SwiftClient Documentation

Release 4.1.1.dev2

OpenStack, LLC.

Sep 13, 2022

CONTENTS

CHAPTER

ONE

INTRODUCTION

1.1 Introduction

1.1.1 Where to Start?

The `python-swiftclient` project comprises a command line tool and two separate APIs for accessing swift programmatically. Choosing the most appropriate method for a given use case is the first problem a user needs to solve.

Use Cases

Alongside the command line tool, the `python-swiftclient` includes two levels of API:

- A low level client API that provides simple Python wrappers around the various authentication mechanisms and the individual HTTP requests.
- A high level service API that provides methods for performing common operations in parallel on a thread pool.

Example use cases:

- **Uploading and retrieving data** Use the command line tool if you are simply uploading and downloading files and directories to and from your filesystem. The command line tool can be integrated into a shell script to automate tasks.
- **Integrating into an automated Python workflow** Use the `SwiftService` API to perform operations offered by the CLI if your use case requires integration with a Python-based workflow. This method offers greater control and flexibility over individual object operations, such as the metadata set on each object. The `SwiftService` class provides methods to perform multiple sets of operations against a swift object store using a configurable shared thread pool. A single instance of the `SwiftService` class can be shared between multiple threads in your own code.
- **Developing an application in Python to access a swift object store** Use the `SwiftService` API to develop Python applications that use swift to store and retrieve objects. A `SwiftService` instance provides a configurable thread pool for performing all operations supported by the CLI.
- **Fine-grained control over threading or the requests being performed** Use the `Connection` API if your use case requires fine grained control over advanced features or you wish to use your own existing threading model. Examples of advanced features requiring the use of the

`Connection` API include creating an SLO manifest that references already existing objects, or fine grained control over the query strings supplied with each HTTP request.

1.1.2 Important considerations

This section covers some important considerations, helpful hints, and things to avoid when integrating an object store into your workflow.

An object store is not a filesystem

It cannot be stressed enough that your usage of the object store should reflect the proper use case, and not treat the storage like a traditional filesystem. There are two main restrictions to bear in mind when designing an application that uses an object store:

- You cannot rename objects. Due to fact that the name of an object is one of the factors that determines where the object and its replicas are stored, renaming would require multiple copies of the data to be moved between physical storage devices. If you want to rename an object you must upload to the new location, or make a server side copy request to the new location, and then delete the original.
- You cannot modify objects. Objects are stored in multiple locations and are checked for integrity based on the MD5 sum calculated during upload. In order to modify the contents of an object, the entire desired contents must be re-uploaded. In certain special cases it is possible to work around this restriction using large objects, but no general file-like access is available to modify a stored object.

Objects cannot be locked

There is no mechanism to perform a combination of reading the data/metadata from an object and writing an update to that data/metadata in an atomic way. Any user with access to a container could update the contents or metadata associated with an object at any time.

Workflows that assume that no updates have been made since the last read of an object should be discouraged. Enabling a workflow of this type requires an external object locking mechanism and/or cooperation between all clients accessing the data.

CHAPTER

TWO

DEVELOPER DOCUMENTATION

2.1 So You Want to Contribute

For general information on contributing to OpenStack, please check out the contributor guide to get started. It covers all the basics that are common to all OpenStack projects: the accounts you need, the basics of interacting with our Gerrit review system, how we communicate as a community, etc.

The python-swiftclient is maintained by the OpenStack Swift project. To understand our development process and how you can contribute to it, please look at the Swift projects general contributors page: http://docs.openstack.org/swift/latest/contributor/contributing.html

2.2 CLI

The `swift` tool is a command line utility for communicating with an OpenStack Object Storage (swift) environment. It allows one to perform several types of operations.

For help on a specific **`swift`** command, enter:

```
$ swift COMMAND --help
```

2.2.1 swift usage

```
Usage: swift [--version] [--help] [--os-help] [--snet] [--verbose]
             [--debug] [--info] [--quiet] [--auth <auth_url>]
             [--auth-version <auth_version> |
                 --os-identity-api-version <auth_version> ]
             [--user <username>]
             [--key <api_key>] [--retries <num_retries>]
             [--os-username <auth-user-name>] [--os-password <auth-password>]
             [--os-user-id <auth-user-id>]
             [--os-user-domain-id <auth-user-domain-id>]
             [--os-user-domain-name <auth-user-domain-name>]
             [--os-tenant-id <auth-tenant-id>]
             [--os-tenant-name <auth-tenant-name>]
             [--os-project-id <auth-project-id>]
             [--os-project-name <auth-project-name>]
             [--os-project-domain-id <auth-project-domain-id>]
```

(continues on next page)

(continued from previous page)

```
            [--os-project-domain-name <auth-project-domain-name>]
            [--os-auth-url <auth-url>] [--os-auth-token <auth-token>]
            [--os-storage-url <storage-url>] [--os-region-name <region-name>]
            [--os-service-type <service-type>]
            [--os-endpoint-type <endpoint-type>]
            [--os-cacert <ca-certificate>] [--insecure]
            [--os-cert <client-certificate-file>]
            [--os-key <client-certificate-key-file>]
            [--no-ssl-compression]
            <subcommand> [--help] [<subcommand options>]
```

Subcommands:

delete Delete a container or objects within a container.

download Download objects from containers.

list Lists the containers for the account or the objects for a container.

post Updates meta information for the account, container, or object; creates containers if not present.

copy Copies object, optionally adds meta

stat Displays information for the account, container, or object.

upload Uploads files or directories to the given container.

capabilities List cluster capabilities.

tempurl Create a temporary URL.

auth Display auth related environment variables.

2.2.2 swift optional arguments

--version show programs version number and exit

-h, --help show this help message and exit

--os-help Show OpenStack authentication options.

-s, --snet Use SERVICENET internal network.

-v, --verbose Print more info.

--debug Show the curl commands and results of all http queries regardless of result status.

--info Show the curl commands and results of all http queries which return an error.

-q, --quiet Suppress status output.

-A AUTH, --auth=AUTH URL for obtaining an auth token.

-V AUTH_VERSION, --auth-version=AUTH_VERSION, --os-identity-api-version=AUTH_VERSION
Specify a version for authentication. Defaults to `env[ST_AUTH_VERSION]`, `env[OS_AUTH_VERSION]`, `env[OS_IDENTITY_API_VERSION]` or 1.0.

-U USER, --user=USER User name for obtaining an auth token.

-K KEY, --key=KEY Key for obtaining an auth token.

-R RETRIES, --retries=RETRIES The number of times to retry a failed connection.

--insecure Allow swiftclient to access servers without having to verify the SSL certificate. Defaults to env[SWIFTCLIENT_INSECURE] (set to true to enable).

--no-ssl-compression This option is deprecated and not used anymore. SSL compression should be disabled by default by the system SSL library.

--prompt Prompt user to enter a password which overrides any password supplied via --key, --os-password or environment variables.

2.2.3 Authentication

This section covers the options for authenticating with a swift object store. The combinations of options required for each authentication version are detailed below, but are just a subset of those that can be used to successfully authenticate. These are the most common and recommended combinations.

You should obtain the details of your authentication version and credentials from your storage provider. These details should make it clearer which of the authentication sections below are most likely to allow you to connect to your storage account.

Keystone v3

```
swift --os-auth-url https://api.example.com:5000/v3 --auth-version 3 \
      --os-project-name project1 --os-project-domain-name domain1 \
      --os-username user --os-user-domain-name domain1 \
      --os-password password list

swift --os-auth-url https://api.example.com:5000/v3 --auth-version 3 \
      --os-project-id 0123456789abcdef0123456789abcdef \
      --os-user-id abcdef0123456789abcdef0123456789 \
      --os-password password list
```

Manually specifying the options above on the command line can be avoided by setting the following combinations of environment variables:

```
ST_AUTH_VERSION=3
OS_USERNAME=user
OS_USER_DOMAIN_NAME=domain1
OS_PASSWORD=password
OS_PROJECT_NAME=project1
OS_PROJECT_DOMAIN_NAME=domain1
OS_AUTH_URL=https://api.example.com:5000/v3

ST_AUTH_VERSION=3
OS_USER_ID=abcdef0123456789abcdef0123456789
OS_PASSWORD=password
OS_PROJECT_ID=0123456789abcdef0123456789abcdef
OS_AUTH_URL=https://api.example.com:5000/v3
```

Keystone v2

```
swift --os-auth-url https://api.example.com:5000/v2.0 \
      --os-tenant-name tenant \
      --os-username user --os-password password list
```

Manually specifying the options above on the command line can be avoided by setting the following environment variables:

```
ST_AUTH_VERSION=2.0
OS_USERNAME=user
OS_PASSWORD=password
OS_TENANT_NAME=tenant
OS_AUTH_URL=https://api.example.com:5000/v2.0
```

Legacy auth systems

You can configure swift to work with any number of other authentication systems that we will not cover in this document. If your storage provider is not using Keystone to provide access tokens, please contact them for instructions on the required options. It is likely that the options will need to be specified as below:

```
swift -A https://api.example.com/v1.0 -U user -K api_key list
```

Specifying the options above manually on the command line can be avoided by setting the following environment variables:

```
ST_AUTH_VERSION=1.0
ST_AUTH=https://api.example.com/v1.0
ST_USER=user
ST_KEY=key
```

It is also possible that you need to use a completely separate auth system, in which case `swiftclient` cannot request a token for you. In this case you should make the authentication request separately and access your storage using the token and storage URL options shown below:

```
swift --os-auth-token 6ee5eb33efad4e45ab46806eac010566 \
      --os-storage-url https://10.1.5.2:8080/v1/AUTH_ced809b6a4baea7aeab61a \
      list
```

Note: Leftover environment variables are a common source of confusion when authorization fails.

2.2.4 CLI commands

Auth

```
Usage: swift auth
```

Display authentication variables in shell friendly format. Command to run to export storage URL and auth token into `OS_STORAGE_URL` and `OS_AUTH_TOKEN`: `swift auth`. Command to append to a runcom file (e.g. `~/.bashrc`, `/etc/profile`) for automatic authentication: `swift auth -v -U test:tester -K testing`.

swift stat

```
Usage: swift stat [--lh] [--header <header:value>]
                  [<container> [<object>]]
```

Displays information for the account, container, or object depending on the arguments given (if any). In verbose mode, the storage URL and the authentication token are displayed as well.

Positional arguments:

[container] Name of container to stat from.

[object] Name of object to stat.

Optional arguments:

--lh Report sizes in human readable format similar to ls -lh.

-H, --header <header:value> Adds a custom request header to use for stat.

swift list

```
Usage: swift list [--long] [--lh] [--totals] [--prefix <prefix>]
                  [--delimiter <delimiter>] [--header <header:value>]
                  [<container>]
```

Lists the containers for the account or the objects for a container. The `-p <prefix>` or `--prefix <prefix>` is an option that will only list items beginning with that prefix. The `-d <delimiter>` or `--delimiter <delimiter>` is an option (for container listings only) that will roll up items with the given delimiter (see *OpenStack Swift general documentation <https://docs.openstack.org/swift/latest/>* for what this means).

The `-l` and `--lh` options provide more detail, similar to `ls -l` and `ls -lh`, the latter providing sizes in human readable format (For example: `3K`, `12M`, etc). The latter two switches use more overhead to retrieve the displayed details, which is directly proportional to the number of container or objects listed.

Positional arguments:

[container] Name of container to list object in.

Optional arguments:

-l, --long Long listing format, similar to ls -l.

--lh Report sizes in human readable format similar to ls -lh.

-t, --totals Used with -l or lh, only report totals.

-p <prefix>, --prefix <prefix> Only list items beginning with the prefix.

-d <delim>, --delimiter <delim> Roll up items with the given delimiter. For containers only. See OpenStack Swift API documentation for what this means.

-H, --header <header:value> Adds a custom request header to use for listing.

swift upload

```
Usage: swift upload [--changed] [--skip-identical] [--segment-size <size>]
                    [--segment-container <container>] [--leave-segments]
                    [--object-threads <thread>] [--segment-threads <threads>]
                    [--header <header>] [--use-slo] [--ignore-checksum]
                    [--object-name <object-name>]
                    <container> <file_or_directory> [<file_or_directory>] [...
↪]
```

Uploads the files and directories specified by the remaining arguments to the given container. The `-c` or `--changed` is an option that will only upload files that have changed since the last upload. The `--object-name <object-name>` is an option that will upload a file and name object to `<object-name>` or upload a directory and use `<object-name>` as object prefix. If the file name is `-`, client reads content from standard input. In this case `--object-name` is required to set the name of the object and no other files may be given. The `-S <size>` or `--segment-size <size>` and `--leave-segments` are options as well (see `--help` for more).

Positional arguments:

<container> Name of container to upload to.

<file_or_directory> Name of file or directory to upload. Specify multiple times for multiple uploads.

Optional arguments:

-c, --changed Only upload files that have changed since the last upload.

--skip-identical Skip uploading files that are identical on both sides.

-S, --segment-size <size> Upload files in segments no larger than <size> (in Bytes) and then create a manifest file that will download all the segments as if it were the original file.

--segment-container <container> Upload the segments into the specified container. If not specified, the segments will be uploaded to a <container>_segments container to not pollute the main <container> listings.

--leave-segments Indicates that you want the older segments of manifest objects left alone (in the case of overwrites).

--object-threads <threads> Number of threads to use for uploading full objects. Default is 10.

--segment-threads <threads> Number of threads to use for uploading object segments. Default is 10.

-H, --header <header:value> Adds a customized request header. This option may be repeated. Example: -H content-type:text/plain -H Content-Length: 4000.

--use-slo When used in conjunction with segment-size it will create a Static Large Object instead of the default Dynamic Large Object.

--object-name <object-name> Upload file and name object to <object-name> or upload dir and use <object-name> as object prefix instead of folder name.

--ignore-checksum Turn off checksum validation for uploads.

swift post

```
Usage: swift post [--read-acl <acl>] [--write-acl <acl>] [--sync-to <sync-to>]
                  [--sync-key <sync-key>] [--meta <name:value>]
                  [--header <header>]
                  [<container> [<object>]]
```

Updates meta information for the account, container, or object depending on the arguments given. If the container is not found, the `swiftclient` will create it automatically, but this is not true for accounts and objects. Containers also allow the `-r <read-acl>` (or `--read-acl <read-acl>`) and `-w <write-acl>` (or `--write-acl <write-acl>`) options. The `-m` or `--meta` option is allowed on accounts, containers and objects, and is used to define the user metadata items to set in the form `Name:Value`. You can repeat this option. For example: `post -m Color:Blue -m Size:Large`

For more information about ACL formats see the documentation: ACLs.

Positional arguments:

[container] Name of container to post to.

[object] Name of object to post.

Optional arguments:

-r, --read-acl <acl> Read ACL for containers. Quick summary of ACL syntax: `.r:*`, `.r:-.example.com`, `.r:www.example.com`, `account1` (v1.0 identity API only), `account1:*`, `account2:user2` (v2.0+ identity API).

-w, --write-acl <acl> Write ACL for containers. Quick summary of ACL syntax: `account1` (v1.0 identity API only), `account1:*`, `account2:user2` (v2.0+ identity API).

-t, --sync-to <sync-to> Sync To for containers, for multi-cluster replication.

-k, --sync-key <sync-key> Sync Key for containers, for multi-cluster replication.

-m, --meta <name:value> Sets a meta data item. This option may be repeated.

Example: -m Color:Blue -m Size:Large

-H, --header <header:value> Adds a customized request header. This option may be repeated.

Example: -H content-type:text/plain -H Content-Length: 4000

swift download

```
Usage: swift download [--all] [--marker <marker>] [--prefix <prefix>]
                      [--output <out_file>] [--output-dir <out_directory>]
                      [--object-threads <threads>] [--ignore-checksum]
                      [--container-threads <threads>] [--no-download]
                      [--skip-identical] [--remove-prefix]
                      [--header <header:value>] [--no-shuffle]
                      [<container> [<object>] [...]]
```

Downloads everything in the account (with `--all`), or everything in a container, or a list of objects depending on the arguments given. For a single object download, you may use the `-o <filename>` or `--output <filename>` option to redirect the output to a specific file or `-` to redirect to stdout. The `--ignore-checksum` is an option that turn off checksum validation. You can specify optional headers with the repeatable cURL-like option `-H [--header <name:value>]`. `--ignore-mtime` ignores the `x-object-meta-mtime` metadata entry on the object (if present) and instead creates the downloaded files with fresh atime and mtime values.

Positional arguments:

<container> Name of container to download from. To download a whole account, omit this and specify all.

<object> Name of object to download. Specify multiple times for multiple objects. Omit this to download all objects from the container.

Optional arguments:

-a, --all Indicates that you really want to download everything in the account.

-m, --marker <marker> Marker to use when starting a container or account download.

-p, --prefix <prefix> Only download items beginning with <prefix>

-r, --remove-prefix An optional flag for prefix <prefix>, use this option to download items without <prefix>

-o, --output <out_file> For a single file download, stream the output to <out_file>. Specifying - as <out_file> will redirect to stdout.

-D, --output-dir <out_directory> An optional directory to which to store objects. By default, all objects are recreated in the current directory.

--object-threads <threads> Number of threads to use for downloading objects. Default is 10.

--container-threads <threads> Number of threads to use for downloading containers. Default is 10.

--no-download Perform download(s), but dont actually write anything to disk.

-H, --header <header:value> Adds a customized request header to the query, like Range or If-Match. This option may be repeated.

Example: header content-type:text/plain

--skip-identical Skip downloading files that are identical on both sides.

--ignore-checksum Turn off checksum validation for downloads.

--no-shuffle By default, when downloading a complete account or container, download order is randomised in order to reduce the load on individual drives when multiple clients are executed simultaneously to download the same set of objects (e.g. a nightly automated download script to multiple servers). Enable this option to submit download jobs to the thread pool in the order they are listed in the object store.

swift delete

```
Usage: swift delete [--all] [--leave-segments]
                    [--object-threads <threads>]
                    [--container-threads <threads>]
                    [--header <header:value>]
                    [<container> [<object>] [...]]
```

Deletes everything in the account (with `--all`), or everything in a container, or a list of objects depending on the arguments given. Segments of manifest objects will be deleted as well, unless you specify the `--leave-segments` option.

Positional arguments:

[<container>] Name of container to delete from.

[<object>] Name of object to delete. Specify multiple times for multiple objects.

Optional arguments:

-a, --all Delete all containers and objects.

--leave-segments Do not delete segments of manifest objects.

-H, --header <header:value> Adds a custom request header to use for deleting objects or an entire container.

--object-threads <threads> Number of threads to use for deleting objects. Default is 10.

--container-threads <threads> Number of threads to use for deleting containers. Default is 10.

swift copy

```
Usage: swift copy [--destination </container/object>] [--fresh-metadata]
                  [--meta <name:value>] [--header <header>] <container>
                  <object> [<object>] [...]
```

Copies an object to a new destination or adds user metadata to an object. Depending on the options supplied, you can preserve existing metadata in contrast to the post command. The `--destination` option sets the copy target destination in the form `/container/object`. If not set, the object will be copied onto itself which is useful for adding metadata. You can use the `-M` or `--fresh-metadata` option to copy an object without existing user meta data, and the `-m` or `--meta` option to define user meta data items to set in the form `Name:Value`. You can repeat this option. For example: `copy -m Color:Blue -m Size:Large`.

Positional arguments:

<container> Name of container to copy from.

<object> Name of object to copy. Specify multiple times for multiple objects

Optional arguments:

-d, --destination </container[/object]> The container and name of the destination object. Name of destination object can be omitted, then will be same as name of source object. Supplying multiple objects and destination with object name is invalid.

-M, --fresh-metadata Copy the object without any existing metadata, If not set, metadata will be preserved or appended

-m, --meta <name:value> Sets a meta data item. This option may be repeated.

Example: -m Color:Blue -m Size:Large

-H, --header <header:value> Adds a customized request header. This option may be repeated.

Example: -H content-type:text/plain -H Content-Length: 4000

swift capabilities

```
Usage: swift capabilities [--json] [<proxy_url>]
```

Displays cluster capabilities. The output includes the list of the activated Swift middlewares as well as relevant options for each ones. Additionally the command displays relevant options for the Swift core. If the `proxy-url` option is not provided, the storage URL retrieved after authentication is used as `proxy-url`.

Optional positional arguments:

<proxy_url> Proxy URL of the cluster to retrieve capabilities.

--json Print the cluster capabilities in JSON format.

swift tempurl

```
Usage: swift tempurl [--absolute] [--prefix-based]
                     <method> <seconds> <path> <key>
```

Generates a temporary URL for a Swift object. `method` option sets an HTTP method to allow for this temporary URL that is usually `GET` or `PUT`. `time` option sets the amount of time the temporary URL will be valid for. `time` can be specified as an integer, denoting the number of seconds from now on until the URL shall be valid; or, if `--absolute` is passed, the Unix timestamp when the temporary URL will expire. But beyond that, `time` can also be specified as an ISO 8601 timestamp in one of following formats:

i) Complete date: YYYY-MM-DD (e.g. 1997-07-16)

ii) Complete date plus hours, minutes and seconds: YYYY-MM-DDThh:mm:ss (e.g. 1997-07-16T19:20:30)

iii) Complete date plus hours, minutes and seconds with UTC designator: YYYY-MM-DDThh:mm:ssZ (e.g. 1997-07-16T19:20:30Z)

Please be aware that if you dont provide the UTC designator (i.e., Z) the timestamp is generated using your local timezone. If only a date is specified, the time part used will equal to **00:00:00**.

`path` option sets the full path to the Swift object. Example: `/v1/AUTH_account/c/o`. `key` option is the secret temporary URL key set on the Swift cluster. To set a key, run `swift post -m "Temp-URL-Key: <your secret key>"`. To generate a prefix-based temporary URL use the `--prefix-based` option. This URL will contain the path to the prefix. Do not forget to append the desired objectname at the end of the path portion (and before the query portion) before sharing the URL. It is possible to use ISO 8601 UTC timestamps within the URL by using the `--iso8601` option.

Positional arguments:

<method> An HTTP method to allow for this temporary URL. Usually GET or PUT.

<seconds> The amount of time in seconds the temporary URL will be valid for; or, if absolute is passed, the Unix timestamp when the temporary URL will expire.

<path> The full path to the Swift object.

Example: /v1/AUTH_account/c/o or: http://saio:8080/v1/AUTH_account/c/o

<key> The secret temporary URL key set on the Swift cluster. To set a key, run swift post -m Temp-URL-Key:b3968d0207b54ece87cccc06515a89d4

Optional arguments:

--absolute Interpret the <seconds> positional argument as a Unix timestamp rather than a number of seconds in the future.

--prefix-based If present, a prefix-based tempURL will be generated.

2.2.5 Examples

In this section we present some example usage of the `swift` CLI. To keep the examples as short as possible, these examples assume that the relevant authentication options have been set using environment variables. You can obtain the full list of commands and options available in the `swift` CLI by executing the following:

```
> swift --help
> swift <command> --help
```

Simple examples

List the existing swift containers:

```
> swift list

container_1
```

Create a new container:

```
> swift post TestContainer
```

Upload an object into a container:

```
> swift upload TestContainer testSwift.txt

testSwift.txt
```

List the contents of a container:

```
> swift list TestContainer

testSwift.txt
```

Copy an object to new destination:

```
> swift copy -d /DestContainer/testSwift.txt SourceContainer testSwift.txt

SourceContainer/testSwift.txt copied to /DestContainer/testSwift.txt
```

Delete an object from a container:

```
> swift delete TestContainer testSwift.txt

testSwift.txt
```

Delete a container:

```
> swift delete TestContainer

TestContainer
```

Display auth related authentication variables in shell friendly format:

```
> swift auth

export OS_STORAGE_URL=http://127.0.0.1:8080/v1/AUTH_
↪bf5e63572f7a420a83fcf0aa8c72c2c7
export OS_AUTH_TOKEN=c597015ae19943a18438b52ef3762e79
```

Download an object from a container:

```
> swift download TestContainer testSwift.txt

testSwift.txt [auth 0.028s, headers 0.045s, total 0.045s, 0.002 MB/s]
```

Note: To upload an object to a container, your current working directory must be where the file is located or you must provide the complete path to the file. In other words, the object-name <object-name> is an option that will upload file and name object to <object-name> or upload directory and use <object-name> as object prefix. In the case that you provide the complete path of the file, that complete path will be the name of the uploaded object.

For example:

```
> swift upload TestContainer /home/swift/testSwift/testSwift.txt

home/swift/testSwift/testSwift.txt

> swift list TestContainer
```

(continues on next page)

(continued from previous page)

```
home/swift/testSwift/testSwift.txt
```

More complex examples

Swift has a single object size limit of 5GiB. In order to upload files larger than this, we must create a large object that consists of smaller segments. The example below shows how to upload a large video file as a static large object in 1GiB segments:

```
> swift upload videos --use-slo --segment-size 1G myvideo.mp4

myvideo.mp4 segment 8
myvideo.mp4 segment 4
myvideo.mp4 segment 2
myvideo.mp4 segment 7
myvideo.mp4 segment 0
myvideo.mp4 segment 1
myvideo.mp4 segment 3
myvideo.mp4 segment 6
myvideo.mp4 segment 5
myvideo.mp4
```

This command will upload segments to a container named `videos_segments`, and create a manifest file describing the entire object in the `videos` container. For more information on large objects, see the documentation here.

```
> swift list videos

myvideo.mp4

> swift list videos_segments

myvideo.mp4/slo/1460229233.679546/9341553868/1073741824/00000000
myvideo.mp4/slo/1460229233.679546/9341553868/1073741824/00000001
myvideo.mp4/slo/1460229233.679546/9341553868/1073741824/00000002
myvideo.mp4/slo/1460229233.679546/9341553868/1073741824/00000003
myvideo.mp4/slo/1460229233.679546/9341553868/1073741824/00000004
myvideo.mp4/slo/1460229233.679546/9341553868/1073741824/00000005
myvideo.mp4/slo/1460229233.679546/9341553868/1073741824/00000006
myvideo.mp4/slo/1460229233.679546/9341553868/1073741824/00000007
myvideo.mp4/slo/1460229233.679546/9341553868/1073741824/00000008
```

Firstly, the key should be set, then generate a temporary URL for a Swift object:

```
> swift post -m "Temp-URL-Key:b3968d0207b54ece87cccc06515a89d4"

> swift tempurl GET 6000 /v1/AUTH_bf5e63572f7a420a83fcf0aa8c72c2c7\
  /firstcontainer/clean.sh b3968d0207b54ece87cccc06515a89d4
```

(continues on next page)

(continued from previous page)

```
/v1/AUTH_/firstcontainer/clean.sh?temp_url_sig=\
9218fc288cc09e5edd857b6a3d43cf2122b906dc&temp_url_expires=1472203614
```

2.3 The swiftclient.SwiftService API

A higher-level API aimed at allowing developers an easy way to perform multiple operations asynchronously using a configurable thread pool. Documentation for each service method call can be found here: *swiftclient.service*.

2.3.1 Authentication

This section covers the various options for authenticating with a swift object store. The combinations of options required for each authentication version are detailed below. Once again, these are just a subset of those that can be used to successfully authenticate, but they are the most common and recommended.

The relevant authentication options are presented as python dictionaries that should be added to any other options you are supplying to your `SwiftService` instance. As indicated in the python code, you can also set these options as environment variables that will be loaded automatically if the relevant option is not specified.

The `SwiftService` authentication attempts to automatically select the auth version based on the combination of options specified, but supplying options from multiple different auth versions can cause unexpected behaviour.

Note: Leftover environment variables are a common source of confusion when authorization fails.

Keystone V3

```
{
    ...
    "auth_version": environ.get('ST_AUTH_VERSION'),  # Should be '3'
    "os_username": environ.get('OS_USERNAME'),
    "os_password": environ.get('OS_PASSWORD'),
    "os_project_name": environ.get('OS_PROJECT_NAME'),
    "os_project_domain_name": environ.get('OS_PROJECT_DOMAIN_NAME'),
    "os_auth_url": environ.get('OS_AUTH_URL'),
    ...
}
```

```
{
    ...
    "auth_version": environ.get('ST_AUTH_VERSION'),  # Should be '3'
    "os_username": environ.get('OS_USERNAME'),
    "os_password": environ.get('OS_PASSWORD'),
    "os_project_id": environ.get('OS_PROJECT_ID'),
```

(continues on next page)

(continued from previous page)

```
    "os_project_domain_id": environ.get('OS_PROJECT_DOMAIN_ID'),
    "os_auth_url": environ.get('OS_AUTH_URL'),
    ...
}
```

Keystone V2

```
{
    ...
    "auth_version": environ.get('ST_AUTH_VERSION'),  # Should be '2.0'
    "os_username": environ.get('OS_USERNAME'),
    "os_password": environ.get('OS_PASSWORD'),
    "os_tenant_name": environ.get('OS_TENANT_NAME'),
    "os_auth_url": environ.get('OS_AUTH_URL'),
    ...
}
```

Legacy Auth

```
{
    ...
    "auth_version": environ.get('ST_AUTH_VERSION'),  # Should be '1.0'
    "auth": environ.get('ST_AUTH'),
    "user": environ.get('ST_USER'),
    "key": environ.get('ST_KEY'),
    ...
}
```

2.3.2 Configuration

When you create an instance of a `SwiftService`, you can override a collection of default options to suit your use case. Typically, the defaults are sensible to get us started, but depending on your needs you might want to tweak them to improve performance (options affecting large objects and thread counts can significantly alter performance in the right situation).

Service level defaults and some extra options can also be overridden on a per-operation (or even in some cases per-object) basis, and you will call out which options affect which operations later in the document.

The configuration of the service API is performed using an options dictionary passed to the `SwiftService` during initialisation. The options available in this dictionary are described below, along with their defaults:

Options

retries: 5 The number of times that the library should attempt to retry HTTP actions before giving up and reporting a failure.

container_threads: 10

object_dd_threads: 10

object_uu_threads: 10

segment_threads: 10 The above options determine the size of the available thread pools for performing swift operations. Container operations (such as listing a container) operate in the container threads, and a similar pattern applies to object and segment threads.

Note: Object threads are separated into two separate thread pools: uu and dd. This stands for upload/update and download/delete, and the corresponding actions will be run on separate threads pools.

---

segment_size: None If specified, this option enables uploading of large objects. Should the object being uploaded be larger than 5G in size, this option is mandatory otherwise the upload will fail. This option should be specified as a size in bytes.

use_slo: False Used in combination with the above option, use_slo will upload large objects as static rather than dynamic. Only static large objects provide error checking for the downloaded object, so we recommend this option.

segment_container: None Allows the user to select the container into which large object segments will be uploaded. We do not recommend changing this value as it could make locating orphaned segments more difficult in the case of errors.

leave_segments: False Setting this option to true means that when deleting or overwriting a large object, its segments will be left in the object store and must be cleaned up manually. This option can be useful when sharing large object segments between multiple objects in more advanced scenarios, but must be treated with care, as it could lead to ever increasing storage usage.

changed: None This option affects uploads and simply means that those objects which already exist in the object store will not be overwritten if the mtime and size of the source is the same as the existing object.

skip_identical: False A slightly more thorough case of the above, but rather than mtime and size uses an objects MD5 sum.

yes_all: False This options affects only download and delete, and in each case must be specified in order to download/delete the entire contents of an account. This option has no effect on any other calls.

no_download: False This option only affects download and means that all operations proceed as normal with the exception that no data is written to disk.

header: [] Used with upload and post operations to set headers on objects. Headers are specified as colon separated strings, e.g. content-type:text/plain.

meta: [] Used to set metadata on an object similarly to headers.

> **Note:** Setting metadata is a destructive operation, so when updating one of many metadata values all desired metadata for an object must be re-applied.

long: False Affects only list operations, and results in more metrics being made available in the results at the expense of lower performance.

fail_fast: False Applies to delete and upload operations, and attempts to abort queued tasks in the event of errors.

prefix: None Affects list operations; only objects with the given prefix will be returned/affected. It is not advisable to set at the service level, as those operations that call list to discover objects on which they should operate will also be affected.

delimiter: None Affects list operations, and means that listings only contain results up to the first instance of the delimiter in the object name. This is useful for working with objects containing / in their names to simulate folder structures.

dir_marker: False Affects uploads, and allows empty pseudofolder objects to be created when the source of an upload is `None`.

checksum: True Affects uploads and downloads. If set check md5 sum for the transfer.

shuffle: False When downloading objects, the default behaviour of the CLI is to shuffle lists of objects in order to spread the load on storage drives when multiple clients are downloading the same files to multiple locations (e.g. in the event of distributing an update). When using the `SwiftService` directly, object downloads are scheduled in the same order as they appear in the container listing. When combined with a single download thread this means that objects are downloaded in lexically-sorted order. Setting this option to `True` gives the same shuffling behaviour as the CLI.

destination: None When copying objects, this specifies the destination where the object will be copied to. The default of None means copy will be the same as source.

fresh_metadata: None When copying objects, this specifies that the object metadata on the source will *not* be applied to the destination object - the destination object will have a new fresh set of metadata that includes *only* the metadata specified in the meta option if any at all.

Other available options can be found in `swiftclient/service.py` in the source code for `python-swiftclient`. Each `SwiftService` method also allows for an optional dictionary to override those specified at init time, and the appropriate docstrings show which options modify each methods behaviour.

2.3.3 Available Operations

Each operation provided by the service API may raise a `SwiftError` or `ClientException` for any call that fails completely (or a call which performs only one operation at an account or container level). In the case of a successful call an operation returns one of the following:

- A dictionary detailing the results of a single operation.
- An iterator that produces result dictionaries (for calls that perform multiple sub-operations).

A result dictionary can indicate either the success or failure of an individual operation (detailed in the `success` key), and will either contain the successful result, or an `error` key detailing the error encountered (usually an instance of Exception).

An example result dictionary is given below:

```
result = {
    'action': 'download_object',
    'success': True,
    'container': container,
    'object': obj,
    'path': path,
    'start_time': start_time,
    'finish_time': finish_time,
    'headers_receipt': headers_receipt,
    'auth_end_time': conn.auth_end_time,
    'read_length': bytes_read,
    'attempts': conn.attempts
}
```

All the possible `action` values are detailed below:

```
[
    'stat_account',
    'stat_container',
    'stat_object',
    'post_account',
    'post_container',
    'post_object',
    'list_part',          # list yields zero or more 'list_part' results
    'download_object',
    'create_container',   # from upload
    'create_dir_marker',  # from upload
    'upload_object',
    'upload_segment',
    'delete_container',
    'delete_object',
    'delete_segment',     # from delete_object operations
    'capabilities',
]
```

Stat

Stat can be called against an account, a container, or a list of objects to get account stats, container stats or information about the given objects. In the first two cases a dictionary is returned containing the results of the operation, and in the case of a list of object names being supplied, an iterator over the results generated for each object is returned.

Information returned includes the amount of data used by the given object/container/account and any headers or metadata set (this includes user set data as well as content-type and modification times).

See `swiftclient.service.SwiftService.stat` for docs generated from the method docstring.

Valid calls for this method are as follows:

- `stat([options])`: Returns stats for the configured account.

- `stat(<container>, [options])`: Returns stats for the given container.
- `stat(<container>, <object_list>, [options])`: Returns stats for each of the given objects in the given container (through the returned iterator).

Results from stat are dictionaries indicating the success or failure of each operation. In the case of a successful stat against an account or container, the method returns immediately with one of the following results:

```
{
    'action': 'stat_account',
    'success': True,
    'items': items,
    'headers': headers
}
```

```
{
    'action': 'stat_container',
    'container': <container>,
    'success': True,
    'items': items,
    'headers': headers
}
```

In the case of stat called against a list of objects, the method returns a generator that returns the results of individual object stat operations as they are performed on the thread pool:

```
{
    'action': 'stat_object',
    'object': <object_name>,
    'container': <container>,
    'success': True,
    'items': items,
    'headers': headers
}
```

In the case of a failure the dictionary returned will indicate that the operation was not successful, and will include the keys below:

```
{
    'action': <'stat_object'|'stat_container'|'stat_account'>,
    'object': <'object_name'>,      # Only for stat with objects list
    'container': <container>,       # Only for stat with objects list or␣
↪container
    'success': False,
    'error': <error>,
    'traceback': <trace>,
    'error_timestamp': <timestamp>
}
```

Example

The code below demonstrates the use of `stat` to retrieve the headers for a given list of objects in a container using 20 threads. The code creates a mapping from object name to headers which is then pretty printed to the log.

```
import logging
import pprint

from swiftclient.service import SwiftService
from sys import argv

logging.basicConfig(level=logging.ERROR)
logging.getLogger("requests").setLevel(logging.CRITICAL)
logging.getLogger("swiftclient").setLevel(logging.CRITICAL)
logger = logging.getLogger(__name__)

_opts = {'object_dd_threads': 20}
with SwiftService(options=_opts) as swift:
    container = argv[1]
    objects = argv[2:]
    header_data = {}
    stats_it = swift.stat(container=container, objects=objects)
    for stat_res in stats_it:
        if stat_res['success']:
            header_data[stat_res['object']] = stat_res['headers']
        else:
            logger.error(
                'Failed to retrieve stats for %s' % stat_res['object']
            )
    pprint.pprint(header_data)
```

List

List can be called against an account or a container to retrieve the containers or objects contained within them. Each call returns an iterator that returns pages of results (by default, up to 10000 results in each page).

See `swiftclient.service.SwiftService.list` for docs generated from the method docstring.

If the given container or account does not exist, the list method will raise a `SwiftError`, but for all other success/failures a dictionary is returned. Each successfully listed page returns a dictionary as described below:

```
{
    'action': <'list_account_part'|'list_container_part'>,
    'container': <container>,       # Only for listing a container
    'prefix': <prefix>,             # The prefix of returned objects/containers
    'success': True,
    'listing': [Item],              # A list of results
```

(continues on next page)

(continued from previous page)

```
                                # (only in the event of success)
    'marker': <marker>          # The last item name in the list
                                # (only in the event of success)
}
```

Where an item contains the following keys:

```
{
    'name': <name>,
    'bytes': 10485760,
    'last_modified': '2014-12-11T12:02:38.774540',
    'hash': 'fb938269cbeabe4c234e1127bbd3b74a',
    'content_type': 'application/octet-stream',
    'meta': <metadata>    # Full metadata listing from stat'ing each object
                          # this key only exists if 'long' is specified in␣
↪options
}
```

Any failure listing an account or container that exists will return a failure dictionary as described below:

```
{
    'action': <'list_account_part'|'list_container_part'>,,
    'container': container,          # Only for listing a container
    'prefix': options['prefix'],
    'success': success,
    'marker': marker,
    'error': error,
    'traceback': <trace>,
    'error_timestamp': <timestamp>
}
```

Example

The code below demonstrates the use of `list` to list all items in a container that are over 10MiB in size:

```
import logging

from swiftclient.service import SwiftService, SwiftError
from sys import argv

logging.basicConfig(level=logging.ERROR)
logging.getLogger("requests").setLevel(logging.CRITICAL)
logging.getLogger("swiftclient").setLevel(logging.CRITICAL)
logger = logging.getLogger(__name__)

container = argv[1]
minimum_size = 10*1024**2
with SwiftService() as swift:
    try:
```

```
        list_parts_gen = swift.list(container=container)
        for page in list_parts_gen:
            if page["success"]:
                for item in page["listing"]:

                    i_size = int(item["bytes"])
                    if i_size > minimum_size:
                        i_name = item["name"]
                        i_etag = item["hash"]
                        print(
                            "%s [size: %s] [etag: %s]" %
                            (i_name, i_size, i_etag)
                        )
            else:
                raise page["error"]

    except SwiftError as e:
        logger.error(e.value)
```

Post

Post can be called against an account, container or list of objects in order to update the metadata attached to the given items. In the first two cases a single dictionary is returned containing the results of the operation, and in the case of a list of objects being supplied, an iterator over the results generated for each object post is returned.

Each element of the object list may be a plain string of the object name, or a `SwiftPostObject` that allows finer control over the options and metadata applied to each of the individual post operations. When a string is given for the object name, the options and metadata applied are a combination of those supplied to the call to `post()` and the defaults of the `SwiftService` object.

If the given container or account does not exist, the `post` method will raise a `SwiftError`. Successful metadata update results are dictionaries as described below:

```
{
    'action': <'post_account'|'post_container'|'post_object'>,
    'success': True,
    'container': <container>,
    'object': <object>,
    'headers': {},
    'response_dict': <HTTP response details>
}
```

Note: Updating user metadata keys will not only add any specified keys, but will also remove user metadata that has previously been set. This means that each time user metadata is updated, the complete set of desired key-value pairs must be specified.

Example

The code below demonstrates the use of `post` to set an archive folder in a given container to expire after a 24 hour delay:

```
import logging

from swiftclient.service import SwiftService, SwiftError
from sys import argv

logging.basicConfig(level=logging.ERROR)
logging.getLogger("requests").setLevel(logging.CRITICAL)
logging.getLogger("swiftclient").setLevel(logging.CRITICAL)
logger = logging.getLogger(__name__)

container = argv[1]
with SwiftService() as swift:
    try:
        list_options = {"prefix": "archive_2016-01-01/"}
        list_parts_gen = swift.list(container=container)
        for page in list_parts_gen:
            if page["success"]:
                objects = [obj["name"] for obj in page["listing"]]
                post_options = {"header": "X-Delete-After:86400"}
                for post_res in swift.post(
                        container=container,
                        objects=objects,
                        options=post_options):
                    if post_res['success']:
                        print("Object '%s' POST success" % post_res['object
↪'])
                    else:
                        print("Object '%s' POST failed" % post_res['object
↪'])
            else:
                raise page["error"]
    except SwiftError as e:
        logger.error(e.value)
```

Download

Download can be called against an entire account, a single container, or a list of objects in a given container. Each element of the object list is a string detailing the full name of an object to download.

In order to download the full contents of an entire account, you must set the value of `yes_all` to `True` in the `options` dictionary supplied to either the `SwiftService` instance or the call to `download`.

If the given container or account does not exist, the `download` method will raise a `SwiftError`, otherwise an iterator over the results generated for each object download is returned.

See `swiftclient.service.SwiftService.download` for docs generated from the method docstring.

For each successfully downloaded object, the results returned by the iterator will be a dictionary as described below (results are not returned for completed container or object segment downloads):

```
{
    'action': 'download_object',
    'container': <container>,
    'object': <object name>,
    'success': True,
    'path': <local path to downloaded object>,
    'pseudodir': <if true, the download created an empty directory>,
    'start_time': <time download started>,
    'end_time': <time download completed>,
    'headers_receipt': <time the headers from the object were retrieved>,
    'auth_end_time': <time authentication completed>,
    'read_length': <bytes_read>,
    'attempts': <attempt count>,
    'response_dict': <HTTP response details>
}
```

Any failure uploading an object will return a failure dictionary as described below:

```
{
    'action': 'download_object',
    'container': <container>,
    'object': <object name>,
    'success': False,
    'path': <local path of the failed download>,
    'pseudodir': <if true, the failed download was an empty directory>,
    'attempts': <attempt count>,
    'error': <error>,
    'traceback': <trace>,
    'error_timestamp': <timestamp>,
    'response_dict': <HTTP response details>
}
```

Example

The code below demonstrates the use of `download` to download all PNG images from a dated archive folder in a given container:

```
import logging

from swiftclient.service import SwiftService, SwiftError
from sys import argv

logging.basicConfig(level=logging.ERROR)
logging.getLogger("requests").setLevel(logging.CRITICAL)
logging.getLogger("swiftclient").setLevel(logging.CRITICAL)
logger = logging.getLogger(__name__)

def is_png(obj):
```

```
    return (
        obj["name"].lower().endswith('.png') or
        obj["content_type"] == 'image/png'
    )

container = argv[1]
with SwiftService() as swift:
    try:
        list_options = {"prefix": "archive_2016-01-01/"}
        list_parts_gen = swift.list(container=container)
        for page in list_parts_gen:
            if page["success"]:
                objects = [
                    obj["name"] for obj in page["listing"] if is_png(obj)
                ]
                for down_res in swift.download(
                        container=container,
                        objects=objects):
                    if down_res['success']:
                        print("'%s' downloaded" % down_res['object'])
                    else:
                        print("'%s' download failed" % down_res['object'])
            else:
                raise page["error"]
    except SwiftError as e:
        logger.error(e.value)
```

Upload

Upload is always called against an account and container and with a list of objects to upload. Each element of the object list may be a plain string detailing the path of the object to upload, or a `SwiftUploadObject` that allows finer control over some aspects of the individual operations.

When a simple string is supplied to specify a file to upload, the name of the object uploaded is the full path of the specified file and the options used for the upload are those supplied to the call to `upload`.

Constructing a `SwiftUploadObject` allows the user to supply an object name for the uploaded file, and modify the options used by `upload` at the granularity of individual files.

If the given container or account does not exist, the `upload` method will raise a `SwiftError`, otherwise an iterator over the results generated for each object upload is returned.

See `swiftclient.service.SwiftService.upload` for docs generated from the method docstring.

For each successfully uploaded object (or object segment), the results returned by the iterator will be a dictionary as described below:

```
{
    'action': 'upload_object',
    'container': <container>,
    'object': <object name>,
```

(continues on next page)

(continued from previous page)

```
    'success': True,
    'status': <'uploaded'|'skipped-identical'|'skipped-changed'>,
    'attempts': <attempt count>,
    'response_dict': <HTTP response details>
}

{
    'action': 'upload_segment',
    'for_container': <container>,
    'for_object': <object name>,
    'segment_index': <segment_index>,
    'segment_size': <segment_size>,
    'segment_location': <segment_path>
    'segment_etag': <etag>,
    'log_line': <object segment n>
    'success': True,
    'response_dict': <HTTP response details>,
    'attempts': <attempt count>
}
```

Any failure uploading an object will return a failure dictionary as described below:

```
{
    'action': 'upload_object',
    'container': <container>,
    'object': <object name>,
    'success': False,
    'attempts': <attempt count>,
    'error': <error>,
    'traceback': <trace>,
    'error_timestamp': <timestamp>,
    'response_dict': <HTTP response details>
}

{
    'action': 'upload_segment',
    'for_container': <container>,
    'for_object': <object name>,
    'segment_index': <segment_index>,
    'segment_size': <segment_size>,
    'segment_location': <segment_path>,
    'log_line': <object segment n>,
    'success': False,
    'error': <error>,
    'traceback': <trace>,
    'error_timestamp': <timestamp>,
    'response_dict': <HTTP response details>,
    'attempts': <attempt count>
}
```

Example

The code below demonstrates the use of `upload` to upload all files and folders in a given directory, and rename each object by replacing the root directory name with my-<d>-objects, where <d> is the name of the uploaded directory:

```
import logging

from os import walk
from os.path import join
from swiftclient.multithreading import OutputManager
from swiftclient.service import SwiftError, SwiftService, SwiftUploadObject
from sys import argv

logging.basicConfig(level=logging.ERROR)
logging.getLogger("requests").setLevel(logging.CRITICAL)
logging.getLogger("swiftclient").setLevel(logging.CRITICAL)
logger = logging.getLogger(__name__)

_opts = {'object_uu_threads': 20}
dir = argv[1]
container = argv[2]
with SwiftService(options=_opts) as swift, OutputManager() as out_manager:
    try:
        # Collect all the files and folders in the given directory
        objs = []
        dir_markers = []
        for (_dir, _ds, _fs) in walk(dir):
            if not (_ds + _fs):
                dir_markers.append(_dir)
            else:
                objs.extend([join(_dir, _f) for _f in _fs])

        # Now that we've collected all the required files and dir markers
        # build the ``SwiftUploadObject``s for the call to upload
        objs = [
            SwiftUploadObject(
                o, object_name=o.replace(
                    dir, 'my-%s-objects' % dir, 1
                )
            ) for o in objs
        ]
        dir_markers = [
            SwiftUploadObject(
                None, object_name=d.replace(
                    dir, 'my-%s-objects' % dir, 1
                ), options={'dir_marker': True}
            ) for d in dir_markers
        ]

        # Schedule uploads on the SwiftService thread pool and iterate
```

```
        # over the results
        for r in swift.upload(container, objs + dir_markers):
            if r['success']:
                if 'object' in r:
                    print(r['object'])
                elif 'for_object' in r:
                    print(
                        '%s segment %s' % (r['for_object'],
                                           r['segment_index'])
                        )
            else:
                error = r['error']
                if r['action'] == "create_container":
                    logger.warning(
                        'Warning: failed to create container '
                        "'%s'%s", container, error
                    )
                elif r['action'] == "upload_object":
                    logger.error(
                        "Failed to upload object %s to container %s: %s" %
                        (container, r['object'], error)
                    )
                else:
                    logger.error("%s" % error)

    except SwiftError as e:
        logger.error(e.value)
```

Delete

Delete can be called against an account or a container to remove the containers or objects contained within them. Each call to `delete` returns an iterator over results of each resulting sub-request.

If the number of requested delete operations is large and the target swift cluster is running the bulk middleware, the call to `SwiftService.delete` will make use of bulk operations and the returned result iterator will return `bulk_delete` results rather than individual `delete_object`, `delete_container` or `delete_segment` results.

See `swiftclient.service.SwiftService.delete` for docs generated from the method docstring.

For each successfully deleted container, object or segment, the results returned by the iterator will be a dictionary as described below:

```
{
    'action': <'delete_object'|'delete_segment'>,
    'container': <container>,
    'object': <object name>,
    'success': True,
    'attempts': <attempt count>,
    'response_dict': <HTTP response details>
```

(continues on next page)

(continued from previous page)

```
}

{
    'action': 'delete_container',
    'container': <container>,
    'success': True,
    'response_dict': <HTTP response details>,
    'attempts': <attempt count>
}

{
    'action': 'bulk_delete',
    'container': <container>,
    'objects': <[objects]>,
    'success': True,
    'attempts': <attempt count>,
    'response_dict': <HTTP response details>
}
```

Any failure in a delete operation will return a failure dictionary as described below:

```
{
    'action': ('delete_object'|'delete_segment'),
    'container': <container>,
    'object': <object name>,
    'success': False,
    'attempts': <attempt count>,
    'error': <error>,
    'traceback': <trace>,
    'error_timestamp': <timestamp>,
    'response_dict': <HTTP response details>
}

{
    'action': 'delete_container',
    'container': <container>,
    'success': False,
    'error': <error>,
    'traceback': <trace>,
    'error_timestamp': <timestamp>,
    'response_dict': <HTTP response details>,
    'attempts': <attempt count>
}

{
    'action': 'bulk_delete',
    'container': <container>,
    'objects': <[objects]>,
    'success': False,
```

(continues on next page)

(continued from previous page)

```
    'attempts': <attempt count>,
    'error': <error>,
    'traceback': <trace>,
    'error_timestamp': <timestamp>,
    'response_dict': <HTTP response details>
}
```

Example

The code below demonstrates the use of `delete` to remove a given list of objects from a specified container. As the objects are deleted the transaction ID of the relevant request is printed along with the object name and number of attempts required. By printing the transaction ID, the printed operations can be easily linked to events in the swift server logs:

```
import logging

from swiftclient.service import SwiftService
from sys import argv


logging.basicConfig(level=logging.ERROR)
logging.getLogger("requests").setLevel(logging.CRITICAL)
logging.getLogger("swiftclient").setLevel(logging.CRITICAL)
logger = logging.getLogger(__name__)

_opts = {'object_dd_threads': 20}
container = argv[1]
objects = argv[2:]
with SwiftService(options=_opts) as swift:
    del_iter = swift.delete(container=container, objects=objects)
    for del_res in del_iter:
        c = del_res.get('container', '')
        o = del_res.get('object', '')
        a = del_res.get('attempts')
        if del_res['success'] and not del_res['action'] == 'bulk_delete':
            rd = del_res.get('response_dict')
            if rd is not None:
                t = dict(rd.get('headers', {}))
                if t:
                    print(
                        'Successfully deleted {0}/{1} in {2} attempts '
                        '(transaction id: {3})'.format(c, o, a, t)
                    )
                else:
                    print(
                        'Successfully deleted {0}/{1} in {2} '
                        'attempts'.format(c, o, a)
                    )
```

Copy

Copy can be called to copy an object or update the metadata on the given items.

Each element of the object list may be a plain string of the object name, or a `SwiftCopyObject` that allows finer control over the options applied to each of the individual copy operations (destination, fresh_metadata, options).

Destination should be in format /container/object; if not set, the object will be copied onto itself. Fresh_metadata sets mode of operation on metadata. If not set, current object user metadata will be copied/preserved; if set, all current user metadata will be removed.

Returns an iterator over the results generated for each object copy (and may also include the results of creating destination containers).

When a string is given for the object name, destination and fresh metadata will default to None and None, which result in adding metadata to existing objects.

Successful copy results are dictionaries as described below:

```
{
    'action': 'copy_object',
    'success': True,
    'container': <container>,
    'object': <object>,
    'destination': <destination>,
    'headers': {},
    'fresh_metadata': <boolean>,
    'response_dict': <HTTP response details>
}
```

Any failure in a copy operation will return a failure dictionary as described below:

```
{
    'action': 'copy_object',
    'success': False,
    'container': <container>,
    'object': <object>,
    'destination': <destination>,
    'headers': {},
    'fresh_metadata': <boolean>,
    'response_dict': <HTTP response details>,
    'error': <error>,
    'traceback': <traceback>,
    'error_timestamp': <timestamp>
}
```

Example

The code below demonstrates the use of `copy` to add new user metadata for objects a and b, and to copy object c to d (with added metadata).

```
import logging
```

```
from swiftclient.service import SwiftService, SwiftCopyObject, SwiftError

logging.basicConfig(level=logging.ERROR)
logging.getLogger("requests").setLevel(logging.CRITICAL)
logging.getLogger("swiftclient").setLevel(logging.CRITICAL)
logger = logging.getLogger(__name__)

with SwiftService() as swift:
    try:
        obj = SwiftCopyObject("c", {"destination": "/cont/d"})
        for i in swift.copy(
                "cont", ["a", "b", obj],
                {"meta": ["foo:bar"], "destination": "/cc"}):
            if i["success"]:
                if i["action"] == "copy_object":
                    print(
                        "object %s copied from /%s/%s" %
                        (i["destination"], i["container"], i["object"])
                    )
                elif i["action"] == "create_container":
                    print(
                        "container %s created" % i["container"]
                    )
            else:
                if "error" in i and isinstance(i["error"], Exception):
                    raise i["error"]
    except SwiftError as e:
        logger.error(e.value)
```

Capabilities

Capabilities can be called against an account or a particular proxy URL in order to determine the capabilities of the swift cluster. These capabilities include details about configuration options and the middlewares that are installed in the proxy pipeline.

See `swiftclient.service.SwiftService.capabilities` for docs generated from the method docstring.

For each successful call to list capabilities, a result dictionary will be returned with the contents described below:

```
{
    'action': 'capabilities',
    'timestamp': <time of the call>,
    'success': True,
    'capabilities': <dictionary containing capability details>
}
```

The contents of the capabilities dictionary contain the core swift capabilities under the key `swift`; all other keys show the configuration options for additional middlewares deployed in the proxy pipeline. An example capabilities dictionary is given below:

```
{
    'account_quotas': {},
    'bulk_delete': {
        'max_deletes_per_request': 10000,
        'max_failed_deletes': 1000
    },
    'bulk_upload': {
        'max_containers_per_extraction': 10000,
        'max_failed_extractions': 1000
    },
    'container_quotas': {},
    'container_sync': {'realms': {}},
    'formpost': {},
    'keystoneauth': {},
    'slo': {
        'max_manifest_segments': 1000,
        'max_manifest_size': 2097152,
        'min_segment_size': 1048576
    },
    'swift': {
        'account_autocreate': True,
        'account_listing_limit': 10000,
        'allow_account_management': True,
        'container_listing_limit': 10000,
        'extra_header_count': 0,
        'max_account_name_length': 256,
        'max_container_name_length': 256,
        'max_file_size': 5368709122,
        'max_header_size': 8192,
        'max_meta_count': 90,
        'max_meta_name_length': 128,
        'max_meta_overall_size': 4096,
        'max_meta_value_length': 256,
        'max_object_name_length': 1024,
        'policies': [
            {'default': True, 'name': 'Policy-0'}
        ],
        'strict_cors_mode': False,
        'version': '2.2.2'
    },
    'tempurl': {
        'methods': ['GET', 'HEAD', 'PUT']
    }
}
```

Example

The code below demonstrates the use of `capabilities` to determine if the Swift cluster supports static large objects, and if so, the maximum number of segments that can be described in a single

manifest file, along with the size restrictions on those objects:

```
import logging

from swiftclient.exceptions import ClientException
from swiftclient.service import SwiftService

logging.basicConfig(level=logging.ERROR)
logging.getLogger("requests").setLevel(logging.CRITICAL)
logging.getLogger("swiftclient").setLevel(logging.CRITICAL)
logger = logging.getLogger(__name__)

with SwiftService() as swift:
    try:
        capabilities_result = swift.capabilities()
        capabilities = capabilities_result['capabilities']
        if 'slo' in capabilities:
            print('SLO is supported')
        else:
            print('SLO is not supported')
    except ClientException as e:
        logger.error(e.value)
```

2.4 The swiftclient.Connection API

A low level API that provides methods for authentication and methods that correspond to the individual REST API calls described in the swift documentation.

For usage details see the client docs: *swiftclient.client*.

2.4.1 Authentication

This section covers the various combinations of kwargs required when creating an instance of the `Connection` object for communicating with a swift object store. The combinations of options required for each authentication version are detailed below, but are just a subset of those that can be used to successfully authenticate. These are the most common and recommended combinations.

Keystone Session

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

# Create a password auth plugin
auth = v3.Password(auth_url='http://127.0.0.1:5000/v3/',
                   username='tester',
                   password='testing',
                   user_domain_name='Default',
                   project_name='Default',
```

(continues on next page)

(continued from previous page)

```
                    project_domain_name='Default')

# Create session
keystone_session = session.Session(auth=auth)

# Create swiftclient Connection
swift_conn = Connection(session=keystone_session)
```

Keystone v3

```
_authurl = 'http://127.0.0.1:5000/v3/'
_auth_version = '3'
_user = 'tester'
_key = 'testing'
_os_options = {
    'user_domain_name': 'Default',
    'project_domain_name': 'Default',
    'project_name': 'Default'
}

conn = Connection(
    authurl=_authurl,
    user=_user,
    key=_key,
    os_options=_os_options,
    auth_version=_auth_version
)
```

Keystone v2

```
_authurl = 'http://127.0.0.1:5000/v2.0/'
_auth_version = '2'
_user = 'tester'
_key = 'testing'
_tenant_name = 'test'

conn = Connection(
    authurl=_authurl,
    user=_user,
    key=_key,
    tenant_name=_tenant_name,
    auth_version=_auth_version
)
```

Legacy Auth

```
_authurl = 'http://127.0.0.1:8080/'
_auth_version = '1'
_user = 'tester'
_key = 'testing'
_tenant_name = 'test'

conn = Connection(
    authurl=_authurl,
    user=_user,
    key=_key,
    tenant_name=_tenant_name,
    auth_version=_auth_version
)
```

2.4.2 Examples

In this section we present some simple code examples that demonstrate the usage of the `Connection` API. You can find full details of the options and methods available to the `Connection` API in the docstring generated documentation: *swiftclient.client*.

List the available containers:

```
resp_headers, containers = conn.get_account()
print("Response headers: %s" % resp_headers)
for container in containers:
    print(container)
```

Create a new container:

```
container = 'new-container'
conn.put_container(container)
resp_headers, containers = conn.get_account()
if container in containers:
    print("The container was created")
```

Create a new object with the contents of a local text file:

```
container = 'new-container'
with open('local.txt', 'r') as local:
    conn.put_object(
        container,
        'local_object.txt',
        contents=local,
        content_type='text/plain'
    )
```

Confirm presence of the object:

```
obj = 'local_object.txt'
container = 'new-container'
try:
    resp_headers = conn.head_object(container, obj)
    print('The object was successfully created')
except ClientException as e:
    if e.http_status = '404':
        print('The object was not found')
    else:
        print('An error occurred checking for the existence of the object')
```

Download the created object:

```
obj = 'local_object.txt'
container = 'new-container'
resp_headers, obj_contents = conn.get_object(container, obj)
with open('local_copy.txt', 'w') as local:
    local.write(obj_contents)
```

Delete the created object:

```
obj = 'local_object.txt'
container = 'new-container'
try:
    conn.delete_object(container, obj)
    print("Successfully deleted the object")
except ClientException as e:
    print("Failed to delete the object with error: %s" % e)
```

CHAPTER

THREE

CODE-GENERATED DOCUMENTATION

3.1 swiftclient

OpenStack Swift Python client binding.

3.2 swiftclient.authv1

Authentication plugin for keystoneauth to support v1 endpoints.

Way back in the long-long ago, there was no Keystone. Swift used an auth mechanism now known as v1, which used only HTTP headers. Auth requests and responses would look something like:

```
> GET /auth/v1.0 HTTP/1.1
> Host: <swift server>
> X-Auth-User: <tenant>:<user>
> X-Auth-Key: <password>
>
< HTTP/1.1 200 OK
< X-Storage-Url: http://<swift server>/v1/<tenant account>
< X-Auth-Token: <token>
< X-Storage-Token: <token>
<
```

This plugin provides a way for Keystone sessions (and clients that use them, like python-openstackclient) to communicate with old auth endpoints that still use this mechanism, such as tempauth, swauth, or https://identity.api.rackspacecloud.com/v1.0

class swiftclient.authv1.**AccessInfoV1**(*auth_url*, *storage_url*, *account*, *username*, *auth_token*, *token_life*)

An object for encapsulating a raw v1 auth token.

classmethod from_state(*data*)

Deserialize the given state.

Returns a new AccessInfoV1 object with the given state

get_state()

Serialize the current state.

will_expire_soon(*stale_duration*)

Determines if expiration is about to occur.

Returns true if expiration is within the given duration

class swiftclient.authv1.**PasswordLoader**

Option handling for the v1password plugin.

property available

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*)

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.

load_from_options(***kwargs*)

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*, ***kwargs*)

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

plugin_class

alias of *swiftclient.authv1.PasswordPlugin*

class swiftclient.authv1.**PasswordPlugin**(*auth_url*, *username*, *password*, *project_name=None*, *reauthenticate=True*)

A plugin for authenticating with a username and password.

Subclassing from BaseIdentityPlugin gets us a few niceties, like handling token invalidation and locking during authentication.

Parameters

- **auth_url** (*string*) Identity v1 endpoint for authorization.
- **username** (*string*) Username for authentication.
- **password** (*string*) Password for authentication.
- **project_name** (*string*) Swift account to use after authentication. We use project_name to be consistent with other auth plugins.
- **reauthenticate** (*string*) Whether to allow re-authentication.

access_class

alias of *swiftclient.authv1.AccessInfoV1*

get_access(*session*, ***kwargs*)

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*, *interface='public'*, *region_name=None*, *service_type=None*, ***kwargs*)

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*, *service_type=None*, *interface=None*, *region_name=None*, *service_name=None*, *version=None*, *allow=None*, *allow_version_hack=True*, *skip_discovery=False*, *discover_versions=False*, *min_version=None*, *max_version=None*, ***kwargs*)

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.
- **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`.

get_auth_ref(*session*, ***kwargs*)

Obtain a token from a v1 endpoint.

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** A session object that can be used for communication.

Returns Token access information.

get_auth_state()

Retrieve the current authentication state for the plugin.

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.

get_cache_id()

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()

Get the elements for this auth plugin that make it unique.

get_connection_params(*session*, ***kwargs*)

Return any additional connection parameters required for the plugin.

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_discovery(**args*, ***kwargs*)

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*, *interface='public'*, ***kwargs*)

Return an endpoint for the client.

get_endpoint_data(*session*, *service_type=None*, *interface=None*, *region_name=None*, *service_name=None*, *allow=None*, *allow_version_hack=True*, *discover_versions=True*, *skip_discovery=False*, *min_version=None*, *max_version=None*, *endpoint_override=None*, ***kwargs*)

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 admin-URL‘

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)
- **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)
- **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)
- **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)
- **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)
- **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*, ***kwargs*)

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.

There are no required kwargs. They are passed directly to the auth plugin and they are implementation specific.

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*, ***kwargs*)

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(**args*, ***kwargs*)

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(*args, **kwargs)

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*, **kwargs)

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*, **kwargs)

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

set_auth_state(*data*)

Install existing authentication state for a plugin.

Take the output of get_auth_state and install that authentication state into the current authentication plugin.

3.3 swiftclient.client

OpenStack Swift client library used internally

3.4 swiftclient.service

3.5 swiftclient.exceptions

3.6 swiftclient.multithreading

3.7 swiftclient.utils

Miscellaneous utility functions for use with Swift.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

FIVE

LICENSE

PYTHON MODULE INDEX

S

INDEX

M

P

S

W