → str | None

Fetch an identifier that uniquely identifies the auth options.

The returned identifier need not be decomposable or otherwise provide any way to recreate the plugin.

This string MUST change if any of the parameters that are used to uniquely identity this plugin change. It should not change upon a reauthentication of the plugin.

Returns

A unique string for the set of options

Return type

str or None if this is unsupported or unavailable.

get_cache_id_elements() → dict[str, str | None]

Get the elements for this auth plugin that make it unique.

As part of the get_cache_id requirement we need to determine what aspects of this plugin and its values that make up the unique elements.

This should be overridden by plugins that wish to allow caching.

Returns
The unique attributes and values of this plugin.

Return type
A flat dict with a str key and str or None value. This is required as we feed these values into a hash. Pairs where the value is None are ignored in the hashed id.

get_discovery(*session: Session*, *url: str*, *authenticated: bool | None = None*) → Discover

Return the discovery object for a URL.

Check the session and the plugin cache to see if we have already performed discovery on the URL and if so return it, otherwise create a new discovery object, cache it and return it.

This function is expected to be used by subclasses and should not be needed by users.

Parameters

- **session** (*keystoneauth1.session.Session*) A session object to discover with.
- **url** (*str*) The url to lookup.
- **authenticated** (*bool*) Include a token in the discovery call. (optional) Defaults to None (use a token if a plugin is installed).

Raises

- **keystoneauth1.exceptions.discovery.DiscoveryFailure** if for some reason the lookup fails.
- **keystoneauth1.exceptions.http.HttpError** An error from an invalid HTTP response.

Returns
A discovery object with the results of looking up that URL.

get_endpoint(*session: Session*, *service_type: str | None = None*, *interface: str | None = None*, *region_name: str | None = None*, *service_name: str | None = None*, *version: str | int | float | Iterable[str | int | float] | None = None*, *allow: dict[str, Any] | None = None*, *allow_version_hack: bool = True*, *skip_discovery: bool = False*, *min_version: str | int | float | Iterable[str | int | float] | None = None*, *max_version: str | int | float | Iterable[str | int | float] | None = None*, ***kwargs: Any*) → str | None

Return a valid endpoint for a service.

If a valid token is not present then a new one will be fetched using the session and kwargs.

version, min_version and max_version can all be given either as a string or a tuple.

Valid interface types: *public* or *publicURL*,
internal or *internalURL*, *admin* or adminURL'

Parameters

- **session** (*keystoneauth1.session.Session*) A session object that can be used for communication.
- **service_type** (*string*) The type of service to lookup the endpoint for. This plugin will return None (failure) if service_type is not provided.
- **interface** Type of endpoint. Can be a single value or a list of values. If its a list of values, they will be looked for in order of preference. Can also be *keystoneauth1.plugin.AUTH_INTERFACE* to indicate that the auth_url should be used instead of the value in the catalog. (optional, defaults to public)
- **region_name** (*string*) The region the endpoint should exist in. (optional)
- **service_name** (*string*) The name of the service in the catalog. (optional)
- **version** The minimum version number required for this endpoint. (optional)
- **allow** (*dict*) Extra filters to pass when discovering API versions. (optional)
- **allow_version_hack** (*bool*) Allow keystoneauth to hack up catalog URLS to support older schemes. (optional, default True)
- **skip_discovery** (*bool*) Whether to skip version discovery even if a version has been given. This is useful if endpoint_override or similar has been given and grabbing additional information about the endpoint is not useful.
- **min_version** The minimum version that is acceptable. Mutually exclusive with version. If min_version is given with no max_version it is as if max version is latest. (optional)
- **max_version** The maximum version that is acceptable. Mutually exclusive with version. If min_version is given with no max_version it is as if max version is latest. (optional)

Raises

`keystoneauth1.exceptions.http.HttpError` An error from an invalid HTTP response.

Returns

A valid endpoint URL or None if not available.

Return type

string or None

get_endpoint_data(*session: Session*, ***, *endpoint_override: str | None = None*, *discover_versions: bool = True*, *service_type: str | None = None*, *interface: str | None = None*, *region_name: str | None = None*, *service_name: str | None = None*, *allow: dict[str, Any] | None = None*, *allow_version_hack: bool = True*, *skip_discovery: bool = False*, *min_version: str | int | float | Iterable[str | int | float] | None = None*, *max_version: str | int | float | Iterable[str | int | float] | None = None*, ***kwargs: Any*) → EndpointData | None

Return a valid endpoint data for a service.

If a valid token is not present then a new one will be fetched using the session and kwargs.

version, min_version and max_version can all be given either as a string or a tuple.

Valid interface types: *public* or *publicURL*,
internal or *internalURL*, *admin* or adminURL'

Parameters

- **session** (*keystoneauth1.session.Session*) A session object that can be used for communication.
- **endpoint_override** (*str*) URL to use instead of looking in the catalog. Catalog lookup will be skipped, but version discovery will be run. Sets allow_version_hack to False (optional)
- **discover_versions** (*bool*) Whether to get version metadata from the version discovery document even if its not neccessary to fulfill the major version request. (optional, defaults to True)
- **service_type** (*string*) The type of service to lookup the endpoint for. This plugin will return None (failure) if service_type is not provided.
- **interface** Type of endpoint. Can be a single value or a list of values. If its a list of values, they will be looked for in order of preference. Can also be *keystoneauth1.plugin.AUTH_INTERFACE* to indicate that the auth_url should be used instead of the value in the catalog. (optional, defaults to public)
- **region_name** (*string*) The region the endpoint should exist in. (optional)
- **service_name** (*string*) The name of the service in the catalog. (optional)
- **allow** (*dict*) Extra filters to pass when discovering API versions. (optional)
- **allow_version_hack** (*bool*) Allow keystoneauth to hack up catalog URLS to support older schemes. (optional, default True)
- **skip_discovery** (*bool*) Whether to skip version discovery even if a version has been given. This is useful if endpoint_override or similar has been given and grabbing additional information about the endpoint is not useful.
- **min_version** The minimum version that is acceptable. Mutually exclusive with version. If min_version is given with no max_version it is as if max version is latest. (optional)
- **max_version** The maximum version that is acceptable. Mutually exclusive with version. If min_version is given with no max_version it is as if max version is latest. (optional)
- **kwargs** Ignored.

Raises
`keystoneauth1.exceptions.http.HttpError` An error from an invalid HTTP response.

Returns
Valid EndpointData or None if not available.

Return type
keystoneauth1.discover.EndpointData or None

get_project_id(*session: Session*) → str | None

Return the project id that we are authenticated to.

Wherever possible the project id should be inferred from the token however there are certain URLs and other places that require access to the currently authenticated project id.

Parameters

session (*keystoneauth1.session.Session*) A session object so the plugin can make HTTP calls.

Returns

A project identifier or None if one is not available.

Return type

str

get_sp_auth_url(*session: Session*, *sp_id: str*) → str | None

Return auth_url from the Service Provider object.

This url is used for obtaining unscoped federated token from remote cloud.

Parameters

sp_id (*string*) ID of the Service Provider to be queried.

Returns

A Service Provider auth_url or None if one is not available.

Return type

str

get_sp_url(*session: Session*, *sp_id: str*) → str | None

Return sp_url from the Service Provider object.

This url is used for passing SAML2 assertion to the remote cloud.

Parameters

sp_id (*str*) ID of the Service Provider to be queried.

Returns

A Service Provider sp_url or None if one is not available.

Return type

str

get_token(*session: Session*) → str | None

Return a valid auth token.

If a valid token is not present then a new one will be fetched.

Parameters

session (*keystoneauth1.session.Session*) A session object that can be used for communication.

Raises

`keystoneauth1.exceptions.http.HttpError` An error from an invalid HTTP response.

Returns

A valid token.

Return type
string

get_user_id(*session: Session*) → str | None

Return a unique user identifier of the plugin.

Wherever possible the user id should be inferred from the token however there are certain URLs and other places that require access to the currently authenticated user id.

Parameters
session (*keystoneauth1.session.Session*) A session object so the plugin can make HTTP calls.

Returns
A user identifier or None if one is not available.

Return type
str

invalidate() → bool

Invalidate the current authentication data.

This should result in fetching a new token on next call.

A plugin may be invalidated if an Unauthorized HTTP response is returned to indicate that the token may have been revoked or is otherwise now invalid.

Returns
True if there was something that the plugin did to invalidate. This means that it makes sense to try again. If nothing happens returns False to indicate give up.

Return type
bool

reauthenticate: bool

set_auth_state(*data: str*) → None

Install existing authentication state for a plugin.

Take the output of get_auth_state and install that authentication state into the current authentication plugin.

keystoneauth1.identity.v2 module

class keystoneauth1.identity.v2.**Auth**(*auth_url: str*, **, *trust_id: str* | *None* = *None*, *tenant_id: str* | *None* = *None*, *tenant_name: str* | *None* = *None*, *reauthenticate: bool* = *True*)

Bases: BaseIdentityPlugin

Identity V2 Authentication Plugin.

Parameters

- **auth_url** (*string*) Identity service endpoint for authorization.
- **trust_id** (*string*) Trust ID for trust scoping.
- **tenant_id** (*string*) Tenant ID for project scoping.
- **tenant_name** (*string*) Tenant name for project scoping.

- **reauthenticate** (*bool*) Allow fetching a new token if the current one is going to expire. (optional) default True

`__abstractmethods__ = frozenset({'get_auth_data'})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': <class 'str'>, 'reauthenticate': 'bool'}`

`__doc__ = 'Identity V2 Authentication Plugin.\n\n :param string auth_url: Identity service endpoint for authorization.\n :param string trust_id: Trust ID for trust scoping.\n :param string tenant_id: Tenant ID for project scoping.\n :param string tenant_name: Tenant name for project scoping.\n :param bool reauthenticate: Allow fetching a new token if the current one\n is going to expire. (optional) default True\n '`

`__init__`(*auth_url: str*, ***, *trust_id: str | None = None*, *tenant_id: str | None = None*, *tenant_name: str | None = None*, *reauthenticate: bool = True*)

`__module__ = 'keystoneauth1.identity.v2'`

`_abc_impl = <_abc._abc_data object>`

`auth_url: str`

`abstract get_auth_data`(*headers: MutableMapping[str, str] | None = None*) → dict[str, object]

Return the authentication section of an auth plugin.

Parameters

headers (*dict*) The headers that will be sent with the auth request if a plugin needs to add to them.

Returns

A dict of authentication data for the auth type.

Return type

dict

`get_auth_ref`(*session: Session*) → AccessInfoV2

Obtain a token from an OpenStack Identity Service.

This method is overridden by the various token version plugins.

This function should not be called independently and is expected to be invoked via the do_authenticate function.

This function will be invoked if the AcessInfo object cached by the plugin is not valid. Thus plugins should always fetch a new AccessInfo when invoked. If you are looking to just retrieve the current auth data then you should use get_access.

Parameters

session (`keystoneauth1.session.Session`) A session object that can be used for communication.

Raises

- **keystoneauth1.exceptions.response.InvalidResponse** The response returned wasnt appropriate.
- **keystoneauth1.exceptions.http.HttpError** An error from an invalid HTTP response.

Returns
Token access information.

Return type
`keystoneauth1.access.AccessInfo`

property has_scope_parameters: bool

Return true if parameters can be used to create a scoped token.

class keystoneauth1.identity.v2.**Password**(*auth_url: str, username: str | None | Unset = Unset.token, password: str | None = None, user_id: str | None | Unset = Unset.token, *, trust_id: str | None = None, tenant_id: str | None = None, tenant_name: str | None = None, reauthenticate: bool = True*)

Bases: `Auth`

A plugin for authenticating with a username and password.

A username or user_id must be provided.

Parameters

- **auth_url** (*string*) Identity service endpoint for authorization.
- **username** (*string*) Username for authentication.
- **password** (*string*) Password for authentication.
- **user_id** (*string*) User ID for authentication.
- **trust_id** (*string*) Trust ID for trust scoping.
- **tenant_id** (*string*) Tenant ID for tenant scoping.
- **tenant_name** (*string*) Tenant name for tenant scoping.
- **reauthenticate** (*bool*) Allow fetching a new token if the current one is going to expire. (optional) default True

Raises
TypeError if a user_id or username is not provided.

__abstractmethods__ = frozenset({})

__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}

__doc__ = 'A plugin for authenticating with a username and password.\n\n A username or user_id must be provided.\n\n :param string auth_url: Identity service endpoint for authorization.\n :param string username: Username for authentication.\n :param string password: Password for authentication.\n :param string user_id: User ID for authentication.\n :param string trust_id: Trust ID for trust scoping.\n :param string tenant_id: Tenant ID for tenant scoping.\n :param string tenant_name: Tenant name for tenant scoping.\n :param bool reauthenticate: Allow fetching a new token if the current one\n is going to expire. (optional) default True\n\n :raises TypeError: if a user_id or username is not provided.\n '

__init__(*auth_url: str, username: str | None | Unset = Unset.token, password: str | None = None, user_id: str | None | Unset = Unset.token, *, trust_id: str | None = None, tenant_id: str | None = None, tenant_name: str | None = None, reauthenticate: bool = True*)

__module__ = 'keystoneauth1.identity.v2'

_abc_impl = <_abc._abc_data object>

get_auth_data(*headers: MutableMapping[str, str] | None = None*) → dict[str, object]

Return the authentication section of an auth plugin.

Parameters
headers (*dict*) The headers that will be sent with the auth request if a plugin needs to add to them.

Returns
A dict of authentication data for the auth type.

Return type
dict

get_cache_id_elements() → dict[str, str | None]

Get the elements for this auth plugin that make it unique.

As part of the get_cache_id requirement we need to determine what aspects of this plugin and its values that make up the unique elements.

This should be overridden by plugins that wish to allow caching.

Returns
The unique attributes and values of this plugin.

Return type
A flat dict with a str key and str or None value. This is required as we feed these values into a hash. Pairs where the value is None are ignored in the hashed id.

class keystoneauth1.identity.v2.**Token**(*auth_url: str, token: str, *, trust_id: str | None = None, tenant_id: str | None = None, tenant_name: str | None = None, reauthenticate: bool = True*)

Bases: Auth

A plugin for authenticating with an existing token.

Parameters

- **auth_url** (*string*) Identity service endpoint for authorization.
- **token** (*string*) Existing token for authentication.
- **tenant_id** (*string*) Tenant ID for tenant scoping.
- **tenant_name** (*string*) Tenant name for tenant scoping.
- **trust_id** (*string*) Trust ID for trust scoping.
- **reauthenticate** (*bool*) Allow fetching a new token if the current one is going to expire. (optional) default True

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}`

`__doc__ = 'A plugin for authenticating with an existing token.\n\n :param string auth_url: Identity service endpoint for authorization.\n :param string token: Existing token for authentication.\n :param string tenant_id: Tenant ID for tenant scoping.\n :param string tenant_name: Tenant name for tenant scoping.\n :param string trust_id: Trust ID for trust scoping.\n :param bool reauthenticate: Allow fetching a new token if the current one\n is going to expire. (optional) default True\n '`

`__init__`(*auth_url: str*, *token: str*, ***, *trust_id: str | None = None*, *tenant_id: str | None = None*, *tenant_name: str | None = None*, *reauthenticate: bool = True*)

`__module__ = 'keystoneauth1.identity.v2'`

`_abc_impl = <_abc._abc_data object>`

`get_auth_data`(*headers: MutableMapping[str, str] | None = None*) → dict[str, object]

Return the authentication section of an auth plugin.

Parameters

headers (*dict*) The headers that will be sent with the auth request if a plugin needs to add to them.

Returns

A dict of authentication data for the auth type.

Return type

dict

`get_cache_id_elements`() → dict[str, str | None]

Get the elements for this auth plugin that make it unique.

As part of the get_cache_id requirement we need to determine what aspects of this plugin and its values that make up the unique elements.

This should be overridden by plugins that wish to allow caching.

Returns

The unique attributes and values of this plugin.

Return type
A flat dict with a str key and str or None value. This is required as we feed these values into a hash. Pairs where the value is None are ignored in the hashed id.

`class keystoneauth1.identity.v2.`**`Unset`**(*value*)

Bases: `Enum`

An enumeration.

`__doc__ = 'An enumeration.'`

`__module__ = 'keystoneauth1.identity.v2'`

`token = 0`

Module contents

`class keystoneauth1.identity.`**`BaseIdentityPlugin`**(*auth_url: str | None = None, reauthenticate: bool = True*)

Bases: `BaseAuthPlugin`

`MIN_TOKEN_LIFE_SECONDS: int = 120`

`__abstractmethods__ = frozenset({'get_auth_ref'})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': <class 'int'>, '_discovery_cache': 'dict[str, discover.Discover]', 'auth_ref': typing.Optional[keystoneauth1.access.access.AccessInfo], 'auth_url': typing.Optional[str], 'reauthenticate': <class 'bool'>}`

`__doc__ = None`

`__init__`(*auth_url: str | None = None, reauthenticate: bool = True*)

`__module__ = 'keystoneauth1.identity.base'`

`_abc_impl = <_abc._abc_data object>`

`_needs_reauthenticate`() → bool

Return if the existing token needs to be re-authenticated.

The token should be refreshed if it is about to expire.

Returns
True if the plugin should fetch a new token. False otherwise.

`auth_ref: AccessInfo | None`

`auth_url: str | None`

`get_access`(*session: Session, **kwargs: Any*) → AccessInfo

Fetch or return a current AccessInfo object.

If a valid AccessInfo is present then it is returned otherwise a new one will be fetched.

Parameters
session (*keystoneauth1.session.Session*) A session object that can be used for communication.

Raises
`keystoneauth1.exceptions.http.HttpError` An error from an invalid HTTP response.

Returns
Valid AccessInfo

Return type
`keystoneauth1.access.AccessInfo`

get_all_version_data(*session: Session*, *interface: str = 'public'*, *region_name: str | None = None*, *service_type: str | None = None*) → dict[str, dict[str, dict[str, list[VersionData]]]]

Get version data for all services in the catalog.

Parameters

- **session** (*keystoneauth1.session.Session*) A session object that can be used for communication.
- **interface** Type of endpoint to get version data for. Can be a single value or a list of values. A value of None indicates that all interfaces should be queried. (optional, defaults to public)
- **region_name** (*string*) Region of endpoints to get version data for. A valueof None indicates that all regions should be queried. (optional, defaults to None)
- **service_type** (*string*) Limit the version data to a single service. (optional, defaults to None)

Returns
A dictionary keyed by region_name with values containing dictionaries keyed by interface with values being a list of `VersionData`.

get_api_major_version(*session: Session*, ***, *endpoint_override: str | None = None*, *service_type: str | None = None*, *interface: str | None = None*, *region_name: str | None = None*, *service_name: str | None = None*, *version: str | None = None*, *allow: dict[str, Any] | None = None*, *allow_version_hack: bool = True*, *skip_discovery: bool = False*, *discover_versions: bool = False*, *min_version: str | int | float | Iterable[str | int | float] | None = None*, *max_version: str | int | float | Iterable[str | int | float] | None = None*, ***kwargs: Any*) → tuple[int | float, ...] | None

Return the major API version for a service.

If a valid token is not present then a new one will be fetched using the session and kwargs.

version, min_version and max_version can all be given either as a string or a tuple.

Valid interface types: *public* or *publicURL*,
internal or *internalURL*, *admin* or adminURL'

Parameters

- **session** (*keystoneauth1.session.Session*) A session object that can be used for communication.

- **endpoint_override** (*str*) URL to use for version discovery.
- **service_type** (*string*) The type of service to lookup the endpoint for. This plugin will return None (failure) if service_type is not provided.
- **interface** Type of endpoint. Can be a single value or a list of values. If its a list of values, they will be looked for in order of preference. Can also be *keystoneauth1.plugin.AUTH_INTERFACE* to indicate that the auth_url should be used instead of the value in the catalog. (optional, defaults to public)
- **region_name** (*string*) The region the endpoint should exist in. (optional)
- **service_name** (*string*) The name of the service in the catalog. (optional)
- **version** The minimum version number required for this endpoint. (optional)
- **allow** (*dict*) Extra filters to pass when discovering API versions. (optional)
- **allow_version_hack** (*bool*) Allow keystoneauth to hack up catalog URLS to support older schemes. (optional, default True)
- **skip_discovery** (*bool*) Whether to skip version discovery even if a version has been given. This is useful if endpoint_override or similar has been given and grabbing additional information about the endpoint is not useful.
- **discover_versions** (*bool*) Whether to get version metadata from the version discovery document even if its not neccessary to fulfill the major version request. Defaults to False because get_endpoint doesnt need metadata. (optional, defaults to False)
- **min_version** The minimum version that is acceptable. Mutually exclusive with version. If min_version is given with no max_version it is as if max version is latest. (optional)
- **max_version** The maximum version that is acceptable. Mutually exclusive with version. If min_version is given with no max_version it is as if max version is latest. (optional)

Raises

`keystoneauth1.exceptions.http.HttpError` An error from an invalid HTTP response.

Returns

The major version of the API of the service discovered.

Return type

tuple or None

Note

Implementation notes follow. Users should not need to wrap their head around these implementation notes. *get_api_major_version* should do what is expected with the least possible cost while still consistently returning a value if possible.

There are many cases when major version can be satisfied without actually calling the discovery endpoint (like when the version is in the url). If the user has a cloud with the versioned endpoint `https://volume.example.com/v3` in the catalog for the `block-storage` service and they do:

```
client = adapter.Adapter(
    session,
    service_type='block-storage',
    min_version=2,
    max_version=3,
)
volume_version = client.get_api_major_version()
```

The version actually be returned with no api calls other than getting the token. For that reason, `get_api_major_version()` first calls `get_endpoint_data()` with `discover_versions=False`.

If their catalog has an unversioned endpoint `https://volume.example.com` for the `block-storage` service and they do this:

```
client = adapter.Adapter(session, service_type='block-storage')
```

client is now set up to use whatever is in the catalog. Since the url doesnt have a version, `get_endpoint_data()` with `discover_versions=False` will result in `api_version=None`. (No version was requested so it didnt need to do the round trip)

In order to find out what version the endpoint actually is, we must make a round trip. Therefore, if `api_version` is `None` after the first call, `get_api_major_version()` will make a second call to `get_endpoint_data()` with `discover_versions=True`.

abstract get_auth_ref(*session: Session*) → AccessInfo

Obtain a token from an OpenStack Identity Service.

This method is overridden by the various token version plugins.

This function should not be called independently and is expected to be invoked via the do_authenticate function.

This function will be invoked if the AcessInfo object cached by the plugin is not valid. Thus plugins should always fetch a new AccessInfo when invoked. If you are looking to just retrieve the current auth data then you should use get_access.

Parameters

session (*keystoneauth1.session.Session*) A session object that can be used for communication.

Raises

- **keystoneauth1.exceptions.response.InvalidResponse** The response returned wasnt appropriate.
- **keystoneauth1.exceptions.http.HttpError** An error from an invalid HTTP response.

Returns

Token access information.

Return type
`keystoneauth1.access.AccessInfo`

`get_auth_state()` → str | None

Retrieve the current authentication state for the plugin.

Retrieve any internal state that represents the authenticated plugin.

This should not fetch any new data if it is not present.

Returns
a string that can be stored or None if there is no auth state present in the plugin. This string can be reloaded with set_auth_state to set the same authentication.

Return type
str or None if no auth present.

`get_cache_id()` → str | None

Fetch an identifier that uniquely identifies the auth options.

The returned identifier need not be decomposable or otherwise provide any way to recreate the plugin.

This string MUST change if any of the parameters that are used to uniquely identity this plugin change. It should not change upon a reauthentication of the plugin.

Returns
A unique string for the set of options

Return type
str or None if this is unsupported or unavailable.

`get_cache_id_elements()` → dict[str, str | None]

Get the elements for this auth plugin that make it unique.

As part of the get_cache_id requirement we need to determine what aspects of this plugin and its values that make up the unique elements.

This should be overridden by plugins that wish to allow caching.

Returns
The unique attributes and values of this plugin.

Return type
A flat dict with a str key and str or None value. This is required as we feed these values into a hash. Pairs where the value is None are ignored in the hashed id.

`get_discovery`(*session: Session*, *url: str*, *authenticated: bool | None = None*) → Discover

Return the discovery object for a URL.

Check the session and the plugin cache to see if we have already performed discovery on the URL and if so return it, otherwise create a new discovery object, cache it and return it.

This function is expected to be used by subclasses and should not be needed by users.

Parameters

- **session** (*keystoneauth1.session.Session*) A session object to discover with.
- **url** (*str*) The url to lookup.

- **authenticated** (*bool*) Include a token in the discovery call. (optional) Defaults to None (use a token if a plugin is installed).

Raises

- **keystoneauth1.exceptions.discovery.DiscoveryFailure** if for some reason the lookup fails.
- **keystoneauth1.exceptions.http.HttpError** An error from an invalid HTTP response.

Returns

A discovery object with the results of looking up that URL.

get_endpoint(*session: Session*, *service_type: str | None = None*, *interface: str | None = None*, *region_name: str | None = None*, *service_name: str | None = None*, *version: str | int | float | Iterable[str | int | float] | None = None*, *allow: dict[str, Any] | None = None*, *allow_version_hack: bool = True*, *skip_discovery: bool = False*, *min_version: str | int | float | Iterable[str | int | float] | None = None*, *max_version: str | int | float | Iterable[str | int | float] | None = None*, ***kwargs: Any*) → str | None

Return a valid endpoint for a service.

If a valid token is not present then a new one will be fetched using the session and kwargs.

version, min_version and max_version can all be given either as a string or a tuple.

Valid interface types: *public* or *publicURL*,
internal or *internalURL*, *admin* or adminURL'

Parameters

- **session** (*keystoneauth1.session.Session*) A session object that can be used for communication.
- **service_type** (*string*) The type of service to lookup the endpoint for. This plugin will return None (failure) if service_type is not provided.
- **interface** Type of endpoint. Can be a single value or a list of values. If its a list of values, they will be looked for in order of preference. Can also be *keystoneauth1.plugin.AUTH_INTERFACE* to indicate that the auth_url should be used instead of the value in the catalog. (optional, defaults to public)
- **region_name** (*string*) The region the endpoint should exist in. (optional)
- **service_name** (*string*) The name of the service in the catalog. (optional)
- **version** The minimum version number required for this endpoint. (optional)
- **allow** (*dict*) Extra filters to pass when discovering API versions. (optional)
- **allow_version_hack** (*bool*) Allow keystoneauth to hack up catalog URLS to support older schemes. (optional, default True)
- **skip_discovery** (*bool*) Whether to skip version discovery even if a version has been given. This is useful if endpoint_override or similar has been given and grabbing additional information about the endpoint is not useful.

- **min_version** The minimum version that is acceptable. Mutually exclusive with version. If min_version is given with no max_version it is as if max version is latest. (optional)
- **max_version** The maximum version that is acceptable. Mutually exclusive with version. If min_version is given with no max_version it is as if max version is latest. (optional)

Raises
`keystoneauth1.exceptions.http.HttpError` An error from an invalid HTTP response.

Returns
A valid endpoint URL or None if not available.

Return type
string or None

get_endpoint_data(*session: Session*, ***, *endpoint_override: str | None = None*, *discover_versions: bool = True*, *service_type: str | None = None*, *interface: str | None = None*, *region_name: str | None = None*, *service_name: str | None = None*, *allow: dict[str, Any] | None = None*, *allow_version_hack: bool = True*, *skip_discovery: bool = False*, *min_version: str | int | float | Iterable[str | int | float] | None = None*, *max_version: str | int | float | Iterable[str | int | float] | None = None*, ***kwargs: Any*) → EndpointData | None

Return a valid endpoint data for a service.

If a valid token is not present then a new one will be fetched using the session and kwargs.

version, min_version and max_version can all be given either as a string or a tuple.

Valid interface types: *public* or *publicURL*,
internal or *internalURL*, *admin* or adminURL‘

Parameters

- **session** (*keystoneauth1.session.Session*) A session object that can be used for communication.
- **endpoint_override** (*str*) URL to use instead of looking in the catalog. Catalog lookup will be skipped, but version discovery will be run. Sets allow_version_hack to False (optional)
- **discover_versions** (*bool*) Whether to get version metadata from the version discovery document even if its not neccessary to fulfill the major version request. (optional, defaults to True)
- **service_type** (*string*) The type of service to lookup the endpoint for. This plugin will return None (failure) if service_type is not provided.
- **interface** Type of endpoint. Can be a single value or a list of values. If its a list of values, they will be looked for in order of preference. Can also be *keystoneauth1.plugin.AUTH_INTERFACE* to indicate that the auth_url should be used instead of the value in the catalog. (optional, defaults to public)
- **region_name** (*string*) The region the endpoint should exist in. (optional)

- **service_name** (*string*) The name of the service in the catalog. (optional)
- **allow** (*dict*) Extra filters to pass when discovering API versions. (optional)
- **allow_version_hack** (*bool*) Allow keystoneauth to hack up catalog URLS to support older schemes. (optional, default True)
- **skip_discovery** (*bool*) Whether to skip version discovery even if a version has been given. This is useful if endpoint_override or similar has been given and grabbing additional information about the endpoint is not useful.
- **min_version** The minimum version that is acceptable. Mutually exclusive with version. If min_version is given with no max_version it is as if max version is latest. (optional)
- **max_version** The maximum version that is acceptable. Mutually exclusive with version. If min_version is given with no max_version it is as if max version is latest. (optional)
- **kwargs** Ignored.

Raises
`keystoneauth1.exceptions.http.HttpError` An error from an invalid HTTP response.

Returns
Valid EndpointData or None if not available.

Return type
keystoneauth1.discover.EndpointData or None

get_project_id(*session: Session*) → str | None

Return the project id that we are authenticated to.

Wherever possible the project id should be inferred from the token however there are certain URLs and other places that require access to the currently authenticated project id.

Parameters
session (*keystoneauth1.session.Session*) A session object so the plugin can make HTTP calls.

Returns
A project identifier or None if one is not available.

Return type
str

get_sp_auth_url(*session: Session*, *sp_id: str*) → str | None

Return auth_url from the Service Provider object.

This url is used for obtaining unscoped federated token from remote cloud.

Parameters
sp_id (*string*) ID of the Service Provider to be queried.

Returns
A Service Provider auth_url or None if one is not available.

Return type
str

get_sp_url(*session: Session*, *sp_id: str*) → str | None

Return sp_url from the Service Provider object.

This url is used for passing SAML2 assertion to the remote cloud.

Parameters
sp_id (*str*) ID of the Service Provider to be queried.

Returns
A Service Provider sp_url or None if one is not available.

Return type
str

get_token(*session: Session*) → str | None

Return a valid auth token.

If a valid token is not present then a new one will be fetched.

Parameters
session (*keystoneauth1.session.Session*) A session object that can be used for communication.

Raises
keystoneauth1.exceptions.http.HttpError An error from an invalid HTTP response.

Returns
A valid token.

Return type
string

get_user_id(*session: Session*) → str | None

Return a unique user identifier of the plugin.

Wherever possible the user id should be inferred from the token however there are certain URLs and other places that require access to the currently authenticated user id.

Parameters
session (*keystoneauth1.session.Session*) A session object so the plugin can make HTTP calls.

Returns
A user identifier or None if one is not available.

Return type
str

invalidate() → bool

Invalidate the current authentication data.

This should result in fetching a new token on next call.

A plugin may be invalidated if an Unauthorized HTTP response is returned to indicate that the token may have been revoked or is otherwise now invalid.

Returns
True if there was something that the plugin did to invalidate. This means that it makes sense to try again. If nothing happens returns False to indicate give up.

Return type
bool

reauthenticate: bool

set_auth_state(*data: str*) → None

Install existing authentication state for a plugin.

Take the output of get_auth_state and install that authentication state into the current authentication plugin.

class keystoneauth1.identity.Password(*auth_url: str*, *username: str | None = None*, *user_id: str | None = None*, *password: str | None = None*, *user_domain_id: str | None = None*, *user_domain_name: str | None = None*, ***, *tenant_id: str | None = None*, *tenant_name: str | None = None*, *project_id: str | None = None*, *project_name: str | None = None*, *project_domain_id: str | None = None*, *project_domain_name: str | None = None*, *domain_id: str | None = None*, *domain_name: str | None = None*, *system_scope: str | None = None*, *trust_id: str | None = None*, *default_domain_id: str | None = None*, *default_domain_name: str | None = None*, *reauthenticate: bool = True*)

Bases: `BaseGenericPlugin`

A common user/password authentication plugin.

Parameters

- **username** (*string*) Username for authentication.
- **user_id** (*string*) User ID for authentication.
- **password** (*string*) Password for authentication.
- **user_domain_id** (*string*) Users domain ID for authentication.
- **user_domain_name** (*string*) Users domain name for authentication.

__abstractmethods__ = frozenset({})

__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', '_plugin': 'ty.Union[v2.Auth, v3.Auth, None]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}

__doc__ = "A common user/password authentication plugin.\n\n :param string username: Username for authentication.\n :param string user_id: User ID for authentication.\n :param string password: Password for authentication.\n :param string user_domain_id: User's domain ID for authentication.\n :param string user_domain_name: User's domain name for authentication.\n "

`__init__`(*auth_url: str, username: str | None = None, user_id: str | None = None, password: str | None = None, user_domain_id: str | None = None, user_domain_name: str | None = None, *, tenant_id: str | None = None, tenant_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, domain_id: str | None = None, domain_name: str | None = None, system_scope: str | None = None, trust_id: str | None = None, default_domain_id: str | None = None, default_domain_name: str | None = None, reauthenticate: bool = True*)

`__module__ = 'keystoneauth1.identity.generic.password'`

`_abc_impl = <_abc._abc_data object>`

`create_plugin`(*session: Session, version: tuple[int | float, ...], url: str, raw_status: str | None = None*) → None | Password | Password

Create a plugin from the given parameters.

This function will be called multiple times with the version and url of a potential endpoint. If a plugin can be constructed that fits the params then it should return it. If not return None and then another call will be made with other available URLs.

Parameters

- **`session`** (*`keystoneauth1.session.Session`*) A session object.
- **`version`** (*`tuple`*) A tuple of the API version at the URL.
- **`url`** (*`str`*) The base URL for this version.
- **`raw_status`** (*`str`*) The status that was in the discovery field.

Returns

A plugin that can match the parameters or None if nothing.

`get_cache_id_elements`() → dict[str, str | None]

Get the elements for this auth plugin that make it unique.

As part of the get_cache_id requirement we need to determine what aspects of this plugin and its values that make up the unique elements.

This should be overridden by plugins that wish to allow caching.

Returns

The unique attributes and values of this plugin.

Return type

A flat dict with a str key and str or None value. This is required as we feed these values into a hash. Pairs where the value is None are ignored in the hashed id.

`property user_domain_id: str | None`

`property user_domain_name: str | None`

class keystoneauth1.identity.**Token**(*auth_url: str, token: str, *, tenant_id: str | None = None, tenant_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, domain_id: str | None = None, domain_name: str | None = None, system_scope: str | None = None, trust_id: str | None = None, default_domain_id: str | None = None, default_domain_name: str | None = None, reauthenticate: bool = True*)

Bases: `BaseGenericPlugin`

Generic token auth plugin.

Parameters
token (*string*) Token for authentication.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {'MIN_TOKEN_LIFE_SECONDS': 'int', '_discovery_cache': 'dict[str, discover.Discover]', '_plugin': 'ty.Union[v2.Auth, v3.Auth, None]', 'auth_ref': 'ty.Optional[access.AccessInfo]', 'auth_url': 'str', 'reauthenticate': 'bool'}`

`__doc__ = 'Generic token auth plugin.\n\n :param string token: Token for authentication.\n '`

`__init__`(*auth_url: str, token: str, *, tenant_id: str | None = None, tenant_name: str | None = None, project_id: str | None = None, project_name: str | None = None, project_domain_id: str | None = None, project_domain_name: str | None = None, domain_id: str | None = None, domain_name: str | None = None, system_scope: str | None = None, trust_id: str | None = None, default_domain_id: str | None = None, default_domain_name: str | None = None, reauthenticate: bool = True*)

`__module__ = 'keystoneauth1.identity.generic.token'`

`_abc_impl = <_abc._abc_data object>`

`create_plugin`(*session: Session, version: tuple[int | float, ...], url: str, raw_status: str | None = None*) → None | Token | Token

Create a plugin from the given parameters.

This function will be called multiple times with the version and url of a potential endpoint. If a plugin can be constructed that fits the params then it should return it. If not return None and then another call will be made with other available URLs.

Parameters

- **session** (*keystoneauth1.session.Session*) A session object.
- **version** (*tuple*) A tuple of the API version at the URL.
- **url** (*str*) The base URL for this version.
- **raw_status** (*str*) The status that was in the discovery field.

Returns
A plugin that can match the parameters or None if nothing.

get_cache_id_elements() → dict[str, str | None]

Get the elements for this auth plugin that make it unique.

As part of the get_cache_id requirement we need to determine what aspects of this plugin and its values that make up the unique elements.

This should be overridden by plugins that wish to allow caching.

Returns
The unique attributes and values of this plugin.

Return type
A flat dict with a str key and str or None value. This is required as we feed these values into a hash. Pairs where the value is None are ignored in the hashed id.

keystoneauth1.identity.**V2Password**
See keystoneauth1.identity.v2.Password

keystoneauth1.identity.**V2Token**
See keystoneauth1.identity.v2.Token

keystoneauth1.identity.**V3ApplicationCredential**
See keystoneauth1.identity.v3.ApplicationCredential

keystoneauth1.identity.**V3MultiFactor**
See keystoneauth1.identity.v3.MultiFactor

keystoneauth1.identity.**V3OAuth2ClientCredential**
See keystoneauth1.identity.v3.OAuth2ClientCredential

keystoneauth1.identity.**V3OAuth2mTlsClientCredential**
See keystoneauth1.identity.v3.OAuth2mTlsClientCredential

keystoneauth1.identity.**V3OidcAccessToken**
See keystoneauth1.identity.v3.oidc.OidcAccessToken

keystoneauth1.identity.**V3OidcAuthorizationCode**
See keystoneauth1.identity.v3.oidc.OidcAuthorizationCode

keystoneauth1.identity.**V3OidcClientCredentials**
See keystoneauth1.identity.v3.oidc.OidcClientCredentials

keystoneauth1.identity.**V3OidcDeviceAuthorization**
See keystoneauth1.identity.v3.oidc.OidcDeviceAuthorization

keystoneauth1.identity.**V3OidcPassword**
See keystoneauth1.identity.v3.oidc.OidcPassword

keystoneauth1.identity.**V3Password**
See keystoneauth1.identity.v3.Password

keystoneauth1.identity.**V3TOTP**
See keystoneauth1.identity.v3.TOTP

`keystoneauth1.identity.`**`V3Token`**

See `keystoneauth1.identity.v3.Token`

`keystoneauth1.identity.`**`V3TokenlessAuth`**

See `keystoneauth1.identity.v3.TokenlessAuth`

keystoneauth1.loading package

Submodules

keystoneauth1.loading.adapter module

`class` `keystoneauth1.loading.adapter.`**`Adapter`**

Bases: `_BaseLoader[Adapter]`

`__abstractmethods__ = frozenset({})`

`__annotations__ = {}`

`__doc__ = None`

`__module__ = 'keystoneauth1.loading.adapter'`

`__orig_bases__ = (keystoneauth1.loading.base._BaseLoader[keystoneauth1.adapter.Adapter],)`

`__parameters__ = ()`

`_abc_impl = <_abc._abc_data object>`

`static get_conf_options`(*include_deprecated: bool = True*, *deprecated_opts: dict[str, list[cfg.DeprecatedOpt]] | None = None*) → list[cfg.Opt]

Get oslo_config options that are needed for a `Adapter`.

These may be useful without being registered for config file generation or to manipulate the options before registering them yourself.

The options that are set are:

service_type
The default service_type for URL discovery.

service_name
The default service_name for URL discovery.

interface
The default interface for URL discovery. (deprecated)

valid_interfaces
List of acceptable interfaces for URL discovery. Can be a list of any of public, internal or admin.

region_name
The default region_name for URL discovery.

endpoint_override
Always use this endpoint URL for requests for this client.

version
The minimum version restricted to a given Major API. Mutually exclusive with min_version and max_version.

min_version
The minimum major version of a given API, intended to be used as the lower bound of a range with max_version. Mutually exclusive with version. If min_version is given with no max_version it is as if max version is latest.

max_version
The maximum major version of a given API, intended to be used as the upper bound of a range with min_version. Mutually exclusive with version.

Parameters

- **include_deprecated** If True (the default, for backward compatibility), deprecated options are included in the result. If False, they are excluded.
- **deprecated_opts** (*dict*) Deprecated options that should be included in the definition of new options. This should be a dict from the name of the new option to a list of oslo.DeprecatedOpts that correspond to the new option. (optional)

  For example, to support the `api_endpoint` option pointing to the new `endpoint_override` option name:

  ```
  old_opt = oslo_cfg.DeprecatedOpt('api_endpoint', 'old_
  ↪group')
  deprecated_opts = {'endpoint_override': [old_opt]}
  ```

Returns
A list of oslo_config options.

get_options() → list[opts.Opt]

Return the list of parameters associated with the auth plugin.

This list may be used to generate CLI or config arguments.

Returns
A list of Param objects describing available plugin parameters.

Return type
list

load_from_conf_options(*conf: cfg.ConfigOpts*, *group: str*, ***kwargs: Any*) → Adapter

Create an Adapter object from an oslo_config object.

The options must have been previously registered with register_conf_options.

Parameters

- **conf** (*oslo_config.Cfg*) config object to register with.
- **group** (*string*) The ini group to register options in.
- **kwargs** (*dict*) Additional parameters to pass to Adapter construction.

Returns
A new Adapter object.

Return type
`Adapter`

`property plugin_class: Type[Adapter]`

`register_conf_options`(*conf: cfg.ConfigOpts*, *group: str*, *include_deprecated: bool = True*, *deprecated_opts: dict[str, list[cfg.DeprecatedOpt]] | None = None*) → list[cfg.Opt]

Register the oslo_config options that are needed for an Adapter.

The options that are set are:

service_type
The default service_type for URL discovery.

service_name
The default service_name for URL discovery.

interface
The default interface for URL discovery. (deprecated)

valid_interfaces
List of acceptable interfaces for URL discovery. Can be a list of any of public, internal or admin.

region_name
The default region_name for URL discovery.

endpoint_override
Always use this endpoint URL for requests for this client.

version
The minimum version restricted to a given Major API. Mutually exclusive with min_version and max_version.

min_version
The minimum major version of a given API, intended to be used as the lower bound of a range with max_version. Mutually exclusive with version. If min_version is given with no max_version it is as if max version is latest.

max_version
The maximum major version of a given API, intended to be used as the upper bound of a range with min_version. Mutually exclusive with version.

connect_retries
The maximum number of retries that should be attempted for connection errors.

status_code_retries
The maximum number of retries that should be attempted for retriable HTTP status codes.

Parameters

- **conf** (*oslo_config.Cfg*) config object to register with.
- **group** (*string*) The ini group to register options in.

- **include_deprecated** If True (the default, for backward compatibility), deprecated options are registered. If False, they are excluded.
- **deprecated_opts** (*dict*) Deprecated options that should be included in the definition of new options. This should be a dict from the name of the new option to a list of oslo.DeprecatedOpts that correspond to the new option. (optional)

  For example, to support the `api_endpoint` option pointing to the new `endpoint_override` option name:

  ```
  old_opt = oslo_cfg.DeprecatedOpt('api_endpoint', 'old_
  ↪group')
  deprecated_opts = {'endpoint_override': [old_opt]}
  ```

Returns
The list of options that was registered.

`keystoneauth1.loading.adapter.`**`get_conf_options`**(*include_deprecated: bool = True, deprecated_opts: dict[str, list[cfg.DeprecatedOpt]] | None = None*) → list[cfg.Opt]

`keystoneauth1.loading.adapter.`**`load_from_conf_options`**(*conf: cfg.ConfigOpts, group: str, **kwargs: Any*) → Adapter

`keystoneauth1.loading.adapter.`**`process_conf_options`**(*confgrp: cfg.OptGroup, kwargs: dict[str, Any]*) → None

Set Adapter constructor kwargs based on conf options.

Parameters

- **confgrp** (*oslo_config.cfg.OptGroup*) Config object group containing options to inspect.
- **kwargs** (*dict*) Keyword arguments suitable for the constructor of keystoneauth1.adapter.Adapter. Will be modified by this method. Values already set remain unaffected.

Raises
TypeError If invalid conf option values or combinations are found.

`keystoneauth1.loading.adapter.`**`register_argparse_arguments`**(*parser: ArgumentParser, service_type: str | None = None*) → None

`keystoneauth1.loading.adapter.`**`register_conf_options`**(*conf: cfg.ConfigOpts, group: str, include_deprecated: bool = True, deprecated_opts: dict[str, list[cfg.DeprecatedOpt]] | None = None*) → list[cfg.Opt]

`keystoneauth1.loading.adapter.`**`register_service_argparse_arguments`**(*parser: ArgumentParser, service_type: str*) → None

keystoneauth1.loading.base module

class keystoneauth1.loading.base.**BaseLoader**

Bases: _BaseLoader[BaseAuthPluginT]

__abstractmethods__ = frozenset({'get_options'})

__annotations__ = {}

__doc__ = None

__module__ = 'keystoneauth1.loading.base'

__orig_bases__ = (keystoneauth1.loading.base._BaseLoader[+BaseAuthPluginT],)

__parameters__ = (+BaseAuthPluginT,)

_abc_impl = <_abc._abc_data object>

class keystoneauth1.loading.base.**_BaseLoader**

Bases: Generic[T]

__abstractmethods__ = frozenset({'get_options'})

__annotations__ = {}

__dict__ = mappingproxy({'__module__': 'keystoneauth1.loading.base', 'plugin_class': <property object>, 'create_plugin': <function _BaseLoader.create_plugin>, 'get_options': <function _BaseLoader.get_options>, 'available': <property object>, 'load_from_options': <function _BaseLoader.load_from_options>, 'load_from_options_getter': <function _BaseLoader.load_from_options_getter>, '__orig_bases__': (typing.Generic[+T],), '__dict__': <attribute '__dict__' of '_BaseLoader' objects>, '__weakref__': <attribute '__weakref__' of '_BaseLoader' objects>, '__doc__': None, '__parameters__': (+T,), '__abstractmethods__': frozenset({'get_options'}), '_abc_impl': <_abc._abc_data object>, '__annotations__': {}})

__doc__ = None

__module__ = 'keystoneauth1.loading.base'

__orig_bases__ = (typing.Generic[+T],)

__parameters__ = (+T,)

__weakref__

list of weak references to the object (if defined)

_abc_impl = <_abc._abc_data object>

property available: bool

Return if the plugin is available for loading.

If a plugin is missing dependencies or for some other reason should not be available to the current system it should override this property and return False to exclude itself from the plugin list.

Return type
bool

create_plugin(***kwargs: Any*) → T

Create a plugin from the options available for the loader.

Given the options that were specified by the loader create an appropriate plugin. You can override this function in your loader.

This used to be specified by providing the plugin_class property and this is still supported, however specifying a property didnt let you choose a plugin type based upon the options that were presented.

Override this function if you wish to return different plugins based on the options presented, otherwise you can simply provide the plugin_class property.

Added 2.9

abstract get_options() → list[opts.Opt]

Return the list of parameters associated with the auth plugin.

This list may be used to generate CLI or config arguments.

Returns
A list of Param objects describing available plugin parameters.

Return type
list

load_from_options(***kwargs: Any*) → T

Create a plugin from the arguments retrieved from get_options.

A client can override this function to do argument validation or to handle differences between the registered options and what is required to create the plugin.

load_from_options_getter(*getter: Callable[[opts.Opt], Any]*, ***kwargs: Any*) → T

Load a plugin from getter function that returns appropriate values.

To handle cases other than the provided CONF and CLI loading you can specify a custom loader function that will be queried for the option value. The getter is a function that takes a `keystoneauth1.loading.Opt` and returns a value to load with.

Parameters
getter (*callable*) A function that returns a value for the given opt.

Returns
An authentication Plugin.

Return type
`keystoneauth1.plugin.BaseAuthPlugin`

property plugin_class: Type[T]

`keystoneauth1.loading.base.`**`_auth_plugin_available`**`(`*ext: Extension*`)` → bool

Read the value of available for whether to load this plugin.

`keystoneauth1.loading.base.`**`get_available_plugin_loaders`**`()` → dict[str, BaseLoader[plugin.BaseAuthPluginT]]

Retrieve all the plugin classes available on the system.

Returns
A dict with plugin entrypoint name as the key and the plugin loader as the value.

Return type
dict

`keystoneauth1.loading.base.`**`get_available_plugin_names`**`()` → FrozenSet[str]

Get the names of all the plugins that are available on the system.

This is particularly useful for help and error text to prompt a user for example what plugins they may specify.

Returns
A list of names.

Return type
frozenset

`keystoneauth1.loading.base.`**`get_plugin_loader`**`(`*name: str*`)` → BaseLoader[BaseAuthPluginT]

Retrieve a plugin class by its entrypoint name.

Parameters
name (*str*) The name of the object to get.

Returns
An auth plugin class.

Return type
`keystoneauth1.loading.BaseLoader`

Raises
`keystoneauth1.exceptions.auth_plugins.NoMatchingPlugin` if a plugin cannot be created.

`keystoneauth1.loading.base.`**`get_plugin_options`**`(`*name: str*`)` → list[opts.Opt]

Get the options for a specific plugin.

This will be the list of options that is registered and loaded by the specified plugin.

Returns
A list of `keystoneauth1.loading.Opt` options.

Raises
`keystoneauth1.exceptions.auth_plugins.NoMatchingPlugin` if a plugin cannot be created.

keystoneauth1.loading.cli module

keystoneauth1.loading.cli.**_register_plugin_argparse_arguments**(*parser: ArgumentParser | _ArgumentGroup*, *plugin: BaseLoader[plugin.BaseAuthPluginT]*) → None

keystoneauth1.loading.cli.**load_from_argparse_arguments**(*namespace: Namespace*, ***kwargs: Any*) → plugin.BaseAuthPluginT | None

Retrieve the created plugin from the completed argparse results.

Loads and creates the auth plugin from the information parsed from the command line by argparse.

Parameters
`namespace` (*Namespace*) The result from CLI parsing.

Returns
An auth plugin, or None if a name is not provided.

Return type
`keystoneauth1.plugin.BaseAuthPlugin`

Raises
`keystoneauth1.exceptions.auth_plugins.NoMatchingPlugin` if a plugin cannot be created.

keystoneauth1.loading.cli.**register_argparse_arguments**(*parser: ArgumentParser*, *argv: list[str]*, *default: Any = None*) → BaseLoader[plugin.BaseAuthPluginT] | None

Register CLI options needed to create a plugin.

The function inspects the provided arguments so that it can also register the options required for that specific plugin if available.

Parameters

- **parser** (*argparse.ArgumentParser*) the parser to attach argparse options to.
- **argv** (*list*) the arguments provided to the appliation.
- **default** (*str/class*) a default plugin name or a plugin object to use if one isnt specified by the CLI. default: None.

Returns
The plugin class that will be loaded or None if not provided.

Return type
`keystoneauth1.loader.BaseLoader`

Raises

`keystoneauth1.exceptions.auth_plugins.NoMatchingPlugin` if a plugin cannot be created.

keystoneauth1.loading.conf module

`keystoneauth1.loading.conf.`**`get_common_conf_options`**`()` → list[cfg.Opt]

Get the oslo_config options common for all auth plugins.

These may be useful without being registered for config file generation or to manipulate the options before registering them yourself.

The options that are set are:

auth_type
The name of the plugin to load.

auth_section
The config file section to load options from.

Returns
A list of oslo_config options.

`keystoneauth1.loading.conf.`**`get_plugin_conf_options`**`(`*plugin: BaseLoader[BaseAuthPluginT] | str*`)` → list[cfg.Opt]

Get the oslo_config options for a specific plugin.

This will be the list of config options that is registered and loaded by the specified plugin.

Parameters
`plugin` (*`str or keystoneauth1.loading.BaseLoader`*) The name of the plugin loader or a plugin loader object

Returns
A list of oslo_config options.

`keystoneauth1.loading.conf.`**`load_from_conf_options`**`(`*conf: cfg.ConfigOpts*, *group: str*, ***kwargs: Any*`)` → keystoneauth1.plugin.BaseAuthPlugin | None

Load a plugin from an oslo_config CONF object.

Each plugin will register their own required options and so there is no standard list and the plugin should be consulted.

The base options should have been registered with register_conf_options before this function is called.

Parameters

- **`conf`** (*`oslo_config.cfg.ConfigOpts`*) A conf object.
- **`group`** (*`str`*) The group name that options should be read from.

Returns
An authentication Plugin or None if a name is not provided

Return type
`keystoneauth1.plugin.BaseAuthPlugin`

Raises

`keystoneauth1.exceptions.auth_plugins.NoMatchingPlugin` if a plugin cannot be created.

`keystoneauth1.loading.conf.`**`register_conf_options`**(*conf: cfg.ConfigOpts*, *group: str*) → None

Register the oslo_config options that are needed for a plugin.

This only registers the basic options shared by all plugins. Options that are specific to a plugin are loaded just before they are read.

The defined options are:

- **auth_type: the name of the auth plugin that will be used for**
  authentication.
- **auth_section: the group from which further auth plugin options should be**
  taken. If section is not provided then the auth plugin options will be taken from the same group as provided in the parameters.

Parameters

- **conf** (*oslo_config.cfg.ConfigOpts*) config object to register with.
- **group** (*string*) The ini group to register options in.

keystoneauth1.loading.identity module

class `keystoneauth1.loading.identity.`**`BaseFederationLoader`**

Bases: `BaseV3Loader[V3FederationPluginT]`

Base Option handling for federation plugins.

This class defines options and handling that should be common to the V3 identity federation API. It provides the options expected by the `keystoneauth1.identity.v3.FederationBaseAuth` class.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {}`

`__doc__ = 'Base Option handling for federation plugins.\n\n This class defines options and handling that should be common to the V3\n identity federation API. It provides the options expected by the\n :py:class:`keystoneauth1.identity.v3.FederationBaseAuth` class.\n '`

`__module__ = 'keystoneauth1.loading.identity'`

`__orig_bases__ = (keystoneauth1.loading.identity.BaseV3Loader[~V3FederationPluginT],)`

`__parameters__ = (~V3FederationPluginT,)`

`_abc_impl = <_abc._abc_data object>`

get_options() → list[Opt]

Return the list of parameters associated with the auth plugin.

This list may be used to generate CLI or config arguments.

Returns
A list of Param objects describing available plugin parameters.

Return type
list

class keystoneauth1.loading.identity.**BaseGenericLoader**

Bases: `BaseIdentityLoader[GenericPluginT]`

Base Option handling for generic plugins.

This class defines options and handling that should be common to generic plugins. These plugins target the OpenStack identity service however are designed to be independent of API version. It provides the options expected by the `keystoneauth1.identity.generic.base.BaseGenericPlugin` class.

__abstractmethods__ = frozenset({})

__annotations__ = {}

__doc__ = 'Base Option handling for generic plugins.\n\n This class defines options and handling that should be common to generic\n plugins. These plugins target the OpenStack identity service however are\n designed to be independent of API version. It provides the options expected\n by the :py:class:`keystoneauth1.identity.generic.base.BaseGenericPlugin`\n class.\n '

__module__ = 'keystoneauth1.loading.identity'

__orig_bases__ = (keystoneauth1.loading.identity.BaseIdentityLoader[~GenericPluginT],)

__parameters__ = (~GenericPluginT,)

_abc_impl = <_abc._abc_data object>

get_options() → list[Opt]

Return the list of parameters associated with the auth plugin.

This list may be used to generate CLI or config arguments.

Returns
A list of Param objects describing available plugin parameters.

Return type
list

class keystoneauth1.loading.identity.**BaseIdentityLoader**

Bases: `BaseLoader[PluginT]`

Base Option handling for identity plugins.

This class defines options and handling that should be common across all plugins that are developed against the OpenStack identity service. It provides the options expected by the `keystoneauth1.identity.BaseIdentityPlugin` class.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {}`

`__doc__ = 'Base Option handling for identity plugins.\n\n This class defines options and handling that should be common across all\n plugins that are developed against the OpenStack identity service. It\n provides the options expected by the\n :py:class:`keystoneauth1.identity.BaseIdentityPlugin` class.\n '`

`__module__ = 'keystoneauth1.loading.identity'`

`__orig_bases__ = (keystoneauth1.loading.base.BaseLoader[~PluginT],)`

`__parameters__ = (~PluginT,)`

`_abc_impl = <_abc._abc_data object>`

`get_options()` → list[Opt]

Return the list of parameters associated with the auth plugin.

This list may be used to generate CLI or config arguments.

Returns
A list of Param objects describing available plugin parameters.

Return type
list

class `keystoneauth1.loading.identity.`**`BaseV2Loader`**

Bases: `BaseIdentityLoader[V2PluginT]`

Base Option handling for identity plugins.

This class defines options and handling that should be common to the V2 identity API. It provides the options expected by the `keystoneauth1.identity.v2.Auth` class.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {}`

`__doc__ = 'Base Option handling for identity plugins.\n\n This class defines options and handling that should be common to the V2\n identity API. It provides the options expected by the\n :py:class:`keystoneauth1.identity.v2.Auth` class.\n '`

`__module__ = 'keystoneauth1.loading.identity'`

`__orig_bases__ = (keystoneauth1.loading.identity.BaseIdentityLoader[~V2PluginT],)`

`__parameters__ = (~V2PluginT,)`

`_abc_impl = <_abc._abc_data object>`

get_options() → list[Opt]

Return the list of parameters associated with the auth plugin.

This list may be used to generate CLI or config arguments.

Returns
A list of Param objects describing available plugin parameters.

Return type
list

class keystoneauth1.loading.identity.**BaseV3Loader**

Bases: BaseIdentityLoader[V3PluginT]

Base Option handling for identity plugins.

This class defines options and handling that should be common to the V3 identity API. It provides the options expected by the `keystoneauth1.identity.v3.Auth` class.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {}`

`__doc__ = 'Base Option handling for identity plugins.\n\n This class defines options and handling that should be common to the V3\n identity API. It provides the options expected by the\n :py:class:`keystoneauth1.identity.v3.Auth` class.\n '`

`__module__ = 'keystoneauth1.loading.identity'`

`__orig_bases__ = (keystoneauth1.loading.identity.BaseIdentityLoader[~V3PluginT],)`

`__parameters__ = (~V3PluginT,)`

`_abc_impl = <_abc._abc_data object>`

get_options() → list[Opt]

Return the list of parameters associated with the auth plugin.

This list may be used to generate CLI or config arguments.

Returns
A list of Param objects describing available plugin parameters.

Return type
list

load_from_options(***kwargs: Any*) → V3PluginT

Create a plugin from the arguments retrieved from get_options.

A client can override this function to do argument validation or to handle differences between the registered options and what is required to create the plugin.

keystoneauth1.loading.opts module

class keystoneauth1.loading.opts.**Opt**(*name: str*, *type: ~typing.Type[~typing.Any] = <class 'str'>*, *help: str | None = None*, *secret: bool = False*, *dest: str | None = None*, *deprecated: list[Opt] | None = None*, *default: ~typing.Any | None = None*, *metavar: str | None = None*, *required: bool = False*, *prompt: str | None = None*)

Bases: `object`

An option required by an authentication plugin.

Opts provide a means for authentication plugins that are going to be dynamically loaded to specify the parameters that are to be passed to the plugin on initialization.

The Opt specifies information about the way the plugin parameter is to be represented in different loading mechanisms.

When defining an Opt with a - the - should be present in the name parameter. This will automatically be converted to an _ when passing to the plugin initialization. For example, you should specify:

```
Opt('user-domain-id')
```

which will pass the value as *user_domain_id* to the plugins initialization.

Parameters

- **name** (*str*) The name of the option.
- **type** (*callable*) The type of the option. This is a callable which is passed the raw option that was loaded (often a string) and is required to return the parameter in the type expected by __init__.
- **help** (*str*) The help text that is shown along with the option.
- **secret** (*bool*) If the parameter is secret it should not be printed or logged in debug output.
- **dest** (*str*) the name of the argument that will be passed to __init__. This allows you to have a different name in loading than is used by the __init__ function. Defaults to the value of name.
- **deprecated** (*keystoneauth1.loading.Opt*) A list of other options that are deprecated in favour of this one. This ensures the old options are still registered.
- **default** A default value that can be used if one is not provided.
- **metavar** (*str*) The <metavar> that should be printed in CLI help text.
- **required** (*bool*) If the option is required to load the plugin. If a required option is not present loading should fail.
- **prompt** (*str*) If the option can be requested via a prompt (where appropriate) set the string that should be used to prompt with.

__annotations__ = {}

```
__dict__ = mappingproxy({'__module__': 'keystoneauth1.loading.opts',
'__doc__': "An option required by an authentication plugin.\n\n Opts
provide a means for authentication plugins that are going to be\n
dynamically loaded to specify the parameters that are to be passed to
the\n plugin on initialization.\n\n The Opt specifies information about
the way the plugin parameter is to be\n represented in different loading
mechanisms.\n\n When defining an Opt with a - the - should be present in
the name\n parameter. This will automatically be converted to an _ when
passing to the\n plugin initialization. For example, you should
specify::\n\n Opt('user-domain-id')\n\n which will pass the value as
`user_domain_id` to the plugin's\n initialization.\n\n :param str name:
The name of the option.\n :param callable type:  The type of the option.
This is a callable which is\n passed the raw option that was loaded (often
a string) and is required\n to return the parameter in the type expected
by __init__.\n :param str help:  The help text that is shown along with
the option.\n :param bool secret:  If the parameter is secret it should
not be printed or\n logged in debug output.\n :param str dest:  the name
of the argument that will be passed to __init__.\n This allows you to have
a different name in loading than is used by the\n __init__ function.
Defaults to the value of name.\n :param keystoneauth1.loading.Opt
deprecated:  A list of other options that\n are deprecated in favour of
this one. This ensures the old options are\n still registered.\n :type
opt:  list(Opt)\n :param default:  A default value that can be used if one
is not provided.\n :param str metavar:  The <metavar> that should be
printed in CLI help text.\n :param bool required:  If the option is
required to load the plugin. If a\n required option is not present loading
should fail.\n :param str prompt:  If the option can be requested via a
prompt (where\n appropriate) set the string that should be used to prompt
with.\n ", '__init__':  <function Opt.__init__>, '__repr__':  <function
Opt.__repr__>, '_to_oslo_opt':  <function Opt._to_oslo_opt>, '__eq__':
<function Opt.__eq__>, '_all_opts':  <property object>, 'argparse_args':
<property object>, 'argparse_envvars':  <property object>,
'argparse_default':  <property object>, '__dict__':  <attribute '__dict__'
of 'Opt' objects>, '__weakref__':  <attribute '__weakref__' of 'Opt'
objects>, '__hash__':  None, '__annotations__':  {}})
```

`__doc__ = "An option required by an authentication plugin.\n\n Opts provide a means for authentication plugins that are going to be\n dynamically loaded to specify the parameters that are to be passed to the\n plugin on initialization.\n\n The Opt specifies information about the way the plugin parameter is to be\n represented in different loading mechanisms.\n\n When defining an Opt with a - the - should be present in the name\n parameter. This will automatically be converted to an _ when passing to the\n plugin initialization. For example, you should specify::\n\n Opt('user-domain-id')\n\n which will pass the value as `user_domain_id` to the plugin's\n initialization.\n\n :param str name: The name of the option.\n :param callable type: The type of the option. This is a callable which is\n passed the raw option that was loaded (often a string) and is required\n to return the parameter in the type expected by __init__.\n :param str help: The help text that is shown along with the option.\n :param bool secret: If the parameter is secret it should not be printed or\n logged in debug output.\n :param str dest: the name of the argument that will be passed to __init__.\n This allows you to have a different name in loading than is used by the\n __init__ function. Defaults to the value of name.\n :param keystoneauth1.loading.Opt deprecated: A list of other options that\n are deprecated in favour of this one. This ensures the old options are\n still registered.\n :type opt: list(Opt)\n :param default: A default value that can be used if one is not provided.\n :param str metavar: The <metavar> that should be printed in CLI help text.\n :param bool required: If the option is required to load the plugin. If a\n required option is not present loading should fail.\n :param str prompt: If the option can be requested via a prompt (where\n appropriate) set the string that should be used to prompt with.\n "`

`__eq__`(*other: Any*) → bool

Define equality operator on option parameters.

`__hash__ = None`

`__init__`(*name: str*, *type: ~typing.Type[~typing.Any] = <class 'str'>*, *help: str | None = None*, *secret: bool = False*, *dest: str | None = None*, *deprecated: list[Opt] | None = None*, *default: ~typing.Any | None = None*, *metavar: str | None = None*, *required: bool = False*, *prompt: str | None = None*)

`__module__ = 'keystoneauth1.loading.opts'`

`__repr__`() → str

Return string representation of option name.

`__weakref__`

list of weak references to the object (if defined)

`property _all_opts: chain[Opt]`

`_to_oslo_opt`() → cfg.Opt

`property argparse_args: list[str]`

`property argparse_default: Any`

`property argparse_envvars: List[str]`

keystoneauth1.loading.session module

`class keystoneauth1.loading.session.Session`

Bases: `_BaseLoader[Session]`

`__abstractmethods__ = frozenset({})`

`__annotations__ = {}`

`__doc__ = None`

`__module__ = 'keystoneauth1.loading.session'`

`__orig_bases__ = (keystoneauth1.loading.base._BaseLoader[keystoneauth1.session.Session],)`

`__parameters__ = ()`

`_abc_impl = <_abc._abc_data object>`

`get_conf_options`(*deprecated_opts: dict[str, list[cfg.DeprecatedOpt]] | None = None*) → list[cfg.Opt]

Get oslo_config options that are needed for a `Session`.

These may be useful without being registered for config file generation or to manipulate the options before registering them yourself.

The options that are set are:

cafile
The certificate authority filename.

certfile
The client certificate file to present.

keyfile
The key for the client certificate.

insecure
Whether to ignore SSL verification.

timeout
The max time to wait for HTTP connections.

collect-timing
Whether to collect API timing information.

split-loggers
Whether to log requests to multiple loggers.

Parameters

deprecated_opts (*dict*) Deprecated options that should be included in the definition of new options. This should be a dict from the name of the new option to a list of oslo.DeprecatedOpts that correspond to the new option. (optional)

For example, to support the `ca_file` option pointing to the new `cafile` option name:

```
old_opt = oslo_cfg.DeprecatedOpt('ca_file', 'old_group')
deprecated_opts = {'cafile': [old_opt]}
```

Returns
A list of oslo_config options.

get_options() → list[opts.Opt]

Return the list of parameters associated with the auth plugin.

This list may be used to generate CLI or config arguments.

Returns
A list of Param objects describing available plugin parameters.

Return type
list

load_from_argparse_arguments(*namespace: Namespace*, ***kwargs: Any*) → Session

load_from_conf_options(*conf: cfg.ConfigOpts*, *group: str*, ***kwargs: Any*) → Session

Create a session object from an oslo_config object.

The options must have been previously registered with register_conf_options.

Parameters

- **conf** (*oslo_config.Cfg*) config object to register with.
- **group** (*string*) The ini group to register options in.
- **kwargs** (*dict*) Additional parameters to pass to session construction.

Returns
A new session object.

Return type
`Session`

load_from_options(*insecure: bool = False*, *verify: bool | str | None = None*, *cacert: str | None = None*, *cert: str | None = None*, *key: str | None = None*, ***kwargs: Any*) → Session

Create a session with individual certificate parameters.

Some parameters used to create a session dont lend themselves to be loaded from config/CLI etc. Create a session by converting those parameters into session __init__ parameters.

property plugin_class: Type[Session]

register_argparse_arguments(*parser: ArgumentParser*) → None

register_conf_options(*conf: cfg.ConfigOpts*, *group: str*, *deprecated_opts: dict[str, list[cfg.DeprecatedOpt]] | None = None*) → list[cfg.Opt]

Register the oslo_config options that are needed for a session.

The options that are set are:

cafile
The certificate authority filename.

certfile
The client certificate file to present.

keyfile
The key for the client certificate.

insecure
Whether to ignore SSL verification.

timeout
The max time to wait for HTTP connections.

collect-timing
Whether to collect API timing information.

split-loggers
Whether to log requests to multiple loggers.

Parameters

- **conf** (*oslo_config.Cfg*) config object to register with.
- **group** (*string*) The ini group to register options in.
- **deprecated_opts** (*dict*) Deprecated options that should be included in the definition of new options. This should be a dict from the name of the new option to a list of oslo.DeprecatedOpts that correspond to the new option. (optional)

  For example, to support the `ca_file` option pointing to the new `cafile` option name:

  ```
  old_opt = oslo_cfg.DeprecatedOpt('ca_file', 'old_group')
  deprecated_opts = {'cafile': [old_opt]}
  ```

Returns
The list of options that was registered.

keystoneauth1.loading.session.**_positive_non_zero_float**(*argument_value: str | None*) → float | None

keystoneauth1.loading.session.**get_conf_options**(*deprecated_opts: dict[str, list[cfg.DeprecatedOpt]] | None = None*) → list[cfg.Opt]

keystoneauth1.loading.session.**load_from_argparse_arguments**(*namespace: Namespace*, ***kwargs: Any*) → Session

keystoneauth1.loading.session.**load_from_conf_options**(*conf: cfg.ConfigOpts*, *group: str*, ***kwargs: Any*) → Session

keystoneauth1.loading.session.**register_argparse_arguments**(*parser: ArgumentParser*) → None `keystoneauth1.loading.session.`**`register_conf_options`**(*conf: cfg.ConfigOpts*, *group: str*, *deprecated_opts: dict[str, list[cfg.DeprecatedOpt]] | None = None*) → list[cfg.Opt]

Module contents

class `keystoneauth1.loading.`**`BaseFederationLoader`**

Bases: `BaseV3Loader[V3FederationPluginT]`

Base Option handling for federation plugins.

This class defines options and handling that should be common to the V3 identity federation API. It provides the options expected by the `keystoneauth1.identity.v3.FederationBaseAuth` class.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {}`

`__doc__ = 'Base Option handling for federation plugins.\n\n This class defines options and handling that should be common to the V3\n identity federation API. It provides the options expected by the\n :py:class:`keystoneauth1.identity.v3.FederationBaseAuth` class.\n '`

`__module__ = 'keystoneauth1.loading.identity'`

`__orig_bases__ = (keystoneauth1.loading.identity.BaseV3Loader[~V3FederationPluginT],)`

`__parameters__ = (~V3FederationPluginT,)`

`_abc_impl = <_abc._abc_data object>`

`get_options`() → list[Opt]

Return the list of parameters associated with the auth plugin.

This list may be used to generate CLI or config arguments.

Returns

A list of Param objects describing available plugin parameters.

Return type

list

class `keystoneauth1.loading.`**`BaseGenericLoader`**

Bases: `BaseIdentityLoader[GenericPluginT]`

Base Option handling for generic plugins.

This class defines options and handling that should be common to generic plugins. These plugins target the OpenStack identity service however are designed to be independent of API version. It provides the options expected by the `keystoneauth1.identity.generic.base.BaseGenericPlugin` class.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {}`

`__doc__ = 'Base Option handling for generic plugins.\n\n This class defines options and handling that should be common to generic\n plugins. These plugins target the OpenStack identity service however are\n designed to be independent of API version. It provides the options expected\n by the :py:class:`keystoneauth1.identity.generic.base.BaseGenericPlugin`\n class.\n '`

`__module__ = 'keystoneauth1.loading.identity'`

`__orig_bases__ = (keystoneauth1.loading.identity.BaseIdentityLoader[~GenericPluginT],)`

`__parameters__ = (~GenericPluginT,)`

`_abc_impl = <_abc._abc_data object>`

`get_options()` → list[Opt]

Return the list of parameters associated with the auth plugin.

This list may be used to generate CLI or config arguments.

Returns
A list of Param objects describing available plugin parameters.

Return type
list

class keystoneauth1.loading.**BaseIdentityLoader**

Bases: `BaseLoader[PluginT]`

Base Option handling for identity plugins.

This class defines options and handling that should be common across all plugins that are developed against the OpenStack identity service. It provides the options expected by the `keystoneauth1.identity.BaseIdentityPlugin` class.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {}`

`__doc__ = 'Base Option handling for identity plugins.\n\n This class defines options and handling that should be common across all\n plugins that are developed against the OpenStack identity service. It\n provides the options expected by the\n :py:class:`keystoneauth1.identity.BaseIdentityPlugin` class.\n '`

`__module__ = 'keystoneauth1.loading.identity'`

`__orig_bases__ = (keystoneauth1.loading.base.BaseLoader[~PluginT],)`

`__parameters__ = (~PluginT,)`

`_abc_impl = <_abc._abc_data object>`

get_options() → list[Opt]

Return the list of parameters associated with the auth plugin.

This list may be used to generate CLI or config arguments.

Returns
A list of Param objects describing available plugin parameters.

Return type
list

class keystoneauth1.loading.**BaseLoader**

Bases: `_BaseLoader[BaseAuthPluginT]`

__abstractmethods__ = frozenset({'get_options'})

__annotations__ = {}

__doc__ = None

__module__ = 'keystoneauth1.loading.base'

__orig_bases__ = (keystoneauth1.loading.base._BaseLoader[+BaseAuthPluginT],)

__parameters__ = (+BaseAuthPluginT,)

_abc_impl = <_abc._abc_data object>

class keystoneauth1.loading.**BaseV2Loader**

Bases: `BaseIdentityLoader[V2PluginT]`

Base Option handling for identity plugins.

This class defines options and handling that should be common to the V2 identity API. It provides the options expected by the `keystoneauth1.identity.v2.Auth` class.

__abstractmethods__ = frozenset({})

__annotations__ = {}

__doc__ = 'Base Option handling for identity plugins.\n\n This class defines options and handling that should be common to the V2\n identity API. It provides the options expected by the\n :py:class:`keystoneauth1.identity.v2.Auth` class.\n '

__module__ = 'keystoneauth1.loading.identity'

__orig_bases__ = (keystoneauth1.loading.identity.BaseIdentityLoader[~V2PluginT],)

__parameters__ = (~V2PluginT,)

_abc_impl = <_abc._abc_data object>

get_options() → list[Opt]

Return the list of parameters associated with the auth plugin.

This list may be used to generate CLI or config arguments.

Returns
A list of Param objects describing available plugin parameters.

Return type
list

class keystoneauth1.loading.**BaseV3Loader**

Bases: `BaseIdentityLoader[V3PluginT]`

Base Option handling for identity plugins.

This class defines options and handling that should be common to the V3 identity API. It provides the options expected by the `keystoneauth1.identity.v3.Auth` class.

`__abstractmethods__ = frozenset({})`

`__annotations__ = {}`

`__doc__ = 'Base Option handling for identity plugins.\n\n This class defines options and handling that should be common to the V3\n identity API. It provides the options expected by the\n :py:class:`keystoneauth1.identity.v3.Auth` class.\n '`

`__module__ = 'keystoneauth1.loading.identity'`

`__orig_bases__ = (keystoneauth1.loading.identity.BaseIdentityLoader[~V3PluginT],)`

`__parameters__ = (~V3PluginT,)`

`_abc_impl = <_abc._abc_data object>`

get_options() → list[Opt]

Return the list of parameters associated with the auth plugin.

This list may be used to generate CLI or config arguments.

Returns
A list of Param objects describing available plugin parameters.

Return type
list

load_from_options(***kwargs: Any*) → V3PluginT

Create a plugin from the arguments retrieved from get_options.

A client can override this function to do argument validation or to handle differences between the registered options and what is required to create the plugin.

class keystoneauth1.loading.**Opt**(*name: str*, *type: ~typing.Type[~typing.Any] = <class 'str'>*, *help: str | None = None*, *secret: bool = False*, *dest: str | None = None*, *deprecated: list[Opt] | None = None*, *default: ~typing.Any | None = None*, *metavar: str | None = None*, *required: bool = False*, *prompt: str | None = None*)

Bases: `object`

An option required by an authentication plugin.

Opts provide a means for authentication plugins that are going to be dynamically loaded to specify the parameters that are to be passed to the plugin on initialization.

The Opt specifies information about the way the plugin parameter is to be represented in different loading mechanisms.

When defining an Opt with a - the - should be present in the name parameter. This will automatically be converted to an _ when passing to the plugin initialization. For example, you should specify:

```
Opt('user-domain-id')
```

which will pass the value as *user_domain_id* to the plugins initialization.

Parameters

- **name** (*str*) The name of the option.
- **type** (*callable*) The type of the option. This is a callable which is passed the raw option that was loaded (often a string) and is required to return the parameter in the type expected by __init__.
- **help** (*str*) The help text that is shown along with the option.
- **secret** (*bool*) If the parameter is secret it should not be printed or logged in debug output.
- **dest** (*str*) the name of the argument that will be passed to __init__. This allows you to have a different name in loading than is used by the __init__ function. Defaults to the value of name.
- **deprecated** (*keystoneauth1.loading.Opt*) A list of other options that are deprecated in favour of this one. This ensures the old options are still registered.
- **default** A default value that can be used if one is not provided.
- **metavar** (*str*) The <metavar> that should be printed in CLI help text.
- **required** (*bool*) If the option is required to load the plugin. If a required option is not present loading should fail.
- **prompt** (*str*) If the option can be requested via a prompt (where appropriate) set the string that should be used to prompt with.

```
__dict__ = mappingproxy({'__module__': 'keystoneauth1.loading.opts',
'__doc__': "An option required by an authentication plugin.\n\n Opts
provide a means for authentication plugins that are going to be\n
dynamically loaded to specify the parameters that are to be passed to
the\n plugin on initialization.\n\n The Opt specifies information about
the way the plugin parameter is to be\n represented in different loading
mechanisms.\n\n When defining an Opt with a - the - should be present in
the name\n parameter. This will automatically be converted to an _ when
passing to the\n plugin initialization. For example, you should
specify::\n\n Opt('user-domain-id')\n\n which will pass the value as
`user_domain_id` to the plugin's\n initialization.\n\n :param str name:
The name of the option.\n :param callable type:  The type of the option.
This is a callable which is\n passed the raw option that was loaded (often
a string) and is required\n to return the parameter in the type expected
by __init__.\n :param str help:  The help text that is shown along with
the option.\n :param bool secret:  If the parameter is secret it should
not be printed or\n logged in debug output.\n :param str dest:  the name
of the argument that will be passed to __init__.\n This allows you to have
a different name in loading than is used by the\n __init__ function.
Defaults to the value of name.\n :param keystoneauth1.loading.Opt
deprecated:  A list of other options that\n are deprecated in favour of
this one. This ensures the old options are\n still registered.\n :type
opt:  list(Opt)\n :param default:  A default value that can be used if one
is not provided.\n :param str metavar:  The <metavar> that should be
printed in CLI help text.\n :param bool required:  If the option is
required to load the plugin. If a\n required option is not present loading
should fail.\n :param str prompt:  If the option can be requested via a
prompt (where\n appropriate) set the string that should be used to prompt
with.\n ", '__init__':  <function Opt.__init__>, '__repr__':  <function
Opt.__repr__>, '_to_oslo_opt':  <function Opt._to_oslo_opt>, '__eq__':
<function Opt.__eq__>, '_all_opts':  <property object>, 'argparse_args':
<property object>, 'argparse_envvars':  <property object>,
'argparse_default':  <property object>, '__dict__':  <attribute '__dict__'
of 'Opt' objects>, '__weakref__':  <attribute '__weakref__' of 'Opt'
objects>, '__hash__':  None, '__annotations__':  {}})
```

`__doc__ = "An option required by an authentication plugin.\n\n Opts provide a means for authentication plugins that are going to be\n dynamically loaded to specify the parameters that are to be passed to the\n plugin on initialization.\n\n The Opt specifies information about the way the plugin parameter is to be\n represented in different loading mechanisms.\n\n When defining an Opt with a - the - should be present in the name\n parameter. This will automatically be converted to an _ when passing to the\n plugin initialization. For example, you should specify::\n\n Opt('user-domain-id')\n\n which will pass the value as `user_domain_id` to the plugin's\n initialization.\n\n :param str name: The name of the option.\n :param callable type: The type of the option. This is a callable which is\n passed the raw option that was loaded (often a string) and is required\n to return the parameter in the type expected by __init__.\n :param str help: The help text that is shown along with the option.\n :param bool secret: If the parameter is secret it should not be printed or\n logged in debug output.\n :param str dest: the name of the argument that will be passed to __init__.\n This allows you to have a different name in loading than is used by the\n __init__ function. Defaults to the value of name.\n :param keystoneauth1.loading.Opt deprecated: A list of other options that\n are deprecated in favour of this one. This ensures the old options are\n still registered.\n :type opt: list(Opt)\n :param default: A default value that can be used if one is not provided.\n :param str metavar: The <metavar> that should be printed in CLI help text.\n :param bool required: If the option is required to load the plugin. If a\n required option is not present loading should fail.\n :param str prompt: If the option can be requested via a prompt (where\n appropriate) set the string that should be used to prompt with.\n "`

`__eq__`(*other: Any*) → bool

Define equality operator on option parameters.

`__hash__` `= None`

`__init__`(*name: str*, *type: ~typing.Type[~typing.Any] = <class 'str'>*, *help: str | None = None*, *secret: bool = False*, *dest: str | None = None*, *deprecated: list[Opt] | None = None*, *default: ~typing.Any | None = None*, *metavar: str | None = None*, *required: bool = False*, *prompt: str | None = None*)

`__module__` `= 'keystoneauth1.loading.opts'`

`__repr__`() → str

Return string representation of option name.

`__weakref__`

list of weak references to the object (if defined)

`property` **`_all_opts`**`: chain[Opt]`

`_to_oslo_opt`() → cfg.Opt

`property` **`argparse_args`**`: list[str]`

property argparse_default: Any

property argparse_envvars: List[str]

`keystoneauth1.loading.`**`get_adapter_conf_options`**(*include_deprecated: bool = True, deprecated_opts: dict[str, list[cfg.DeprecatedOpt]] | None = None*) → list[cfg.Opt]

`keystoneauth1.loading.`**`get_auth_common_conf_options`**() → list[cfg.Opt]

Get the oslo_config options common for all auth plugins.

These may be useful without being registered for config file generation or to manipulate the options before registering them yourself.

The options that are set are:

auth_type
The name of the plugin to load.

auth_section
The config file section to load options from.

Returns
A list of oslo_config options.

`keystoneauth1.loading.`**`get_auth_plugin_conf_options`**(*plugin: BaseLoader[BaseAuthPluginT] | str*) → list[cfg.Opt]

Get the oslo_config options for a specific plugin.

This will be the list of config options that is registered and loaded by the specified plugin.

Parameters
plugin (*str or keystoneauth1.loading.BaseLoader*) The name of the plugin loader or a plugin loader object

Returns
A list of oslo_config options.

`keystoneauth1.loading.`**`get_available_plugin_loaders`**() → dict[str, BaseLoader[plugin.BaseAuthPluginT]]

Retrieve all the plugin classes available on the system.

Returns
A dict with plugin entrypoint name as the key and the plugin loader as the value.

Return type
dict

`keystoneauth1.loading.`**`get_available_plugin_names`**() → FrozenSet[str]

Get the names of all the plugins that are available on the system.

This is particularly useful for help and error text to prompt a user for example what plugins they may specify.

Returns
A list of names.

Return type
frozenset

`keystoneauth1.loading.`**`get_plugin_loader`**(*name: str*) → BaseLoader[BaseAuthPluginT]

Retrieve a plugin class by its entrypoint name.

Parameters
`name` (*`str`*) The name of the object to get.

Returns
An auth plugin class.

Return type
`keystoneauth1.loading.BaseLoader`

Raises
`keystoneauth1.exceptions.auth_plugins.NoMatchingPlugin` if a plugin cannot be created.

`keystoneauth1.loading.`**`get_plugin_options`**(*name: str*) → list[opts.Opt]

Get the options for a specific plugin.

This will be the list of options that is registered and loaded by the specified plugin.

Returns
A list of `keystoneauth1.loading.Opt` options.

Raises
`keystoneauth1.exceptions.auth_plugins.NoMatchingPlugin` if a plugin cannot be created.

`keystoneauth1.loading.`**`get_session_conf_options`**(*deprecated_opts: dict[str, list[cfg.DeprecatedOpt]] | None = None*) → list[cfg.Opt]

`keystoneauth1.loading.`**`load_adapter_from_conf_options`**(*conf: cfg.ConfigOpts*, *group: str*, ***kwargs: Any*) → Adapter

`keystoneauth1.loading.`**`load_auth_from_argparse_arguments`**(*namespace: Namespace*, ***kwargs: Any*) → plugin.BaseAuthPluginT | None

Retrieve the created plugin from the completed argparse results.

Loads and creates the auth plugin from the information parsed from the command line by argparse.

Parameters
`namespace` (*`Namespace`*) The result from CLI parsing.

Returns
An auth plugin, or None if a name is not provided.

Return type
`keystoneauth1.plugin.BaseAuthPlugin`

Raises
`keystoneauth1.exceptions.auth_plugins.NoMatchingPlugin` if a plugin cannot be created.

`keystoneauth1.loading.`**`load_auth_from_conf_options`**(*conf: cfg.ConfigOpts*, *group: str*, ***kwargs: Any*) → keystoneauth1.plugin.BaseAuthPlugin | None

Load a plugin from an oslo_config CONF object.

Each plugin will register their own required options and so there is no standard list and the plugin should be consulted.

The base options should have been registered with register_conf_options before this function is called.

Parameters

- **conf** (`oslo_config.cfg.ConfigOpts`) A conf object.
- **group** (`str`) The group name that options should be read from.

Returns
An authentication Plugin or None if a name is not provided

Return type
`keystoneauth1.plugin.BaseAuthPlugin`

Raises
`keystoneauth1.exceptions.auth_plugins.NoMatchingPlugin` if a plugin cannot be created.

`keystoneauth1.loading.`**`load_session_from_argparse_arguments`**(*namespace: Namespace*, ***kwargs: Any*) → Session

`keystoneauth1.loading.`**`load_session_from_conf_options`**(*conf: cfg.ConfigOpts*, *group: str*, ***kwargs: Any*) → Session

`keystoneauth1.loading.`**`register_adapter_argparse_arguments`**(*parser: ArgumentParser*, *service_type: str | None = None*) → None

`keystoneauth1.loading.`**`register_adapter_conf_options`**(*conf: cfg.ConfigOpts*, *group: str*, *include_deprecated: bool = True*, *deprecated_opts: dict[str, list[cfg.DeprecatedOpt]] | None = None*) → list[cfg.Opt]

`keystoneauth1.loading.`**`register_auth_argparse_arguments`**(*parser: ArgumentParser*, *argv: list[str]*, *default: Any = None*) → BaseLoader[plugin.BaseAuthPluginT] | None

Register CLI options needed to create a plugin.

The function inspects the provided arguments so that it can also register the options required for that specific plugin if available.

Parameters

- **parser** (*argparse.ArgumentParser*) the parser to attach argparse options to.
- **argv** (*list*) the arguments provided to the appliation.
- **default** (*str/class*) a default plugin name or a plugin object to use if one isnt specified by the CLI. default: None.

Returns
The plugin class that will be loaded or None if not provided.

Return type
`keystoneauth1.loader.BaseLoader`

Raises
`keystoneauth1.exceptions.auth_plugins.NoMatchingPlugin` if a plugin cannot be created.

`keystoneauth1.loading.`**`register_auth_conf_options`**(*conf: cfg.ConfigOpts*, *group: str*) → None

Register the oslo_config options that are needed for a plugin.

This only registers the basic options shared by all plugins. Options that are specific to a plugin are loaded just before they are read.

The defined options are:

- **auth_type: the name of the auth plugin that will be used for**
  authentication.
- **auth_section: the group from which further auth plugin options should be**
  taken. If section is not provided then the auth plugin options will be taken from the same group as provided in the parameters.

Parameters

- **conf** (*oslo_config.cfg.ConfigOpts*) config object to register with.
- **group** (*string*) The ini group to register options in.

`keystoneauth1.loading.`**`register_service_adapter_argparse_arguments`**(*parser: ArgumentParser*, *service_type: str*) → None

`keystoneauth1.loading.`**`register_session_argparse_arguments`**(*parser: ArgumentParser*) → None

`keystoneauth1.loading.`**`register_session_conf_options`**(*conf: cfg.ConfigOpts*, *group: str*, *deprecated_opts: dict[str, list[cfg.DeprecatedOpt]] | None = None*) → list[cfg.Opt]

6.1.2 Submodules

keystoneauth1.adapter module

`class keystoneauth1.adapter.Adapter`(*session: Session*, *service_type: str | None = None*, *service_name: str | None = None*, *interface: str | None = None*, *region_name: str | None = None*, *endpoint_override: str | None = None*, *version: str | None = None*, *auth: plugin.BaseAuthPlugin | None = None*, *user_agent: str | None = None*, *connect_retries: int | None = None*, *logger: Logger | None = None*, *allow: dict[str, Any] | None = None*, *additional_headers: MutableMapping[str, str] | None = None*, *client_name: str | None = None*, *client_version: str | None = None*, *allow_version_hack: bool | None = None*, *global_request_id: str | None = None*, *min_version: str | None = None*, *max_version: str | None = None*, *default_microversion: str | None = None*, *status_code_retries: int | None = None*, *retriable_status_codes: list[int] | None = None*, *raise_exc: bool | None = None*, *rate_limit: float | None = None*, *concurrency: int | None = None*, *connect_retry_delay: float | None = None*, *status_code_retry_delay: float | None = None*)

Bases: `_BaseAdapter`

`__annotations__ = {}`

`__doc__ = None`

`__module__ = 'keystoneauth1.adapter'`

`delete`(*url: str*, ***kwargs: Any*) → Response

Perform a DELETE request.

This calls `request()` with `method` set to `DELETE`.

`get`(*url: str*, ***kwargs: Any*) → Response

Perform a GET request.

This calls `request()` with `method` set to `GET`.

`head`(*url: str*, ***kwargs: Any*) → Response

Perform a HEAD request.

This calls `request()` with `method` set to `HEAD`.

`patch`(*url: str*, ***kwargs: Any*) → Response

Perform a PATCH request.

This calls `request()` with `method` set to `PATCH`.

`post`(*url: str*, ***kwargs: Any*) → Response

Perform a POST request.

This calls `request()` with `method` set to `POST`.

put(*url: str*, ***kwargs: Any*) → Response

Perform a PUT request.

This calls `request()` with `method` set to PUT.

request(*url: str*, *method: str*, ***kwargs: Any*) → Response

`class keystoneauth1.adapter.`**LegacyJsonAdapter**(*session: Session*, *service_type: str | None = None*, *service_name: str | None = None*, *interface: str | None = None*, *region_name: str | None = None*, *endpoint_override: str | None = None*, *version: str | None = None*, *auth: plugin.BaseAuthPlugin | None = None*, *user_agent: str | None = None*, *connect_retries: int | None = None*, *logger: Logger | None = None*, *allow: dict[str, Any] | None = None*, *additional_headers: MutableMapping[str, str] | None = None*, *client_name: str | None = None*, *client_version: str | None = None*, *allow_version_hack: bool | None = None*, *global_request_id: str | None = None*, *min_version: str | None = None*, *max_version: str | None = None*, *default_microversion: str | None = None*, *status_code_retries: int | None = None*, *retriable_status_codes: list[int] | None = None*, *raise_exc: bool | None = None*, *rate_limit: float | None = None*, *concurrency: int | None = None*, *connect_retry_delay: float | None = None*, *status_code_retry_delay: float | None = None*)

Bases: `_BaseAdapter`

Make something that looks like an old HTTPClient.

A common case when using an adapter is that we want an interface similar to the HTTPClients of old which returned the body as JSON as well.

You probably dont want this if you are starting from scratch.

`__annotations__ = {}`

`__doc__ = "Make something that looks like an old HTTPClient.\n\n A common case when using an adapter is that we want an interface similar to\n the HTTPClients of old which returned the body as JSON as well.\n\n You probably don't want this if you are starting from scratch.\n "`

`__module__ = 'keystoneauth1.adapter'`

delete(*url: str*, ***kwargs: Any*) → tuple[Response, object]

Perform a DELETE request.

This calls `request()` with `method` set to `DELETE`.

`get`(*url: str*, ***kwargs: Any*) → tuple[Response, object]

Perform a GET request.

This calls `request()` with `method` set to `GET`.

`head`(*url: str*, ***kwargs: Any*) → tuple[Response, object]

Perform a HEAD request.

This calls `request()` with `method` set to `HEAD`.

`patch`(*url: str*, ***kwargs: Any*) → tuple[Response, object]

Perform a PATCH request.

This calls `request()` with `method` set to `PATCH`.

`post`(*url: str*, ***kwargs: Any*) → tuple[Response, object]

Perform a POST request.

This calls `request()` with `method` set to `POST`.

`put`(*url: str*, ***kwargs: Any*) → tuple[Response, object]

Perform a PUT request.

This calls `request()` with `method` set to `PUT`.

`request`(*url: str*, *method: str*, ***kwargs: Any*) → tuple[Response, object]

`class keystoneauth1.adapter.`**`_BaseAdapter`**(*session: Session*, *service_type: str | None = None*, *service_name: str | None = None*, *interface: str | None = None*, *region_name: str | None = None*, *endpoint_override: str | None = None*, *version: str | None = None*, *auth: plugin.BaseAuthPlugin | None = None*, *user_agent: str | None = None*, *connect_retries: int | None = None*, *logger: Logger | None = None*, *allow: dict[str, Any] | None = None*, *additional_headers: MutableMapping[str, str] | None = None*, *client_name: str | None = None*, *client_version: str | None = None*, *allow_version_hack: bool | None = None*, *global_request_id: str | None = None*, *min_version: str | None = None*, *max_version: str | None = None*, *default_microversion: str | None = None*, *status_code_retries: int | None = None*, *retriable_status_codes: list[int] | None = None*, *raise_exc: bool | None = None*, *rate_limit: float | None = None*, *concurrency: int | None = None*, *connect_retry_delay: float | None = None*, *status_code_retry_delay: float | None = None*)

Bases: `object`

An instance of a session with local variables.

A session is a global object that is shared around amongst many clients. It therefore contains state that is relevant to everyone. There is a lot of state such as the service type and region_name that are only relevant to a particular client that is using the session. An adapter provides a wrapper of client local data around the global session object.

version, min_version, max_version and default_microversion can all be given either as a string or a tuple.

Parameters

- **session** (*keystoneauth1.session.Session*) The session object to wrap.
- **service_type** (*str*) The default service_type for URL discovery.
- **service_name** (*str*) The default service_name for URL discovery.
- **interface** (*str*) The default interface for URL discovery.
- **region_name** (*str*) The default region_name for URL discovery.
- **endpoint_override** (*str*) Always use this endpoint URL for requests for this client.
- **version** The minimum version restricted to a given Major API. Mutually exclusive with min_version and max_version. (optional)
- **auth** (*keystoneauth1.plugin.BaseAuthPlugin*) An auth plugin to use instead of the session one.
- **user_agent** (*str*) The User-Agent string to set.
- **connect_retries** (*int*) The maximum number of retries that should be attempted for connection errors. Default None - use session default which is dont retry.
- **logger** (*logging.Logger*) A logging object to use for requests that pass through this adapter.
- **allow** (*dict*) Extra filters to pass when discovering API versions. (optional)
- **additional_headers** (*dict*) Additional headers that should be attached to every request passing through the adapter. Headers of the same name specified per request will take priority.
- **client_name** (*str*) The name of the client that created the adapter. This will be used to create the user_agent.
- **client_version** (*str*) The version of the client that created the adapter. This will be used to create the user_agent.
- **allow_version_hack** (*bool*) Allow keystoneauth to hack up catalog URLS to support older schemes. (optional, default True)
- **global_request_id** (*str*) A global_request_id (in the form of `req-$uuid`) that will be passed on all requests. Enables cross project request id tracking.
- **min_version** The minimum major version of a given API, intended to be used as the lower bound of a range with max_version. Mutually exclusive with version. If min_version is given with no max_version it is as if max version is latest. (optional)
- **max_version** The maximum major version of a given API, intended to be used as the upper bound of a range with min_version. Mutually exclusive with version. (optional)

- **default_microversion** The default microversion value to send with API requests. While microversions are a per-request feature, a user may know they want to default to sending a specific value. (optional)
- **status_code_retries** (*int*) The maximum number of retries that should be attempted for retriable HTTP status codes (optional, defaults to 0 - never retry).
- **retriable_status_codes** (*list*) List of HTTP status codes that should be retried (optional, defaults to HTTP 503, has no effect when status_code_retries is 0).
- **raise_exc** (*bool*) If True, requests returning failing HTTP responses will raise an exception; if False, the response is returned. This can be overridden on a per-request basis via the kwarg of the same name.
- **rate_limit** (*float*) A client-side rate limit to impose on requests made through this adapter in requests per second. For instance, a rate_limit of 2 means to allow no more than 2 requests per second, and a rate_limit of 0.5 means to allow no more than 1 request every two seconds. (optional, defaults to None, which means no rate limiting will be applied).
- **concurrency** (*int*) How many simultaneous http requests this Adapter can be used for. (optional, defaults to None, which means no limit).
- **connect_retry_delay** (*float*) Delay (in seconds) between two connect retries (if enabled). By default exponential retry starting with 0.5 seconds up to a maximum of 60 seconds is used.
- **status_code_retry_delay** (*float*) Delay (in seconds) between two status code retries (if enabled). By default exponential retry starting with 0.5 seconds up to a maximum of 60 seconds is used.

`__annotations__ = {'client_name': typing.Optional[str], 'client_version': typing.Optional[str]}`

__dict__ = mappingproxy({'__module__': 'keystoneauth1.adapter', '__annotations__': {'client_name': typing.Optional[str], 'client_version': typing.Optional[str]}, '__doc__': "An instance of a session with local variables.\n\n A session is a global object that is shared around amongst many clients. It\n therefore contains state that is relevant to everyone. There is a lot of\n state such as the service type and region_name that are only relevant to a\n particular client that is using the session. An adapter provides a wrapper\n of client local data around the global session object.\n\n version, min_version, max_version and default_microversion can all be\n given either as a string or a tuple.\n\n :param session: The session object to wrap.\n :type session: keystoneauth1.session.Session\n :param str service_type: The default service_type for URL discovery.\n :param str service_name: The default service_name for URL discovery.\n :param str interface: The default interface for URL discovery.\n :param str region_name: The default region_name for URL discovery.\n :param str endpoint_override:\n Always use this endpoint URL for requests for this client.\n :param version:\n The minimum version restricted to a given Major API.\n Mutually exclusive with min_version and max_version.\n (optional)\n :param auth: An auth plugin to use instead of the session one.\n :type auth: keystoneauth1.plugin.BaseAuthPlugin\n :param str user_agent: The User-Agent string to set.\n :param int connect_retries:\n The maximum number of retries that should be attempted for\n connection errors. Default None - use session default which\n is don't retry.\n :param logger:\n A logging object to use for requests that pass through this\n adapter.\n :type logger: logging.Logger\n :param dict allow:\n Extra filters to pass when discovering API versions. (optional)\n :param dict additional_headers:\n Additional headers that should be attached to every request\n passing through the adapter. Headers of the same name specified\n per request will take priority.\n :param str client_name:\n The name of the client that created the adapter. This will be\n used to create the user_agent.\n :param str client_version:\n The version of the client that created the adapter. This will\n be used to create the user_agent.\n :param bool allow_version_hack:\n Allow keystoneauth to hack up catalog URLS to support older schemes.\n (optional, default True)\n :param str global_request_id:\n A global_request_id (in the form of ``req-$uuid``) that will be\n passed on all requests. Enables cross project request id tracking.\n :param min_version:\n The minimum major version of a given API, intended to be used as\n the lower bound of a range with max_version. Mutually exclusive with\n version. If min_version is given with no max_version it is as\n if max version is 'latest'. (optional)\n :param max_version:\n The maximum major version of a given API, intended to be used as\n the upper bound of a range with min_version. Mutually exclusive with\n version. (optional)\n :param default_microversion:\n The default microversion value to send with API requests. While\n microversions are a per-request feature, a user may know they\n want to default to sending a specific value. (optional)\n :param int status_code_retries:\n The maximum number of retries that should be attempted for retriable\n HTTP status codes (optional, defaults to 0 - never retry).\n :param list retriable_status_codes:\n List of HTTP status codes that should be retried (optional,\n defaults to HTTP 503, has no effect when status_code_retries is 0).\n :param bool raise_exc:\n If True, requests returning failing HTTP responses will raise an\n exception; if False, the response is returned. This can be\n overridden on a per-request basis via the kwarg of the same name.\n :param float rate_limit:\n A client-side rate limit to impose on requests made through this\n adapter in requests per second. For instance, a rate_limit of 2\n means to allow

```
__doc__ = "An instance of a session with local variables.\n\n A session is
a global object that is shared around amongst many clients. It\n therefore
contains state that is relevant to everyone. There is a lot of\n state
such as the service type and region_name that are only relevant to a\n
particular client that is using the session. An adapter provides a
wrapper\n of client local data around the global session object.\n\n
version, min_version, max_version and default_microversion can all be\n
given either as a string or a tuple.\n\n :param session:  The session
object to wrap.\n :type session:  keystoneauth1.session.Session\n :param
str service_type:  The default service_type for URL discovery.\n :param
str service_name:  The default service_name for URL discovery.\n :param
str interface:  The default interface for URL discovery.\n :param str
region_name:  The default region_name for URL discovery.\n :param str
endpoint_override:\n Always use this endpoint URL for requests for this
client.\n :param version:\n The minimum version restricted to a given
Major API.\n Mutually exclusive with min_version and max_version.\n
(optional)\n :param auth:  An auth plugin to use instead of the session
one.\n :type auth:  keystoneauth1.plugin.BaseAuthPlugin\n :param str
user_agent:  The User-Agent string to set.\n :param int connect_retries:\n
The maximum number of retries that should be attempted for\n connection
errors. Default None - use session default which\n is don't retry.\n
:param logger:\n A logging object to use for requests that pass through
this\n adapter.\n :type logger:  logging.Logger\n :param dict allow:\n
Extra filters to pass when discovering API versions. (optional)\n :param
dict additional_headers:\n Additional headers that should be attached to
every request\n passing through the adapter. Headers of the same name
specified\n per request will take priority.\n :param str client_name:\n
The name of the client that created the adapter. This will be\n used to
create the user_agent.\n :param str client_version:\n The version of the
client that created the adapter. This will\n be used to create the
user_agent.\n :param bool allow_version_hack:\n Allow keystoneauth to hack
up catalog URLS to support older schemes.\n (optional, default True)\n
:param str global_request_id:\n A global_request_id (in the form of
``req-$uuid``) that will be\n passed on all requests. Enables cross
project request id tracking.\n :param min_version:\n The minimum major
version of a given API, intended to be used as\n the lower bound of a
range with max_version. Mutually exclusive with\n version. If min_version
is given with no max_version it is as\n if max version is 'latest'.
(optional)\n :param max_version:\n The maximum major version of a given
API, intended to be used as\n the upper bound of a range with min_version.
Mutually exclusive with\n version. (optional)\n :param
default_microversion:\n The default microversion value to send with API
requests. While\n microversions are a per-request feature, a user may know
they\n want to default to sending a specific value. (optional)\n :param
int status_code_retries:\n The maximum number of retries that should be
attempted for retriable\n HTTP status codes (optional, defaults to 0 -
never retry).\n :param list retriable_status_codes:\n List of HTTP status
codes that should be retried (optional,\n defaults to HTTP 503, has no
effect when status_code_retries is 0).\n :param bool raise_exc:\n If True,
requests returning failing HTTP responses will raise an\n exception; if
False, the response is returned. This can be\n overridden on a per-request
basis via the kwarg of the same name.\n :param float rate_limit:\n A
client-side rate limit to impose on requests made through this\n adapter
in requests per second. For instance, a rate_limit of 2\n means to allow
no more than 2 requests per second, and a rate_limit\n of 0.5 means to
allow no more than 1 request every two seconds.\n (optional, defaults to
None, which means no rate limiting will be\n applied).\n :param int
```

__init__(*session: Session*, *service_type: str | None = None*, *service_name: str | None = None*, *interface: str | None = None*, *region_name: str | None = None*, *endpoint_override: str | None = None*, *version: str | None = None*, *auth: plugin.BaseAuthPlugin | None = None*, *user_agent: str | None = None*, *connect_retries: int | None = None*, *logger: Logger | None = None*, *allow: dict[str, Any] | None = None*, *additional_headers: MutableMapping[str, str] | None = None*, *client_name: str | None = None*, *client_version: str | None = None*, *allow_version_hack: bool | None = None*, *global_request_id: str | None = None*, *min_version: str | None = None*, *max_version: str | None = None*, *default_microversion: str | None = None*, *status_code_retries: int | None = None*, *retriable_status_codes: list[int] | None = None*, *raise_exc: bool | None = None*, *rate_limit: float | None = None*, *concurrency: int | None = None*, *connect_retry_delay: float | None = None*, *status_code_retry_delay: float | None = None*)

__module__ = 'keystoneauth1.adapter'

__weakref__

list of weak references to the object (if defined)

_request(*url: str*, *method: str*, ***kwargs: Any*) → Response

_set_endpoint_filter_kwargs(*kwargs: dict[str, object]*) → None

client_name: str | None = None

client_version: str | None = None

get_all_version_data(*interface: str = 'public'*, *region_name: str | None = None*) → dict[str, dict[str, dict[str, list[VersionData]]]]

Get data about all versions of a service.

Parameters

- **interface** Type of endpoint to get version data for. Can be a single value or a list of values. A value of None indicates that all interfaces should be queried. (optional, defaults to public)
- **region_name** (*string*) Region of endpoints to get version data for. A valueof None indicates that all regions should be queried. (optional, defaults to None)

Returns

A dictionary keyed by region_name with values containing dictionaries keyed by interface with values being a list of `VersionData`.

get_api_major_version(*auth: plugin.BaseAuthPlugin | None = None*, ***kwargs: Any*) → tuple[int | float, ...] | None

Get the major API version as provided by the auth plugin.

Parameters

auth (*keystoneauth1.plugin.BaseAuthPlugin*) The auth plugin to use for token. Overrides the plugin on the session. (optional)

Raises
: `keystoneauth1.exceptions.auth_plugins.MissingAuthPlugin` if a plugin is not available.

Returns
: The major version of the API of the service discovered.

Return type
: tuple or None

get_endpoint(*auth: plugin.BaseAuthPlugin* | *None* = *None*, ***kwargs: Any*) → str | None

Get an endpoint as provided by the auth plugin.

Parameters
: **auth** (*keystoneauth1.plugin.BaseAuthPlugin*) The auth plugin to use for token. Overrides the plugin on the session. (optional)

Raises
: `keystoneauth1.exceptions.auth_plugins.MissingAuthPlugin` if a plugin is not available.

Returns
: An endpoint if available or None.

Return type
: `str`

get_endpoint_data(*auth: plugin.BaseAuthPlugin* | *None* = *None*) → discover.EndpointData | None

Get the endpoint data for this Adapters endpoint.

Parameters
: **auth** (*keystoneauth1.plugin.BaseAuthPlugin*) The auth plugin to use for token. Overrides the plugin on the session. (optional)

Raises

- `keystoneauth1.exceptions.auth_plugins.MissingAuthPlugin` if a plugin is not available.
- `TypeError` If arguments are invalid

Returns
: Endpoint data if available or None.

Return type
: keystoneauth1.discover.EndpointData

get_project_id(*auth: plugin.BaseAuthPlugin* | *None* = *None*) → str | None

Return the authenticated project_id as provided by the auth plugin.

Parameters
: **auth** (*keystoneauth1.plugin.BaseAuthPlugin*) The auth plugin to use for token. Overrides the plugin on the session. (optional)

Raises

- `keystoneauth1.exceptions.auth.AuthorizationFailure` if a new token fetch fails.

- `keystoneauth1.exceptions.auth_plugins.MissingAuthPlugin` if a plugin is not available.

Returns
Current *project_id* or None if not supported by plugin.

Return type
`str`

get_token(*auth: plugin.BaseAuthPlugin | None = None*) → str | None

Return a token as provided by the auth plugin.

Parameters
auth (*keystoneauth1.plugin.BaseAuthPlugin*) The auth plugin to use for token. Overrides the plugin on the session. (optional)

Raises
`keystoneauth1.exceptions.auth.AuthorizationFailure` if a new token fetch fails.

Returns
A valid token.

Return type
`str`

get_user_id(*auth: plugin.BaseAuthPlugin | None = None*) → str | None

Return the authenticated user_id as provided by the auth plugin.

Parameters
auth (*keystoneauth1.plugin.BaseAuthPlugin*) The auth plugin to use for token. Overrides the plugin on the session. (optional)

Raises

- `keystoneauth1.exceptions.auth.AuthorizationFailure` if a new token fetch fails.
- `keystoneauth1.exceptions.auth_plugins.MissingAuthPlugin` if a plugin is not available.

Returns
Current *user_id* or None if not supported by plugin.

Return type
`str`

invalidate(*auth: plugin.BaseAuthPlugin | None = None*) → bool

Invalidate an authentication plugin.

Parameters
auth (*keystoneauth1.plugin.BaseAuthPlugin*) The auth plugin to invalidate. Overrides the plugin on the session. (optional)

classmethod register_argparse_arguments(*parser: ArgumentParser, service_type: str | None = None*) → None

Attach arguments to a given argparse Parser for Adapters.

Parameters

- **parser** (*argparse.ArgumentParser*) The argparse parser to attach options to.
- **service_type** (*str*) Default service_type value. (optional)

classmethod register_service_argparse_arguments(*parser: ArgumentParser*, *service_type: str*) → None

Attach arguments to a given argparse Parser for Adapters.

Parameters

- **parser** (*argparse.ArgumentParser*) The argparse parser to attach options to.
- **service_type** (*str*) Name of a service to generate additional arguments for.

keystoneauth1.adapter.**register_adapter_argparse_arguments**(*parser: ArgumentParser*, *service_type: str | None = None*) → None

keystoneauth1.adapter.**register_service_adapter_argparse_arguments**(*parser: ArgumentParser*, *service_type: str*) → None

keystoneauth1.discover module

The passive components to version discovery.

The Discover object in discover.py contains functions that can create objects on your behalf. These functions are not usable from within the keystoneauth1 library because you will get dependency resolution issues.

The Discover object in this file provides the querying components of Discovery. This includes functions like url_for which allow you to retrieve URLs and the raw data specified in version discovery responses.

class keystoneauth1.discover.**Discover**(*session: ks_session.Session*, *url: str*, *authenticated: bool | None = None*)

Bases: object

CURRENT_STATUSES = ('stable', 'current', 'supported')

DEPRECATED_STATUSES = ('deprecated',)

EXPERIMENTAL_STATUSES = ('experimental',)

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.discover', 'CURRENT_STATUSES': ('stable', 'current', 'supported'), 'DEPRECATED_STATUSES': ('deprecated',), 'EXPERIMENTAL_STATUSES': ('experimental',), '__init__': <function Discover.__init__>, 'raw_version_data': <function Discover.raw_version_data>, 'version_data': <function Discover.version_data>, 'version_string_data': <function Discover.version_string_data>, 'data_for': <function Discover.data_for>, 'url_for': <function Discover.url_for>, 'versioned_data_for': <function Discover.versioned_data_for>, 'versioned_url_for': <function Discover.versioned_url_for>, '__dict__': <attribute '__dict__' of 'Discover' objects>, '__weakref__': <attribute '__weakref__' of 'Discover' objects>, '__doc__': None, '__annotations__': {}})`

`__doc__ = None`

`__init__`(*session: ks_session.Session*, *url: str*, *authenticated: bool | None = None*)

`__module__ = 'keystoneauth1.discover'`

`__weakref__`

list of weak references to the object (if defined)

`data_for`(*version: str | int | float | Iterable[str | int | float]*, ***, *allow_experimental: bool = False*, *allow_deprecated: bool = True*, *allow_unknown: bool = False*) → VersionData | None

Return endpoint data for a version.

NOTE: This method raises a TypeError if version is None. It is
kept for backwards compatability. New code should use versioned_data_for instead.

Parameters
`version` (*tuple*) The version is always a minimum version in the same major release as there should be no compatibility issues with using a version newer than the one asked for.

Returns
the endpoint data for a URL that matches the required version (the format is described in version_data) or None if no match.

Return type
dict

`raw_version_data`(*allow_experimental: bool = False*, *allow_deprecated: bool = True*, *allow_unknown: bool = False*) → list[dict[str, Any]]

Get raw version information from URL.

Raw data indicates that only minimal validation processing is performed on the data, so what is returned here will be the data in the same format it was received from the endpoint.

Parameters

- `allow_experimental` (*bool*) Allow experimental version endpoints.
- `allow_deprecated` (*bool*) Allow deprecated version endpoints.
- `allow_unknown` (*bool*) Allow endpoints with an unrecognised status.

Returns
The endpoints returned from the server that match the criteria.

Return type
list

url_for(*version: str | int | float | Iterable[str | int | float]*, ***, *allow_experimental: bool = False*, *allow_deprecated: bool = True*, *allow_unknown: bool = False*) → str | None

Get the endpoint url for a version.

NOTE: This method raises a TypeError if version is None. It is
kept for backwards compatability. New code should use versioned_url_for instead.

Parameters
version (*tuple*) The version is always a minimum version in the same major release as there should be no compatibility issues with using a version newer than the one asked for.

Returns
The url for the specified version or None if no match.

Return type
str

version_data(*reverse: bool = False*, ***, *allow_experimental: bool = False*, *allow_deprecated: bool = True*, *allow_unknown: bool = False*) → list[VersionData]

Get normalized version data.

Return version data in a structured way.

Parameters
reverse (*bool*) Reverse the list. reverse=true will mean the returned list is sorted from newest to oldest version.

Returns
A list of `VersionData` sorted by version number.

Return type
list(VersionData)

version_string_data(*reverse: bool = False*, ***, *allow_experimental: bool = False*, *allow_deprecated: bool = True*, *allow_unknown: bool = False*) → list[VersionData]

Get normalized version data with versions as strings.

Return version data in a structured way.

Parameters
reverse (*bool*) Reverse the list. reverse=true will mean the returned list is sorted from newest to oldest version.

Returns
A list of `VersionData` sorted by version number.

Return type
list(VersionData)

versioned_data_for(*url: str | None = None*, *min_version: str | int | float | Iterable[str | int | float] | None = None*, *max_version: str | int | float | Iterable[str | int | float] | None = None*, ***, *allow_experimental: bool = False*, *allow_deprecated: bool = True*, *allow_unknown: bool = False*) → VersionData | None

Return endpoint data for the service at a url.

min_version and max_version can be given either as strings or tuples.

Parameters

- **url** (*string*) If url is given, the data will be returned for the endpoint data that has a self link matching the url.
- **min_version** The minimum endpoint version that is acceptable. If min_version is given with no max_version it is as if max version is latest. If min_version is latest, max_version may only be latest or None.
- **max_version** The maximum endpoint version that is acceptable. If min_version is given with no max_version it is as if max version is latest. If min_version is latest, max_version may only be latest or None.

Returns

the endpoint data for a URL that matches the required version (the format is described in version_data) or None if no match.

Return type

dict

versioned_url_for(*min_version: str | int | float | Iterable[str | int | float] | None = None*, *max_version: str | int | float | Iterable[str | int | float] | None = None*, ***, *allow_experimental: bool = False*, *allow_deprecated: bool = True*, *allow_unknown: bool = False*) → str | None

Get the endpoint url for a version.

min_version and max_version can be given either as strings or tuples.

Parameters

- **min_version** The minimum version that is acceptable. If min_version is given with no max_version it is as if max version is latest.
- **max_version** The maximum version that is acceptable. If min_version is given with no max_version it is as if max version is latest.

Returns

The url for the specified version or None if no match.

Return type

str

class keystoneauth1.discover.**EndpointData**(*catalog_url: str | None = None, service_url: str | None = None, service_type: str | None = None, service_name: str | None = None, service_id: str | None = None, region_name: str | None = None, interface: str | None = None, endpoint_id: str | None = None, raw_endpoint: str | None = None, api_version: tuple[int | float, ...] | None = None, major_version: str | None = None, min_microversion: tuple[int | float, ...] | None = None, max_microversion: tuple[int | float, ...] | None = None, next_min_version: str | None = None, not_before: str | None = None, status: str | None = None*)

Bases: `object`

Normalized information about a discovered endpoint.

Contains url, version, microversion, interface and region information. This is essentially the data contained in the catalog and the version discovery documents about an endpoint that is used to select the endpoint desired by the user. It is returned so that a user can know which qualities a discovered endpoint had, in case their request allowed for a range of possibilities.

Refer to the microversion specification for more information.

https://specs.openstack.org/openstack/api-wg/guidelines/microversion_specification.html

__copy__() → EndpointData

Return a new EndpointData based on this one.

__dict__ = mappingproxy({'__module__': 'keystoneauth1.discover', '__doc__': 'Normalized information about a discovered endpoint.\n\n Contains url, version, microversion, interface and region information.\n This is essentially the data contained in the catalog and the version\n discovery documents about an endpoint that is used to select the endpoint\n desired by the user. It is returned so that a user can know which qualities\n a discovered endpoint had, in case their request allowed for a range of\n possibilities.\n\n Refer to the microversion specification for more information.\n\n https://specs.openstack.org/openstack/api-wg/guidelines/microversion_specification.html\n ', '__init__': <function EndpointData.__init__>, '__copy__': <function EndpointData.__copy__>, '__str__': <function EndpointData.__str__>, 'url': <property object>, 'get_current_versioned_data': <function EndpointData.get_current_versioned_data>, 'get_versioned_data': <function EndpointData.get_versioned_data>, 'get_all_version_string_data': <function EndpointData.get_all_version_string_data>, '_infer_version_data': <function EndpointData._infer_version_data>, '_set_version_info': <function EndpointData._set_version_info>, '_run_discovery': <function EndpointData._run_discovery>, '_get_discovery_url_choices': <function EndpointData._get_discovery_url_choices>, '_get_catalog_discover_hack': <function EndpointData._get_catalog_discover_hack>, '__dict__': <attribute '__dict__' of 'EndpointData' objects>, '__weakref__': <attribute '__weakref__' of 'EndpointData' objects>, '__annotations__': {'_saved_project_id': 'ty.Optional[str]', '_catalog_matches_exactly': 'bool', '_disc': 'ty.Optional[Discover]'}})

`__doc__` = **`'Normalized information about a discovered endpoint.\n\n Contains url, version, microversion, interface and region information.\n This is essentially the data contained in the catalog and the version\n discovery documents about an endpoint that is used to select the endpoint\n desired by the user. It is returned so that a user can know which qualities\n a discovered endpoint had, in case their request allowed for a range of\n possibilities.\n\n Refer to the microversion specification for more information.\n\n https://specs.openstack.org/ openstack/api-wg/guidelines/microversion_specification.html\n '`**

`__init__`(*catalog_url: str | None = None*, *service_url: str | None = None*, *service_type: str | None = None*, *service_name: str | None = None*, *service_id: str | None = None*, *region_name: str | None = None*, *interface: str | None = None*, *endpoint_id: str | None = None*, *raw_endpoint: str | None = None*, *api_version: tuple[int | float, ...] | None = None*, *major_version: str | None = None*, *min_microversion: tuple[int | float, ...] | None = None*, *max_microversion: tuple[int | float, ...] | None = None*, *next_min_version: str | None = None*, *not_before: str | None = None*, *status: str | None = None*)

`__module__` = **`'keystoneauth1.discover'`**

`__str__`() → str

Produce a string like EndpointData{key=val, }, for debugging.

`__weakref__`

list of weak references to the object (if defined)

`_get_catalog_discover_hack`() → str

Apply the catalog hacks and figure out an unversioned endpoint.

This function is internal to keystoneauth1.

Returns

A url that has been transformed by the regex hacks that match the service_type.

`_get_discovery_url_choices`(*project_id: str | None*, *allow_version_hack: bool = True*, *min_version: tuple[int | float, ...] | None = None*, *max_version: tuple[int | float, ...] | None = None*) → Generator[str, None, None]

Find potential locations for version discovery URLs.

min_version and max_version are already normalized, so will either be None or a tuple.

`_infer_version_data`(*project_id: str | None = None*) → list[VersionData]

Return version data dict for when discovery fails.

Parameters

`project_id` (*`string`*) ID of the currently scoped project. Used for removing project_id components of URLs from the catalog. (optional)

Returns

A list of `VersionData` sorted by version number.

Return type

list(VersionData)

_run_discovery(*session: ks_session.Session*, *cache: dict[str, Discover] | None*, *min_version: tuple[int | float, ...] | None*, *max_version: tuple[int | float, ...] | None*, *project_id: str | None*, *allow_version_hack: bool*, *discover_versions: bool*) → None

_set_version_info(*session: ks_session.Session*, *allow: dict[str, Any]*, *cache: dict[str, Discover] | None*, *allow_version_hack: bool*, *project_id: str | None*, *discover_versions: bool*, *min_version: tuple[int | float, ...] | None*, *max_version: tuple[int | float, ...] | None*) → None

get_all_version_string_data(*session: ks_session.Session*, *project_id: str | None = None*) → list[VersionData]

Return version data for all versions discovery can find.

Parameters

project_id (*string*) ID of the currently scoped project. Used for removing project_id components of URLs from the catalog. (optional)

Returns

A list of `VersionData` sorted by version number.

Return type

list(VersionData)

get_current_versioned_data(*session: ks_session.Session*, *allow: dict[str, Any] | None = None*, *cache: dict[str, Discover] | None = None*, *project_id: str | None = None*) → EndpointData

Run version discovery on the current endpoint.

A simplified version of get_versioned_data, get_current_versioned_data runs discovery but only on the endpoint that has been found already.

It can be useful in some workflows where the user wants version information about the endpoint they have.

Parameters

- **session** (*keystoneauth1.session.Session*) A session object that can be used for communication.
- **allow** (*dict*) Extra filters to pass when discovering API versions. (optional)
- **cache** (*dict*) A dict to be used for caching results in addition to caching them on the Session. (optional)
- **project_id** (*string*) ID of the currently scoped project. Used for removing project_id components of URLs from the catalog. (optional)

Returns

A new EndpointData with the requested versioned data.

Return type

`keystoneauth1.discover.EndpointData`

Raises

keystoneauth1.exceptions.discovery.DiscoveryFailure If the appropriate versioned data could not be discovered.

get_versioned_data(*session: ks_session.Session*, *allow: dict[str, Any] | None = None*, *cache: dict[str, Any] | None = None*, *allow_version_hack: bool = True*, *project_id: str | None = None*, *discover_versions: bool = True*, *min_version: str | int | float | Iterable[str | int | float] | None = None*, *max_version: str | int | float | Iterable[str | int | float] | None = None*) → EndpointData

Run version discovery for the service described.

Performs Version Discovery and returns a new EndpointData object with information found.

min_version and max_version can be given either as strings or tuples.

Parameters

- **session** (*keystoneauth1.session.Session*) A session object that can be used for communication.
- **allow** (*dict*) Extra filters to pass when discovering API versions. (optional)
- **cache** (*dict*) A dict to be used for caching results in addition to caching them on the Session. (optional)
- **allow_version_hack** (*bool*) Allow keystoneauth to hack up catalog URLS to support older schemes. (optional, default True)
- **project_id** (*string*) ID of the currently scoped project. Used for removing project_id components of URLs from the catalog. (optional)
- **discover_versions** (*bool*) Whether to get version metadata from the version discovery document even if its not neccessary to fulfill the major version request. (optional, defaults to True)
- **min_version** The minimum version that is acceptable. If min_version is given with no max_version it is as if max version is latest.
- **max_version** The maximum version that is acceptable. If min_version is given with no max_version it is as if max version is latest.

Returns

A new EndpointData with the requested versioned data.

Return type

`keystoneauth1.discover.EndpointData`

Raises

keystoneauth1.exceptions.discovery.DiscoveryFailure If the appropriate versioned data could not be discovered.

property url: str | None

class keystoneauth1.discover.**Status**

Bases: `object`

CURRENT = 'CURRENT'

DEPRECATED = 'DEPRECATED'

EXPERIMENTAL = 'EXPERIMENTAL'

`KNOWN = ('CURRENT', 'SUPPORTED', 'DEPRECATED', 'EXPERIMENTAL')`

`SUPPORTED = 'SUPPORTED'`

`UNKNOWN = 'UNKNOWN'`

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.discover', 'CURRENT': 'CURRENT', 'SUPPORTED': 'SUPPORTED', 'DEPRECATED': 'DEPRECATED', 'EXPERIMENTAL': 'EXPERIMENTAL', 'UNKNOWN': 'UNKNOWN', 'KNOWN': ('CURRENT', 'SUPPORTED', 'DEPRECATED', 'EXPERIMENTAL'), 'normalize': <classmethod(<function Status.normalize>)>, '__dict__': <attribute '__dict__' of 'Status' objects>, '__weakref__': <attribute '__weakref__' of 'Status' objects>, '__doc__': None, '__annotations__': {}})`

`__doc__ = None`

`__module__ = 'keystoneauth1.discover'`

`__weakref__`

list of weak references to the object (if defined)

`classmethod` **`normalize`**(*raw_status: str*) → str

Turn a status into a canonical status value.

If the status from the version discovery document does not match one of the known values, it will be set to UNKNOWN.

Parameters

raw_status (*str*) Status value from a discovery document.

Returns

A canonicalized version of the status. Valid values are CURRENT, SUPPORTED, DEPRECATED, EXPERIMENTAL and UNKNOWN

Return type

str

`class keystoneauth1.discover.`**`VersionData`**(*version: tuple[int | float, ...] | str | None, url: str, collection: str | None = None, max_microversion: tuple[int | float, ...] | str | None = None, min_microversion: tuple[int | float, ...] | str | None = None, next_min_version: tuple[int | float, ...] | str | None = None, not_before: str | None = None, status: str = 'CURRENT', raw_status: str | None = None*)

Bases: `dict[str, Any]`

Normalized Version Data about an endpoint.

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.discover', '__doc__': 'Normalized Version Data about an endpoint.', '__init__': <function VersionData.__init__>, 'version': <property object>, 'url': <property object>, 'collection': <property object>, 'min_microversion': <property object>, 'max_microversion': <property object>, 'status': <property object>, 'raw_status': <property object>, '__orig_bases__': (dict[str, typing.Any],), '__dict__': <attribute '__dict__' of 'VersionData' objects>, '__weakref__': <attribute '__weakref__' of 'VersionData' objects>, '__annotations__': {}})`

`__doc__ = 'Normalized Version Data about an endpoint.'`

`__init__`(*version: tuple[int | float, ...] | str | None, url: str, collection: str | None = None, max_microversion: tuple[int | float, ...] | str | None = None, min_microversion: tuple[int | float, ...] | str | None = None, next_min_version: tuple[int | float, ...] | str | None = None, not_before: str | None = None, status: str = 'CURRENT', raw_status: str | None = None*)

`__module__ = 'keystoneauth1.discover'`

`__orig_bases__ = (dict[str, typing.Any],)`

`__weakref__`

list of weak references to the object (if defined)

`property collection: str | None`

The URL for the discovery document.

May be None.

`property max_microversion: tuple[int | float, ...] | None`

The maximum microversion supported by the endpoint.

May be None.

`property min_microversion: tuple[int | float, ...] | None`

The minimum microversion supported by the endpoint.

May be None.

`property raw_status: str | None`

The status as provided by the server.

`property status: str`

A canonicalized version of the status.

Valid values are CURRENT, SUPPORTED, DEPRECATED and EXPERIMENTAL.

`property url: str`

The url for the endpoint.

`property version: tuple[int | float, ...] | None`

The normalized version of the endpoint.

class keystoneauth1.discover.**_VersionHacks**

Bases: object

A container to abstract the list of version hacks.

This could be done as simply a dictionary but is abstracted like this to make for easier testing.

__dict__ = mappingproxy({'__module__': 'keystoneauth1.discover', '__doc__': 'A container to abstract the list of version hacks.\n\n This could be done as simply a dictionary but is abstracted like this to\n make for easier testing.\n ', '__init__': <function _VersionHacks.__init__>, 'add_discover_hack': <function _VersionHacks.add_discover_hack>, 'get_discover_hack': <function _VersionHacks.get_discover_hack>, '__dict__': <attribute '__dict__' of '_VersionHacks' objects>, '__weakref__': <attribute '__weakref__' of '_VersionHacks' objects>, '__annotations__': {'_discovery_data': 'dict[str, list[tuple[re.Pattern[str], str]]]'}})

__doc__ = 'A container to abstract the list of version hacks.\n\n This could be done as simply a dictionary but is abstracted like this to\n make for easier testing.\n '

__init__() → None

__module__ = 'keystoneauth1.discover'

__weakref__

list of weak references to the object (if defined)

add_discover_hack(*service_type: str*, *old: Pattern[str]*, *new: str = ''*) → None

Add a new hack for a service type.

Parameters

- **service_type** (*str*) The service_type in the catalog.
- **old** (*re.RegexObject*) The pattern to use.
- **new** (*str*) What to replace the pattern with.

get_discover_hack(*service_type: str*, *url: str*) → str

Apply the catalog hacks and figure out an unversioned endpoint.

Parameters

- **service_type** (*str*) the service_type to look up.
- **url** (*str*) The original url that came from a service_catalog.

Returns

Either the unversioned url or the one from the catalog to try.

keystoneauth1.discover.**_combine_relative_url**(*discovery_url: str*, *version_url: str*) → str

keystoneauth1.discover.**_int_or_latest**(*val: str | float*) → int | float

Convert val to an int or the special value LATEST.

Parameters

val An int()-able, or the string latest, or the special value LATEST.

Returns
An int, or the special value LATEST

`keystoneauth1.discover.`**`_latest_soft_match`**(*required: tuple[int | float, ...] | None*, *candidate: tuple[int | float, ...]*) → bool

`keystoneauth1.discover.`**`_normalize_version_args`**(*version: str | int | float | Iterable[str | int | float] | None*, *min_version: str | int | float | Iterable[str | int | float] | None*, *max_version: str | int | float | Iterable[str | int | float] | None*, *service_type: str | None = None*) → tuple[tuple[int | float, ...] | None, tuple[int | float, ...] | None]

`keystoneauth1.discover.`**`_str_or_latest`**(*val: str | int | float*) → str

Convert val to a string, handling LATEST => latest.

Parameters
`val` An int or the special value LATEST.

Returns
A string representation of val. If val was LATEST, the return is latest.

`keystoneauth1.discover.`**`_version_from_url`**(*url: str | None*) → tuple[int | float, ...] | None

`keystoneauth1.discover.`**`add_catalog_discover_hack`**(*service_type: str*, *old: Pattern[str]*, *new: str*) → None

Add a version removal rule for a particular service.

Originally deployments of OpenStack would contain a versioned endpoint in the catalog for different services. E.g. an identity service might look like `http://localhost:5000/v2.0`. This is a problem when we want to use a different version like v3.0 as there is no way to tell where it is located. We cannot simply change all service catalogs either so there must be a way to handle the older style of catalog.

This function adds a rule for a given service type that if part of the URL matches a given regular expression in *old* then it will be replaced with the *new* value. This will replace all instances of old with new. It should therefore contain a regex anchor.

For example the included rule states:

```
add_catalog_version_hack('identity', re.compile('/v2.0/?$'), '/')
```

so if the catalog retrieves an *identity* URL that ends with /v2.0 or /v2.0/ then it should replace it simply with / to fix the users catalog.

Parameters

- **`service_type`** (*`str`*) The service type as defined in the catalog that the rule will apply to.
- **`old`** (*`re.RegexObject`*) The regular expression to search for and replace if found.
- **`new`** (*`str`*) The new string to replace the pattern with.

`keystoneauth1.discover.get_discovery`(*session: ks_session.Session*, *url: str*, *cache: dict[str, Discover] | None = None*, *authenticated: bool | None = False*) → Discover

Return the discovery object for a URL.

Check the session and the plugin cache to see if we have already performed discovery on the URL and if so return it, otherwise create a new discovery object, cache it and return it.

NOTE: This function is expected to be used by keystoneauth and should not be needed by users part of normal usage. A normal user should use get_endpoint or get_endpoint_data on *keystoneauth.session.Session* or endpoint_filters on *keystoneauth.session.Session* or *keystoneauth.session.Session*. However, should the user need to perform direct discovery for some reason, this function should be used so that the discovery caching is used.

Parameters

- **session** (*keystoneauth1.session.Session*) A session object to discover with.
- **url** (*str*) The url to lookup.
- **cache** (*dict*) A dict to be used for caching results, in addition to caching them on the Session. (optional) Defaults to None.
- **authenticated** (*bool*) Include a token in the discovery call. (optional) Defaults to None, which will use a token if an auth plugin is installed.

Raises

- **keystoneauth1.exceptions.discovery.DiscoveryFailure** if for some reason the lookup fails.
- **keystoneauth1.exceptions.http.HttpError** An error from an invalid HTTP response.

Returns

A discovery object with the results of looking up that URL.

Return type

keystoneauth1.discover.Discovery

`keystoneauth1.discover.get_version_data`(*session: ks_session.Session*, *url: str*, *authenticated: bool | None = None*, *version_header: str | None = None*) → list[dict[str, Any]]

Retrieve raw version data from a url.

The return is a list of dicts of the form:

```
[
    {
        'status': 'STABLE',
        'id': 'v2.3',
        'links': [
            {'href': 'http://network.example.com/v2.3', 'rel': 'self'},
            {'href': 'http://network.example.com/', 'rel': 'collection'},
        ],
```

(continues on next page)

(continued from previous page)

```
        'min_version': '2.0',
        'max_version': '2.7',
    },
    ...,
]
```

Note: The maximum microversion may be specified by *max_version* or *version*, the former superseding the latter. All **version* keys are optional. Other keys and links entries are permitted, but ignored.

Parameters

- **session** (*keystoneauth1.session.Session*) A Session object that can be used for communication.
- **url** (*string*) Endpoint or discovery URL from which to retrieve data.
- **authenticated** (*bool*) Include a token in the discovery call. (optional) Defaults to None.
- **version_header** (*string*) provide the OpenStack-API-Version header for services which dont return version information without it, for backward compatibility.

Returns
A list of dicts containing version information.

Return type
list(dict)

keystoneauth1.discover.**normalize_version_number**(*version: str | int | float | Iterable[str | int | float]*) → tuple[int | float, ...]

Turn a version representation into a tuple.

Examples:

The following all produce a return value of (1, 0):

```
1, '1', 'v1', [1], (1,), ['1'], 1.0, '1.0', 'v1.0', (1, 0)
```

The following all produce a return value of (1, 20, 3):

```
'v1.20.3', '1.20.3', (1, 20, 3), ['1', '20', '3']
```

The following all produce a return value of (LATEST, LATEST):

```
'latest', 'vlatest', ('latest', 'latest'), (LATEST, LATEST)
```

The following all produce a return value of (2, LATEST):

```
'2.latest', 'v2.latest', (2, LATEST), ('2', 'latest')
```

Parameters
version A version specifier in any of the following forms: String, possibly prefixed with v, containing one or more numbers *or* the string latest, separated by

periods. Examples: v1, v1.2, 1.2.3, 123, latest, 1.latest, v1.latest. Integer. This will be assumed to be the major version, with a minor version of 0. Float. The integer part is assumed to be the major version; the decimal part the minor version. Non-string iterable comprising integers, integer strings, the string latest, or the special value LATEST. Examples: (1,), [1, 2], (12, 34, 56), (LATEST,), (2, latest)

Returns

A tuple of len >= 2 comprising integers and/or LATEST.

Raises

`TypeError` If the input version cannot be interpreted.

`keystoneauth1.discover.`**`version_between`**(*min_version: str | int | float | Iterable[str | int | float] | None*, *max_version: str | int | float | Iterable[str | int | float] | None*, *candidate: str | int | float | Iterable[str | int | float]*) → bool

Determine whether a candidate version is within a specified range.

Parameters

- **`min_version`** The minimum version that is acceptable. None/empty indicates no lower bound.
- **`max_version`** The maximum version that is acceptable. None/empty indicates no upper bound.
- **`candidate`** Candidate version to test. May not be None/empty.

Returns

True if candidate is between min_version and max_version; False otherwise.

Raises

- `ValueError` If candidate is None.
- `TypeError` If any input cannot be normalized.

`keystoneauth1.discover.`**`version_match`**(*required: tuple[int | float, ...]*, *candidate: tuple[int | float, ...]*) → bool

Test that an available version satisfies the required version.

To be suitable a version must be of the same major version as required and be at least a match in minor/patch level.

eg. 3.3 is a match for a required 3.1 but 4.1 is not.

Parameters

- **`required`** (*`tuple`*) the version that must be met.
- **`candidate`** (*`tuple`*) the version to test against required.

Returns

True if candidate is suitable False otherwise.

Return type

bool

keystoneauth1.discover.**version_to_string**(*version: tuple[int | float, ...]*) → str

Turn a version tuple into a string.

Parameters

version (*tuple*) A version represented as a tuple of ints. As a special case, a tuple member may be LATEST, which translates to latest.

Returns

A version represented as a period-delimited string.

keystoneauth1.http_basic module

class keystoneauth1.http_basic.**HTTPBasicAuth**(*endpoint: str | None = None*, *username: str | None = None*, *password: str | None = None*)

Bases: FixedEndpointPlugin

A provider that will always use HTTP Basic authentication.

This is useful to unify session/adapter loading for services that might be deployed in standalone mode.

__annotations__ = {}

__doc__ = 'A provider that will always use HTTP Basic authentication.\n\n This is useful to unify session/adapter loading for services\n that might be deployed in standalone mode.\n '

__init__(*endpoint: str | None = None*, *username: str | None = None*, *password: str | None = None*)

__module__ = 'keystoneauth1.http_basic'

get_headers(*session: ks_session.Session*) → dict[str, str] | None

Fetch authentication headers for message.

This is a more generalized replacement of the older get_token to allow plugins to specify different or additional authentication headers to the OpenStack standard X-Auth-Token header.

How the authentication headers are obtained is up to the plugin. If the headers are still valid they may be re-used, retrieved from cache or the plugin may invoke an authentication request against a server.

The default implementation of get_headers calls the *get_token* method to enable older style plugins to continue functioning unchanged. Subclasses should feel free to completely override this function to provide the headers that they want.

Returning None will indicate that no token was able to be retrieved and that authorization was a failure. Adding no authentication data can be achieved by returning an empty dictionary.

Parameters

session (*keystoneauth1.session.Session*) The session object that the auth_plugin belongs to.

Returns

Headers that are set to authenticate a message or None for failure. Note that when checking this value that the empty dict is a valid, non-failure response.

Return type
dict

get_token(*session: ks_session.Session*) → str | None

Obtain a token.

How the token is obtained is up to the plugin. If it is still valid it may be re-used, retrieved from cache or invoke an authentication request against a server.

Returning None will indicate that no token was able to be retrieved.

This function is misplaced as it should only be required for auth plugins that use the X-Auth-Token header. However due to the way plugins evolved this method is required and often called to trigger an authentication request on a new plugin.

When implementing a new plugin it is advised that you implement this method, however if you dont require the X-Auth-Token header override the *get_headers* method instead.

Parameters
session (*keystoneauth1.session.Session*) A session object so the plugin can make HTTP calls.

Returns
A token to use.

Return type
string

keystoneauth1.noauth module

class keystoneauth1.noauth.**NoAuth**(*endpoint: str | None = None*)

Bases: FixedEndpointPlugin

A provider that will always use no auth.

This is useful to unify session/adapter loading for services that might be deployed in standalone/noauth mode.

__annotations__ = {}

__doc__ = 'A provider that will always use no auth.\n\n This is useful to unify session/adapter loading for services\n that might be deployed in standalone/noauth mode.\n '

__module__ = 'keystoneauth1.noauth'

get_token(*session: ks_session.Session*) → str | None

Obtain a token.

How the token is obtained is up to the plugin. If it is still valid it may be re-used, retrieved from cache or invoke an authentication request against a server.

Returning None will indicate that no token was able to be retrieved.

This function is misplaced as it should only be required for auth plugins that use the X-Auth-Token header. However due to the way plugins evolved this method is required and often called to trigger an authentication request on a new plugin.

When implementing a new plugin it is advised that you implement this method, however if you dont require the X-Auth-Token header override the *get_headers* method instead.

Parameters
`session` (*keystoneauth1.session.Session*) A session object so the plugin can make HTTP calls.

Returns
A token to use.

Return type
string

keystoneauth1.plugin module

`class keystoneauth1.plugin.BaseAuthPlugin`

Bases: `object`

The basic structure of an authentication plugin.

Note

See *Authentication Plugins* for a description of plugins provided by this library.

`__annotations__ = {}`

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.plugin', '__doc__': 'The basic structure of an authentication plugin.\n\n .. note::\n See :doc:`/authentication-plugins` for a description of plugins\n provided by this library.\n\n ', '__init__': <function BaseAuthPlugin.__init__>, 'get_token': <function BaseAuthPlugin.get_token>, 'get_auth_ref': <function BaseAuthPlugin.get_auth_ref>, 'get_headers': <function BaseAuthPlugin.get_headers>, 'get_endpoint_data': <function BaseAuthPlugin.get_endpoint_data>, 'get_api_major_version': <function BaseAuthPlugin.get_api_major_version>, 'get_all_version_data': <function BaseAuthPlugin.get_all_version_data>, 'get_endpoint': <function BaseAuthPlugin.get_endpoint>, 'get_connection_params': <function BaseAuthPlugin.get_connection_params>, 'invalidate': <function BaseAuthPlugin.invalidate>, 'get_user_id': <function BaseAuthPlugin.get_user_id>, 'get_project_id': <function BaseAuthPlugin.get_project_id>, 'get_sp_auth_url': <function BaseAuthPlugin.get_sp_auth_url>, 'get_sp_url': <function BaseAuthPlugin.get_sp_url>, 'get_cache_id': <function BaseAuthPlugin.get_cache_id>, 'get_auth_state': <function BaseAuthPlugin.get_auth_state>, 'set_auth_state': <function BaseAuthPlugin.set_auth_state>, '__dict__': <attribute '__dict__' of 'BaseAuthPlugin' objects>, '__weakref__': <attribute '__weakref__' of 'BaseAuthPlugin' objects>, '__annotations__': {'_discovery_cache': 'dict[str, discover.Discover]'}})`

`__doc__ = 'The basic structure of an authentication plugin.\n\n .. note::\n See :doc:`/authentication-plugins` for a description of plugins\n provided by this library.\n\n '`

`__init__()` → None

`__module__ = 'keystoneauth1.plugin'`

`__weakref__`

list of weak references to the object (if defined)

`get_all_version_data`(*session: ks_session.Session*, *interface: str = 'public'*, *region_name: str | None = None*, *service_type: str | None = None*) → dict[str, dict[str, dict[str, list[VersionData]]]]

Get version data for all services in the catalog.

Parameters

- **`session`** (*`keystoneauth1.session.Session`*) A session object that can be used for communication.
- **`interface`** Type of endpoint to get version data for. Can be a single value or a list of values. A value of None indicates that all interfaces should be queried. (optional, defaults to public)
- **`region_name`** (*`string`*) Region of endpoints to get version data for. A valueof None indicates that all regions should be queried. (optional, defaults to None)
- **`service_type`** (*`string`*) Limit the version data to a single service. (optional, defaults to None)

Returns

A dictionary keyed by region_name with values containing dictionaries keyed by interface with values being a list of `VersionData`.

`get_api_major_version`(*session: ks_session.Session*, ***, *endpoint_override: str | None = None*, ***kwargs: Any*) → tuple[int | float, ...] | None

Get the major API version from the endpoint.

Parameters

- **`session`** (*`keystoneauth1.session.Session`*) A session object that can be used for communication.
- **`endpoint_override`** (*`str`*) URL to use for version discovery.
- **`kwargs`** Ignored.

Raises

`keystoneauth1.exceptions.http.HttpError` An error from an invalid HTTP response.

Returns

Valid EndpointData or None if not available.

Return type

keystoneauth1.discover.EndpointData or None

`get_auth_ref`(*session: ks_session.Session*) → access.AccessInfo | None

Return the authentication reference of an auth plugin.

Parameters

`session` (*`keystoneauth1.session.session`*) A session object to be used for communication

Returns
An access info object if supported.

Return type
keystoneauth1.access.AccessInfo or None

`get_auth_state()` → object

Retrieve the current authentication state for the plugin.

Retrieve any internal state that represents the authenticated plugin.

This should not fetch any new data if it is not present.

Raises
`NotImplementedError` if the plugin does not support this feature.

Returns
raw python data (which can be JSON serialized) that can be moved into another plugin (of the same type) to have the same authenticated state.

Return type
object or None if unauthenticated.

`get_cache_id()` → str | None

Fetch an identifier that uniquely identifies the auth options.

The returned identifier need not be decomposable or otherwise provide anyway to recreate the plugin. It should not contain sensitive data in plaintext.

This string MUST change if any of the parameters that are used to uniquely identity this plugin change.

If get_cache_id returns a str value suggesting that caching is supported then get_auth_cache and set_auth_cache must also be implemented.

Returns
A unique string for the set of options

Return type
str or None if this is unsupported or unavailable.

`get_connection_params`(*session: ks_session.Session*) → ConnectionParams

Return any additional connection parameters required for the plugin.

Parameters
session (*keystoneauth1.session.Session*) The session object that the auth_plugin belongs to.

Returns
Parameters that are passed to the requests library. Only the `cert` and `verify` parameters may be returned.

Return type
dict

`get_endpoint`(*session: ks_session.Session*, ***kwargs: Any*) → str | None

Return an endpoint for the client.

There are no required keyword arguments to `get_endpoint` as a plugin implementation should use best effort with the information available to determine the endpoint. However there are certain standard options that will be generated by the clients and should be used by plugins:

- `service_type`: what sort of service is required.
- `service_name`: the name of the service in the catalog.
- `interface`: what visibility the endpoint should have.
- `region_name`: the region the endpoint exists in.

Parameters

- **session** (*keystoneauth1.session.Session*) The session object that the auth_plugin belongs to.
- **kwargs** Ignored.

Returns

The base URL that will be used to talk to the required service or None if not available.

Return type

string

get_endpoint_data(*session: ks_session.Session*, **, *endpoint_override: str | None = None*, *discover_versions: bool = True*, ***kwargs: Any*) → EndpointData | None

Return a valid endpoint data for a the service.

Parameters

- **session** (*keystoneauth1.session.Session*) A session object that can be used for communication.
- **endpoint_override** (*str*) URL to use for version discovery.
- **discover_versions** (*bool*) Whether to get version metadata from the version discovery document even if it major api version info can be inferred from the url. (optional, defaults to True)
- **kwargs** Ignored.

Raises

`keystoneauth1.exceptions.http.HttpError` An error from an invalid HTTP response.

Returns

Valid EndpointData or None if not available.

Return type

keystoneauth1.discover.EndpointData or None

get_headers(*session: ks_session.Session*) → dict[str, str] | None

Fetch authentication headers for message.

This is a more generalized replacement of the older get_token to allow plugins to specify different or additional authentication headers to the OpenStack standard X-Auth-Token header.

How the authentication headers are obtained is up to the plugin. If the headers are still valid they may be re-used, retrieved from cache or the plugin may invoke an authentication request against a server.

The default implementation of get_headers calls the *get_token* method to enable older style plugins to continue functioning unchanged. Subclasses should feel free to completely override this function to provide the headers that they want.

Returning None will indicate that no token was able to be retrieved and that authorization was a failure. Adding no authentication data can be achieved by returning an empty dictionary.

Parameters
: **session** (*keystoneauth1.session.Session*) The session object that the auth_plugin belongs to.

Returns
: Headers that are set to authenticate a message or None for failure. Note that when checking this value that the empty dict is a valid, non-failure response.

Return type
: dict

get_project_id(*session: ks_session.Session*) → str | None

Return the project id that we are authenticated to.

Wherever possible the project id should be inferred from the token however there are certain URLs and other places that require access to the currently authenticated project id.

Parameters
: **session** (*keystoneauth1.session.Session*) A session object so the plugin can make HTTP calls.

Returns
: A project identifier or None if one is not available.

Return type
: str

get_sp_auth_url(*session: ks_session.Session*, *sp_id: str*) → str | None

Return auth_url from the Service Provider object.

This url is used for obtaining unscoped federated token from remote cloud.

Parameters
: **sp_id** (*string*) ID of the Service Provider to be queried.

Returns
: A Service Provider auth_url or None if one is not available.

Return type
: str

get_sp_url(*session: ks_session.Session*, *sp_id: str*) → str | None

Return sp_url from the Service Provider object.

This url is used for passing SAML2 assertion to the remote cloud.

Parameters
: **sp_id** (*str*) ID of the Service Provider to be queried.

Returns
: A Service Provider sp_url or None if one is not available.

Return type
str

`get_token`(*session: ks_session.Session*) → str | None

Obtain a token.

How the token is obtained is up to the plugin. If it is still valid it may be re-used, retrieved from cache or invoke an authentication request against a server.

Returning None will indicate that no token was able to be retrieved.

This function is misplaced as it should only be required for auth plugins that use the X-Auth-Token header. However due to the way plugins evolved this method is required and often called to trigger an authentication request on a new plugin.

When implementing a new plugin it is advised that you implement this method, however if you dont require the X-Auth-Token header override the *get_headers* method instead.

Parameters
session (*keystoneauth1.session.Session*) A session object so the plugin can make HTTP calls.

Returns
A token to use.

Return type
string

`get_user_id`(*session: ks_session.Session*) → str | None

Return a unique user identifier of the plugin.

Wherever possible the user id should be inferred from the token however there are certain URLs and other places that require access to the currently authenticated user id.

Parameters
session (*keystoneauth1.session.Session*) A session object so the plugin can make HTTP calls.

Returns
A user identifier or None if one is not available.

Return type
str

`invalidate`() → bool

Invalidate the current authentication data.

This should result in fetching a new token on next call.

A plugin may be invalidated if an Unauthorized HTTP response is returned to indicate that the token may have been revoked or is otherwise now invalid.

Returns
True if there was something that the plugin did to invalidate. This means that it makes sense to try again. If nothing happens returns False to indicate give up.

Return type
bool

set_auth_state(*data: str*) → None

Install existing authentication state for a plugin.

Take the output of get_auth_state and install that authentication state into the current authentication plugin.

Raises

`NotImplementedError` if the plugin does not support this feature.

class keystoneauth1.plugin.**ConnectionParams**

Bases: `TypedDict`

__annotations__ = {'cert': typing_extensions.NotRequired[typing.Union[str, typing.Tuple[str, str], NoneType]], 'verify': typing_extensions.NotRequired[typing.Union[bool, str, NoneType]]}

__dict__ = mappingproxy({'__module__': 'keystoneauth1.plugin', '__annotations__': {'cert': typing_extensions.NotRequired[typing.Union[str, typing.Tuple[str, str], NoneType]], 'verify': typing_extensions.NotRequired[typing.Union[bool, str, NoneType]]}, '__orig_bases__': (<function TypedDict>,), '__dict__': <attribute '__dict__' of 'ConnectionParams' objects>, '__weakref__': <attribute '__weakref__' of 'ConnectionParams' objects>, '__doc__': None, '__required_keys__': frozenset({'cert', 'verify'}), '__optional_keys__': frozenset(), '__total__': True})

__doc__ = None

__module__ = 'keystoneauth1.plugin'

__optional_keys__ = frozenset({})

__orig_bases__ = (<function TypedDict>,)

__required_keys__ = frozenset({'cert', 'verify'})

__total__ = True

__weakref__

list of weak references to the object (if defined)

cert: NotRequired[str | Tuple[str, str] | None]

verify: NotRequired[bool | str | None]

class keystoneauth1.plugin.**FixedEndpointPlugin**(*endpoint: str | None = None*)

Bases: `BaseAuthPlugin`

A base class for plugins that have one fixed endpoint.

__annotations__ = {}

__doc__ = 'A base class for plugins that have one fixed endpoint.'

__init__(*endpoint: str | None = None*) `__module__ = 'keystoneauth1.plugin'`

get_endpoint(*session: ks_session.Session*, *endpoint_override: str | None = None*, ***kwargs: Any*) → str | None

Return the supplied endpoint.

Using this plugin the same endpoint is returned regardless of the parameters passed to the plugin. endpoint_override overrides the endpoint specified when constructing the plugin.

get_endpoint_data(*session: ks_session.Session*, ***, *endpoint_override: str | None = None*, *discover_versions: bool = True*, ***kwargs: Any*) → EndpointData | None

Return a valid endpoint data for a the service.

Parameters

- **session** (*keystoneauth1.session.Session*) A session object that can be used for communication.
- **endpoint_override** (*str*) URL to use for version discovery.
- **discover_versions** (*bool*) Whether to get version metadata from the version discovery document even if it major api version info can be inferred from the url. (optional, defaults to True)
- **kwargs** Ignored.

Raises

`keystoneauth1.exceptions.http.HttpError` An error from an invalid HTTP response.

Returns

Valid EndpointData or None if not available.

Return type

keystoneauth1.discover.EndpointData or None

keystoneauth1.service_token module

class keystoneauth1.service_token.**ServiceTokenAuthWrapper**(*user_auth: BaseAuthPlugin*, *service_auth: BaseAuthPlugin*)

Bases: `BaseAuthPlugin`

`__annotations__ = {}`

`__doc__ = None`

__init__(*user_auth: BaseAuthPlugin*, *service_auth: BaseAuthPlugin*)

`__module__ = 'keystoneauth1.service_token'`

get_connection_params(*session: ks_session.Session*) → ConnectionParams

Return any additional connection parameters required for the plugin.

Parameters

session (*keystoneauth1.session.Session*) The session object that the auth_plugin belongs to.

Returns
Parameters that are passed to the requests library. Only the `cert` and `verify` parameters may be returned.

Return type
dict

get_endpoint(*session: ks_session.Session*, ***kwargs: Any*) → str | None

Return an endpoint for the client.

There are no required keyword arguments to `get_endpoint` as a plugin implementation should use best effort with the information available to determine the endpoint. However there are certain standard options that will be generated by the clients and should be used by plugins:

- `service_type`: what sort of service is required.
- `service_name`: the name of the service in the catalog.
- `interface`: what visibility the endpoint should have.
- `region_name`: the region the endpoint exists in.

Parameters

- **session** (*keystoneauth1.session.Session*) The session object that the auth_plugin belongs to.
- **kwargs** Ignored.

Returns
The base URL that will be used to talk to the required service or None if not available.

Return type
string

get_headers(*session: ks_session.Session*) → dict[str, str] | None

Fetch authentication headers for message.

This is a more generalized replacement of the older get_token to allow plugins to specify different or additional authentication headers to the OpenStack standard X-Auth-Token header.

How the authentication headers are obtained is up to the plugin. If the headers are still valid they may be re-used, retrieved from cache or the plugin may invoke an authentication request against a server.

The default implementation of get_headers calls the *get_token* method to enable older style plugins to continue functioning unchanged. Subclasses should feel free to completely override this function to provide the headers that they want.

Returning None will indicate that no token was able to be retrieved and that authorization was a failure. Adding no authentication data can be achieved by returning an empty dictionary.

Parameters
session (*keystoneauth1.session.Session*) The session object that the auth_plugin belongs to.

Returns
Headers that are set to authenticate a message or None for failure. Note that when checking this value that the empty dict is a valid, non-failure response.

Return type
dict

`get_project_id`(*session: ks_session.Session*) → str | None

Return the project id that we are authenticated to.

Wherever possible the project id should be inferred from the token however there are certain URLs and other places that require access to the currently authenticated project id.

Parameters
session (*keystoneauth1.session.Session*) A session object so the plugin can make HTTP calls.

Returns
A project identifier or None if one is not available.

Return type
str

`get_sp_auth_url`(*session: ks_session.Session*, *sp_id: str*) → str | None

Return auth_url from the Service Provider object.

This url is used for obtaining unscoped federated token from remote cloud.

Parameters
sp_id (*string*) ID of the Service Provider to be queried.

Returns
A Service Provider auth_url or None if one is not available.

Return type
str

`get_sp_url`(*session: ks_session.Session*, *sp_id: str*) → str | None

Return sp_url from the Service Provider object.

This url is used for passing SAML2 assertion to the remote cloud.

Parameters
sp_id (*str*) ID of the Service Provider to be queried.

Returns
A Service Provider sp_url or None if one is not available.

Return type
str

`get_token`(*session: ks_session.Session*) → str | None

Obtain a token.

How the token is obtained is up to the plugin. If it is still valid it may be re-used, retrieved from cache or invoke an authentication request against a server.

Returning None will indicate that no token was able to be retrieved.

This function is misplaced as it should only be required for auth plugins that use the X-Auth-Token header. However due to the way plugins evolved this method is required and often called to trigger an authentication request on a new plugin.

When implementing a new plugin it is advised that you implement this method, however if you dont require the X-Auth-Token header override the *get_headers* method instead.

Parameters
`session` (*keystoneauth1.session.Session*) A session object so the plugin can make HTTP calls.

Returns
A token to use.

Return type
string

get_user_id(*session: ks_session.Session*) → str | None

Return a unique user identifier of the plugin.

Wherever possible the user id should be inferred from the token however there are certain URLs and other places that require access to the currently authenticated user id.

Parameters
`session` (*keystoneauth1.session.Session*) A session object so the plugin can make HTTP calls.

Returns
A user identifier or None if one is not available.

Return type
str

invalidate() → bool

Invalidate the current authentication data.

This should result in fetching a new token on next call.

A plugin may be invalidated if an Unauthorized HTTP response is returned to indicate that the token may have been revoked or is otherwise now invalid.

Returns
True if there was something that the plugin did to invalidate. This means that it makes sense to try again. If nothing happens returns False to indicate give up.

Return type
bool

keystoneauth1.session module

class keystoneauth1.session.**NoOpSemaphore**

Bases: `object`

Empty context manager for use as a default semaphore.

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.session', '__doc__': 'Empty context manager for use as a default semaphore.', '__enter__': <function NoOpSemaphore.__enter__>, '__exit__': <function NoOpSemaphore.__exit__>, '__dict__': <attribute '__dict__' of 'NoOpSemaphore' objects>, '__weakref__': <attribute '__weakref__' of 'NoOpSemaphore' objects>, '__annotations__': {}})`

`__doc__ = 'Empty context manager for use as a default semaphore.'`

`__enter__`() → None

Enter the context manager and do nothing.

`__exit__`(*exc_type: Type[BaseException] | None*, *exc_value: BaseException | None*, *traceback: TracebackType | None*) → None

Exit the context manager and do nothing.

`__module__ = 'keystoneauth1.session'`

`__weakref__`

list of weak references to the object (if defined)

class `keystoneauth1.session.`**`RequestTiming`**(*method: str | None*, *url: str | None*, *elapsed: timedelta*)

Bases: `object`

Contains timing information for an HTTP interaction.

`__annotations__ = {'elapsed': <class 'datetime.timedelta'>, 'method': typing.Optional[str], 'url': typing.Optional[str]}`

`__dict__ = mappingproxy({'__module__': 'keystoneauth1.session', '__annotations__': {'method': typing.Optional[str], 'url': typing.Optional[str], 'elapsed': <class 'datetime.timedelta'>}, '__doc__': 'Contains timing information for an HTTP interaction.', '__init__': <function RequestTiming.__init__>, '__dict__': <attribute '__dict__' of 'RequestTiming' objects>, '__weakref__': <attribute '__weakref__' of 'RequestTiming' objects>})`

`__doc__ = 'Contains timing information for an HTTP interaction.'`

`__init__`(*method: str | None*, *url: str | None*, *elapsed: timedelta*)

`__module__ = 'keystoneauth1.session'`

`__weakref__`

list of weak references to the object (if defined)

`elapsed: timedelta`

Elapsed time information

`method: str | None`

HTTP method used for the call (GET, POST, etc)

`url: str | None`

URL against which the call was made

class keystoneauth1.session.Session(*auth: plugin.BaseAuthPlugin | None = None, session: Session | None = None, original_ip: str | None = None, verify: bool | str | None = True, cert: str | tuple[str, str] | None = None, timeout: int | None = None, user_agent: str | None = None, redirect: int | bool = 30, additional_headers: MutableMapping[str, str] | None = None, app_name: str | None = None, app_version: str | None = None, additional_user_agent: list[tuple[str, str]] | None = None, discovery_cache: dict[str, Any] | None = None, split_loggers: bool | None = None, collect_timing: bool = False, rate_semaphore: ContextManager[None] | None = None, connect_retries: int = 0*)

Bases: `object`

Maintains client communication state and common functionality.

As much as possible the parameters to this class reflect and are passed directly to the `requests` library.

Parameters

- **auth** (*keystoneauth1.plugin.BaseAuthPlugin*) An authentication plugin to authenticate the session with. (optional, defaults to None)
- **session** (*requests.Session*) A requests session object that can be used for issuing requests. (optional)
- **original_ip** (*str*) The original IP of the requesting user which will be sent to identity service in a Forwarded header. (optional)
- **verify** The verification arguments to pass to requests. These are of the same form as requests expects, so True or False to verify (or not) against system certificates or a path to a bundle or CA certs to check against or None for requests to attempt to locate and use certificates. (optional, defaults to True)
- **cert** A client certificate to pass to requests. These are of the same form as requests expects. Either a single filename containing both the certificate and key or a tuple containing the path to the certificate then a path to the key. (optional)
- **timeout** (*float*) A timeout to pass to requests. This should be a numerical value indicating some amount (or fraction) of seconds or 0 for no timeout. (optional, defaults to 0)
- **user_agent** (*str*) A User-Agent header string to use for the request. If not provided, a default of `DEFAULT_USER_AGENT` is used, which contains the keystoneauth1 version as well as those of the requests library and which Python is being used. When a non-None value is passed, it will be prepended to the default.
- **redirect** (*int/bool*) Controls the maximum number of redirections that can be followed by a request. Either an integer for a specific count or True/False for forever/never. (optional, default to 30)
- **additional_headers** (*dict*) Additional headers that should be attached to every request passing through the session. Headers of the same name specified

per request will take priority.

- **app_name** (*str*) The name of the application that is creating the session. This will be used to create the user_agent.
- **app_version** (*str*) The version of the application creating the session. This will be used to create the user_agent.
- **additional_user_agent** (*list*) A list of tuple of name, version that will be added to the user agent. This can be used by libraries that are part of the communication process.
- **discovery_cache** (*dict*) A dict to be used for caching of discovery information. This is normally managed transparently, but if the user wants to share a single cache across multiple sessions that do not share an auth plugin, it can be provided here. (optional, defaults to None which means automatically manage)
- **split_loggers** (*bool*) Split the logging of requests across multiple loggers instead of just one. Defaults to False.
- **collect_timing** (*bool*) Whether or not to collect per-method timing information for each API call. (optional, defaults to False)
- **rate_semaphore** Semaphore to be used to control concurrency and rate limiting of requests. (optional, defaults to no concurrency or rate control)
- **connect_retries** (*int*) the maximum number of retries that should be attempted for connection errors. (optional, defaults to 0 - never retry).

_DEFAULT_REDIRECT_LIMIT = 30

_REDIRECT_STATUSES = (301, 302, 303, 305, 307, 308)

__del__() → None

Clean up resources on delete.

```
__dict__ = mappingproxy({'__module__': 'keystoneauth1.session',
'__doc__': "Maintains client communication state and common
functionality.\n\n As much as possible the parameters to this class
reflect and are passed\n directly to the :mod:`requests` library.\n\n
:param auth: An authentication plugin to authenticate the session with.\n
(optional, defaults to None)\n :type auth:
keystoneauth1.plugin.BaseAuthPlugin\n :param requests.Session session: A
requests session object that can be used\n for issuing requests.
(optional)\n :param str original_ip: The original IP of the requesting
user which will\n be sent to identity service in a 'Forwarded'\n header.
(optional)\n :param verify: The verification arguments to pass to
requests. These are of\n the same form as requests expects, so True or
False to\n verify (or not) against system certificates or a path to a\n
bundle or CA certs to check against or None for requests to\n attempt to
locate and use certificates. (optional, defaults\n to True)\n :param cert:
A client certificate to pass to requests. These are of the\n same form as
requests expects. Either a single filename\n containing both the
certificate and key or a tuple containing\n the path to the certificate
then a path to the key. (optional)\n :param float timeout: A timeout to
pass to requests. This should be a\n numerical value indicating some
amount (or fraction)\n of seconds or 0 for no timeout. (optional,
defaults\n to 0)\n :param str user_agent: A User-Agent header string to
use for the request.\n If not provided, a default of\n
:attr:`~keystoneauth1.session.DEFAULT_USER_AGENT` is\n used, which
contains the keystoneauth1 version as\n well as those of the requests
library and which\n Python is being used. When a non-None value is\n
passed, it will be prepended to the default.\n :param int/bool redirect:
Controls the maximum number of redirections that\n can be followed by a
request. Either an integer\n for a specific count or True/False for\n
forever/never. (optional, default to 30)\n :param dict additional_headers:
Additional headers that should be attached\n to every request passing
through the\n session. Headers of the same name specified\n per request
will take priority.\n :param str app_name: The name of the application
that is creating the\n session. This will be used to create the
user_agent.\n :param str app_version: The version of the application
creating the\n session. This will be used to create the\n user_agent.\n
:param list additional_user_agent: A list of tuple of name, version
that\n will be added to the user agent. This\n can be used by libraries
that are part\n of the communication process.\n :param dict
discovery_cache: A dict to be used for caching of discovery\n
information. This is normally managed\n transparently, but if the user
wants to\n share a single cache across multiple sessions\n that do not
share an auth plugin, it can\n be provided here. (optional, defaults to\n
None which means automatically manage)\n :param bool split_loggers: Split
the logging of requests across multiple\n loggers instead of just one.
Defaults to False.\n :param bool collect_timing: Whether or not to
collect per-method timing\n information for each API call. (optional,\n
defaults to False)\n :param rate_semaphore: Semaphore to be used to
control concurrency\n and rate limiting of requests. (optional,\n defaults
to no concurrency or rate control)\n :param int connect_retries: the
maximum number of retries that should\n be attempted for connection
errors.\n (optional, defaults to 0 - never retry).\n ", 'user_agent':
None, '_REDIRECT_STATUSES': (301, 302, 303, 305, 307, 308),
'_DEFAULT_REDIRECT_LIMIT': 30, '__init__': <function Session.__init__>,
'__del__': <function Session.__del__>, 'adapters': <property object>,
'mount': <function Session.mount>, '_remove_service_catalog': <function
Session._remove_service_catalog>, '_process_header':
```

```
__doc__ = "Maintains client communication state and common
functionality.\n\n As much as possible the parameters to this class
reflect and are passed\n directly to the :mod:`requests` library.\n\n
:param auth:  An authentication plugin to authenticate the session with.\n
(optional, defaults to None)\n :type auth:
keystoneauth1.plugin.BaseAuthPlugin\n :param requests.Session session:  A
requests session object that can be used\n for issuing requests.
(optional)\n :param str original_ip:  The original IP of the requesting
user which will\n be sent to identity service in a 'Forwarded'\n header.
(optional)\n :param verify:  The verification arguments to pass to
requests. These are of\n the same form as requests expects, so True or
False to\n verify (or not) against system certificates or a path to a\n
bundle or CA certs to check against or None for requests to\n attempt to
locate and use certificates. (optional, defaults\n to True)\n :param cert:
A client certificate to pass to requests. These are of the\n same form as
requests expects. Either a single filename\n containing both the
certificate and key or a tuple containing\n the path to the certificate
then a path to the key. (optional)\n :param float timeout:  A timeout to
pass to requests. This should be a\n numerical value indicating some
amount (or fraction)\n of seconds or 0 for no timeout. (optional,
defaults\n to 0)\n :param str user_agent:  A User-Agent header string to
use for the request.\n If not provided, a default of\n
:attr:`~keystoneauth1.session.DEFAULT_USER_AGENT` is\n used, which
contains the keystoneauth1 version as\n well as those of the requests
library and which\n Python is being used. When a non-None value is\n
passed, it will be prepended to the default.\n :param int/bool redirect:
Controls the maximum number of redirections that\n can be followed by a
request. Either an integer\n for a specific count or True/False for\n
forever/never. (optional, default to 30)\n :param dict additional_headers:
Additional headers that should be attached\n to every request passing
through the\n session. Headers of the same name specified\n per request
will take priority.\n :param str app_name:  The name of the application
that is creating the\n session. This will be used to create the
user_agent.\n :param str app_version:  The version of the application
creating the\n session. This will be used to create the\n user_agent.\n
:param list additional_user_agent:  A list of tuple of name, version
that\n will be added to the user agent. This\n can be used by libraries
that are part\n of the communication process.\n :param dict
discovery_cache:  A dict to be used for caching of discovery\n
information. This is normally managed\n transparently, but if the user
wants to\n share a single cache across multiple sessions\n that do not
share an auth plugin, it can\n be provided here. (optional, defaults to\n
None which means automatically manage)\n :param bool split_loggers:  Split
the logging of requests across multiple\n loggers instead of just one.
Defaults to False.\n :param bool collect_timing:  Whether or not to
collect per-method timing\n information for each API call. (optional,\n
defaults to False)\n :param rate_semaphore:  Semaphore to be used to
control concurrency\n and rate limiting of requests. (optional,\n defaults
to no concurrency or rate control)\n :param int connect_retries:  the
maximum number of retries that should\n be attempted for connection
errors.\n (optional, defaults to 0 - never retry).\n "
```

`__init__`(*auth: plugin.BaseAuthPlugin | None = None*, *session: Session | None = None*, *original_ip: str | None = None*, *verify: bool | str | None = True*, *cert: str | tuple[str, str] | None = None*, *timeout: int | None = None*, *user_agent: str | None = None*, *redirect: int | bool = 30*, *additional_headers: MutableMapping[str, str] | None = None*, *app_name: str | None = None*, *app_version: str | None = None*, *additional_user_agent: list[tuple[str, str]] | None = None*, *discovery_cache: dict[str, Any] | None = None*, *split_loggers: bool | None = None*, *collect_timing: bool = False*, *rate_semaphore: ContextManager[None] | None = None*, *connect_retries: int = 0*)

`__module__` = 'keystoneauth1.session'

`__weakref__`

list of weak references to the object (if defined)

`_auth_required`(*auth: plugin.BaseAuthPlugin | None*, *msg: str*) → plugin.BaseAuthPlugin

`_get_split_loggers`(*split_loggers: bool | None*) → bool | None

Get a boolean value from the various argument sources.

We default split_loggers to None in the kwargs of the Session constructor so we can track set vs. not set. We also accept split_loggers as a parameter in a few other places. In each place we want the parameter, if given by the user, to win.

`_http_log_request`(*url: str*, *method: str | None = None*, *data: str | bytes | None = None*, *json: object | None = None*, *headers: MutableMapping[str, str] | None = None*, *query_params: dict[str, Any] | None = None*, *logger: Logger | None = None*, *split_loggers: bool | None = None*) → None

`_http_log_response`(*response: Response | None = None*, *json: object | None = None*, *status_code: int | None = None*, *headers: MutableMapping[str, str] | None = None*, *text: str | None = None*, ***, *logger: Logger*, *split_loggers: bool | None = None*) → None

static `_process_header`(*header: tuple[str, str]*) → tuple[str, str]

Redact the secure headers to be logged.

`_remove_service_catalog`(*body: str*) → str

`_send_request`(*url: str*, *method: str*, *redirect: int | bool*, *log: bool*, *logger: Logger*, *split_loggers: bool | None*, *connect_retries: int*, *status_code_retries: int*, *retriable_status_codes: list[int]*, *rate_semaphore: ContextManager[None]*, *connect_retry_delays: _Retries*, *status_code_retry_delays: _Retries*, ***kwargs: Any*) → Response

static `_set_microversion_headers`(*headers: MutableMapping[str, str]*, *microversion: str*, *service_type: str | None*, *endpoint_filter: dict[str, Any] | None*) → None

property `adapters`: `MutableMapping[Any, Any]`

`delete`(*url: str*, ***kwargs: Any*) → Response

Perform a DELETE request.

This calls `request()` with `method` set to `DELETE`.

`get`(*url: str*, ***kwargs: Any*) → Response

Perform a GET request.

This calls `request()` with `method` set to `GET`.

`get_all_version_data`(*auth: plugin.BaseAuthPlugin | None = None*, *interface: str = 'public'*, *region_name: str | None = None*, *service_type: str | None = None*) → dict[str, dict[str, dict[str, list[VersionData]]]]

Get version data for all services in the catalog.

Parameters

- **auth** (*keystoneauth1.plugin.BaseAuthPlugin*) The auth plugin to use for token. Overrides the plugin on the session. (optional)
- **interface** Type of endpoint to get version data for. Can be a single value or a list of values. A value of None indicates that all interfaces should be queried. (optional, defaults to public)
- **region_name** (*string*) Region of endpoints to get version data for. A valueof None indicates that all regions should be queried. (optional, defaults to None)
- **service_type** (*string*) Limit the version data to a single service. (optional, defaults to None)

Returns

A dictionary keyed by region_name with values containing dictionaries keyed by interface with values being a list of *~keystoneauth1.discover.VersionData*.

`get_api_major_version`(*auth: plugin.BaseAuthPlugin | None = None*, ***kwargs: Any*) → tuple[int | float, ...] | None

Get the major API version as provided by the auth plugin.

Parameters

auth (*keystoneauth1.plugin.BaseAuthPlugin*) The auth plugin to use for token. Overrides the plugin on the session. (optional)

Raises

`keystoneauth1.exceptions.auth_plugins.MissingAuthPlugin` if a plugin is not available.

Returns

The major version of the API of the service discovered.

Return type

tuple or None

`get_auth_connection_params`(*auth: plugin.BaseAuthPlugin | None = None*) → plugin.ConnectionParams

Return auth connection params as provided by the auth plugin.

An auth plugin may specify connection parameters to the request like providing a client certificate for communication.

We restrict the values that may be returned from this function to prevent an auth plugin overriding values unrelated to connection parameters. The values that are currently accepted are:

- *cert*: a path to a client certificate, or tuple of client certificate and key pair that are used with this request.
- *verify*: a boolean value to indicate verifying SSL certificates against the system CAs or a path to a CA file to verify with.

These values are passed to the requests library and further information on accepted values may be found there.

Parameters
`auth` (*keystoneauth1.plugin.BaseAuthPlugin*) The auth plugin to use for tokens. Overrides the plugin on the session. (optional)

Raises

- `keystoneauth1.exceptions.auth.AuthorizationFailure` if a new token fetch fails.
- `keystoneauth1.exceptions.auth_plugins.MissingAuthPlugin` if a plugin is not available.
- `keystoneauth1.exceptions.auth_plugins.UnsupportedParameters` if the plugin returns a parameter that is not supported by this session.

Returns
Authentication headers or None for failure.

Return type
dict

`get_auth_headers`(*auth: plugin.BaseAuthPlugin* | *None* = *None*) → dict[str, str] | None

Return auth headers as provided by the auth plugin.

Parameters
`auth` (*keystoneauth1.plugin.BaseAuthPlugin*) The auth plugin to use for token. Overrides the plugin on the session. (optional)

Raises

- `keystoneauth1.exceptions.auth.AuthorizationFailure` if a new token fetch fails.
- `keystoneauth1.exceptions.auth_plugins.MissingAuthPlugin` if a plugin is not available.

Returns
Authentication headers or None for failure.

Return type
dict

`get_endpoint`(*auth: plugin.BaseAuthPlugin* | *None* = *None*, ***, *endpoint_override: str* | *None* = *None*, ***kwargs: Any*) → str | None

Get an endpoint as provided by the auth plugin.

Parameters
`auth` (*keystoneauth1.plugin.BaseAuthPlugin*) The auth plugin to use for token. Overrides the plugin on the session. (optional)

Raises
`keystoneauth1.exceptions.auth_plugins.MissingAuthPlugin` if a plugin is not available.

Returns
An endpoint if available or None.

Return type
string

get_endpoint_data(*auth: plugin.BaseAuthPlugin* | *None* = *None*, ***kwargs: Any*) → EndpointData | None

Get endpoint data as provided by the auth plugin.

Parameters
auth (*keystoneauth1.plugin.BaseAuthPlugin*) The auth plugin to use for token. Overrides the plugin on the session. (optional)

Raises

- `keystoneauth1.exceptions.auth_plugins.MissingAuthPlugin` if a plugin is not available.
- `TypeError` If arguments are invalid

Returns
Endpoint data if available or None.

Return type
keystoneauth1.discover.EndpointData

get_project_id(*auth: plugin.BaseAuthPlugin* | *None* = *None*) → str | None

Return the authenticated project_id as provided by the auth plugin.

Parameters
auth (*keystoneauth1.plugin.BaseAuthPlugin*) The auth plugin to use for token. Overrides the plugin on the session. (optional)

Raises

- `keystoneauth1.exceptions.auth.AuthorizationFailure` if a new token fetch fails.
- `keystoneauth1.exceptions.auth_plugins.MissingAuthPlugin` if a plugin is not available.

Returns
Current project_id or None if not supported by plugin.

Return type
`str`

get_timings() → list[RequestTiming]

Return collected API timing information.

Returns
List of *RequestTiming* objects.

get_token(*auth: plugin.BaseAuthPlugin* | *None* = *None*) → str | None

Return a token as provided by the auth plugin.

Parameters

auth (*keystoneauth1.plugin.BaseAuthPlugin*) The auth plugin to use for token. Overrides the plugin on the session. (optional)

Raises

- **keystoneauth1.exceptions.auth.AuthorizationFailure** if a new token fetch fails.
- **keystoneauth1.exceptions.auth_plugins.MissingAuthPlugin** if a plugin is not available.

Warning

DEPRECATED: This assumes that the only header that is used to authenticate a message is `X-Auth-Token`. This may not be correct. Use `get_auth_headers()` instead.

Returns

A valid token.

Return type

string

get_user_id(*auth: plugin.BaseAuthPlugin* | *None* = *None*) → str | None

Return the authenticated user_id as provided by the auth plugin.

Parameters

auth (*keystoneauth1.plugin.BaseAuthPlugin*) The auth plugin to use for token. Overrides the plugin on the session. (optional)

Raises

- **keystoneauth1.exceptions.auth.AuthorizationFailure** if a new token fetch fails.
- **keystoneauth1.exceptions.auth_plugins.MissingAuthPlugin** if a plugin is not available.

Returns

Current user_id or None if not supported by plugin.

Return type

`str`

head(*url: str*, ***kwargs: Any*) → Response

Perform a HEAD request.

This calls `request()` with `method` set to `HEAD`.

invalidate(*auth: plugin.BaseAuthPlugin* | *None* = *None*) → bool

Invalidate an authentication plugin.

Parameters

auth (*keystoneauth1.plugin.BaseAuthPlugin*) The auth plugin to invalidate. Overrides the plugin on the session. (optional)

mount(*scheme: str*, *adapter: BaseAdapter*) → None

patch(*url: str*, ***kwargs: Any*) → Response

Perform a PATCH request.

This calls `request()` with `method` set to `PATCH`.

post(*url: str*, ***kwargs: Any*) → Response

Perform a POST request.

This calls `request()` with `method` set to `POST`.

put(*url: str*, ***kwargs: Any*) → Response

Perform a PUT request.

This calls `request()` with `method` set to `PUT`.

request(*url: str*, *method: str*, *json: object | None = None*, *original_ip: str | None = None*, *user_agent: str | None = None*, *redirect: int | bool | None = None*, *authenticated: bool | None = None*, *endpoint_filter: dict[str, Any] | None = None*, *auth: plugin.BaseAuthPlugin | None = None*, *requests_auth: requests.auth.AuthBase | None = None*, *raise_exc: bool = True*, *allow_reauth: bool = True*, *log: bool = True*, *endpoint_override: str | None = None*, *connect_retries: int | None = None*, *logger: Logger | None = None*, *allow: dict[str, Any] | None = None*, *client_name: str | None = None*, *client_version: str | None = None*, *microversion: str | None = None*, *microversion_service_type: str | None = None*, *status_code_retries: int = 0*, *retriable_status_codes: list[int] | None = None*, *rate_semaphore: ContextManager[None] | None = None*, *global_request_id: str | None = None*, *connect_retry_delay: float | None = None*, *status_code_retry_delay: float | None = None*, ***kwargs: Any*) → Response

Send an HTTP request with the specified characteristics.

Wrapper around *requests.Session.request* to handle tasks such as setting headers, JSON encoding/decoding, and error handling.

Arguments that are not handled are passed through to the requests library.

Parameters

- **url** (*str*) Path or fully qualified URL of HTTP request. If only a path is provided then endpoint_filter must also be provided such that the base URL can be determined. If a fully qualified URL is provided then endpoint_filter will be ignored.
- **method** (*str*) The http method to use. (e.g. GET, POST)
- **json** Some data to be represented as JSON. (optional)
- **original_ip** (*str*) Mark this request as forwarded for this ip. (optional)
- **headers** (*dict*) Headers to be included in the request. (optional)
- **user_agent** (*str*) A user_agent to use for the request. If present will override one present in headers. (optional)
- **redirect** (*int/bool*) the maximum number of redirections that can be followed by a request. Either an integer for a specific count or True/False for forever/never. (optional)

- **connect_retries** (*int*) the maximum number of retries that should be attempted for connection errors. (optional, defaults to None - never retry).
- **authenticated** (*bool*) True if a token should be attached to this request, False if not or None for attach if an auth_plugin is available. (optional, defaults to None)
- **endpoint_filter** (*dict*) Data to be provided to an auth plugin with which it should be able to determine an endpoint to use for this request. If not provided then URL is expected to be a fully qualified URL. (optional)
- **endpoint_override** (*str*) The URL to use instead of looking up the endpoint in the auth plugin. This will be ignored if a fully qualified URL is provided but take priority over an endpoint_filter. This string may contain the values `%(project_id)s` and `%(user_id)s` to have those values replaced by the project_id/user_id of the current authentication. (optional)
- **auth** (*keystoneauth1.plugin.BaseAuthPlugin*) The auth plugin to use when authenticating this request. This will override the plugin that is attached to the session (if any). (optional)
- **requests_auth** (`requests.auth.AuthBase`) A requests library auth plugin that cannot be passed via kwarg because the *auth* kwarg collides with our own auth plugins. (optional)
- **raise_exc** (*bool*) If True then raise an appropriate exception for failed HTTP requests. If False then return the request object. (optional, default True)
- **allow_reauth** (*bool*) Allow fetching a new token and retrying the request on receiving a 401 Unauthorized response. (optional, default True)
- **log** (*bool*) If True then log the request and response data to the debug log. (optional, default True)
- **logger** (*logging.Logger*) The logger object to use to log request and responses. If not provided the keystoneauth1.session default logger will be used.
- **allow** (*dict*) Extra filters to pass when discovering API versions. (optional)
- **microversion** Microversion to send for this request. microversion can be given as a string or a tuple. (optional)
- **microversion_service_type** (*str*) The service_type to be sent in the microversion header, if a microversion is given. Defaults to the value of service_type from endpoint_filter if one exists. If endpoint_filter is not provided or does not have a service_type, microversion is given and microversion_service_type is not provided, an exception will be raised.
- **status_code_retries** (*int*) the maximum number of retries that should be attempted for retriable HTTP status codes (optional, defaults to 0 - never retry).
- **retriable_status_codes** (*list*) list of HTTP status codes that should be retried (optional, defaults to HTTP 503, has no effect when status_code_retries is 0).

- **rate_semaphore** Semaphore to be used to control concurrency and rate limiting of requests. (optional, defaults to no concurrency or rate control)
- **global_request_id** Value for the X-Openstack-Request-Id header.
- **connect_retry_delay** (*float*) Delay (in seconds) between two connect retries (if enabled). By default exponential retry starting with 0.5 seconds up to a maximum of 60 seconds is used.
- **status_code_retry_delay** (*float*) Delay (in seconds) between two status code retries (if enabled). By default exponential retry starting with 0.5 seconds up to a maximum of 60 seconds is used.
- **kwargs** any other parameter that can be passed to `requests.Session.request()` (such as *headers*). Except:
  - *data* will be overwritten by the data in the *json* param.
  - *allow_redirects* is ignored as redirects are handled by the session.

Raises

`keystoneauth1.exceptions.base.ClientException` For connection failure, or to indicate an error response code.

Returns

The response to the request.

reset_timings() → None

Clear API timing information.

user_agent = None

class keystoneauth1.session.**TCPKeepAliveAdapter**(*pool_connections=10*, *pool_maxsize=10*, *max_retries=0*, *pool_block=False*)

Bases: `HTTPAdapter`

The custom adapter used to set TCP Keep-Alive on all connections.

This Adapter also preserves the default behaviour of Requests which disables Nagles Algorithm. See also: https://blogs.msdn.com/b/windowsazurestorage/archive/2010/06/25/nagle-s-algorithm-is-not-friendly-towards-small-requests.aspx

__doc__ = "The custom adapter used to set TCP Keep-Alive on all connections.\n\n This Adapter also preserves the default behaviour of Requests which\n disables Nagle's Algorithm. See also:\n https://blogs.msdn.com/b/windowsazurestorage/archive/2010/06/25/nagle-s-algorithm-is-not-friendly-towards-small-requests.aspx\n "

__module__ = 'keystoneauth1.session'

init_poolmanager(**args: Any*, ***kwargs: Any*) → None

Initializes a urllib3 PoolManager.

This method should not be called from user code, and is only exposed for use when subclassing the `HTTPAdapter`.

Parameters

- **connections** The number of urllib3 connection pools to cache.
- **maxsize** The maximum number of connections to save in the pool.
- **block** Block when no free connections are available.
- **pool_kwargs** Extra keyword arguments used to initialize the Pool Manager.

class keystoneauth1.session.**_JSONEncoder**(**, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, sort_keys=False, indent=None, separators=None, default=None*)

Bases: `JSONEncoder`

__doc__ = None

__module__ = 'keystoneauth1.session'

default(*o: object*) → Any

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

class keystoneauth1.session.**_Retries**(*fixed_delay: float | None = None*)

Bases: `object`

__annotations__ = {'_current': <class 'float'>, '_fixed_delay': typing.Optional[float]}

__doc__ = None

__init__(*fixed_delay: float | None = None*)

__module__ = 'keystoneauth1.session'

__next__() → float

__slots__ = ('_fixed_delay', '_current')

_current: float

_fixed_delay: float | None

reset() → None

class keystoneauth1.session.**_StringFormatter**(*session: Session*, *auth: plugin.BaseAuthPlugin* | *None*)

Bases: object

A String formatter that fetches values on demand.

__dict__ = mappingproxy({'__module__': 'keystoneauth1.session', '__doc__': 'A String formatter that fetches values on demand.', '__init__': <function _StringFormatter.__init__>, '__getitem__': <function _StringFormatter.__getitem__>, '__dict__': <attribute '__dict__' of '_StringFormatter' objects>, '__weakref__': <attribute '__weakref__' of '_StringFormatter' objects>, '__annotations__': {}})

__doc__ = 'A String formatter that fetches values on demand.'

__getitem__(*item: str*) → Any

__init__(*session: Session*, *auth: plugin.BaseAuthPlugin* | *None*)

__module__ = 'keystoneauth1.session'

__weakref__

list of weak references to the object (if defined)

keystoneauth1.session.**_construct_session**(*session_obj: Session* | *None* = *None*) → Session

keystoneauth1.session.**_determine_calling_package**() → str

Walk the call frames trying to identify what is using this module.

keystoneauth1.session.**_determine_user_agent**() → str

Attempt to programmatically generate a user agent string.

First, look at the name of the process. Return this unless it is in the *ignored* list. Otherwise, look at the function call stack and try to find the name of the code that invoked this module.

keystoneauth1.session.**_mv_legacy_headers_for_service**(*mv_service_type: str*) → list[str]

Workaround for services that predate standardization.

TODO(sdague): eventually convert this to using os-service-types and put the logic there. However, right now this is so little logic, inlining it for release is a better call.

keystoneauth1.session.**_sanitize_headers**(*headers: dict[str* | *bytes, Any]*) → dict[str, Any]

Ensure headers are strings and not bytes.

keystoneauth1.token_endpoint module

class keystoneauth1.token_endpoint.**Token**(*endpoint: str* | *None*, *token: str* | *None*)

Bases: BaseAuthPlugin

A provider that will always use the given token and endpoint.

This is really only useful for testing and in certain CLI cases where you have a known endpoint and admin token that you want to use.

__annotations__ = {}

`__doc__ = 'A provider that will always use the given token and endpoint.\n\n This is really only useful for testing and in certain CLI cases where you\n have a known endpoint and admin token that you want to use.\n '`

`__init__`(*endpoint: str | None*, *token: str | None*)

`__module__ = 'keystoneauth1.token_endpoint'`

`get_auth_ref`(*session: ks_session.Session*) → access.AccessInfo | None

Return the authentication reference of an auth plugin.

Parameters

`session` (*`keystoneauth1.session.session`*) A session object to be used for communication

`get_endpoint`(*session: ks_session.Session*, ***kwargs: Any*) → str | None

Return the supplied endpoint.

Using this plugin the same endpoint is returned regardless of the parameters passed to the plugin.

`get_endpoint_data`(*session: ks_session.Session*, ***, *endpoint_override: str | None = None*, *discover_versions: bool = True*, ***kwargs: Any*) → EndpointData | None

Return a valid endpoint data for a the service.

Parameters

- `session` (*`keystoneauth1.session.Session`*) A session object that can be used for communication.
- `endpoint_override` (*`str`*) URL to use for version discovery other than the endpoint stored in the plugin. (optional, defaults to None)
- `discover_versions` (*`bool`*) Whether to get version metadata from the version discovery document even if it major api version info can be inferred from the url. (optional, defaults to True)
- `kwargs` Ignored.

Raises

`keystoneauth1.exceptions.http.HttpError` An error from an invalid HTTP response.

Returns

Valid EndpointData or None if not available.

Return type

keystoneauth1.discover.EndpointData or None

`get_token`(*session: ks_session.Session*) → str | None

Obtain a token.

How the token is obtained is up to the plugin. If it is still valid it may be re-used, retrieved from cache or invoke an authentication request against a server.

Returning None will indicate that no token was able to be retrieved.

This function is misplaced as it should only be required for auth plugins that use the X-Auth-Token header. However due to the way plugins evolved this method is required and often called to trigger an authentication request on a new plugin.

When implementing a new plugin it is advised that you implement this method, however if you dont require the X-Auth-Token header override the *get_headers* method instead.

Parameters
`session` (*keystoneauth1.session.Session*) A session object so the plugin can make HTTP calls.

Returns
A token to use.

Return type
string

6.1.3 Module contents