CTERA SDK for Python Documentation

CTERA Networks

Mar 09, 2023

CONTENTS:

CHAPTER

ONE

CTERA FOR PYTHON

A Python SDK for integrating with the CTERA Global File System API. Compatible with Python 3.5+.

1.1 Documentation

User documentation is available on Read the Docs.

1.2 Installation

Installing via pip:

```
$ pip install cterasdk
```

If you receive a certificate error, add the following trusted hosts:

```
$ pip install cterasdk --trusted-host pypi.org --trusted-host files.pythonhosted.org  #␣
↪[SSL: CERTIFICATE_VERIFY_FAILED]
```

Installation via proxy:

```
$ pip install cterasdk --proxy http://user:password@proxyserver:port  # use proxy
```

Install from source:

```
$ git clone https://github.com/ctera/ctera-python-sdk.git
$ cd ctera-python-sdk
$ python setup.py install
```

1.3 Importing the Library

After installation, to get started, open a Python console:

```
>>> from cterasdk import *
```

1.4 Building Documentation

Documentation can be compiled by running `make html` from the `docs` folder. After compilation, open `docs/build/html/index.html`.

1.5 Testing

We use the tox package to run tests in Python 3. To install, use `pip install tox`. Once installed, run *tox* from the root directory.

```
$ tox
```

CHAPTER

TWO

USER GUIDES

2.1 Global File System

2.1.1 Global Administration

Table of Contents

2.1.1.1 Instantiate a Global Admin object

class cterasdk.object.Portal.**GlobalAdmin**(*host*, *port=None*, *https=True*)

Main class for Global Admin operations on a Portal

Variables

- **portals** (cterasdk.core.portals.Portals) – Object holding the Portals Management APIs
- **cli** (cterasdk.core.cli.CLI) – Object holding the Portal GlobalAdmin CLI APIs
- **servers** (cterasdk.core.servers.Servers) – Object holding the Servers Management APIs
- **setup** (cterasdk.core.setup.Setup) – Object holding the Portal setup APIs
- **ssl** (cterasdk.core.ssl.SSL) – Object holding the Portal SSL Certificate APIs
- **startup** (cterasdk.core.startup.Startup) – Object holding the Portal startup APIs
- **syslog** (cterasdk.core.syslog.Syslog) – Object holding the Portal syslog APIs
- **antivirus** (cterasdk.core.antivirus.Antivirus) – Object holding the Portal Antivirus APIs
- **buckets** (cterasdk.core.buckets.Buckets) – Object holding the Portal Storage Node APIs
- **messaging** (cterasdk.core.messaging.Messaging) – Object holding the Portal Messaging Service Management APIs

__init__(*host*, *port=None*, *https=True*)

Parameters

- **host** (*str*) – The fully qualified domain name, hostname or an IPv4 address of the Portal
- **port** (*int,optional*) – Set a custom port number (0 - 65535), If not set defaults to 80 for http and 443 for https
- **https** (*bool,optional*) – Set to True to require HTTPS, defaults to True

```
admin = GlobalAdmin('portal.ctera.com') # will use HTTPS over port 443
```

Warning: for any certificate related error, this library will prompt for your consent in order to proceed. to avoid the prompt, you may configure this library to automatically trust the server's certificate, using: `config.http['ssl'] = 'Trust'`

2.1.1.2 Setup

Setup.**init_master**(*name*, *email*, *first_name*, *last_name*, *password*, *domain*)

Initialize the CTERA Portal master server.

Parameters

- **name** (*str*) – User name for the new user
- **email** (*str*) – E-mail address of the new user
- **first_name** (*str*) – The first name of the new user

- **last_name** (*str*) – The last name of the new user
- **password** (*str*) – Password for the new user
- **domain** (*str*) – The domain suffix for CTERA Portal

```
admin.init_master('admin', 'bruce.wayne@we.com', 'Bruce', 'Wayne', 'password1!', 'ctera.
↪me')
```

Setup.**init_application_server**(*ipaddr*, *secret*)

Initialize a CTERA Portal Application Server.

Parameters

- **ipaddr** (*str*) – The CTERA Portal master server IP address
- **secret** (*str*) – A password or a PEM-encoded private key

```
"""Connect a secondary Portal server using a password"""
master_ipaddr = '172.31.53.246'
master_password = 'secret'
admin.init_application_server(master_ipaddr, master_password)

"""Connect a secondary Portal server using a private key"""
master_ipaddr = '172.31.53.246'
master_pk = """...PEM-encoded private key..."""
admin.init_application_server(master_ipaddr, master_pk)
```

Setup.**init_replication_server**(*ipaddr*, *secret*, *replicate_from=None*)

Initialize a CTERA Portal Database Replication Server.

Parameters

- **ipaddr** (*str*) – The CTERA Portal master server IP address
- **secret** (*str*) – A password or a PEM-encoded private key
- **replicate_from** (*str*) – The name of a CTERA Portal server to replicate from

2.1.1.3 Logging in

GlobalAdmin.**test**()

Verification check to ensure the target host is a Portal.

```
admin.test()
```

GlobalAdmin.**login**(*username*, *password*)

Log in

Parameters

- **username** (*str*) – User name to log in
- **password** (*str*) – User password

```
admin.login('admin', 'G3neralZ0d!')
```

GlobalAdmin.**logout**()

Log out

```
admin.logout()
```

GlobalAdmin.**whoami**()

Return the name of the logged in user.

Return cterasdk.common.object.Object
The session object of the current user

```
admin.whoami()
```

2.1.1.4 Navigating

Portals.**browse_global_admin**()

Browse the Global Admin

```
admin.portals.browse_global_admin()
```

Portals.**browse**(*tenant*)

Browse a tenant

Parameters
tenant (*str*) – Name of the tenant to browse

```
admin.portals.browse('portal')
```

2.1.1.5 Core Methods

GlobalAdmin.**show**(*path*, *use_file_url=False*)

Print a schema object as a JSON string.

GlobalAdmin.**show_multi**(*path*, *paths*, *use_file_url=False*)

Print one or more schema objects as a JSON string.

GlobalAdmin.**get**(*path*, *params=None*, *use_file_url=False*)

Retrieve a schema object as a Python object.

GlobalAdmin.**put**(*path*, *value*, *use_file_url=False*)

Update a schema object or attribute.

GlobalAdmin.**execute**(*path*, *name*, *param=None*, *use_file_url=False*)

Execute a schema object method.

GlobalAdmin.**query**(*path*, *name*, *param*)

GlobalAdmin.**show_query**(*path*, *name*, *param*)

2.1.1.6 Storage Classes

StorageClasses.**add**(*name*)

Add a storage class

Parameters
name (*str*) – Name

```
admin.storage_classes.add('Archive')
```

StorageClasses.**all**()

Get storage classes

Returns
List of storage classes

Return type
list(*cterasdk.common.object.Object*)

```
for storage_class in admin.storage_classes.all():
    print(storage_class)
```

StorageClasses.**get**(*name*)

Get storage class

Parameters
name (*str*) – Storage class name, defaults to None

Returns
Storage class

Return type
cterasdk.common.object.Object

```
print(admin.storage_classes.get('Archive'))
```

2.1.1.7 Storage Nodes

Buckets.**get**(*name*, *include=None*)

Get a Bucket

Parameters

- **name** (*str*) – Name of the bucket
- **include** (*list[str]*) – List of fields to retrieve, defaults to ['name']

```
bucket = admin.buckets.get('MainStorage')
print(bucket)

bucket = admin.buckets.get('MainStorage', include=['bucket', 'driver'])
print(bucket.name, bucket.bucket, bucket.driver)
```

Buckets.**add**(*name*, *bucket*, *read_only=False*, *dedicated_to=None*)

Add a Bucket

Parameters

- **name** (*str*) – Name of the bucket
- **bucket** (cterasdk.core.types.Bucket) – Storage bucket to add
- **read_only** (*bool,optional*) – Set bucket to read-delete only, defaults to False
- **dedicated_to** (*str,optional*) – Name of a tenant, defaults to None

```
"""Add an Amazon S3 bucket called 'mybucket'"""
bucket = portal_types.AmazonS3('mybucket', 'access-key', 'secret-key')
admin.buckets.add('cterabucket', bucket)

"""Add an Amazon S3 bucket called 'mybucket', dedicated to a tenant called 'mytenant'"""
bucket = portal_types.AmazonS3('mybucket', 'access-key', 'secret-key')
admin.buckets.add('cterabucket', bucket, dedicated_to='mytenant')

"""Add a bucket in read-delete only mode"""
bucket = portal_types.AmazonS3('mybucket', 'access-key', 'secret-key')
admin.buckets.add('cterabucket', bucket, read_only=True)
```

Buckets.**modify**(*current_name*, *new_name=None*, *read_only=None*, *dedicated_to=None*)

Modify a Bucket

Parameters

- **current_name** (*str*) – The current bucket name
- **new_name** (*str,optional*) – New name
- **read_only** (*bool,optional*) – Set bucket to read-delete only
- **dedicated** (*bool,optional*) – Dedicate bucket to a tenant
- **dedicated_to** (*str,optional*) – Tenant name

```
"""Modify an existing bucket, set it to read-delete only and dedicate it to 'mytenant'"""
admin.buckets.modify('MainStorage', read_only=True, dedicated_to='mytenant')
```

Buckets.**list_buckets**(*include=None*)

List Buckets.

To retrieve buckets, you must first browse the Global Administration Portal, using: *GlobalAdmin.portals.browse_global_admin()*

Parameters

include (*list[str],optional*) – List of fields to retrieve, defaults to ['name']

```
for bucket in admin.buckets.list_buckets():
    print(bucket)
```

Buckets.**delete**(*name*)

Delete a Bucket

Parameters

name (*str*) – Name of the bucket

```
admin.buckets.delete('MainStorage')
```

Buckets.**read_write**(*name*)

Set bucket to Read Write

Parameters

name (*str*) – Name of the bucket

```
admin.buckets.read_write('MainStorage')
```

Buckets.**read_only**(*name*)

Set bucket to Read Only

Parameters

name (*str*) – Name of the bucket

```
admin.buckets.read_only('MainStorage')
```

2.1.1.8 Portals

Retrieve Portals

Portals.**list_tenants**(*include=None*, *portal_type=None*)

List tenants.

To retrieve tenants, you must first browse the Global Administration Portal, using: *GlobalAdmin.portals.browse_global_admin()*

Parameters

- **include** (*list[str],optional*) – List of fields to retrieve, defaults to ['name']
- **portal_type** (cterasdk.core.enum.PortalType) – The Portal type

```
"""List all tenants"""
for tenant in admin.portals.list_tenants():
    print(tenant)

"""List Team Portals. For each tenant, retrieve its creation date, subscription plan and␣
↪activation status"""
for tenant in admin.portals.list_tenants(include=['createDate', 'plan', 'activationStatus
↪'], portal_type=portal_enum.PortalType.Team):
    print(tenant)
```

Portals.**tenants**(*include_deleted=False*)

Get all tenants

Parameters

include_deleted (*bool,optional*) – Include deleted tenants, defaults to False

```
for tenant in admin.portals.tenants():
    print(tenant.name, tenant.usedStorageQuota, tenant.totalStorageQuota)
```

Create a Team Portal

Portals.**add**(*name*, *display_name=None*, *billing_id=None*, *company=None*, *plan=None*, *comment=None*)

Add a new tenant

Parameters

- **name** (*str*) – Name of the new tenant
- **display_name** (*str,optional*) – Display Name of the new tenant, defaults to None
- **billing_id** (*str,optional*) – Billing ID of the new tenant, defaults to None
- **company** (*str,optional*) – Company Name of the new tenant, defaults to None
- **plan** (*str,optional*) – Subscription plan name to assign to the new tenant, defaults to None
- **comment** (*str,optional*) – Assign a comment to the new tenant, defaults to None

Return str

A relative url path to the Team Portal

```
"""Create a Team Portal"""

admin.portals.add('acme')

"""Create a Team Portal, including a display name, billing id and a company name"""

admin.portals.add('ctera', 'CTERA', 'Tz9YRDSd8LNfaouzr3Db', 'CTERA Networks')

"""Create a Team Portal and assign it to a pre-configured subscription plan"""
admin.portals.add('ctera', plan = 'Default')
```

Subscribe a Team Portal to a Plan

Portals.**subscribe**(*tenant*, *plan*)

Subscribe a tenant to a plan

Parameters

- **name** (*str*) – Name of the tenant
- **str,plan** – Name of the subscription plan

```
admin.portals.subscribe('ctera', '10tb')
```

Apply Provisioning Changes

Portals.**apply_changes**(*wait=False*)

Apply provisioning changes.

Parameters

wait (*bool,optional*) – Wait for all changes to apply

```
"""Apply Portal Provisioning Changes"""
admin.portals.apply_changes()
admin.portals.apply_changes(wait=True)  # wait for all changes to apply

"""Apply User Provisioning Changes"""
admin.users.apply_changes()
admin.users.apply_changes(wait=True)  # wait for all changes to apply
```

Delete a Team Portal

Portals.**delete**(*name*)

Delete an existing tenant

Parameters

name (*str*) – Name of the tenant to delete

```
admin.portals.delete_tenant('acme')
```

Recover a Team Portal

Portals.**undelete**(*name*)

Undelete a previously deleted tenant

Parameters

name (*str*) – Name of the tenant to undelete

```
admin.portals.undelete_tenant('acme')
```

2.1.1.9 Plans

Plans.**get**(*name*, *include=None*)

Retrieve subscription plan properties

Parameters

- **name** (*str*) – Name of the subscription plan
- **include** (*list[str]*) – List of fields to retrieve, defaults to ['name']

Returns

The subscription plan, including the requested fields

```
plan = admin.plans.get('good_plan', ['createDate', 'modifiedDate'])
```

Plans.**list_plans**(*include=None*, *filters=None*)

List Plans

Parameters

- **include** (*list[str],optional*) – List of fields to retrieve, defaults to ['name']
- **filters** (*list[],optional*) – List of additional filters, defaults to None

Returns

Iterator for all matching Plans

Return type

cterasdk.lib.iterator.Iterator

```
"""List plans and include their creation date"""
for plan in admin.plans.list_plans(['createDate']):
    print(plan)
```

Plans.**by_name**(*names*, *include=None*)

Get Plans by their names

Parameters

- **names** (*list[str],optional*) – List of names of plans
- **include** (*list[str],optional*) – List of fields to retrieve, defaults to ['name']
- **filters** (*list[cterasdk.core.query.FilterBuilder],optional*) – List of additional filters, defaults to None

Returns

Iterator for all matching Plans

Return type

cterasdk.lib.iterator.Iterator

```
"""List plans 'PlanOne' and 'PlanTwo'; and retrieve the 'modifiedDate', 'uid' and 'isDefault'␣
↪properties"""
for plan in admin.plans.by_name(['PlanOne', 'PlanTwo'], ['modifiedDate', 'uid',
↪'isDefault']):
    print(plan)
```

Plans.**add**(*name*, *retention=None*, *quotas=None*)

Add a subscription plan

Parameters

- **retention** (*dict,optional*) – The data retention policy
- **quotas** (*dict,optional*) – The items included in the plan and their respective quota

```
"""
Retention Policy (portal_enum.PlanRetention):
- All: All Versions
- Hourly: Hourly Versions
- Daily: Daily Versions
- Weekly: Weekly Versions
- Monthly: Monthly Versions
- Quarterly: Quarterly Versions
```

(continues on next page)

(continued from previous page)

```
- Yearly: Yearly Versions
- Deleted: Recycle Bin

Quotas (portal_enum.PlanItem):
- Storage: Storage Quota, in Gigabytes
- EV4: CTERA Edge Filer, Up to 4 TB of Local Cache
- EV8: CTERA Edge Filer, Up to 8 TB of Local Cache
- EV16: CTERA Edge Filer, Up to 16 TB of Local Cache
- EV32: CTERA Edge Filer, Up to 32 TB of Local Cache
- EV64: CTERA Edge Filer, Up to 64 TB of Local Cache
- EV128: CTERA Edge Filer, Up to 128 TB of Local Cache
- WA: Workstation Backup Agent
- SA: Server Agent
- Share: CTERA Drive Share
- Connect: CTERA Drive Connect
"""

"""
Create the 'good_plan' subscription plan:
1) Retention: 7 daily versions, 12 monthly versions
2) Quotas: 10 x EV16, 5 x EV32, 100 x Cloud Drive (Share)
"""

name = 'good_plan'
retention = {portal_enum.PlanRetention.Daily: 7, portal_enum.PlanRetention.Monthly: 12}
quotas = {portal_enum.PlanItem.EV16: 10, portal_enum.PlanItem.EV32: 5, portal_enum.
↪PlanItem.Share: 100}
admin.plans.add(name, retention, quotas)
```

Plans.**modify**(*name*, *retention=None*, *quotas=None*, *apply_changes=True*)

Modify a subscription plan

Parameters

- **retention** (*dict,optional*) – The data retention policy
- **quotas** (*dict,optional*) – The items included in the plan and their respective quota
- **apply_changes** (*bool,optional*) – Apply provisioning changes immediately

```
"""
Modify 'good_plan' subscription plan:
1) Retention: 30 daily versions, 36 monthly versions
2) Quotas: 20 x EV16, 10 x EV32, 200 x Cloud Drive (Share)
"""

name = 'good_plan'
retention = {portal_enum.PlanRetention.Daily: 30, portal_enum.PlanRetention.Monthly: 36}
quotas = {portal_enum.PlanItem.EV16: 20, portal_enum.PlanItem.EV32: 10, portal_enum.
↪PlanItem.Share: 200}
admin.plans.modify(name, retention, quotas)
```

Plans.**delete**(*name*)

Delete a subscription plan

Parameters
username (*str*) – The name of the subscription plan

```
name = 'good_plan'
admin.plan.delete(name)
```

Plan Auto Assignment Rules

PlanAutoAssignPolicy.**get_policy**()
Get plans auto assignment policy

PlanAutoAssignPolicy.**set_policy**(*rules*, *apply_default=None*, *default=None*, *apply_changes=True*)
Set plans auto assignment policy

Parameters

- **rules** (*list[cterasdk.common.types.PolicyRule]*) – List of policy rules
- **apply_default** (*bool,optional*) – If no match found, apply default plan. If not passed, the current config will be kept
- **default** (*str,optional*) – Name of a plan to assign if no match found. Ignored unless the `apply_default` is set to `True`
- **apply_changes** (*bool,optional*) – Apply provisioning changes upon update, defaults to `True`

```
"""Apply the '100GB' plan to all user names that start with 'adm'"""
c1 = portal_types.PlanCriteriaBuilder.username().startswith('adm').build()
r1 = PolicyRule('100GB', c1)

"""Apply the '200GB' plan to all user names that end with 'inc'"""
c2 = portal_types.PlanCriteriaBuilder.username().endswith('inc').build()
r2 = PolicyRule('200GB', c2)

"""Apply the 'Bingo' plan to all user names that contain 'bing'"""
c3 = portal_types.PlanCriteriaBuilder.username().contains('bing').build()
r3 = PolicyRule('Bingo', c3)

"""Apply the 'ABC' plan to 'alice', 'bob' and 'charlie'"""
c4 = portal_types.PlanCriteriaBuilder.username().isoneof(['alice', 'bob', 'charlie']).
↪build()
r4 = PolicyRule('ABC', c4)

"""Apply the '10TB' plan to read write, read only and support administrators"""
roles = [portal_enum.Role.ReadWriteAdmin, portal_enum.Role.ReadOnlyAdmin, portal_enum.
↪Role.Support]
c5 = portal_types.PlanCriteriaBuilder.role().include(roles).build()
r5 = PolicyRule('10TB', c5)

"""Apply the 'TechStaff' plan to the 'Product' and 'Support' groups"""
c6 = portal_types.PlanCriteriaBuilder.user_groups().include(['Product', 'Support']).
↪build()
r6 = PolicyRule('TechStaff', c6)
```

(continues on next page)

(continued from previous page)

```
admin.plans.auto_assign.set_policy([r1, r2, r3, r4, r5, r6])

"""Remove all policy rules"""
admin.plans.auto_assign.set_policy([])

"""Do not assign a default plan if no match applies"""
admin.plans.auto_assign.set_policy([], False)

"""Assign 'Default' if no match applies"""
admin.plans.auto_assign.set_policy([], True, 'Default')
```

2.1.1.10 Configuration Templates

Templates.**get**(*name*, *include=None*)

Get a Configuration Template

Parameters

- **name** (*str*) – Name of the template
- **include** (*list[str]*) – List of fields to retrieve, defaults to `['name']`

```
admin.templates.get('MyTemplate')
```

Templates.**list_templates**(*include=None*, *filters=None*)

List Configuration Templates.

To retrieve templates, you must first browse the tenant, using: *GlobalAdmin.portals.browse()*

Parameters

- **include** (*list[str],optional*) – List of fields to retrieve, defaults to `['name']`
- **filters** (*list[],optional*) – List of additional filters, defaults to None

```
for template in admin.templates.list_templates(include=['name', 'description',
↪'modifiedDate']):
    print(template.name, template.description, template.modifiedDate)
```

Templates.**add**(*name*, *description=None*, *include_sets=None*, *exclude_sets=None*, *apps=None*, *backup_schedule=None*, *versions=None*, *scripts=None*, *cli_commands=None*)

Add a Configuration Template

Parameters

- **name** (*str*) – Name of the template
- **description** (*str,optional*) – Template description
- **include_sets** (*list[cterasdk.common.types.FilterBackupSet],optional*) – List of backup sets to include
- **exclude_sets** (*list[cterasdk.common.types.FilterBackupSet],optional*) – List of backup sets to exclude
- **apps** (*list[cterasdk.common.enum.Application],optional*) – List of applications to back up

- **backup_schedule** (cterasdk.common.types.TaskSchedule,*optional*) – Backup schedule
- **versions** (*list[*cterasdk.core.types.PlatformVersion*],optional*) – List of platforms and their associated versions. Pass *None* to inehrit the default settings from the Global Administration Portal
- **scripts** (*list[*cterasdk.core.types.TemplateScript*],optional*) – Scripts to execute after logon, before or after backup
- **cli_commands** (*list[str],optional*) – Template CLI commands to execute

This library provides several classes, methods and enumerators to assist in creating configuration templates: #. Builder class for filtered backup sets. *cterasdk.common.types.FileFilterBuilder* #. A class representing a backup include or exclude set. *cterasdk.common.types.FilterBackupSet* #. Builder class for defining backup schedule. *cterasdk.common.types.BackupScheduleBuilder* #. A time-range class, used to configure backups to run at a specific time. *cterasdk.common.types.TimeRange* #. Enumerator containing applications supported for backup. cterasdk.common.enum.Application #. A named tuple defining a platform and a software version. *cterasdk.core.types.PlatformVersion* #. Enumerator containing a list of platforms. *cterasdk.core.enum.Platform*

```
"""Include all 'pptx', 'xlsx' and 'docx' file types for all users"""
docs = common_types.FileFilterBuilder.extensions().include(['pptx', 'xlsx', 'docx']).
↪build()
include_sets = common_types.FilterBackupSet('Documents', filter_rules=[docs],
                                                    template_dirs=[portal_enum.
↪EnvironmentVariables.ALLUSERSPROFILE])

"""Exclude all 'cmd', 'exe' and 'bat' file types for all users"""
programs = common_types.FileFilterBuilder.extensions().include(['cmd', 'exe', 'bat']).
↪build()
exclude_sets = common_types.FilterBackupSet('Programs', filter_rules=[programs],
                                                     template_dirs=[portal_enum.
↪EnvironmentVariables.ALLUSERSPROFILE])

"""Schedule backup to run periodically"""
backup_schedule = common_types.BackupScheduleBuilder.interval(hours=6)  # periodically,␣
↪every 6 hours
backup_schedule = common_types.BackupScheduleBuilder.interval(hours=0, minutes=30)  #␣
↪periodically, every 30 minutes

"""Schedule backup for a specific time"""
time_range = common_types.TimeRange().start('07:00:00').days(common_enum.DayOfWeek.
↪Weekdays).build()  # 7am, on weekdays
backup_schedule = common_types.BackupScheduleBuilder.window(time_range)

"""Backup applications"""
apps = [common_enum.Application.NTDS, common_enum.Application.HyperV]  # back up Active␣
↪Directory and Hyper-V
apps = common_enum.Application.All  # back up all applications

"""Configure software versions"""
versions = [portal_types.PlatformVersion(portal_enum.Platform.Edge_7, '7.0.981.7')]  #␣
↪use 7.0.981.7 for v7 Edge Filers
```

(continues on next page)

(continued from previous page)

```
"""Configure Scripts"""
scripts = [
    portal_types.TemplateScript.windows().after_logon('echo Current directory: %cd%'),
    portal_types.TemplateScript.linux().before_backup('./mysqldump -u admin website > /
↪mnt/backup/backup.sql'),
    portal_types.TemplateScript.linux().after_backup('rm /mnt/backup/backup.sql')
]

"""Configure CLI Commands"""
cli_commands = [
    'set /config/agent/stubs/deleteFilesOfCachedFolderOnDisable false',
    'add /config/agent/stubs/allowedExplorerExtensions url'
]

admin.templates.add('MyTemplate', 'woohoo', include_sets=[include_sets], exclude_
↪sets=[exclude_sets],
                    backup_schedule=backup_schedule, apps=apps, versions=versions,␣
↪scripts=scripts, cli_commands=cli_commands)
```

Templates.**set_default**(*name*, *wait=False*)

Set a Configuration Template as the default template

Parameters

- **name** (*str*) – Name of the template
- **wait** (*bool, optional*) – Wait for all changes to apply, defaults to *False*

```
admin.templates.set_default('MyTemplate')

admin.templates.set_default('MyTemplate', wait=True)  # wait for template changes to␣
↪apply
```

Templates.**remove_default**(*name*, *wait=False*)

Set a Configuration Template not to be the default template

Parameters

- **name** (*str*) – Name of the template
- **wait** (*bool, optional*) – Wait for all changes to apply, defaults to *False*

```
admin.templates.remove_default('MyTemplate')

admin.templates.remove_default('MyTemplate', wait=True)  # wait for template changes to␣
↪apply
```

TemplateAutoAssignPolicy.**apply_changes**(*wait=False*)

Apply provisioning changes.

Parameters

wait (*bool, optional*) – Wait for all changes to apply, defaults to *False*

```
admin.templates.auto_assign.apply_changes()
```

(continues on next page)

(continued from previous page)

```
admin.templates.auto_assign.apply_changes(wait=True)  # wait for template changes to␣
↪apply
```

Template Auto Assignment Rules

TemplateAutoAssignPolicy.**get_policy**()

Get templates auto assignment policy

TemplateAutoAssignPolicy.**set_policy**(*rules*, *apply_default=None*, *default=None*, *apply_changes=True*)

Set templates auto assignment policy

Parameters

- **rules** (*list[cterasdk.common.types.PolicyRule]*) – List of policy rules
- **apply_default** (*bool,optional*) – If no match found, apply default template. If not passed, the current config will be kept
- **default** (*str,optional*) – Name of a template to assign if no match found. Ignored unless the apply_default is set to True
- **apply_changes** (*bool,optional*) – Apply changes upon update, defaults to True

```
"""Apply the 'ESeries' template to devices of type: C200, C400, C800, C800P"""
device_types = [portal_enum.DeviceType.C200, portal_enum.DeviceType.C400, portal_enum.
↪DeviceType.C800, portal_enum.DeviceType.C800P]
c1 = portal_types.TemplateCriteriaBuilder.type().include(device_types).build()
r1 = PolicyRule('ESeries', c1)

"""Apply the 'Windows' template to devices that use a 'Windows' operating system"""
c2 = portal_types.TemplateCriteriaBuilder.os().contains('Windows').build()
r2 = PolicyRule('Windows', c2)

"""Apply the 'CTERA7' template to devices running version 7"""
c3 = portal_types.TemplateCriteriaBuilder.version().startswith('7.0').build()
r3 = PolicyRule('CTERA7', c3)

"""Apply the 'WD5' template to devices that their hostname ends with 'WD5'"""
c4 = portal_types.TemplateCriteriaBuilder.hostname().endswith('WD5').build()
r4 = PolicyRule('WD5', c4)

"""Apply the 'Beta' template to devices that their name is one of"""
c5 = portal_types.TemplateCriteriaBuilder.name().isoneof(['DEV1', 'DEV2', 'DEV3']).
↪build()
r5 = PolicyRule('Beta', c5)

admin.templates.auto_assign.set_policy([r1, r2, r3, r4, r5])

"""Remove all policy rules"""
admin.templates.auto_assign.set_policy([])

"""Do not assign a default template if no match applies"""
admin.templates.auto_assign.set_policy([], False)
```

(continues on next page)

(continued from previous page)

```
"""Assign 'Default' if no match applies"""
admin.templates.auto_assign.set_policy([], True, 'Default')
```

2.1.1.11 Servers

Servers.**get**(*name*, *include=None*)

Retrieve server properties

Parameters

- **name** (*str*) – Name of the server
- **include** (*list[str]*) – List of fields to retrieve, defaults to ['name']

Returns

The server, including the requested fields

```
"""Retrieve a server"""

server = admin.servers.get('server', ['isApplicationServer', 'renderingServer'])
print(server.isApplicationServer, server.renderingServer)
```

Servers.**list_servers**(*include=None*)

Retrieve the servers that comprise CTERA Portal.

To retrieve servers, you must first browse the Global Administration Portal, using: *GlobalAdmin.portals.browse_global_admin()*

Parameters

include (*list[str],optional*) – List of fields to retrieve, defaults to ['name']

```
"""Retrieve all servers"""
servers = admin.servers.list_servers() # will only retrieve the server name
for server in servers:
    print(server.name)

"""Retrieve multiple server attributes"""
servers = admin.servers.list_servers(include = ['name', 'connected', 'isApplicationServer
↪', 'mainDB'])
for server in servers:
    print(server)
```

Servers.**modify**(*name*, *server_name=None*, *app=None*, *preview=None*, *enable_public_ip=None*, *public_ip=None*, *allow_user_login=None*, *enable_replication=None*, *replica_of=None*)

Modify a Portal server

Parameters

- **name** (*str*) – The current server name
- **server_name** (*str,optional*) – New server name
- **app** (*bool,optional*) – Application server
- **preview** (*bool,optional*) – Preview server

- **enable_public_ip** (*bool,optional*) – Enable or disable public NAT address
- **public_ip** (*str,optional*) – Public NAT address
- **allow_user_login** (*bool,optional*) – Allow or disallow logins to this server
- **enable_replication** (*bool,optional*) – Enable or disable database replication
- **replica_of** (*str,optional*) – Configure as a replicate of another Portal server. *enable_replication* must be set to *True*

```
admin.servers.modify('server2', server_name='replica', app=False, enable_
↪replication=True, replica_of='maindb')  # rename and enable database replication

admin.servers.modify('server2', allow_user_login=False)  # disable logins to this server

admin.servers.modify('server2', enable_public_ip=True, public_ip='33.191.55.2')  #␣
↪configure a public NAT ip address
```

Server Tasks

Tasks.**background**(*name*)

Get all background tasks

Parameters
name (*str*) – Name of the server

Returns
List of tasks

```
for task in admin.servers.tasks.background('database'):
    print(task.name)
```

Tasks.**scheduled**(*name*)

Get all scheduled tasks

Parameters
name (*str*) – Name of the server

Returns
List of tasks

```
for task in admin.servers.tasks.scheduled('database'):
   print(task.name)
```

2.1.1.12 Messaging Service

Messaging.**get_status**()

Retrieve the global status of messaging service

```
"""Retrieve the global status of Messaging service"""

print(admin.messaging.get_status())
```

Messaging.**get_servers_status**()

Retrieve the status of the messaging servers

```
"""Retrieve the status of the Messaging servers"""

print(admin.messaging.get_servers_status())
```

Messaging.**add**(*servers*)

Add messaging servers to cluster

Parameters

servers (*list[str]*) – Server names (number of allowed servers: 1 or 3)

```
"""Add Messaging servers to cluster"""

servers = ["server1", "server2", "server3"]
admin.messaging.add(servers)
```

2.1.1.13 Antivirus

Antivirus.**list_servers**(*include=None*)

List the antivirus servers

Parameters

include (*list[str],optional*) – List of fields to retrieve, defaults to `['name', 'type']`

Antivirus.**status**()

Get antivirus service status

Antivirus.**rescan**()

Scan all files using the latest antivirus update. This may take a while

Antivirus.**suspend**()

Suspend antivirus scanning

Antivirus.**unsuspend**()

Unsuspend antivirus scanning

Antivirus Servers

AntivirusServers.**get**(*name*)

Get an antivirus server's configuration

Parameters

name (*str*) – Server name

AntivirusServers.**add**(*name*, *vendor*, *url*, *connection_timeout=5*)

Add an antivirus server

Parameters

- **name** (*str*) – Server name
- **vendor** (cterasdk.core.enum.AntivirusType) – Server type

- **url** (*str*) – Server URL (example: `http://your-antivirus.server.local:1234/signature`)
- **connection_timeout** (*int,optional*) – Server connection timeout (in seconds), defaults to 5 seconds

`AntivirusServers.`**delete**(*name*)

Remove an antivirus server

`AntivirusServers.`**suspend**(*name*)

Suspend an antivirus server

`AntivirusServers.`**unsuspend**(*name*)

Unsuspend antivirus scanning

2.1.1.14 Global Administrators

`Administrators.`**list_admins**(*include=None*)

List local administrators

Parameters
include (*list[str]*) – List of fields to retrieve, defaults to ['name']

Returns
Iterator for local administrators

Return type
cterasdk.lib.iterator.Iterator

```
"""list all global admins"""
for admin in admin.admins.list_global_administrators():
    print(admin.name)

for admin in admin.admins.list_global_administrators(include=['name', 'email', 'firstName
↪', 'lastName']):
    print(admin)
```

`Administrators.`**add**(*name*, *email*, *first_name*, *last_name*, *password*, *role*, *company=None*, *comment=None*, *password_change=False*)

Create a Global Administrator

Parameters

- **name** (*str*) – User name for the new GlobalAdmin
- **email** (*str*) – E-mail address of the new GlobalAdmin
- **first_name** (*str*) – The first name of the new GlobalAdmin
- **last_name** (*str*) – The last name of the new GlobalAdmin
- **password** (*str*) – Password for the new GlobalAdmin
- **role** (`cterasdk.core.enum.Role`) – User role of the new GlobalAdmin
- **company** (*str,optional*) – The name of the company of the new GlobalAdmin, defaults to None
- **comment** (*str,optional*) – Additional comment for the new GlobalAdmin, defaults to None

- **password_change** (*variable,optional*) – Require the user to change the password on the first login. Pass datetime.date for a specific date, integer for days from creation, or True for immediate , defaults to False

```
"""Create a global admin"""
admin.admins.add('bruce', 'bruce.wayne@we.com', 'Bruce', 'Wayne', 'G0th4amCity!')
```

Administrators.**modify**(*current_username*, *new_username=None*, *email=None*, *first_name=None*, *last_name=None*, *password=None*, *role=None*, *company=None*, *comment=None*)

Modify a Global Administrator user account

Parameters

- **current_username** (*str*) – The current GlobalAdmin username
- **new_username** (*str,optional*) – New name
- **email** (*str,optional*) – E-mail address
- **first_name** (*str,optional*) – First name
- **last_name** (*str,optional*) – Last name
- **password** (*str,optional*) – Password
- **role** (cterasdk.core.enum.Role,*optional*) – User role
- **company** (*str,optional*) – Company name
- **comment** (*str,optional*) – Comment

```
"""Modify a global admin"""
admin.admins.modify('bruce', 'bwayne@we.com', 'Bruce', 'Wayne', 'Str0ngP@ssword!',
↪'Wayne Enterprises')
```

Administrators.**delete**(*name*)

Delete a Global Administrator

Parameters

username (*str*) – Global administrator username

```
"""Delete a global admin"""
admin.admins.delete('alice')
```

2.1.1.15 Users

Users.**delete**(*user*)

Delete a user

Parameters

user (cterasdk.core.types.UserAccount) – the user account

```
"""Delete a local user"""

alice = portal_types.UserAccount('alice')
admin.users.delete(alice)

"""Delete a domain user"""
```

(continues on next page)

(continued from previous page)

```
bruce = portal_types.UserAccount('bruce', 'domain.ctera.local')
admin.users.delete(bruce)
```

Local Users

Users.**list_local_users**(*include=None*)

List all local users

Parameters
include (*list[str]*) – List of fields to retrieve, defaults to ['name']

Returns
Iterator for all local users

Return type
cterasdk.lib.iterator.Iterator

```
users = admin.users.list_local_users()

for user in users:

    print(user.name)

users = admin.users.list_local_users(include = ['name', 'email', 'firstName', 'lastName
↪'])

for user in users:

    print(user)
```

Users.**add**(*name*, *email*, *first_name*, *last_name*, *password*, *role*, *company=None*, *comment=None*, *password_change=False*)

Create a local user account

Parameters

- **name** (*str*) – User name for the new user
- **email** (*str*) – E-mail address of the new user
- **first_name** (*str*) – The first name of the new user
- **last_name** (*str*) – The last name of the new user
- **password** (*str*) – Password for the new user
- **role** (cterasdk.core.enum.Role) – User role of the new user
- **company** (*str,optional*) – The name of the company of the new user, defaults to None
- **comment** (*str,optional*) – Additional comment for the new user, defaults to None
- **password_change** (*variable,optional*) – Require the user to change the password on the first login. Pass datetime.date for a specific date, integer for days from creation, or True for immediate , defaults to False

```
"""Create a local user"""
admin.users.add('bruce', 'bruce.wayne@we.com', 'Bruce', 'Wayne', 'G0th4amCity!')
```

Users.**modify**(*current_username*, *new_username=None*, *email=None*, *first_name=None*, *last_name=None*, *password=None*, *role=None*, *company=None*, *comment=None*)

Modify a local user account

Parameters

- **current_username** (*str*) – The current user name
- **new_username** (*str,optional*) – New name
- **email** (*str,optional*) – E-mail address
- **first_name** (*str,optional*) – First name
- **last_name** (*str,optional*) – Last name
- **password** (*str,optional*) – Password
- **role** (cterasdk.core.enum.Role,*optional*) – User role
- **company** (*str,optional*) – Company name
- **comment** (*str,optional*) – Comment

```
"""Modify a local user"""
admin.users.modify('bruce', 'bwayne@we.com', 'Bruce', 'Wayne', 'Str0ngP@ssword!', 'Wayne␣
↪Enterprises')
```

Domain Users

Users.**list_domains**()

List all domains

Return list
List of all domains

Users.**list_domain_users**(*domain*, *include=None*)

List all the users in the domain

Parameters
include (*list[str]*) – List of fields to retrieve, defaults to ['name']

Returns
Iterator for all the domain users

Return type
cterasdk.lib.iterator.Iterator

```
users = admin.users.list_domain_users('domain.ctera.local') # will only retrieve the 'name
↪' attribute
for user in users:
    print(user.name)

"""Retrieve additional user attributes"""
users = admin.users.list_domain_users('domain.ctera.local', include = ['name', 'email',
```

(continues on next page)

(continued from previous page)

```
↪'firstName', 'lastName'])
print(user)
```

Fetch Users & Groups

DirectoryService.**fetch**(*active_directory_accounts*)

Instruct the Portal to fetch the provided Active Directory Accounts

Parameters

active_directory_accounts (*list[cterasdk.core.types.PortalAccount]*) – List of Active Directory Accounts to fetch

Returns

Response Code

```
"""Fetch domain users"""

alice = portal_types.UserAccount('alice', 'domain.ctera.local')
bruce = portal_types.UserAccount('bruce', 'domain.ctera.local')

admin.directoryservice.fetch([alice, bruce])
```

2.1.1.16 Directory Services

DirectoryService.**connect**(*domain*, *username*, *password*, *directory='ActiveDirectory'*, *domain_controllers=None*, *ou=None*, *ssl=False*, *krb=False*, *mapping=None*, *acl=None*, *default='Disabled'*, *fetch='Lazy'*)

Connect a Portal tenant to directory services

Parameters

- **domain** (*str*) – The directory services domain to connect to
- **username** (*str*) – The user name to use when connecting to the active directory services
- **password** (*str*) – The password to use when connecting to the active directory services
- **ou** (*str,optional*) – The OU path to use when connecting to the active directory services, defaults to *None*
- **directory** (cterasdk.core.enum.DirectoryServiceType,*optional*) – The directory service type, deafults to *'ActiveDirectory'*
- **domain_controllers** (cterasdk.core.types.DomainControllers,*optional*) – Connect to a primary and a secondary domain controllers, defaults to *None*
- **ssl** (*bool,optional*) – Connect using SSL, defaults to *False*
- **krb** (*bool,optional*) – Connect using Kerberos, defaults to *False*
- **list[cterasdk.common.types.ADDomainIDMapping],optional** – The directory services UID/GID mapping
- **acl** (*list[cterasdk.core.types.AccessControlEntry],optional*) – List of access control entries and their associated roles
- **default** (cterasdk.core.enum.Role) – Default role if no match applies, defaults to *None*

- **fetch** (*str,optional*) – Configure identity fetching mode, defaults to *'Lazy'*

```
"""Connect to Active Directory using a primary domain controller, configure domain UID/
↪GID mapping and access control"""
mapping = [portal_types.ADDomainIDMapping('demo.local', 200001, 5000000), portal_types.
↪ADDomainIDMapping('trusted.local', 5000001, 10000000)]
rw_admin_group = portal_types.AccessControlEntry(
    portal_types.GroupAccount('ctera_admins', 'demo.local'),
    portal_enum.Role.ReadWriteAdmin
)
ro_admin_user = portal_types.AccessControlEntry(
    portal_types.UserAccount('jsmith', 'demo.local'),
    portal_enum.Role.ReadOnlyAdmin
)
admin.directoryservice.connect('demo.local', 'svc_account', 'P@ssw0rd1', mapping=mapping,
↪ domain_controllers=portal_types.DomainControllers('172.54.3.52'), acl=[rw_admin, ro_
↪admin])
```

DirectoryService.**domains**()

Get domains

Return list(str)
List of names of all discovered domains

```
print(admin.directoryservice.domains())
```

DirectoryService.**get_connected_domain**()

Get the connected domain information. Returns *None* if the Portal tenant is not connected to a domain

Return str
The connected domain

```
print(admin.directoryservice.get_connected_domain())
```

DirectoryService.**get_advanced_mapping**()

Retrieve directory services advanced mapping configuration

Returns
A dictionary of domain mapping objects

Return type
dict

```
for domain, mapping in admin.directoryservice.get_advanced_mapping().items():
    print(domain, mapping)
```

DirectoryService.**set_advanced_mapping**(*mapping*)

Configure advanced mapping

Parameters
mapping (*list[cterasdk.common.types.ADDomainIDMapping]*) – List of domains and their UID/GID mapping range

```
"""Configure UID/GID mapping"""
mapping = [portal_types.ADDomainIDMapping('demo.local', 200001, 5000000), portal_types.
```

(continues on next page)

(continued from previous page)

```
↪ADDomainIDMapping('trusted.local', 5000001, 10000000)]
admin.directoryservice.set_advanced_mapping(mapping)
```

DirectoryService.**get_access_control**()

Retrieve directory services access control list

Returns
List of access control entries

Return type
list[*cterasdk.core.types.AccessControlEntry*]

```
for ace in admin.directoryservice.get_access_control():
    print(ace.account, ace.role)
```

DirectoryService.**set_access_control**(*acl=None*, *default=None*)

Configure directory services access control

Parameters

- **acl** (*list[cterasdk.core.types.AccessControlEntry],optional*) – List of access control entries and their associated roles
- **default** (cterasdk.core.enum.Role) – Default role if no match applies, defaults to *None*

```
"""Configure access control for a domain group and a domain user. Set the default role␣
↪to 'Disabled'"""
rw_admin_group = portal_types.AccessControlEntry(
    portal_types.GroupAccount('ctera_admins', 'demo.local'),
    portal_enum.Role.ReadWriteAdmin
)
ro_admin_user = portal_types.AccessControlEntry(
    portal_types.UserAccount('jsmith', 'demo.local'),
    portal_enum.Role.ReadOnlyAdmin
)
admin.directoryservice.set_access_control([rw_admin_group, ro_admin_user], portal_enum.
↪Role.Disabled)
```

DirectoryService.**get_default_role**()

Retrieve the default role assigned when no access control entry match was found

```
print(admin.directoryservice.get_default_role())
```

DirectoryService.**disconnect**()

Disconnect a Portal tenant from directory services

```
admin.directoryservice.disconnect()
```

2.1.1.17 Devices

Devices.**device**(*device_name*, *tenant=None*, *include=None*)

Get a Device by its name

Parameters

- **device_name** (*str*) – Name of the device to retrieve
- **tenant** (*str,optional*) – Tenant of the device, defaults to the tenant in the current session
- **include** (*list[str],optional*) – List of fields to retrieve, defaults to ['name', 'portal', 'deviceType']

Returns

Managed Device

Return type

ctera.object.Gateway.Gateway or ctera.object.Agent.Agent

Devices.**filers**(*include=None*, *allPortals=False*, *deviceTypes=None*)

Get Filers

Parameters

- **include** (*list[str],optional*) – List of fields to retrieve, defaults to ['name', 'portal', 'deviceType']
- **allPortals** (*bool,optional*) – Search in all portals, defaults to False
- **deviceTypes** (*list[cterasdk.core.enum.DeviceType.Gateways]*) – Types of Filers, defaults to all Filer types

Returns

Iterator for all matching Filers

Return type

cterasdk.lib.iterator.Iterator[*cterasdk.object.Gateway.Gateway*]

```
"""Retrieve all Gateways from the current tenant"""

filers = admin.devices.filers()

for filer in filers:

    print(filer.name) # will print the Gateway name

"""Retrieve additional Gateway attributes"""

filers = admin.devices.filers(['owner', 'deviceConnectionStatus'])

"""Retrieve nested attributes using the '.' delimiter"""

filers = admin.devices.filers(['deviceReportedStatus.status.device.runningFirmware'])

"""Retrieve filers from all portals"""

admin.portals.browse_global_admin()
```

(continues on next page)

(continued from previous page)

```
filers = admin.devices.filers(allPortals = True)

"""Retrieve C200's and C400's from all portals"""

admin.portals.browse_global_admin()

filers = admin.devices.filers(allPortals = True, deviceTypes = ['C200', 'C400'])
```

Devices.**agents**(*include=None*, *allPortals=False*)

Get Agents

Parameters

- **include** (*list[str],optional*) – List of fields to retrieve, defaults to ['name', 'portal', 'deviceType']
- **allPortals** (*bool,optional*) – Search in all portals, defaults to False

Returns

Iterator for all matching Agents

Return type

cterasdk.lib.iterator.Iterator[*cterasdk.object.Agent.Agent*]

```
"""Retrieve all Agents from the current tenant"""

agents = admin.devices.agents()

for agent in agents:

    print(agent.name) # will print the Agent name

"""Retrieve all Agents and the underlying OS name"""

agents = admin.devices.agents(['deviceReportedStatus.status.agent.details.osName'])
```

Devices.**servers**(*include=None*, *allPortals=False*)

Get Servers

Parameters

- **include** (*list[str],optional*) – List of fields to retrieve, defaults to ['name', 'portal', 'deviceType']
- **allPortals** (*bool,optional*) – Search in all portals, defaults to False

Returns

Iterator for all matching Servers

Return type

cterasdk.lib.iterator.Iterator

```
server_agents = admin.devices.server()
```

Devices.**desktops**(*include=None*, *allPortals=False*)

Get Desktops

Parameters

- **include** (*list[str],optional*) – List of fields to retrieve, defaults to ['name', 'portal', 'deviceType']
- **allPortals** (*bool,optional*) – Search in all portals, defaults to False

Returns
Iterator for all matching Desktops

Return type
cterasdk.lib.iterator.Iterator

```
desktop_agents = admin.devices.desktop_agents()
```

Devices.**by_name**(*names*, *include=None*)

Get Devices by their names

Parameters

- **names** (*list[str],optional*) – List of names of devices
- **include** (*list[str],optional*) – List of fields to retrieve, defaults to ['name', 'portal', 'deviceType']

Returns
Iterator for all matching Devices

Return type
cterasdk.lib.iterator.Iterator

Devices.**get_comment**(*device_name*, *tenant=None*)

Get Portal device comment

Parameters
device (*str*) – Device name

Returns
Comment

Return type
str

```
print(admin.devices.get_comment('FSRV'))
```

Devices.**set_comment**(*device_name*, *comment*, *tenant=None*)

Set a comment to a Portal device

Parameters

- **device** (*str*) – Device name
- **comment** (*str*) – Comment

```
admin.devices.set_comment('FSRV', 'Production')
```

Generate Activation Codes

Activation.**generate_code**(*username=None*, *tenant=None*)

Generate device activation code

Parameters

- **username** (*str,optional*) – User name used for activation, defaults to None
- **tenant** (*str,optional*) – Tenant name used for activation, defaults to None

Returns

Portal Activation Code

Return type

str

```
"""Generate a device activation code"""

code = admin.activation.generate_code() # will generate a code for the current, logged␣
↪on, user

code = admin.activation.generate_code('bruce') # will generate a code for 'bruce' in the␣
↪current tenant

code = admin.activation.generate_code('batman', 'gotham') # will generate a code for
↪'bruce' in the 'gotham' tenant
```

Note: Read Write Administrator, granted with the "Super User" role permission, can generate 200 codes every 5 minutes

Code Snippets

Generate activation codes for all domain users

```
# ... login ...

users = admin.users.list_domain_users('dc.ctera.local') # obtain a list of domain users

for user in users:

    activation_code = admin.activation.generate_code(user.name) # generate activation␣
↪code

    print((user.name, activation_code))

# ... logout ...
```

2.1.1.18 Zones

To manage zones, you must be a Read Write Administrator

Retrieve a Zone

Zones.**get**(*name*)

Get zone by name

Parameters
name (*str*) – The name of the zone to get

Returns
The requested zone

```
zone = admin.zones.get('ZN-001')
```

List & Search Zones

Zones.**list_zones**(*filters=None*)

List Zones :param list[],optional filters: List of additional filters, defaults to None

Returns
Iterator for all Zones

Return type
cterasdk.lib.iterator.Iterator

```
for zone in admin.zones.list_zones():
    print(zone)
```

Zones.**search**(*name*)

Search for Zones by name :param str name: Search query

Returns
Iterator for all matching Zones

Return type
cterasdk.lib.iterator.Iterator

```
for zone in admin.zones.search('ZN'):
    print(zone)
```

Create a Zone

Zones.**add**(*name*, *policy_type='selectedFolders'*, *description=None*)

Add a new zone

Parameters

- **name** (*str*) – The name of the new zone
- **policy_type** (cterasdk.core.enum.PolicyType,*optional*) – Policy type of the new zone, defaults to cterasdk.core.enum.PolicyType.SELECT

- **description** (*str, optional*) – The description of the new zone

```
"""
Policy Types:
- All: Include all cloud folders
- Select: Select one or more cloud folders to include
- None: Create an empty zone
"""

"""Create a zone with a description"""

admin.zones.add('ZN-NYS-001', description = 'The New York State Zone')

"""Create a zone and include all folders"""

admin.zones.add('ZN-NYS-002', 'All', 'All Folders')

"""Create an empty zone"""

admin.zones.add('ZN-NYS-003', 'None', 'Empty Zone')
```

Add Folders to a Zone

Zones.**add_folders**(*name*, *folder_finding_helpers*)

Add the folders to the zone

Parameters

- **name** (*str*) – The name of the zone
- **folder_finding_helpers** (*list[*cterasdk.core.types.CloudFSFolderFindingHelper*]*) – List of folder names and owners

```
"""
Add the following cloud folders to zone: 'ZN-001'

1) 'Accounting' folder owned by 'Bruce'
2) 'HR' folder owned by 'Diana'
"""

accounting = portal_types.CloudFSFolderFindingHelper('Accounting', 'Bruce')
hr = portal_types.CloudFSFolderFindingHelper('HR', 'Diana')

admin.zones.add_folders('ZN-001', [accounting, hr])
```

Add Devices to a Zone

Zones.**add_devices**(*name*, *device_names*)

Add devices to a zone

Parameters

- **name** (*str*) – The name of the zone to add devices to
- **device_names** (*list[str]*) – The names of the devices to add to the zone

```
admin.zones.add_devices('ZN-001', ['vGateway-01ba', 'vGateway-bd02'])
```

Delete a Zone

Zones.**delete**(*name*)

Delete a zone

Parameters

name (*str*) – The name of the zone to delete

```
admin.zones.delete('ZN-001')
```

2.1.1.19 CloudFS

To manage the Cloud File System, folder groups, backup and cloud drive folders, you must be a Read Write Administrator

Folder Groups

CloudFS.**list_folder_groups**(*include=None*, *user=None*)

List folder groups

Parameters

- **include** (*str,optional*) – List of fields to retrieve, defaults to ['name', 'owner']
- **user** (cterasdk.core.types.UserAccount) – User account of the folder group owner

Returns

Iterator for all folder groups

```
"""List all folder groups"""
folder_groups = admin.cloudfs.list_folder_groups()
for folder_group in folder_groups:
    print(folder_group.name, folder_group.owner)

"""List folder groups owned by a domain user"""
bruce = portal_types.UserAccount('bruce', 'domain.ctera.local')
folder_groups = admin.cloudfs.list_folder_groups(user=bruce)
```

CloudFS.**mkfg**(*name*, *user=None*, *deduplication_method_type=None*, *storage_class=None*)

Create a new Folder Group

Parameters

- **name** (*str*) – Name of the new folder group
- **user** (cterasdk.core.types.UserAccount) – User account, the user directory and name of the new folder group owner (default to None)
- **deduplication_method_type** (cterasdk.core.enum.DeduplicationMethodType) – Deduplication-Method
- **storage_class** (*str,optional*) – Storage class, defaults to the Default storage class

```
"""Create a Folder Group, owned by a local user account 'svc_account'"""
svc_account = portal_types.UserAccount('svc_account')
admin.cloudfs.mkfg('FG-001', svc_account)

"""Create a Folder Group, owned by the domain user 'ctera.local\wbruce'"""
wbruce = portal_types.UserAccount('wbruce', 'ctera.local')
admin.cloudfs.mkfg('FG-002', wbruce)

admin.cloudfs.mkfg('FG-003') # without an owner

"""Create a Folder Group, assigned to an 'Archive' storage class"""
admin.cloudfs.mkfg('Archive', portal_types.UserAccount('svc_account'), storage_class=
↪'Archive')
```

CloudFS.**rmfg**(*name*)

Remove a Folder Group

Parameters

name (*str*) – Name of the folder group to remove

```
admin.cloudfs.rmfg('FG-001')
```

Cloud Drive Folders

CloudFS.**list_folders**(*include=None*, *list_filter='NonDeleted'*, *user=None*)

List Cloud Drive folders.

Parameters

- **include** (*str,optional*) – List of fields to retrieve, defaults to ['name', 'group', 'owner']
- **list_filter** (cterasdk.core.enum.ListFilter) – Filter the list of Cloud Drive folders, defaults to non-deleted folders
- **user** (cterasdk.core.types.UserAccount) – User account of the cloud folder owner

Returns

Iterator for all Cloud Drive folders

```
"""List all cloud drive folders"""
cloud_drive_folders = admin.cloudfs.list_folders()
for cloud_drive_folder in cloud_drive_folders:
```

(continues on next page)

(continued from previous page)

```
    print(cloud_drive_folder)

"""List cloud drive folders owned by a domain user"""
bruce = portal_types.UserAccount('bruce', 'domain.ctera.local')
cloud_drive_folders = admin.cloudfs.list_folders(user=bruce)

"""List both deleted and non-deleted cloud drive folders"""
cloud_drive_folders = admin.cloudfs.list_folders(list_filter=portal_enum.ListFilter.All)

"""List deleted cloud drive folders"""
cloud_drive_folders = admin.cloudfs.list_folders(list_filter=portal_enum.ListFilter.
↪Deleted)
```

CloudFS.**mkdir**(*name*, *group*, *owner*, *winacls=True*, *description=None*)

Create a new directory

Parameters

- **name** (*str*) – Name of the new directory
- **group** (*str*) – The Folder Group to which the directory belongs
- **owner** (cterasdk.core.types.UserAccount) – User account, the owner of the new directory
- **winacls** (*bool,optional*) – Use Windows ACLs, defaults to True
- **description** (*str,optional*) – Cloud drive folder description

```
"""Create a Cloud Drive folder, owned by a local user account 'svc_account'"""
svc_account = portal_types.UserAccount('svc_account')
admin.cloudfs.mkdir('DIR-001', 'FG-001', svc_account)
admin.cloudfs.mkdir('DIR-003', 'FG-003', svc_account, winacls = False) # disable Windows␣
↪ACL's

"""Create a Cloud Drive folder, owned by the domain user 'ctera.local\wbruce'"""
wbruce = portal_types.UserAccount('wbruce', 'ctera.local')
admin.cloudfs.mkdir('DIR-002', 'FG-002', wbruce)
```

CloudFS.**delete**(*name*, *owner*)

Delete a Cloud Drive Folder

Parameters

- **name** (*str*) – Name of the Cloud Drive Folder to delete
- **owner** (cterasdk.core.types.UserAccount) – User account, the owner of the Cloud Drive Folder to delete

```
"""Delete a Cloud Drive folder, owned by the local user account 'svc_account'"""
svc_account = portal_types.UserAccount('svc_account')
admin.cloudfs.delete('DIR-001', svc_account)

"""Delete a Cloud Drive folder, owned by the domain user 'ctera.local\wbruce'"""
wbruce = portal_types.UserAccount('wbruce', 'ctera.local')
admin.cloudfs.delete('DIR-002', wbruce)
```

CloudFS.**undelete**(*name*, *owner*)

Un-Delete a Cloud Drive Folder

Parameters

- **name** (*str*) – Name of the Cloud Drive Folder to un-delete
- **owner** (cterasdk.core.types.UserAccount) – User account, the owner of the Cloud Drive Folder to delete

```
"""Recover a deleted Cloud Drive folder, owned by the local user account 'svc_account'"""
svc_account = portal_types.UserAccount('svc_account')
admin.cloudfs.undelete('DIR-001', svc_account)

"""Recover a deleted Cloud Drive folder, owned by the domain user 'ctera.local\wbruce'"""
wbruce = portal_types.UserAccount('wbruce', 'ctera.local')
admin.cloudfs.undelete('DIR-002', wbruce)
```

NT ACLs

CloudFS.**set_folders_acl**(*folders_path*, *sddl_string*, *is_recursive=False*)

Changing the file or Folder ACLs

Parameters

- **folders_path** (*list*) – A list of paths
- **sddl_string** (*str*) – The SDDL string with the ACL permissions
- **is_recursive** (*bool*) – For path that is not a file but a folder

Returns

execution response

```
"""Changing the file or Folder ACLs"""
folders_paths = ["portaladmin/cloudFolder/diagrams", "adrian/data/docs"]
sddl_string = 'O:S-1-12-1-1536910496-1126310805-1188065941-1612002142' \
              'G:S-1-12-1-1536910496-1126310805-1188065941-1612002142' \
              'D:AI(A;ID;FA;;;BA)(A;ID;FA;;;SY)(A;ID;0x1200a9;;;BU)(A;ID;0x1301bf;;;AU)'
admin.cloudfs.set_folders_acl(folders_paths, sddl_string, True)
```

CloudFS.**set_owner_acl**(*folders_path*, *owner_sid*, *is_recursive=False*)

Changing the File or Folder Owner SID or ACLs

Parameters

- **folders_path** (*list*) – A list of paths
- **owner_sid** (*str*) – The SID string that identifies the object's owner.
- **is_recursive** (*bool*) – If the path is not a file but a folder

Returns

execution response

```
"""Changing the File or Folder Owner SID or ACLs"""
folders_paths = ["portaladmin/cloudFolder/diagrams", "dorian/data/docs"]
owner_sid = 'S-1-12-1-1536910496-1126310805-1188065941-1612002142'
admin.cloudfs.set_folders_acl(folders_paths, owner_sid, True)
```

2.1.1.20 Timezone

GlobalSettings.**get_timezone**()

Get timezone

```
admin.settings.global_settings.get_timezone()
```

GlobalSettings.**set_timezone**(*timezone*)

Set timezone

Parameters
timezone (*str*) – Timezone

```
admin.settings.global_settings.set_timzeone('(GMT-05:00) Eastern Time (US , Canada)')
```

2.1.1.21 SSL Certificate

SSL.**get**()

Retrieve details of the current installed SSL certificate

Return cterasdk.common.object.Object
An object including the SSL certificate details

```
certificate = admin.ssl.get()
print(certificate)
```

SSL.**thumbprint**()

Get the SHA1 thumbprint of the Portal SSL certificate

```
print(admin.ssl.thumbprint)
```

SSL.**export**(*destination=None*)

Export the Portal SSL Certificate to a ZIP archive

Parameters
destination (*str,optional*) – File destination, defaults to the default directory

```
admin.ssl.export()

admin.ssl.export(r'C:\Temp')  # export to an alternate location
```

SSL.**import_from_zip**(*zipfile*)

Import an SSL Certificate to CTERA Portal from a ZIP archive

Parameters
zipfile (*str*) – A zip archive including the private key and SSL certificate chain

```
admin.ssl.import_from_zip(r'C:\Users\jsmith\Downloads\certificate.zip')
```

SSL.**import_from_chain**(*private_key*, **certificates*)

Import an SSL Certificate to CTERA Portal from a chain

Parameters

- **private_key** (*str*) – The PEM-encoded private key, or a path to the PEM-encoded private key file

- **certificates** (*list[str]*) – The PEM-encoded certificates, or a list of paths of the PEM-encoded certificate files

```
admin.ssl.import_from_chain(
    r'C:\Users\jsmith\Downloads\private.key',
    r'C:\Users\jsmith\Downloads\domain.crt',
    r'C:\Users\jsmith\Downloads\intermediate.crt',
    r'C:\Users\jsmith\Downloads\root.crt'
)
```

2.1.1.22 Logs

Logs.**get**(*topic='system'*, *min_severity='info'*, *origin_type='Portal'*, *origin=None*, *before=None*, *after=None*, *filters=None*)

Get logs from the Portal

Parameters

- **topic** (cterasdk.core.enum.LogTopic,*optional*) – Log topic to get, defaults to cterasdk.core.enum.LogTopic.System
- **min_severity** (cterasdk.core.enum.Severity,*optional*) – Minimun severity of logs to get, defaults to cterasdk.core.enum.Severity.INFO
- **origin_type** (cterasdk.core.enum.OriginType,*optional*) – Origin type of the logs to get, defaults to cterasdk.core.enum.OriginType.Portal
- **origin** (*str,optional*) – Log origin (e.g. device name, Portal server name), defaults to None
- **before** (*str,optional*) – Get logs before this date (in format "%m/%d/%Y %H:%M:%S"), defaults to None
- **after** (*str,optional*) – Get logs after this date (in format "%m/%d/%Y %H:%M:%S"), defaults to None
- **filters** (*list[cterasdk.core.query.FilterBuilder],optional*) – List of additional filters, defaults to None

Returns

Iterator for all matching logs

Return type

cterasdk.lib.iterator.Iterator[cterasdk.object.Object]

Logs.**device**(*name*, *topic='system'*, *min_severity='info'*, *before=None*, *after=None*, *filters=None*)

Get device logs from the Portal

Parameters

- **name** (*str*) – Name of a device
- **topic** (cterasdk.core.enum.LogTopic,*optional*) – Log topic to get, defaults to cterasdk.core.enum.LogTopic.System
- **min_severity** (cterasdk.core.enum.Severity,*optional*) – Minimun severity of logs to get, defaults to cterasdk.core.enum.Severity.INFO
- **before** (*str,optional*) – Get logs before this date (in format "%m/%d/%Y %H:%M:%S"), defaults to None

- **after** (*str,optional*) – Get logs after this date (in format "%m/%d/%Y %H:%M:%S"), defaults to None
- **filters** (*list[cterasdk.core.query.FilterBuilder],optional*) – List of additional filters, defaults to None

Returns
Iterator for all matching logs

Return type
cterasdk.lib.iterator.Iterator[cterasdk.object.Object]

```
"""Retrieve all cloud backup logs for device 'WIN-SRV2019'"""
admin.logs.device('WIN-SRV2019', topic='backup')
```

Log Based Alerts

Alerts.**get**()

Get a List of Log Based Alerts

Returns
A list of alerts

Return type
list[*cterasdk.common.object.Object*]

```
"""Get a list of log based alerts"""
for alert in admin.logs.alerts.get():
   print(alert)
```

Alerts.**add**(*name*, *description=None*, *topic=None*, *log=None*, *min_severity=None*, *origin_type=None*, *content=None*)

Add a Log Based Alert

Parameters

- **name** (*str*) – Alert name
- **description** (*str,optional*) – Alert description
- **topic** (*cterasdk.core.enum.LogTopic,optional*) – Log topic to get, defaults to any topic
- **log** (*str,optional*) – Class name of the log
- **min_severity** (*cterasdk.core.enum.Severity,optional*) – Minimun severity for triggering an alert, defaults to any severity
- **origin_type** (*cterasdk.core.enum.OriginType,optional*) – Origin type of the log, defaults to any origin
- **content** (*str*) – Content of the log message

Returns
A list of alerts

Return type
list[*cterasdk.common.object.Object*]

```
"""Alert on a volume full error event"""
admin.logs.alerts.add('Volume Full', topic='system', log='VolumeFull', origin_type=
↪'Device', min_severity='error')
```

Alerts.**put**(*alerts*)

Set Log Based Alerts

Use *cterasdk.core.types.AlertBuilder()* to build log based alerts`

Parameters

alerts (*list[cterasdk.core.types.Alert]*) – List of alerts

```
"""Set alerts. Overrides all existing alerts"""
volume_full = portal_types.AlertBuilder.name('volume_full').log('VolumeFull').build()
agent_repo = portal_types.AlertBuilder.name('agent_repo').log('AgentRepositoryNotReady').
↪build()
admin.logs.alerts.put([volume_full, agent_repo])
```

Alerts.**delete**(*name*)

Remove a Log Based Alert

Parameters

name (*str*) – Alert name

```
"""Delete an alert by name"""
admin.logs.alerts.delete('volume_full')

"""Delete all alerts"""
admin.logs.alerts.put([])
```

2.1.1.23 Syslog

Syslog.**is_enabled**()

Check if forwarding log messages over syslog is enabled

Syslog.**get_configuration**()

Retrieve the syslog server configuration

Syslog.**enable**(*server*, *port=514*, *protocol='UDP'*, *min_severity='info'*)

Enable Syslog

Parameters

- **server** (*str*) – Syslog server address
- **port** (*int*,*optional*) – Syslog server port
- **protocol** (cterasdk.core.enum.IPProtocol,*optional*) – Syslog server IP protocol
- **min_severity** (cterasdk.core.enum.Severity,*optional*) – Minimum log severity to forward

Syslog.**disable**()

2.1.1.24 CLI Execution

CLI.**run_command**(*cli_command*)

Run a CLI command

Parameters
cli_command (*str*) – The CLI command to run on the gateway

Return str
The response of the Portal

```
result = admin.cli.run_command('show /settings')
print(result)
```

2.1.2 End User Portal

Table of Contents

2.1.2.1 Instantiate a Services Portal object

class cterasdk.object.Portal.**ServicesPortal**(*host*, *port=None*, *https=True*)

Main class for Service operations on a Portal

__init__(*host*, *port=None*, *https=True*)

Parameters

- **host** (*str*) – The fully qualified domain name, hostname or an IPv4 address of the Portal
- **port** (*int,optional*) – Set a custom port number (0 - 65535), If not set defaults to 80 for http and 443 for https
- **https** (*bool,optional*) – Set to True to require HTTPS, defaults to True

```
user = ServicesPortal('portal.ctera.com') # will use HTTPS over port 443
```

Warning: for any certificate related error, this library will prompt for your consent in order to proceed. to avoid the prompt, you may configure *chopin-core* to automatically trust the server's certificate, using: `config.http['ssl'] = 'Trust'`

Logging in

ServicesPortal.**test**()

Verification check to ensure the target host is a Portal.

```
user.test()
```

ServicesPortal.**login**(*username*, *password*)

Log in

Parameters

- **username** (*str*) – User name to log in
- **password** (*str*) – User password

```
user.login('walice', 'G3neralZ0d!')
```

ServicesPortal.**logout**()

Log out

```
user.logout()
```

2.1.3 File Browser

Table of Contents

2.1.3.1 File Browser

List

FileBrowser.**ls**(*path*, *include_deleted=False*)

Execute ls on the provided path

Parameters

- **path** (*str*) – Path to list
- **include_deleted** (*bool,optional*) – Include deleted files, defaults to False

```
file_browser.ls('')  # List the contents of the Cloud Drive

file_browser.ls('My Files')  # List the contents of the 'My Files' folder

file_browser.ls('My Files', True)  # Include deleted files
```

FileBrowser.**walk**(*path*, *include_deleted=False*)

Perform walk on the provided path

Parameters

- **path** (*str*) – Path to perform walk on
- **include_deleted** (*bool,optional*) – Include deleted files, defaults to False

```
file_browser.walk('My Files')
```

Download

FileBrowser.**download**(*path*, *destination=None*)

Download a file

Parameters

- **path** (*str*) – Path of the file to download
- **destination** (*str,optional*) – File destination, if it is a directory, the original filename will be kept, defaults to the default directory

```
file_browser.download('My Files/Documents/Sample.docx')
```

Copy

FileBrowser.**copy**(*src*, *dest*)

Copy a file or directory

Parameters

- **src** (*str*) – The source path of the file or directory
- **dst** (*str*) – The destination path of the file or directory

```
file_browser.copy('My Files/Documents/Sample.docx', 'The/quick/brown/fox')
```

FileBrowser.**copy_multi**(*src*, *dest*)

```
file_browser.copy_multi(['My Files/Documents/Sample.docx', 'My Files/Documents/Burndown.
↪xlsx'], 'The/quick/brown/fox')
```

Create Public Link

FileBrowser.**mklink**(*path*, *access='RO'*, *expire_in=30*)

Create a link to a file

Parameters

- **path** (*str*) – The path of the file to create a link to
- **access** (*str,optional*) – Access policy of the link, defaults to 'RO'
- **expire_in** (*int,optional*) – Number of days until the link expires, defaults to 30

```
"""
Access:
- RW: Read Write
- RO: Read Only
- NA: No Access
"""

"""Create a Read Only public link to a file that expires in 30 days"""

file_browser.mklink('My Files/Documents/Sample.docx')

"""Create a Read Write public link to a folder that expires in 45 days"""

file_browser.mklink('My Files/Documents/Sample.docx', 'RW', 45)
```

Warning: you cannot use this tool to create read write public links to files.

2.1.3.2 Cloud Drive

The CloudDrive class is a subclass to `cterasdk.common.files.browser.FileBrowser` providing file access to the user's Cloud Drive

```
from getpass import getpass

"""Accessing Cloud Drive Files and Folders as a Global Administrator"""
admin = GlobalAdmin('portal.ctera.com')  # logging in to /admin
admin.login('admin', getpass())
file_browser = admin.files # the field is an instance of CloudDrive class object

"""Accessing Cloud Drive Files and Folders as a Tenant User Account"""
user = ServicesPortal('portal.ctera.com')  # logging in to /ServicesPortal
```

(continues on next page)

(continued from previous page)

```
user.login('bwayne', getpass())
file_browser = user.files # the field is an instance of CloudDrive class object
```

Create Directory

CloudDrive.**mkdir**(*path*, *recurse=False*)

Create a new directory

Parameters

- **path** (*str*) – Path of the directory to create
- **recurse** (*bool,optional*) – Whether to create the path recursivly, defaults to False

```
file_browser.mkdir('My Files/Documents')

file_browser.mkdir('The/quick/brown/fox', recurse = True)
```

Rename

CloudDrive.**rename**(*path*, *name*)

Rename a file

Parameters

- **path** (*str*) – Path of the file or directory to rename
- **name** (*str*) – The name to rename to

```
file_browser.rename('My Files/Documents/Sample.docx', 'Wizard Of Oz.docx')
```

Delete

CloudDrive.**delete**(*path*)

Delete a file

Parameters

path (*str*) – Path of the file or directory to delete

```
file_browser.delete('My Files/Documents/Sample.docx')
```

CloudDrive.**delete_multi**(**args*)

Delete multiple files and/or directories

Parameters

***args** – Variable lengthed list of paths of files and/or directories to delete

```
file_browser.delete_multi('My Files/Documents/Sample.docx', 'The/quick/brown/fox')
```

Undelete

CloudDrive.**undelete**(*path*)

Restore a previously deleted file or directory

Parameters

path (*str*) – Path of the file or directory to restore

```
file_browser.undelete('My Files/Documents/Sample.docx')
```

CloudDrive.**undelete_multi**(**args*)

Restore previously deleted multiple files and/or directories

Parameters

***args** – Variable length list of paths of files and/or directories to restore

```
file_browser.undelete_multi('My Files/Documents/Sample.docx', 'The/quick/brown/fox')
```

Move

CloudDrive.**move**(*src*, *dest*)

Move a file or directory

Parameters

- **src** (*str*) – The source path of the file or directory
- **dst** (*str*) – The destination path of the file or directory

```
file_browser.move('My Files/Documents/Sample.docx', 'The/quick/brown/fox')
```

CloudDrive.**move_multi**(*src*, *dest*)

```
file_browser.move_multi(['My Files/Documents/Sample.docx', 'My Files/Documents/Burndown.
↪xlsx'], 'The/quick/brown/fox')
```

Upload

CloudDrive.**upload**(*file_path*, *server_path*)

Upload a file

Parameters

- **file_path** (*str*) – Path to the local file to upload
- **server_path** (*str*) – Path to the directory to upload the file to

```
"""
Upload the 'Tree.jpg' file as an End User to 'Forest' directory
"""
file_browser.files.upload(r'C:\Users\BruceWayne\Downloads\Tree.jpg', 'Images/Forest')

"""
Upload the 'Tree.jpg' file as an Administrator to an End User's Cloud Drive
```

(continues on next page)

(continued from previous page)

```
"""
file_browser.files.upload(r'C:\Users\Administrator\Downloads\Tree.jpg', 'Bruce Wayne/
↪Images/Forest')
```

Collaboration Shares

CloudDrive.**share**(*path*, *recipients*, *as_project=True*, *allow_reshare=True*, *allow_sync=True*)

Share a file or a folder

Parameters

- **path** (*str*) – The path of the file or folder to share
- **recipients** (*list[cterasdk.core.types.ShareRecipient]*) – A list of share recipients
- **as_project** (*bool,optional*) – Share as a team project, defaults to True when the item is a cloud folder else False
- **allow_reshare** (*bool,optional*) – Allow recipients to re-share this item, defaults to True
- **allow_sync** (*bool,optional*) – Allow recipients to sync this item, defaults to True when the item is a cloud folder else False

Returns

A list of all recipients added to the collaboration share

Return type

list[*cterasdk.core.types.ShareRecipient*]

```
"""
Share with a local user and a local group.
- Grant the local user with read only access for 30 days
- Grant the local group with read write access with no expiration
"""

alice = portal_types.UserAccount('alice')
engineers = portal_types.GroupAccount('Engineers')

recipients = []

alice_rcpt = portal_types.ShareRecipient.local_user(alice).expire_in(30).read_only()
engineers_rcpt = portal_types.ShareRecipient.local_group(engineering).read_write()

file_browser.share('Codebase', [alice_rcpt, engineers_rcpt])
```

```
"""
Share with an external recipient
- Grant the external user with preview only access for 10 days
"""
jsmith = portal_types.ShareRecipient.external('jsmith@hotmail.com').expire_in(10).
↪preview_only())
file_browser.share('My Files/Projects/2020/ProjectX', [jsmith])
```

(continues on next page)

(continued from previous page)

```
"""
Share with an external recipient, and require 2 factor authentication
- Grant the external user with read only access for 5 days, and require 2 factor␣
↪authentication over e-mail
"""
jsmith = portal_types.ShareRecipient.external('jsmith@hotmail.com', True).expire_in(5).
↪read_only())
file_browser.share('My Files/Projects/2020/ProjectX', [jsmith])
```

```
"""
Share with a domain groups
- Grant the Albany domain group with read write access with no expiration
- Grant the Cleveland domain group with read only access with no expiration
"""
albany_group = portal_types.GroupAccount('Albany', 'ctera.com')
cleveland_group = portal_types.GroupAccount('Cleveland', 'ctera.com')

albany_rcpt = portal_types.ShareRecipient.domain_group(albany_group).read_write()
cleveland_rcpt = portal_types.ShareRecipient.domain_group(cleveland_group).read_only()

file_browser.share('Cloud/Albany', [albany_rcpt, cleveland_rcpt])
```

CloudDrive.**add_share_recipients**(*path*, *recipients*)

Add share recipients

Parameters

- **path** (*str*) – The path of the file or folder
- **recipients** (*list[cterasdk.core.types.ShareRecipient]*) – A list of share recipients

Returns

A list of all recipients added

Return type

list[*cterasdk.core.types.ShareRecipient*]

Note: if the share recipients provided as an argument already exist, they will be skipped and not updated

---

CloudDrive.**remove_share_recipients**(*path*, *accounts*)

Remove share recipients

Parameters

- **path** (*str*) – The path of the file or folder
- **accounts** (*list[cterasdk.core.types.PortalAccount]*) – A list of portal user or group accounts

Returns

A list of all share recipients removed

Return type

list[*cterasdk.core.types.PortalAccount*]

CloudDrive.**unshare**(*path*)

Unshare a file or a folder

```
"""
Unshare a file or a folder
"""
file_browser.unshare('Codebase')
file_browser.unshare('My Files/Projects/2020/ProjectX')
file_browser.unshare('Cloud/Albany')
```

2.1.3.3 Backups

The Backups class is a subclass to `cterasdk.common.files.browser.FileBrowser` providing access to files stored in backup folders

```
from getpass import getpass

"""Accessing Backups as a Global Administrator"""
admin = GlobalAdmin('portal.ctera.com')  # logging in to /admin
admin.login('admin', getpass())
file_browser = admin.files # the field is an instance of Backups class object

"""Accessing Backups as a Tenant User Account"""
user = ServicesPortal('portal.ctera.com')  # logging in to /ServicesPortal
user.login('bwayne', getpass())
file_browser = user.backups  # the field is an instance of Backups class object
```

2.2 Edge Filer

2.2.1 Gateway

Table of Contents

2.2.1.1 Instantiate a Gateway object

class cterasdk.object.Gateway.**Gateway**(*host*, *port=None*, *https=False*, *Portal=None*)

Main class operating on a Gateway

Variables

- **config** (cterasdk.edge.config.Config) – Object holding the Gateway Configuration APIs
- **network** (cterasdk.edge.network.Network) – Object holding the Gateway Network APIs
- **licenses** (cterasdk.edge.licenses.Licenses) – Object holding the Gateway Licenses APIs
- **services** (cterasdk.edge.services.Services) – Object holding the Gateway Services APIs
- **directoryservice** (cterasdk.edge.directoryservice.DirectoryService) – Object holding the Gateway Active Directory APIs
- **telnet** (cterasdk.edge.telnet.Telnet) – Object holding the Gateway Telnet APIs
- **syslog** (cterasdk.edge.syslog.Syslog) – Object holding the Gateway Syslog APIs
- **tasks** (cterasdk.edge.taskmgr.Tasks) – Object holding the Gateway Background Tasks APIs
- **audit** (cterasdk.edge.audit.Audit) – Object holding the Gateway Audit APIs
- **mail** (cterasdk.edge.mail.Mail) – Object holding the Gateway Mail APIs
- **backup** (cterasdk.edge.backup.Backup) – Object holding the Gateway Backup APIs
- **sync** (cterasdk.edge.sync.Sync) – Object holding the Gateway Sync APIs
- **cache** (cterasdk.edge.cache.Cache) – Object holding the Gateway Cache APIs
- **snmp** (*cterasdk.edge.snmp.SNMP*) – Object holding the Gateway SNMP APIs
- **ssl** (cterasdk.edge.ssl.SSL) – Object holding the Gateway SSL APIs
- **ssh** (*cterasdk.edge.ssl.SSH*) – Object holding the Gateway SSH APIs
- **power** (cterasdk.edge.power.Power) – Object holding the Gateway Power APIs
- **users** (cterasdk.edge.users.Users) – Object holding the Gateway Users APIs
- **groups** (cterasdk.edge.groups.Groups) – Object holding the Gateway Groups APIs
- **mtool** (cterasdk.edge.migration_tool.MigrationTool) – Object holding the Edge Filer's Migration Tool APIs
- **drive** (cterasdk.edge.drive.Drive) – Object holding the Gateway Drive APIs
- **volumes** (cterasdk.edge.volumes.Volumes) – Object holding the Gateway Volumes APIs
- **array** (cterasdk.edge.array.Array) – Object holding the Gateway Array APIs
- **shares** (cterasdk.edge.shares.Shares) – Object holding the Gateway Shares APIs
- **smb** (cterasdk.edge.smb.SMB) – Object holding the Gateway SMB APIs
- **aio** (cterasdk.edge.aio.AIO) – Object holding the Gateway AIO APIs
- **ftp** (cterasdk.edge.ftp.FTP) – Object holding the Gateway FTP APIs

- **afp** (cterasdk.edge.afp.AFP) – Object holding the Gateway AFP APIs
- **nfs** (cterasdk.edge.nfs.NFS) – Object holding the Gateway NFS APIs
- **rsync** (cterasdk.edge.rsync.RSync) – Object holding the Gateway RSync APIs
- **timezone** (cterasdk.edge.timezone.Timezone) – Object holding the Gateway Timezone APIs
- **logs** (cterasdk.edge.logs.Logs) – Object holding the Gateway Logs APIs
- **ntp** (cterasdk.edge.ntp.NTP) – Object holding the Gateway NTP APIs
- **shell** (cterasdk.edge.shell.Shell) – Object holding the Gateway Shell APIs
- **cli** (cterasdk.edge.cli.CLI) – Object holding the Gateway CLI APIs
- **dedup** (cterasdk.edge.dedup.Dedup) – Object holding the Gateway Local Deduplication APIs
- **support** (cterasdk.edge.support.Support) – Object holding the Gateway Support APIs
- **files** (*cterasdk.edge.files.FileBrowser*) – Object holding the Gateway File Browsing APIs
- **firmware** (cterasdk.edge.firmware.Firmware) – Object holding the Gateway Firmware APIs

__init__(*host*, *port=None*, *https=False*, *Portal=None*)

Parameters

- **host** (*str*) – The fully qualified domain name, hostname or an IPv4 address of the Gateway
- **port** (*int,optional*) – Set a custom port number (0 - 65535), If not set defaults to 80 for http and 443 for https
- **https** (*bool,optional*) – Set to True to require HTTPS, defaults to False
- **Portal** (cterasdk.object.Portal.Portal*,optional*) – The portal throught which the remote session was created, defaults to None

```
filer = Gateway('10.100.102.4') # will use HTTP over port 80

filer = Gateway('10.100.102.4', 8080) # will use HTTP over port 8080

filer = Gateway('vGateway-0dbc', 443, True) # will use HTTPS over port 443
```

Warning: for any certificate related error, this library will prompt for your consent in order to proceed. to avoid the prompt, you may configure *chopin-core* to automatically trust the server's certificate, using: `config.http['ssl'] = 'Trust'`

Logging in

Gateway.**test**()

Verification check to ensure the target host is a Gateway.

```
filer.test()
```

Gateway.**login**(*username*, *password*)

Log in

Parameters

- **username** (*str*) – User name to log in
- **password** (*str*) – User password

```
filer.login('admin', 'G3neralZ0d!')
```

Gateway.**logout**()

Log out

```
filer.logout()
```

Gateway.**whoami**()

Return the name of the logged in user.

Return cterasdk.common.object.Object
The session object of the current user

```
filer.whoami()
```

Core Methods

Gateway.**show**(*path*, *use_file_url=False*)

Print a schema object as a JSON string.

```
filer.show('/status/storage/volumes')
```

Gateway.**show_multi**(*path*, *paths*, *use_file_url=False*)

Print one or more schema objects as a JSON string.

```
filer.show_multi(['/config/storage/volumes', '/status/storage/volumes'])
```

Gateway.**get**(*path*, *params=None*, *use_file_url=False*)

Retrieve a schema object as a Python object.

```
"""Retrieve the device configuration and print it as JSON string"""

config = filer.get('/config')
print(config)

"""Retrieve the device settings and print the hostname and location settings"""
```

(continues on next page)

(continued from previous page)

```
settings = filer.get('/config/device')

print(settings.hostname)
print(settings.location)

"""Retrieve a list of volumes and print the name of the first volume"""

volumes  = filer.get('/status/storage/volumes') # returns a list of volumes

print(volumes[0].name) # will print the name of the first volume

"""Retrieve the network settings and print the MTU setting"""

network  = filer.get('/config/network') # returns network settings

print(network.ports[0].ethernet.mtu) # will print the MTU setting
```

Gateway.**get_multi**(*path*, *paths*, *use_file_url=False*)

Retrieve one or more schema objects as a Python object.

```
"""Retrieve '/config/cloudsync' and '/proc/cloudsync' at once"""

device = filer.get_multi(['/config/cloudsync', '/proc/cloudsync'])

print(device.config.cloudsync.cloudExtender.operationMode)
print(device.proc.cloudsync.serviceStatus.uploadingFiles)
```

Gateway.**put**(*path*, *value*, *use_file_url=False*)

Update a schema object or attribute.

```
"""Disable the first time wizard"""

filer.put('/config/gui/openFirstTimeWizard', False)

"""Turn off FTP access on all shares"""

shares = filer.get('/config/fileservices/share')

for share in shares:

    share.exportToFTP = False

    filer.put('/config/fileservices/share/' + share.name, share)
```

Gateway.**execute**(*path*, *name*, *param=None*, *use_file_url=False*)

Execute a schema object method.

```
"""Execute the file-eviction process"""

filer.execute('/config/cloudsync', 'forceExecuteEvictor') # doesn't require a param

"""Reboot the Gateway"""
```

(continues on next page)

(continued from previous page)

```
filer.execute('/statuc/device', 'reboot') # doesn't require a param

"""TCP Connect"""

param = Object()

param.address = 'chopin.ctera.com'

param.port = 995 # CTTP

bgTask = filer.execute('/status/network', 'tcpconnect', param)

print(bgTask)
```

See also:

Execute the file-eviction process: *cterasdk.edge.cache.Cache.force_eviction()*, Reboot the Gateway: `cterasdk.edge.power.reboot()`, Execute tcp connect: `Gateway.tcp_connect()`

`Gateway.`**`add`**(*path*, *param*, *use_file_url=False*)

Add a schema object.

```
"""Add a user account"""

user = Object()

user.username = 'mickey'

user.fullName = 'Mickey Mouse'

user.email = 'm.mouse@disney.com'

user.uid = 1940

user.password = 'M!niM0us3'

filer.add('/config/auth/users', user)
```

`Gateway.`**`delete`**(*path*, *use_file_url=False*)

Delete a schema object.

```
"""Delete a user account"""

user = 'mickey'

filer.delete('/config/auth/users/' + user)
```

Device Configuration

Config.**get_hostname**()

Get the hostname of the gateway

Return str
The hostname of the gateway

```
hostname = filer.config.hostname()
```

Config.**set_hostname**(*hostname*)

Set the hostname of the gateway

Parameters
hostname (*str*) – New hostname to set

Return str
The new hostname

```
filer.config.set_hostname('Chopin')
```

Config.**get_location**()

Get the location of the gateway

Return str
The location of the gateway

```
location = filer.config.location()
```

Config.**set_location**(*location*)

Set the location of the gateway

Parameters
location (*str*) – New location to set

Return str
The new location

```
filer.config.set_location('Jupiter')
```

Config.**disable_wizard**()

Disable the first time wizard

```
filer.config.disable_wizard()
```

Config.**export**(*destination=None*)

Export the Edge Filer configuration

Parameters
destination (*str,optional*) – File destination, defaults to the default directory

```
filer.config.export()
```

Config.**import_config**(*config*, *exclude=None*)

Import the Edge Filer configuration

Parameters

- **config** (*str*) – A string or a path to the Edge Filer configuration file
- **delete_attrs** (*list[str],optional*) – List of configuration properties to exclude from import

```
"""Import Edge Filer configuration from file"""
filer.config.import_config(r'C:\Users\bwayne\Downloads\EdgeFiler.xml')

"""Import configuration without network settings"""
filer.config.import_config(r'C:\Users\bwayne\Downloads\EdgeFiler.xml', exclude=[
    '/config/network'
])

"""Import configuration without the 'logs' and 'public' shares"""
filer.config.import_config(r'C:\Users\bwayne\Downloads\EdgeFiler.xml', exclude=[
    '/config/fileservices/share/logs',
    '/config/fileservices/share/public'
])
```

Code Snippets

```
filer.get('/status/device/runningFirmware')  # Get firmware version

"""Get the entire device status object"""
status = filer.get('/status/device')
print(status.runningFirmware, status.MacAddress)
```

Storage

Format

Drive.**format**(*name*)

Format a drive

Parameters

name (*str*) – The name of the drive to format

```
filer.drive.format('SATA1')
```

Drive.**format_all**()

Format all drives

```
filer.drive.format_all()
```

Volumes

Volumes.**add**(*name*, *size=None*, *filesystem='xfs'*, *device=None*, *passphrase=None*)

Add a new volume to the gateway

Parameters

- **name** (*str*) – Name of the new volume
- **size** (*int,optional*) – Size of the new volume, defaults to the device's size
- **filesystem** (*str,optional*) – Filesystem to use, defaults to xfs
- **device** (*str,optional*) – Name of the device to use for the new volume, can be left as None if there the gateway has only one
- **passphrase** (*str,optional*) – Passphrase for the volume

Returns

Gateway response

```
filer.volumes.add('localcache')
```

Volumes.**delete**(*name*)

Delete a volume

Parameters

name (*str*) – Name of the volume to delete

```
filer.volumes.delete('localcache')
```

Volumes.**delete_all**()

Delete all volumes

```
filer.volumes.delete_all()
```

Local Deduplication

Dedup.**enable**(*reboot=False*, *wait=False*)

Enable local deduplication

Parameters

- **reboot** (*bool*) – Reboot, defaults to False
- **wait** (*bool,optional*) – Wait for reboot to complete, defaults to False

```
"""Enable local de-duplication without rebooting the Edge Filer"""
filer.dedup.enable()

"""Enable local de-duplication and wait for reboot to complete"""
filer.dedup.enable(reboot=True, wait=True)
```

Dedup.**disable**(*reboot=False*, *wait=False*)

Disable local deduplication

Parameters

- **reboot** (*bool*) – Reboot, defaults to `False`
- **wait** (*bool,optional*) – Wait for reboot to complete, defaults to False

```
"""Disable local de-duplication without rebooting the Edge Filer"""
filer.dedup.disable()

"""Disable local de-duplication and wait for reboot to complete"""
filer.dedup.disable(reboot=True, wait=True)
```

`Dedup.`**`status`**`()`

Get the de-duplication status

Returns
An object including the deduplication status

Return type
cterasdk.edge.types.DeduplicationStatus

```
print(filer.dedup.status())
```

`Regeneration.`**`run`**`()`

Run the regeneration process

```
filer.dedup.regen.run()
```

`Regeneration.`**`status`**`()`

Get the regeneration process statistics

```
print(filer.dedup.regen.status())
```

Shares

`Shares.`**`add`**`(`*name*, *directory*, *acl=None*, *access='winAclMode'*, *csc='manual'*, *dir_permissions=777*, *comment=None*, *export_to_afp=False*, *export_to_ftp=False*, *export_to_nfs=False*, *export_to_pc_agent=False*, *export_to_rsync=False*, *indexed=False*, *trusted_nfs_clients=None*, *uuid=None*`)`

Add a network share.

Parameters

- **name** (*str*) – The share name
- **directory** (*str*) – Full directory path
- **acl** (*list[cterasdk.edge.types.ShareAccessControlEntry]*) – List of access control entries
- **access** (`cterasdk.edge.enum.Acl`) – The Windows File Sharing authentication mode, defaults to `winAclMode`
- **csc** (`cterasdk.edge.enum.ClientSideCaching`) – The client side caching (offline files) configuration, defaults to `manual`
- **dir_permissions** (*int*) – Directory Permission, defaults to 777
- **comment** (*str*) – Comment

- **export_to_afp** (*bool*) – Whether to enable AFP access, defaults to `False`
- **export_to_ftp** (*bool*) – Whether to enable FTP access, defaults to `False`
- **export_to_nfs** (*bool*) – Whether to enable NFS access, defaults to `False`
- **export_to_pc_agent** (*bool*) – Whether to allow as a destination share for CTERA Backup Agents, defaults to `False`
- **export_to_rsync** (*bool*) – Whether to enable access over rsync, defaults to `False`
- **indexed** (*bool*) – Whether to enable indexing for search, defaults to `False`
- **trusted_nfs_clients** (*list[cterasdk.edge.types.NFSv3AccessControlEntry]*) – Trusted NFS v3 clients, defaults to `None`

```
"""
Create an ACL-enabled cloud share called 'Accounting' and define four access control␣
↪entries:

1) Everyone - Read Only (Local Group)
2) admin - Read Write (Local User)
3) Domain Admins - Read Only (Domain Group)
4) bruce.wayne@ctera.com - Read Write (Domain User)

Principal Type:
- LG: Local Group
- LU: Local User
- DG: Domain Group
- DU: Domain User

Access:
- RW: Read Write
- RO: Read Only
- NA: No Access
"""

everyone = gateway_types.ShareAccessControlEntry(gateway_enum.PrincipalType.LG, 'Everyone
↪', gateway_enum.FileAccessMode.RO)
local_admin = gateway_types.ShareAccessControlEntry(gateway_enum.PrincipalType.LU, 'admin
↪', gateway_enum.FileAccessMode.RW)
domain_admins = gateway_types.ShareAccessControlEntry(gateway_enum.PrincipalType.DG,
↪'CTERA\Domain Admins', gateway_enum.FileAccessMode.RO)
bruce_wayne = gateway_types.ShareAccessControlEntry(gateway_enum.PrincipalType.DU,
↪'bruce.wayne@ctera.com', gateway_enum.FileAccessMode.RW)

filer.shares.add('Accounting', 'cloud/users/Service Account/Accounting', acl = [ \
    everyone, local_admin, domain_admins, bruce_wayne \
])

"""Create an 'Only Authenticated Users' cloud share called 'FTP' and enable FTP access to␣
↪everyone"""

everyone = gateway_types.ShareAccessControlEntry(gateway_enum.PrincipalType.LG, 'Everyone
↪', gateway_enum.FileAccessMode.RW)
```

(continues on next page)

(continued from previous page)

```
filer.shares.add('FTP', 'cloud/users/Service Account/FTP', acl = [everyone], export_to_
↪ftp = True)

"""Add an NFS share and enable access to two hosts"""
nfs_client_1 = gateway_types.NFSv3AccessControlEntry('192.168.0.1', '255.255.255.0',␣
↪gateway_enum.FileAccessMode.RW)  # read write
nfs_client_2 = gateway_types.NFSv3AccessControlEntry('192.168.0.2', '255.255.255.0',␣
↪gateway_enum.FileAccessMode.RO)  # read only
filer.shares.add('NFS', 'cloud/users/Service Account/NFS', export_to_nfs=True, trusted_
↪nfs_clients=[nfs_client_1, nfs_client_2])
```

Shares.**add_acl**(*name*, *acl*)

Add one or more access control entries to an existing share.

Parameters

- **name** (*str*) – The share name
- **acl** (*list[cterasdk.edge.types.ShareAccessControlEntry]*) – List of access control entries to add

```
"""Add two access control entries to the 'Accounting' share"""

domain_group = gateway_types.ShareAccessControlEntry(gateway_enum.PrincipalType.DG,
↪'CTERA\leadership', gateway_enum.FileAccessMode.RW)
domain_user = gateway_types.ShareAccessControlEntry(gateway_enum.PrincipalType.DU,
↪'clark.kent@ctera.com', gateway_enum.FileAccessMode.RO)

filer.shares.add_acl('Accounting', [domain_group, domain_user])
```

Shares.**set_acl**(*name*, *acl*)

Set a network share's access control entries.

Parameters

- **name** (*str*) – The share name
- **acl** (*list[cterasdk.edge.types.ShareAccessControlEntry]*) – List of access control entries

Warning: this method will override the existing access control entries

```
"""Set the access control entries of the 'Accounting' share"""

domain_group = gateway_types.ShareAccessControlEntry(gateway_enum.PrincipalType.DG,
↪'CTERA\leadership', gateway_enum.FileAccessMode.RW)
domain_user = gateway_types.ShareAccessControlEntry(gateway_enum.PrincipalType.DU,
↪'clark.kent@ctera.com', gateway_enum.FileAccessMode.RO)

filer.shares.set_acl('Accounting', [domain_group, domain_user])
```

Shares.**remove_acl**(*name*, *acl*)

Remove one or more access control entries from an existing share.

Parameters

- **name** (*str*) – The share name
- **acl** (*list[cterasdk.edge.types.RemoveShareAccessControlEntry]*) – List of access control entries to remove

```
"""Remove access control entries from the 'Accounting' share"""

domain_group = gateway_types.RemoveShareAccessControlEntry(gateway_enum.PrincipalType.DG,
↪ 'CTERA\leadership')
domain_user = gateway_types.RemoveShareAccessControlEntry(gateway_enum.PrincipalType.DU,
↪'clark.kent@ctera.com')

filer.shares.remove_acl('Accounting', [domain_group, domain_user])
```

Shares.**set_share_winacls**(*name*)

Set a network share to use Windows ACL Emulation Mode

Parameters

name (*str*) – The share name

```
filer.shares.set_share_winacls('cloud')
```

Shares.**block_files**(*name*, *extensions*)

Configure a share to block one or more file extensions

Parameters

- **name** (*str*) – The share name
- **extensions** (*list[str]*) – List of file extensions to block

```
filer.shares.block_files('Accounting', ['exe', 'cmd', 'bat'])
```

Shares.**modify**(*name*, *directory=None*, *acl=None*, *access=None*, *csc=None*, *dir_permissions=None*, *comment=None*, *export_to_afp=None*, *export_to_ftp=None*, *export_to_nfs=None*, *export_to_pc_agent=None*, *export_to_rsync=None*, *indexed=None*, *trusted_nfs_clients=None*)

Modify an existing network share. All parameters but name are optional and default to None

Parameters

- **name** (*str*) – The share name
- **directory** (*str,optional*) – Full directory path
- **acl** (*list[cterasdk.edge.types.ShareAccessControlEntry],optional*) – List of access control entries
- **access** (*cterasdk.edge.enum.Acl,optional*) – The Windows File Sharing authentication mode
- **csc** (*cterasdk.edge.enum.ClientSideCaching,optional*) – The client side caching (offline files) configuration
- **dir_permissions** (*int,optional*) – Directory Permission
- **comment** (*str,optional*) – Comment
- **export_to_afp** (*bool,optional*) – Whether to enable AFP access
- **export_to_ftp** (*bool,optional*) – Whether to enable FTP access

- **export_to_nfs** (*bool, optional*) – Whether to enable NFS access
- **export_to_pc_agent** (*bool, optional*) – Whether to allow as a destination share for CTERA Backup Agents
- **export_to_rsync** (*bool, optional*) – Whether to enable access over rsync
- **indexed** (*bool, optional*) – Whether to enable indexing for search
- **trusted_nfs_clients** (*list[cterasdk.edge.types.NFSv3AccessControlEntry]*) – Trusted NFS v3 clients, defaults to None

```
""" Disable all file-access protocols on all shares """
shares = filer.shares.get() # obtain a list of all shares

for share in shares:
   filer.share.modify(
      share.name,
      export_to_afp=False,       # Apple File Sharing
      export_to_ftp=False,       # FTP
      export_to_nfs=False,       # NFS
      export_to_pc_agent=False,  # CTERA Agent
      export_to_rsync=False,     # rsync
      indexed=False              # Search
   )
```

Shares.**delete**(*name*)

Delete a share.

Parameters

name (*str*) – The share name

```
filer.shares.delete('Accounting')
```

Users

Users.**add**(*username*, *password*, *full_name=None*, *email=None*, *uid=None*)

Add a user of the Gateway

Parameters

- **username** (*str*) – User name for the new user
- **password** (*str*) – Password for the new user
- **full_name** (*str, optional*) – The full name of the new user, defaults to None
- **email** (*str, optional*) – E-mail address of the new user, defaults to None
- **uid** (*str, optional*) – The uid of the new user, defaults to None

```
filer.users.add('Clark', 'Kryptonite1!') # without a full name, email or custom uid

filer.users.add('alice', 'W!z4rd0fOz!', 'Alice Wonderland') # including a full name

filer.users.add('Bruce', 'GothamCity1!', 'Bruce Wayne', 'bruce.wayne@we.com', uid =␣
↪1940) # all
```

Users.**modify**(*username*, *password=None*, *full_name=None*, *email=None*, *uid=None*)

Modify an existing user of the Gateway

Parameters

- **username** (*str*) – User name to modify
- **password** (*str,optional*) – New password, defaults to None
- **full_name** (*str,optional*) – The full name of the user, defaults to None
- **email** (*str,optional*) – E-mail address of the user, defaults to None
- **uid** (*str,optional*) – The uid of the user, defaults to None

```
filer.users.modify('Clark', 'Passw0rd1!') # Change a user's password
filer.users.modify('Clark', email='clark.kent@krypton.com') # Change a user's email
```

Users.**delete**(*username*)

Delete an existing user

Parameters

username (*str*) – User name of the user to delete

```
filer.users.delete('alice')
```

Users.**add_first_user**(*username*, *password*, *email=''*)

Add the first user of the Gateway and login

Parameters

- **username** (*str*) – User name for the new user
- **password** (*str*) – Password for the new user
- **email** (*str,optional*) – E-mail address of the new user, defaults to an empty string

```
filer.users.add_first_user('admin', 'L3tsG3tR34dyT0Rumbl3!')
```

Groups

Groups.**add_members**(*group*, *members*)

Add members to a group

Parameters

- **group** (*str*) – Name of the group
- **members** (*list[cterasdk.edge.types.UserGroupEntry]*) – List of users and groups to add to the group

```
"""Add Bruce Wayne to the local Administrators group"""
member = gateway_types.UserGroupEntry(gateway_enum.PrincipalType.DU, 'bruce.wayne@we.com
↪')
filer.groups.add_members('Administrators', [member])

"""Add Bruce Wayne and Domain Admins to the local Administrators group"""
```

(continues on next page)

(continued from previous page)

```
domain_user = gateway_types.UserGroupEntry(gateway_enum.PrincipalType.DU, 'bruce.
↪wayne@we.com')
domain_group = gateway_types.UserGroupEntry(gateway_enum.PrincipalType.DG, 'WE\Domain␣
↪Admins')
filer.groups.add_members('Administrators', [domain_user, domain_group])
```

Groups.**remove_members**(*group*, *members*)

Remove members from a group

Parameters

- **group** (*str*) – Name of the group
- **members** (*list[cterasdk.edge.types.UserGroupEntry]*) – List of users and groups to remove from the group

```
"""Remove Bruce Wayne from the local Administrators group"""

filer.groups.remove_members('Administrators', [('DU', 'bruce.wayne@we.com')])

"""Remove Bruce Wayne and Domain Admins from the local Administrators group"""

filer.groups.remove_members('Administrators', [('DU', 'bruce.wayne@we.com'), ('DG', 'WE\
↪Domain Admins')])
```

Active Directory

DirectoryService.**connect**(*domain*, *username*, *password*, *ou=None*, *check_connection=False*)

Connect the Gateway to an Active Directory

Parameters

- **domain** (*str*) – The active directory domain to connect to
- **username** (*str*) – The user name to use when connecting to the active directory services
- **password** (*str*) – The password to use when connecting to the active directory services
- **ou** (*str,optional*) – The OU path to use when connecting to the active directory services, defaults to None
- **check_connection** (*bool,optional*) – Check connectivity before attempting to connect to directory services, defaults to *False*

```
filer.directoryservice.connect('ctera.local', 'administrator', 'B4tMob!l3')

"""Connect to the EMEA Organizational Unit"""

filer.directoryservice.connect('ctera.local', 'administrator', 'B4tMob!l3', 'ou=EMEA,␣
↪dc=ctera, dc=local')
```

Note: the *ou* parameter must specify the distinguished name of the organizational unit

DirectoryService.**get_advanced_mapping**()

Retrieve directory services advanced mapping configuration

Returns
A dictionary of domain mapping objects

Return type
dict

```
for domain, mapping in filer.directoryservice.get_advanced_mapping().items():
    print(domain)
    print(mapping)
```

Note: to retrieve a list of domain flat names, use `cterasdk.edge.directoryservice.domains()`

DirectoryService.**set_advanced_mapping**(*mappings*)

Configure advanced mapping

Parameters
mappings (*list[cterasdk.common.types.ADDomainIDMapping]*) – List of domains and their UID/GID mapping range

```
"""Create a list of domain mappings"""

advanced_mapping = [
    common_types.ADDomainIDMapping('CTERA-PRD', 1000001, 2000000),
    common_types.ADDomainIDMapping('CTERA-LAB', 2000001, 3000000),
    common_types.ADDomainIDMapping('CTERA-LDR', 3000001, 4000000)
]

filer.directoryservice.set_advanced_mapping(advanced_mapping)  # this function will skip␣
↪domains that are not found
```

Note: to retrieve a list of domain flat names, use `cterasdk.edge.directoryservice.domains()`

DirectoryService.**disconnect**()

Disconnect from Active Directory Service

```
filer.directoryservice.disconnect()
```

DirectoryService.**domains**()

Get all domains

Return list(str)
List of names of all discovered domains

```
domains = filer.directoryservice.domains()

print(domains)
```

DirectoryService.**set_static_domain_controller**(*dc*)

Configure the Gateway to use a static domain controller

Parameters
dc (*str*) – The FQDN, hostname or ip address of the domain controller

Returns
The FQDN, hostname or ip address of the domain controller

Return type
str

```
filer.directoryservice.set_static_domain_controller('192.168.90.1')
```

DirectoryService.**get_static_domain_controller**()

Retrieve the static domain controller configuration

Returns
A FQDN, hostname or ip address of the domain controller

Return type
str

```
domain_controller = filer.directoryservice.get_static_domain_controller()
print(domain_controller)
```

DirectoryService.**remove_static_domain_controller**()

Delete the static domain controller configuration

```
filer.directoryservice.remove_static_domain_controller()
```

Cloud Services

Services.**connect**(*server*, *user*, *password*, *ctera_license='EV16'*)

Connect to a Portal.

The connect method will first validate the *license* argument,
ensure the Gateway can establish a TCP connection over port 995 to *server* using `Gateway.tcp_connect()` and verify the Portal does not require device activation via code

Parameters

- **server** (*str*) – Address of the Portal
- **user** (*str*) – User for the Portal connection
- **password** (*str*) – Password for the Portal connection
- **ctera_license** (`cterasdk.edge.enum.License`,*optional*) – CTERA License, defaults to cterasdk.edge.enum.License.EV16

Warning: for any certificate related error, this library will prompt for your consent in order to proceed. to avoid the prompt, you may configure *chopin-core* to automatically trust the server's certificate, using: `config.connect['ssl'] = 'Trust'`

```
filer.services.connect('chopin.ctera.com', 'svc_account', 'Th3AmazingR4ce!', 'EV32') #␣
↪activate as an EV32
```

```
filer.services.connect('52.204.15.122', 'svc_account', 'Th3AmazingR4ce!', 'EV64') #␣
↪activate as an EV64
```

Services.**activate**(*server*, *user*, *code*, *ctera_license='EV16'*)

Activate the gateway using an activation code

Parameters

- **server** (*str*) – Address of the Portal
- **user** (*str*) – User for the Portal connection
- **code** (*str*) – Activation code for the Portal connection
- **ctera_license** (cterasdk.edge.enum.License,*optional*) – CTERA License, defaults to cterasdk.edge.enum.License.EV16

This method's behavior is identical to *cterasdk.edge.services.Services.connect()*

```
filer.services.activate('chopin.ctera.com', 'svc_account', 'fd3a-301b-88d5-e1a9-cbdb') #␣
↪activate as an EV16
```

Services.**reconnect**()

Reconnect to the Portal

```
filer.services.reconnect()
```

Services.**disconnect**()

Disconnect from the Portal

```
filer.services.disconnect()
```

Services.**enable_sso**()

Enable SSO connection

Applying a License

Licenses.**apply**(*ctera_license*)

Apply a license

Parameters

ctera_license (cterasdk.edge.enum.License) – License type

```
filer.license.apply('EV32')
```

Note: you can specify a license upon connecting the Gateway to CTERA Portal. See *cterasdk.edge.services.Services.connect()*

Caching

Cache.**enable**()

Enable caching

```
filer.cache.enable()
```

Cache.**disable**()

Disable caching

```
filer.cache.disable()
```

Warning: all data synchronized from the cloud will be deleted and all unsynchronized changes will be lost.

Cache.**force_eviction**()

Force eviction

```
filer.cache.force_eviction()
```

Cache.**pin**(*path*)

Pin a folder

Parameters
path (*str*) – Directory path

```
""" Pin a cloud folder named 'data' owned by 'Service Account' """
filer.cache.pin('users/Service Account/data')
```

Cache.**pin_exclude**(*path*)

Exclude a sub-folder from a pinned folder

Parameters
path (*str*) – Directory path

```
""" Exclude a subfolder from a pinned cloud folder """
filer.cache.pin_exclude('users/Service Account/data/accounting')
```

Cache.**remove_pin**(*path*)

Remove a pin from a previously pinned folder

Parameters
path (*str*) – Directory path

```
""" Remove a pin from a previously pinned folder """
filer.cache.remove_pin('users/Service Account/data')
```

Cache.**pin_all**()

Pin all folders

```
""" Pin all folders """
filer.cache.pin_all()
```

Cache.**unpin_all**()

Remove all folder pins

```
""" Remove all folder pins """
filer.cache.unpin_all()
```

Cloud Backup

Backup.**configure**(*passphrase=None*)

Gateway backup configuration

Parameters

passphrase (*str,optional*) – Passphrase for the backup, defaults to None

```
"""Configure backup without a passphrase"""

filer.backup.configure()
```

Backup.**start**()

Start backup

```
filer.backup.start()
```

Backup.**suspend**()

Suspend backup

```
filer.backup.suspend()
```

Backup.**unsuspend**()

Unsuspend backup

```
filer.backup.unsuspend()
```

Cloud Backup Files

BackupFiles.**unselect_all**()

Unselect all files from backup

```
filer.backup.files.unselect_all()
```

Cloud Sync

Sync.**suspend**(*wait=True*)

Suspend Cloud Sync

Parameters

wait (*bool*) – Wait for synchronization to stop

```
filer.sync.suspend()
```

Sync.**unsuspend**()

Unsuspend Cloud Sync

```
filer.sync.unsuspend()
```

Sync.**exclude_files**(*extensions=None*, *filenames=None*, *paths=None*, *custom_exclusion_rules=None*)

Exclude files from Cloud Sync. This method will override any existing file exclusion rules Use *cterasdk.common.types.FileFilterBuilder()* to build custom file exclusion rules`

Parameters

- **extensions** (*list[str]*) – List of file extensions
- **filenames** (*list[str]*) – List of file names
- **paths** (*list[str]*) – List of file paths
- **rules** (*list[cterasdk.common.types.FilterBackupSet]*) – Set of custom exclusion rules

```
filer.sync.exclude_files(['exe', 'cmd', 'bat'])  # exclude file extensions

filer.sync.exclude_files(filenames=['Cloud Sync.lnk', 'The quick brown fox.docx'])  #␣
↪exclude file names

"""Exclude file extensions and file names"""
filer.sync.exclude_files(['exe', 'cmd'], ['Cloud Sync.lnk'])

"""
Create a custom exclusion rule

Exclude files that their name starts with 'tmp' and smaller than 1 MB (1,048,576 bytes)
"""
name_filter_rule = common_types.FileFilterBuilder.name().startswith('tmp')
size_filter_rule = common_types.FileFilterBuilder.size().less_than(1048576)
exclusion_rule = common_types.FilterBackupSet('Custom exclusion rule', filter_
↪rules=[name_filter_rule, size_filter_rule])

filer.sync.exclude_files(custom_exclusion_rules=[exclusion_rule])
```

Sync.**remove_file_exclusion_rules**()

Remove previously configured sync exclusion rules

```
filer.sync.remove_file_exclusion_rules()
```

Sync.**evict**(*path*, *wait=False*)

Evict a directory from the Edge Filer

Parameters

- **path** (*str*) – Directory path
- **wait** (*bool*) – Wait for eviction task to complete, defaults to `False`

Returns

A reference to the background task

Return type

str

```
"""Evict a directory"""
background_task_ref = filer.sync.evict('/Share/path/to/sub/directory')  # non-blocking␣
↪call
print(background_task_ref)

"""Evict a directory and wait for eviction to complete - blocking"""
filer.sync.evict('/Share/path/to/sub/directory', wait=True)  # blocking call
```

Cloud Sync Bandwidth Throttling

CloudSyncBandwidthThrottling.**get_policy**()

Get the bandwidth throttling policy

Returns
a list of bandwidth throttling rules

Return type
list[*cterasdk.common.types.ThrottlingRule*]

CloudSyncBandwidthThrottling.**set_policy**(*rules*)

Set the bandwidth throttling policy

Parameters
rules (*list[cterasdk.common.types.ThrottlingRule]*) – List of bandwidth throttling rules

```
"""Throttle bandwidth during business hours on week days: Monday - Friday"""
schedule1 = common_types.TimeRange().start('07:00:00').end('19:00:00').days(common_enum.
↪DayOfWeek.Weekdays).build()
rule1 = common_types.ThrottlingRuleBuilder().upload(50).download(50).schedule(schedule1).
↪build()

"""Throttle bandwidth off business hours on week days: Monday - Friday"""
schedule2 = common_types.TimeRange().start('19:00:00').end('07:00:00').days(common_enum.
↪DayOfWeek.Weekdays).build()
rule2 = common_types.ThrottlingRuleBuilder().upload(100).download(100).
↪schedule(schedule2).build()

"""Throttle bandwidth during weekends: Saturday, Sunday"""
schedule3 = common_types.TimeRange().start('00:00:00').end('23:59:00').days(common_enum.
↪DayOfWeek.Weekend).build()
rule3 = common_types.ThrottlingRuleBuilder().upload(500).download(500).
↪schedule(schedule3).build()

filer.sync.throttling.set_policy([rule1, rule2, rule3])
```

File Access Protocols

FTP.**disable**()

Disable FTP

```
filer.ftp.disable()
```

AFP.**disable**()

Disable AFP

```
filer.afp.disable()
```

NFS.**disable**()

Disable NFS

```
filer.nfs.disable()
```

RSync.**disable**()

Disable FTP

```
filer.rsync.disable()
```

Windows File Sharing (CIFS/SMB)

SMB.**enable**()

Enable SMB

```
filer.smb.enable()
```

SMB.**disable**()

Disable SMB

```
filer.smb.disable()
```

SMB.**set_packet_signing**(*packet_signing*)

Set Packet signing

Parameters

packet_signing (cterasdk.edge.enum.CIFSPacketSigning) – Packet signing type

```
filer.smb.set_packet_signing('If client agrees')
```

SMB.**enable_abe**()

Enable ABE

```
filer.smb.enable_abe()
```

SMB.**disable_abe**()

Disable ABE

```
filer.smb.disable_abe()
```

AIO.**enable**()

Enable AIO

```
filer.aio.enable()
```

AIO.**disable**()

Disable AIO

```
filer.aio.disable()
```

Network

Network.**set_static_ipaddr**(*address*, *subnet*, *gateway*, *primary_dns_server*, *secondary_dns_server=None*)

Set a Static IP Address

Parameters

- **address** (*str*) – The static address
- **subnet** (*str*) – The subnet for the static address
- **gateway** (*str*) – The default gateway
- **primary_dns_server** (*str*) – The primary DNS server
- **secondary_dns_server** (*str,optinal*) – The secondary DNS server, defaults to None

```
filer.network.set_static_ipaddr('10.100.102.4', '255.255.255.0', '10.100.102.1', '10.100.
↪102.1')

filer.show('/status/network/ports/0/ip') # will print the IP configuration
```

Network.**set_static_nameserver**(*primary_dns_server*, *secondary_dns_server=None*)

Set the DNS Server addresses statically

Parameters

- **primary_dns_server** (*str*) – The primary DNS server
- **secondary_dns_server** (*str,optinal*) – The secondary DNS server, defaults to None

```
filer.network.set_static_nameserver('10.100.102.1') # to set the primary name server

filer.network.set_static_nameserver('10.100.102.1', '10.100.102.254') # to set both␣
↪primary and secondary
```

Network.**enable_dhcp**()

Enable DHCP

```
filer.network.enable_dhcp()
```

Network.**set_mtu**(*mtu*)

Set a custom network maximum transmission unit (MTU)

Parameters

mtu (*int*) – Maximum transmission unit

```
filer.network.set_mtu(1320)  # set the maximum transmission unit (MTU) to 1320

filer.network.set_mtu(9000)  # configure 'jumbo' frames (MTU: 9000)
```

Network.**reset_mtu**()

Set the default maximum transmission unit (MTU) settings

```
filer.network.reset_mtu()  # disable custom mtu configuration and restore default␣
↪setting (1500)
```

Network.**get_static_routes**()

Get all Static Routes

```
# get static routes
filer.network.get_static_routes()
```

Network.**add_static_route**(*source_ip*, *destination_ip_mask*)

Set a Static Route

Parameters

- **source_ip** (*str*) – The source IP (192.168.15.55)
- **destination_ip_mask** (*str*) – The destination IP and CIDR block (10.5.0.1/32)

```
# add static route from 10.10.12.1 to 192.168.55.7/32
filer.network.add_static_route('10.10.12.1', '192.168.55.7/32')

# add static route from 10.100.102.4 to 172.18.100.0/24
filer.network.add_static_route('10.100.102.4', '172.18.100.0/24')
```

Network.**remove_static_route**(*destination_ip_mask*)

Delete a Static Route

Parameters

destination_ip_mask (*str*) – The destination IP and CIDR block (10.5.0.1/32)

```
# remove static route 192.168.55.7/32
filer.network.remove_static_route('192.168.55.7/32')
```

Network.**clean_all_static_routes**()

Clean all Static routes

```
# remove all static routes -  (clean)
filer.network.clean_all_static_routes()
```

Network Diagnostics

Network.**tcp_connect**(*service*)

Test a TCP connection between the Gateway and the provided host address

Parameters
service (cterasdk.edge.types.TCPService) – A service, identified by a host and a port

Returns
A named-tuple including the host, port and a boolean value indicating whether TCP connection can be established

Return type
cterasdk.edge.types.TCPConnectResult

```
cttp_service = gateway_types.TCPService('chopin.ctera.com', 995)
result = filer.network.tcp_connect(cttp_service)
if result.is_open:
    print('Success')
    # do something...
else:
    print('Failure')

ldap_service = gateway_types.TCPService('dc.ctera.com', 389)
filer.network.tcp_connect(ldap_service)
```

Network.**diagnose**(*services*)

Test a TCP connection to a host over a designated port

Parameters
services (*list[*cterasdk.edge.types.TCPService*]*) – List of services, identified by a host and a port

Returns
A list of named-tuples including the host, port and a boolean value indicating whether TCP connection can be established

Return type
list[*cterasdk.edge.types.TCPConnectResult*]

```
services = []
services.append(gateway_types.TCPService('192.168.90.1', 389))  # LDAP
services.append(gateway_types.TCPService('ctera.portal.com', 995))  # CTTP
services.append(gateway_types.TCPService('ctera.portal.com', 443))  # HTTPS
result = filer.network.diagnose(services)
for result in results:
    print(result.host, result.port, result.is_open)
```

Network.**iperf**(*address*, *port=5201*, *threads=1*, *protocol='TCP'*, *direction='Upload'*, *retries=120*, *seconds=1*)

Invoke a network throughput test

Parameters

- **address** (*str*) – The host running the iperf server
- **port** (*int,optional*) – The iperf server port, defaults to 5201
- **threads** (*int,optional*) – The number of threads, defaults to 1

- **protocol** (cterasdk.edge.enum.IPProtocol,*optional*) – IP protocol, defaults to *'TCP'*
- **direction** (cterasdk.edge.enum.Traffic,*optional*) – Traffic direction, defaults to *'Upload'*
- **retries** (*int*,*optional*) – Number of retries when sampling the iperf task status, defaults to 120
- **seconds** (*int*,*optional*) – Number of seconds to wait between retries, defaults to 1

Returns
A string containing the iperf output

Return type
str

```
filer.network.iperf('192.168.1.145')  # iperf server: 192.168.1.145, threads: 1, measure␣
↪upload over TCP port 5201

filer.network.iperf('192.168.1.145', port=85201, threads=5)  # Customized port and␣
↪number of threads

filer.network.iperf('192.168.1.145', direction=gateway_enum.Traffic.Download)  # Measure␣
↪download speed

filer.network.iperf('192.168.1.145', protocol=gateway_enum.IPProtocol.UDP)  # Use UDP
```

Mail Server

Mail.**enable**(*smtp_server*, *port=25*, *username=None*, *password=None*, *use_tls=True*)

Enable e-mail delivery using a custom SMTP server

Parameters

- **smtp_server** (*str*) – Address of the SMTP Server
- **port** (*int*,*optional*) – The listening port of the SMTP Server, defaults to 25
- **username** (*str*,*optional*) – The user name of the SMTP Server, defaults to None
- **password** (*str*,*optional*) – The password of the SMTP Server, defaults to None
- **use_tls** (*bool*,*optional*) – Use TLS when connecting to the SMTP Server, defaults to True

```
filer.mail.enable('smtp.ctera.com') # default settings

filer.mail.enable('smtp.ctera.com', 465) # custom port number

"""Use default port number, use authentication and require TLS"""

filer.mail.enable('smtp.ctera.com', username = 'user', password = 'secret', useTLS =␣
↪True)
```

Mail.**disable**()

Disable e-mail delivery using a custom SMTP server

```
filer.mail.disable()
```

Logging

Logs.**settings**(*retention*, *min_severity=None*)

Configure log settings

Parameters

- **retention** (*int*) – Log retention period in days
- **min_severity** (cterasdk.edge.enum.Severity,*optional*) – Minimal log severity

Logs.**logs**(*topic*, *include=None*, *minSeverity='info'*)

Fetch Gateway logs

Parameters

- **topic** (*str*) – Log Topic to fetch
- **include** (*list[str],optional*) – List of fields to include in the response, defaults to Logs.default_include
- **minSeverity** (cterasdk.edge.enum.Severity,*optional*) – Minimal log severity to fetch, defaults to cterasdk.edge.enum.Severity.INFO

Returns

Log lines

Return type

cterasdk.lib.iterator.Iterator

Syslog.**enable**(*server*, *port=514*, *proto='UDP'*, *min_severity='info'*)

Enable Syslog

Parameters

- **server** (*str*) – Server address to send syslog logs
- **port** (*int,optional*) – Syslog server communication port, defaults to 514
- **proto** (cterasdk.edge.enum.IPProtocol,*optional*) – Syslog server communication protocol, defaults to cterasdk.edge.enum.IPProtocol.UDP
- **min_severity** (cterasdk.edge.enum.Severity,*optional*) – Minimal log severity to fetch, defaults to cterasdk.edge.enum.Severity.INFO

```
filer.syslog.enable('syslog.ctera.com') # default settings

filer.syslog.enable('syslog.ctera.com', proto = 'TCP') # use TCP

filer.syslog.enable('syslog.ctera.com', 614, minSeverity = 'error') # use 614 UDP,␣
↪severity >= error
```

Syslog.**disable**()

Disable Syslog

```
filer.syslog.disable()
```

SMB Audit Logs

Audit.**enable**(*path*, *auditEvents=None*, *logKeepPeriod=30*, *maxLogKBSize=102400*, *maxRotateTime=1440*, *includeAuditLogTag=True*, *humanReadableAuditLog=False*)

Enable Gateway Audit log

Parameters

- **path** (*str*) – Path to save the audit log
- **auditEvents** (*list[cterasdk.edge.enum.AuditEvents],optional*) – List of audit event types to save, defaults to Audit.defaultAuditEvents
- **logKeepPeriod** (*int,optional*) – Period to key the logs in days, defaults to 30
- **maxLogKBSize** (*int,optional*) – The maximum size of the log file in KB, defaults to 102400 (100 MB)
- **maxRotateTime** (*int,optional*) – The maximal time before rotating the log file in Minutes, defaults to 1440 (24 hours)
- **includeAuditLogTag** (*bool,optional*) – Include audit log tag, defaults to True
- **humanReadableAuditLog** (*bool,optional*) – Human readable audit log, defaults to False

```
filer.audit.enable('/logs')
```

Audit.**disable**()

Disable Gateway Audit log

```
filer.audit.disable()
```

Reset

Power.**reset**(*wait=False*)

Reset the Gateway setting

Parameters

wait (*bool,optional*) – Wait for reset to complete, defaults to False

```
filer.power.reset() # will reset and immediately return

filer.power.reset(True) # will reset and wait for the Gateway to boot
```

See also:

create the first admin account after resetting the Gateway to its default settings: *cterasdk.edge.users.Users.add_first_user()*

SSL

SSL.**disable_http**()

Disable HTTP access

```
filer.ssl.disable_http()
```

SSL.**enable_http**()

Enable HTTP access

```
filer.ssl.enable_http()
```

SSL.**is_http_disabled**()

Check if HTTP access is disabled

```
filer.ssl.is_http_disabled()
```

SSL.**is_http_enabled**()

Check if HTTP access is enabled

```
filer.ssl.is_http_enabled()
```

```
"""
ca_certificate = './certs/certificate.crt'
"""

filer.ssl.set_trusted_ca(ca_certificate)
```

SSL.**get_storage_ca**()

Get object storage trusted CA certificate

SSL.**remove_storage_ca**()

Remove object storage trusted CA certificate

SSL.**import_certificate**(*private_key*, **certificates*)

Import the Edge Filer's web server's SSL certificate

Parameters

- **private_key** (*str*) – The PEM-encoded private key, or a path to the PEM-encoded private key file
- **certificates** (*list[str]*) – The PEM-encoded certificates, or a list of paths to the PEM-encoded certificates

```
"""
certificate = './certs/certificate.crt'
intermediate_cert = './certs/certificate1.crt'
ca_certificate = './certs/certificate2.crt'
private_key = './certs/private.key'
"""

"""
Specify certificates in the following order: domain cert, intermediary certs, CA cert
```

(continues on next page)

(continued from previous page)

```
You may include as many intermediate certificates as needed
"""
filer.ssl.import_certificate(private_key, certificate, intermediate_cert, ca_certificate)
```

Power Management

Power.**reboot**(*wait=False*)

Reboot the Gateway

Parameters

wait (*bool, optional*) – Wait for reboot to complete, defaults to False

```
filer.power.reboot() # will reboot and immediately return

filer.power.reboot(True) # will reboot and wait
```

Power.**shutdown**()

Shutdown the Gateway

```
filer.power.shutdown()
```

SNMP

SNMP.**is_enabled**()

Check if SNMP is enabled

Returns

True is SNMP is enabled, else False

Return type

bool

```
filer.snmp.is_enabled()
```

SNMP.**enable**(*port=161*, *community_str=None*, *username=None*, *auth_password=None*, *privacy_password=None*)

Enable SNMP

Parameters

- **port** (*int, optional*) – SNMP server port, defaults to 161
- **community_str** (*str, optional*) – SNMPv2c community string
- **username** (*str, optional*) – SNMPv3 username
- **auth_password** (*str, optional*) – SNMPv3 authentication password
- **privacy_password** (*str, optional*) – SNMPv3 privacy password

```
filer.snmp.enable(community_str='MpPcKl2sArSdTLZ4URj4')  # enable SNMP v2c
filer.snmp.enable(username='snmp_user', auth_password='gVQBaHSOGV', privacy_password=
↪'VG0zbn5aJ')  # enable SNMP v3
```

SNMP.**disable**()

Disable SNMP

filer.snmp.disable()

SNMP.**modify**(*port=None*, *community_str=None*, *username=None*, *auth_password=None*, *privacy_password=None*)

Modify current SNMP configuration. Only configurations that are not *None* will be changed. SNMP must be enabled

Parameters

- **port** (*int,optional*) – SNMP server port, defaults to 161
- **community_str** (*str,optional*) – SNMPv2c community string
- **username** (*str,optional*) – SNMPv3 username
- **auth_password** (*str,optional*) – SNMPv3 authentication password
- **privacy_password** (*str,optional*) – SNMPv3 privacy password

filer.snmp.modify(community_str='L0K2zGpgmOQH2CXaUSuB', username='snmp_user', auth_password='gVQBaHSOGV', privacy_password='VG0zbn5aJ')

SNMP.**get_configuration**()

filer.snmp.get_configuration()

Support

Support Report

Support.**get_support_report**()

Download support report

Debug

Support.**set_debug_level**(**levels*)

Set the debug level

```
filer.support.set_debug_level('backup', 'process', 'cttp', 'samba')

filer.support.set_debug_level('info')

filer.support.set_debug_level('caching', 'evictor')
```

Telnet Access

Telnet.**enable**(*code*)

Enable Telnet

```
filer.telnet.enable('a7df639a')
```

Telnet.**disable**()

Disable Telnet

```
filer.telnet.disable()
```

SSH Access

SSH.**enable**(*public_key=None*, *public_key_file=None*, *exponent=65537*, *key_size=2048*)

Enable the Edge Filer's SSH daemon

Parameters

- **public_key** (*str,optional*) – A PEM-encoded public key in OpenSSH format. If neither a public key nor public key file were specified, an RSA key pair will be generated automatically. The PEM-encoded private key will be saved to the default Downloads directory
- **public_key_file** (*str,optional*) – A path to the public key file
- **exponent** (*int,optional*) – The public exponent of the new key, defaults to *65537*
- **key_size** (*int,optional*) – The length of the modulus in bits, defaults to *2048*

```
"""Enable SSH access"""
filer.ssh.enable()

"""Enable SSH access using a public key file"""
filer.ssh.enable(public_key_file='./public_key.pub')  # relative to the current directory
filer.ssh.enable(public_key_file='C:\\Users\\jsmith\\Desktop\\public_key.pub')  # full␣
↪path

"""Generate an RSA key pair and enable SSH access"""

from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives.serialization import Encoding, PrivateFormat,␣
↪PublicFormat, NoEncryption

private_key = rsa.generate_private_key(public_exponent=exponent, key_size=key_size)
public_key = private_key.public_key().public_bytes(Encoding.OpenSSH, PublicFormat.
↪OpenSSH).decode('utf-8')

filer.ssh.enable(public_key)

"""Print PEM-encoded RSA private key"""
print(private_key.private_bytes(Encoding.PEM, PrivateFormat.OpenSSH, NoEncryption()).
↪decode('utf-8'))
```

(continues on next page)

(continued from previous page)

```
"""Print OpenSSH formatted RSA public key"""
print(public_key)
```

SSH.**disable**()

```
filer.ssh.disable()
```

CTERA Migrate

MigrationTool.**list_shares**(*credentials*)

Log in

Parameters
credentials (cterasdk.edge.types.HostCredentials) – Target host credentials

```
""" List all shares available on a source host """
credentials = gateway_types.HostCredentials('source-hostname', 'username', 'password')
filer.mtool.list_shares(credentials)

""" List all shares available on the current Edge Filer """
credentials = gateway_types.HostCredentials.localhost()
filer.mtool.list_shares(credentials)
```

MigrationTool.**list_tasks**(*deleted=False*)

List tasks

Parameters
deleted (*bool, optional*) – List deleted tasks, defaults to False

Returns
List of tasks

Return type
list(*cterasdk.common.object.Object*)

```
""" Print all tasks, optional flag to list deleted tasks """
for task in filer.mtool.list_tasks(deleted=False):
    print(task)
```

MigrationTool.**start**(*task*)

Start a task

Parameters
task (cterasdk.common.object.Object) – Task object

```
filer.mtool.start(task)
```

MigrationTool.**stop**(*task*)

Stop a task

Parameters
task (cterasdk.common.object.Object) – Task object

```
filer.mtool.stop(task)
```

MigrationTool.**delete**(*tasks*)

Delete tasks

Parameters
tasks (*list(cterasdk.common.object.Object)*) – List of tasks

```
filer.mtool.delete(filer.mtool.list_tasks())  # delete all tasks
```

MigrationTool.**restore**(*tasks*)

Recover tasks

Parameters
tasks (*list(cterasdk.common.object.Object)*) – List of tasks

```
filer.mtool.restore(filer.mtool.list_tasks(deleted=True))  # recover all deleted tasks
```

MigrationTool.**details**(*task*)

Get task details

```
filer.mtool.details(task)
```

MigrationTool.**results**(*task*)

```
filer.mtool.results(task)
```

Discovery Tasks

Discovery.**list_tasks**(*deleted=False*)

```
""" Print all discovery tasks, optional flag to list deleted tasks """
for task in filer.mtool.discovery.list_tasks(deleted=False):
    print(task)
```

Discovery.**add**(*name*, *credentials*, *shares*, *auto_start=False*, *log_every_file=False*, *notes=None*)

Create a discovery task

Parameters

- **name** (*str*) – Task name
- **credentials** (*cterasdk.edge.types.HostCredentials*) – Target host credentials
- **auto_start** (*bool,optional*) – Start task after creation, defaults to `False`
- **log_every_file** (*bool,optional*) – Log every file, defaults to `False`
- **notes** (*str,optional*) – Task notes

Returns
Task

Return type
cterasdk.common.object.Object

```
credentials = gateway_types.HostCredentials('source-hostname', 'username', 'password')
task = filer.mtool.discovery.add('my-discovery', credentials, ['share1', 'share2'], auto_
↪start=False, log_every_file=True, notes='job 1')


"""Add a local discovery task"""
credentials = gateway_types.HostCredentials.localhost()
task = filer.mtool.discovery.add('my-discovery', credentials, ['share1', 'share2'], log_
↪every_file=True, notes='local discovery job')

"""Run the task"""
filer.mtool.start(task)
```

Discovery.**update**(*task*, *name=None*, *notes=None*)

Update a discovery task

Parameters

- **name** (*str,optional*) – Task name
- **notes** (*str,optional*) – Task notes

Migration Tasks

Migration.**list_tasks**(*deleted=False*)

```
""" Print all migration tasks, optional flag to list deleted tasks """
for task in filer.mtool.migration.list_tasks(deleted=False):
    print(task)
```

Migration.**add**(*name*, *credentials*, *shares*, *auto_start=False*, *access_time=None*, *winacls=True*, *cloud_folder=None*, *create_cloud_folder_per_share=False*, *compute_checksum=False*, *exclude=None*, *include=None*, *notes=None*)

Create a discovery task

Parameters

- **name** (*str*) – Task name
- **credentials** (cterasdk.edge.types.HostCredentials) – Target host credentials
- **auto_start** (*bool,optional*) – Start task after creation, defaults to False
- **access_time** (*bool,optional*) – Copy last access time, defaults to None
- **winacls** (*bool,optional*) – Copy NTFS ACL's, defaults to True
- **cloud_folder** (*str,optional*) – Target cloud folder, if create_cloud_folder_per_share was set to True then this attribute serves as the cloud folder name prefix
- **create_cloud_folder_per_share** (*bool,optional*) – Create cloud folder per share, defaults to False
- **compute_checksum** (*bool,optional*) – Validate and report checksums post-migration, defaults to False
- **exclude** (*list(str),optional*) – List of patterns to exclude, defaults to None

- **include** (*list(str),optional*) – List of patterns to include, defaults to None
- **notes** (*str,optional*) – Task notes

Returns
Task

Return type
cterasdk.common.object.Object

```
credentials = gateway_types.HostCredentials('source-hostname', 'username', 'password')
task = filer.mtool.migration.add('my-discovery', credentials, ['share1', 'share2'], auto_
↪start=False, winacls=True, cloud_folder='my_cloud_folder', create_cloud_folder_per_
↪share=False, compute_checksum=False, exclude=['*.pdf', '*.jpg'], include=['*.png', '*.
↪avi'], notes='migration job 1')

"""Run the task"""
filer.mtool.start(task)
```

2.2.2 File Browser

Table of Contents

2.2.2.1 Obtaining Access to the Gateway's File System

```
filer = Gateway('vGateway-0dbc')

filer.login('USERNAME', 'PASSWORD')

file_browser = filer.files # the field is an instance of FileBrowser class object
```

List

static FileBrowser.**ls**(*_path*)

Download

FileBrowser.**download**(*path*, *destination=None*)

Download a file

Parameters

- **path** (*str*) – The file path on the Edge Filer
- **destination** (*str,optional*) – File destination, if it is a directory, the original filename will be kept, defaults to the default directory

```
file_browser.download('cloud/users/Service Account/My Files/Documents/Sample.docx')
```

Create Directory

FileBrowser.**mkdir**(*path*, *recurse=False*)

Create a new directory

Parameters

- **path** (*str*) – The path of the new directory
- **recurse** (*bool,optional*) – Create subdirectories if missing, defaults to False

```
file_browser.mkdir('cloud/users/Service Account/My Files/Documents')

file_browser.mkdir('cloud/users/Service Account/My Files/The/quick/brown/fox', recurse =␣
↪True)
```

Copy

FileBrowser.**copy**(*src*, *dest*, *overwrite=False*)

Copy a file or a folder

Parameters

- **src** (*str*) – Source file or folder path
- **dest** (*str*) – Destination folder path
- **overwrite** (*bool,optional*) – Overwrite on conflict, defaults to False

```
"""
Copy the 'Documents' folder from Bruce Wayne to Alice Wonderland
The full path of the documents folder after the copy: 'cloud/users/Alice Wonderland/My␣
↪Files/Documents'
"""
file_browser.copy('cloud/users/Bruce Wayne/My Files/Documents', 'cloud/users/Alice␣
↪Wonderland/My Files')

"""Copy the file Summary.xlsx to another directory, and overwrite on conflict"""
file_browser.copy('cloud/users/Bruce Wayne/My Files/Summary.xlsx', 'cloud/users/Bruce␣
↪Wayne/Spreadsheets', True)
```

Move

FileBrowser.**move**(*src*, *dest*, *overwrite=False*)

Move a file or a folder

Parameters

- **src** (*str*) – Source file or folder path
- **dest** (*str*) – Destination folder path
- **overwrite** (*bool,optional*) – Overwrite on conflict, defaults to False

```
"""
Move the 'Documents' folder from Bruce Wayne to Alice Wonderland
The full path of the documents folder after the move: 'cloud/users/Alice Wonderland/My␣
↪Files/Documents'
"""
file_browser.move('cloud/users/Bruce Wayne/My Files/Documents', 'cloud/users/Alice␣
↪Wonderland/My Files')

"""Move the file Summary.xlsx to another directory, and overwrite on conflict"""
file_browser.move('cloud/users/Bruce Wayne/My Files/Summary.xlsx', 'cloud/users/Bruce␣
↪Wayne/Spreadsheets', True)
```

Delete

FileBrowser.**delete**(*path*)

Delete a file

Parameters

path (*str*) – The file's path on the gateway

```
file_browser.delete('cloud/users/Service Account/My Files/Documents')
```

2.3 Agent

class cterasdk.object.Agent.**Agent**(*host*, *port=80*, *https=False*, *Portal=None*)

Main class operating on a Agent

Variables

- **backup** (cterasdk.edge.backup.Backup) – Object holding the Agent Backup APIs
- **cli** (cterasdk.edge.cli.CLI) – Object holding the Agent CLI APIs
- **logs** (cterasdk.edge.logs.Logs) – Object holding the Agent Logs APIs
- **services** (cterasdk.edge.services.Services) – Object holding the Agent Services APIs
- **support** (cterasdk.edge.support.Support) – Object holding the Agent Support APIs
- **sync** (cterasdk.edge.sync.Sync) – Object holding the Agent Sync APIs

__init__(*host*, *port=80*, *https=False*, *Portal=None*)

Parameters

- **host** (*str*) – The fully qualified domain name, hostname or an IPv4 address of the Gateway
- **port** (*int,optional*) – Set a custom port number (0 - 65535), defaults to 80
- **https** (*bool,optional*) – Set to True to require HTTPS, defaults to False
- **Portal** (cterasdk.object.Portal.Portal*,optional*) – The portal throught which the remote session was created, defaults to None

2.4 Miscellaneous

2.4.1 Logging

The library includes a built-in console logger. The logger's configuration is controlled by the config.Logging.get() class object.

2.4.1.1 Redirecting the log to a file

You can redirect the cterasdk log to a file by setting the environment variable **CTERASDK_LOG_FILE**

2.4.1.2 Disabling the Logger

The logger is enabled by default. To disable the logger, run:

```
config.Logging.get().disable()
```

2.4.1.3 Changing the Log Level

The default logging level is set to `logging.INFO`. To change the log level, run:

```
config.Logging.get().setLevel(logging.ERROR) # will log severity >= error

config.Logging.get().setLevel(logging.WARNING) # will log severity >= warning
```

Log Levels

The available log levels are:

Level	Numeric Value
CRITICAL	50
ERROR	40
WARNING	30
INFO	20
DEBUG	10

2.4.2 Formatting

The following formatting functions are included in this library:

`cterasdk.convert.format.`**`tojsonstr`**(*obj*, *pretty_print=True*, *no_log=True*)

Convert a Python object to a JSON string.

Parameters

- **`obj`** (*`object`*) – the Python object
- **`pretty_print`** (*`bool`*) – Whether to format the JSON string, defaults to `True`
- **`no_log`** (*`book`*) – Hide sensitive values in the log messages

Returns

JSON string of the object

Return type

str

```
user = Object()
user.name = 'alice'
user.firstName = 'Alice'
user.lastName = 'Wonderland'
user.email = 'alice@adventures.com'
user.password = 'Passw0rd1!'
print(tojsonstr(user))
{
     "lastName": "Wonderland",
     "password": "Passw0rd1!",
     "name": "alice",
     "firstName": "Alice",
     "email": "alice@adventures.com"
}
print(tojsonstr(user, False))
{"lastName": "Wonderland", "password": "Passw0rd1!", "name": "alice", "firstName": "Alice
↪", "email": "alice@adventures.com"}
```

cterasdk.convert.format.**toxmlstr**(*obj*, *pretty_print=False*)

Convert a Python object to an XML string

Parameters

- **obj** (*object*) – the Python object
- **pretty_print** (*bool*) – whether to format the XML string, defaults to `False`

Returns

XML string of the object

Return type

str

```
user = Object()
user.name = 'alice'
user.firstName = 'Alice'
user.lastName = 'Wonderland'
user.email = 'alice@adventures.com'
user.password = 'Passw0rd1!'
print(toxmlstr(user))
print(toxmlstr(user, True))
```

CHAPTER

THREE

CTERASDK PACKAGE

3.1 Subpackages

3.1.1 cterasdk.client package

3.1.1.1 Submodules

cterasdk.client.cteraclient module

class cterasdk.client.cteraclient.**CTERAClient**(*session_id_key*)

Bases: object

copy(*baseurl*, *src*, *dest*, *overwrite*)

db(*baseurl*, *path*, *name*, *param*)

delete(*baseurl*, *path*)

download(*baseurl*, *path*, *params*)

download_zip(*baseurl*, *path*, *form_data*)

execute(*baseurl*, *path*, *name*, *param=None*)

static file_descriptor(*request*, *response*)

form_data(*baseurl*, *path*, *form_data*)

static fromxmlstr(*request*, *response*)

get(*baseurl*, *path*, *params=None*)

get_multi(*baseurl*, *path*, *paths*)

get_session_id()

mkcol(*baseurl*, *path*)

move(*baseurl*, *src*, *dest*, *overwrite*)

multipart(*baseurl*, *path*, *form_data*)

post(*baseurl*, *path*, *data*)

put(*baseurl*, *path*, *data*)

set_authorization_headers(*headers*)

set_session_id(*session_id*)

upload(*baseurl*, *path*, *form_data*)

class cterasdk.client.cteraclient.**MigrationClient**(*http_client=None*, *session_id_key=None*)

Bases: *RESTClient*

XSRF_TOKEN_ID = 'x-mt-x'

login(*baseurl*, *path*)

class cterasdk.client.cteraclient.**RESTClient**(*http_client=None*, *session_id_key=None*)

Bases: `object`

delete(*baseurl*, *path*)

static **fromjsonstr**(*request*, *response*)

get(*baseurl*, *path*, *params=None*)

post(*baseurl*, *path*, *data*)

put(*baseurl*, *path*, *data*)

cterasdk.client.cteraclient.**execute_request**(*request_function*, *return_function*)

Execute an HTTP request

cterasdk.client.cteraclient.**transcribe_request**(*request*, *response*, *transcribe_response=True*)

Transcribe the HTTP request

cterasdk.client.host module

class cterasdk.client.host.**CTERAHost**(*host*, *port*, *https*)

Bases: *NetworkHost*

add(*path*, *param*, *use_file_url=False*)

Add a schema object.

property base_api_url

property base_file_url

copy(*src*, *dest*, *overwrite*, *use_file_url=False*)

db(*path*, *name*, *param*, *use_file_url=False*)

default_class(*name*)

delete(*path*, *use_file_url=False*)

Delete a schema object.

download_zip(*path*, *form_data*, *use_file_url=False*)

execute(*path*, *name*, *param=None*, *use_file_url=False*)

Execute a schema object method.

form_data(*path*, *form_data*, *use_file_url=False*)

get(*path*, *params=None*, *use_file_url=False*)

Retrieve a schema object as a Python object.

get_multi(*path*, *paths*, *use_file_url=False*)

Retrieve one or more schema objects as a Python object.

get_session_id()

Get the id of the current session

Return str
Current session id

login(*username*, *password*)

Log in

Parameters

- **username** (*str*) – User name to log in
- **password** (*str*) – User password

logout()

Log out

mkcol(*path*, *use_file_url=False*)

move(*src*, *dest*, *overwrite*, *use_file_url=False*)

multipart(*path*, *form_data*, *use_file_url=False*)

openfile(*path*, *params=None*, *use_file_url=False*)

post(*path*, *value*, *use_file_url=False*)

put(*path*, *value*, *use_file_url=False*)

Update a schema object or attribute.

register_session(*session*)

session()

set_authorization_headers(*headers*)

Start a session using authorization headers id instead of logging in

Parameters
headers (*dict*) – the authorization headers, represented as a key-value str dict

set_session_id(*session_id*)

Start a session with the session id instead of logging in

Parameters
session_id (*str*) – Session id for the new session

show(*path*, *use_file_url=False*)

Print a schema object as a JSON string.

show_multi(*path*, *paths*, *use_file_url=False*)

Print one or more schema objects as a JSON string.

upload(*path*, *form_data*, *use_file_url=False*)

whoami()

Return the name of the logged in user.

Return cterasdk.common.object.Object
The session object of the current user

class cterasdk.client.host.**MigrationHost**(*host*, *port*, *https*, *is_authenticated=None*, *http_client=None*)

Bases: *NetworkHost*

delete(*path*)

static from_ctera_host(*ctera_host*)

Create a RESTful host instance from an existing CTERA host instance

get(*path*, *params=None*)

login(*path*)

post(*path*, *value*)

put(*path*, *value*)

class cterasdk.client.host.**NetworkHost**(*host*, *port*, *https*)

Bases: object

baseurl()

host()

https()

port()

scheme()

test_conn()

cterasdk.client.host.**authenticated**(*function*)

cterasdk.client.http module

class cterasdk.client.http.**ContentType**

Bases: object

application_json = {'Content-Type': 'application/json'}

textplain = {'Content-Type': 'text/plain'}

urlencoded = {'Content-Type': 'application/x-www-form-urlencoded'}

class cterasdk.client.http.**HTTPClient**(*session_id_key*)

Bases: *HttpClientBase*

copy(*src*, *dest*, *overwrite*, *headers=None*)

delete(*url*, *headers=None*)

get(*url*, *params=None*, *headers=None*, *stream=None*)

mkcol(*url*, *headers=None*)

move(*src*, *dest*, *overwrite*, *headers=None*)

multipart(*url*, *form_data*, *monitor_function_generator=None*)

post(*url*, *headers=None*, *data=''*, *urlencode=False*)

put(*url*, *headers=None*, *data=''*)

upload(*url*, *form_data*)

exception cterasdk.client.http.**HTTPException**(*http_error*)

Bases: Exception

class cterasdk.client.http.**HTTPResponse**(*response*)

Bases: object

getcode()

geturl()

read()

class cterasdk.client.http.**HttpClientBase**(*session_id_key*)

Bases: object

dispatch(*ctera_request*)

get_session_id()

on_ssl_error(*request*)

static on_timeout(*attempt*)

set_custom_headers(*headers*)

Add custom headers that will be included in every http request.

Parameters

headers (*dict*) – the headers, represented as a key-value str dict

set_session_id(*session_id*)

should_trust(*host*, *port*)

trust(*_host*, *_port*)

class cterasdk.client.http.**HttpClientRequest**(*method*, *url*, ***kwargs*)

Bases: object

class cterasdk.client.http.**HttpClientRequestCopy**(*src*, *dest*, *overwrite*, *headers=None*)

Bases: *HttpClientRequestCopyMove*

class cterasdk.client.http.**HttpClientRequestCopyMove**(*method*, *src*, *dest*, *overwrite*, *headers=None*)

Bases: *HttpClientRequest*

class cterasdk.client.http.**HttpClientRequestDelete**(*url*, *headers=None*)

Bases: *HttpClientRequest*

class cterasdk.client.http.**HttpClientRequestGet**(*url*, *params=None*, *headers=None*, *stream=None*)

Bases: *HttpClientRequest*

class cterasdk.client.http.**HttpClientRequestMkcol**(*url*, *headers=None*)

Bases: *HttpClientRequest*

class cterasdk.client.http.**HttpClientRequestMove**(*src*, *dest*, *overwrite*, *headers=None*)

Bases: *HttpClientRequestCopyMove*

class cterasdk.client.http.**HttpClientRequestPost**(*url*, *headers=None*, *data=None*)

Bases: *HttpClientRequest*

class cterasdk.client.http.**HttpClientRequestPut**(*url*, *headers=None*, *data=None*)

Bases: *HttpClientRequest*

cterasdk.client.http.**geturi**(*baseurl*, *path*)

cterasdk.client.ssl module

class cterasdk.client.ssl.**CertificateServices**

Bases: object

static add_trusted_cert(*host*, *port*)

static save_cert_from_server(*host*, *port*)

3.1.2 cterasdk.common package

3.1.2.1 Submodules

cterasdk.common.datetime_utils module

class cterasdk.common.datetime_utils.**DateTimeUtils**

Bases: object

static get_expiration_date(*expiration*)

Get a datetime.date representation of the expiration date

Parameters

expiration (*variable,optional*) – The expiration value. Pass datetime.date for a specific date, integer for days from now, or True for immediate (yesterday)

Return datetime.date

datetime.date representation of the expiration date

cterasdk.common.item module

class cterasdk.common.item.**Item**

Bases: object

cterasdk.common.object module

class cterasdk.common.object.**Device**(*uid*, *version*, *firmware*)

Bases: *Object*

class cterasdk.common.object.**Object**

Bases: object

cterasdk.common.object.**delete_attr**(*obj*, *path*)

Delete attribute

Parameters

- **object** (cterasdk.common.object.Object) – The object
- **path** (*str*) – Attribute path

Returns
The modified object

Return type
cterasdk.common.object.Object

cterasdk.common.object.**delete_attrs**(*obj*, *paths*)

Delete attributes

Parameters

- **object** (cterasdk.common.object.Object) – The object
- **paths** (*list[str]*) – List of attributes to remove

Returns
The modified object

Return type
cterasdk.common.object.Object

cterasdk.common.object.**find_attr**(*obj*, *path*)

Find attribute

Parameters

- **object** (cterasdk.common.object.Object) – The object
- **path** (*str*) – A string or an array of the attribute path

Returns
The attribute, or None if not found

cterasdk.common.object.**get_attr**(*obj*, *attr*)

Get attribute

Parameters

- **object** (cterasdk.common.object.Object) – The object

- **attr** (*str*) – The name of the attribute to retrieve

Returns
The attribute, or None if not found

cterasdk.common.object.**remove_array_element**(*array*, *attr*)

cterasdk.common.object.**remove_array_element_by_key**(*array*, *key*, *value*)

cterasdk.common.object.**remove_attr**(*obj*, *attr*)

Remove attribute

Parameters

- **object** (cterasdk.common.object.Object) – The object
- **attr** (*str*) – The name of the attribute to remove

cterasdk.common.types module

class cterasdk.common.types.**ADDomainIDMapping**(*domain*, *start*, *end*)

Bases: *Object*

Base Class for Directory Service ID Mapping

Variables
domain (*str*) – The domain flat name

Parameters

- **start** (*int*) – The minimum id to use for mapping
- **end** (*int*) – The maximum id to use for mapping

class cterasdk.common.types.**AdvancedFilterRule**(*classname*, *field*, *operator*)

Bases: *Object*

class cterasdk.common.types.**AfterOperator**(*right*)

Bases: *Operator*

class cterasdk.common.types.**ApplicationBackupSet**(*apps*)

Bases: *BackupSet*

class cterasdk.common.types.**BackupScheduleBuilder**

Bases: object

static interval(*hours=None*, *minutes=None*)

Schedule backup to periodically, defaults to 24 hours

Parameters

- **hours** (*int*) – Hours
- **minutes** (*int*) – Minutes

static window(*time_range*)

Schedule backup to run at a specific time

Parameters
time_range (cterasdk.common.types.TimeRange) – Time range

class cterasdk.common.types.**BackupSet**(*name*, *directory_tree=None*, *filter_rules=None*, *defaults_dirs=None*, *template_dirs=None*, *enabled=True*, *boolean_function=None*, *comment=None*)

Bases: *Object*

class cterasdk.common.types.**BeforeOperator**(*right*)

Bases: *Operator*

class cterasdk.common.types.**BeginsWithOperator**(*right*)

Bases: *Operator*

class cterasdk.common.types.**ContainsOperator**(*right*)

Bases: *Operator*

class cterasdk.common.types.**CriteriaBuilder**(*criteria_type*, *criteria_field*)

Bases: object

build()

class cterasdk.common.types.**DateTimeCriteriaBuilder**(*criteria_type*, *criteria_field*)

Bases: *CriteriaBuilder*

after(*value*)

before(*value*)

class cterasdk.common.types.**DirEntry**(*name*, *display_name=None*, *included=None*, *children=None*)

Bases: *FileEntry*

class cterasdk.common.types.**DirectoryEntryFactory**

Bases: object

static root(*included*)

class cterasdk.common.types.**EndsWithOperator**(*right*)

Bases: *Operator*

class cterasdk.common.types.**FileEntry**(*name*, *display_name=None*, *included=None*)

Bases: *Object*

class cterasdk.common.types.**FileFilterBuilder**

Bases: object

Type = 'File'

static extensions()

Filter files by extension

static last_modified()

Filter files by last modification date

static name()

Filter files by name pattern

static names()

Filter files by names

static path()

Filter files by path pattern

static paths()

Filter files by path

static size()

Filter files by size

class cterasdk.common.types.**FilterBackupSet**(*name*, *directory_tree=None*, *filter_rules=None*, *defaults_dirs=None*, *template_dirs=None*, *enabled=True*, *boolean_function=None*, *comment=None*)

Bases: *BackupSet*

class cterasdk.common.types.**IntegerCriteriaBuilder**(*criteria_type*, *criteria_field*)

Bases: *CriteriaBuilder*

less_than(*value*)

more_than(*value*)

class cterasdk.common.types.**IsOneOfOperator**(*right*)

Bases: *Operator*

class cterasdk.common.types.**IsOperator**(*right*)

Bases: *Operator*

class cterasdk.common.types.**LessThanOperator**(*right*)

Bases: *Operator*

class cterasdk.common.types.**ListCriteriaBuilder**(*criteria_type*, *criteria_field*)

Bases: *CriteriaBuilder*

include(*values*)

class cterasdk.common.types.**MoreThanOperator**(*right*)

Bases: *Operator*

class cterasdk.common.types.**Operator**(*right*)

Bases: *Object*

class cterasdk.common.types.**PolicyRule**(*assignment*, *criteria*)

Bases: object

class cterasdk.common.types.**PolicyRuleConverter**

Bases: object

static convert(*rule*, *classname*, *property_name*, *assignment=None*)

class cterasdk.common.types.**StringCriteriaBuilder**(*criteria_type*, *criteria_field*)

Bases: *CriteriaBuilder*

contains(*value*)

endswith(*value*)

equals(*value*)

isoneof(*values*)

startswith(*value*)

class cterasdk.common.types.**TaskSchedule**

Bases: *Object*

class cterasdk.common.types.**ThrottlingRule**

Bases: object

Throttling Rule

Variables

- **upload** (*int*) – Throttling rate upstream (Kilobits)
- **download** (*int*) – Throttling rate downstream (Kilobits)
- **start** (*str*) – Start time
- **end** (*str*) – End time
- **days** (*list[str]*) – Days

static from_server_object(*param*)

to_server_object()

class cterasdk.common.types.**ThrottlingRuleBuilder**

Bases: object

Bandwidth Throttling Rule Builder

build()

Build the throttling rule

download(*kbps*)

Throttle bandwidth downstream

Parameters
kbps (*int*) – Kilobits per second

schedule(*schedule*)

Set the throttling rule schedule

Parameters
schedule (cterasdk.common.types.TimeRange) – Schedule

upload(*kbps*)

Throttle bandwidth upstream

Parameters
kbps (*int*) – Kilobits per second

class cterasdk.common.types.**TimeRange**

Bases: object

Class representing a task schedule

build()

Build the time range

days(*days*)

Set days

Parameters
days (*list[cterasdk.common.enum.DayOfWeek]*) – A list of days

end(*end*)

End time

Parameters
end (*str*) – A military time string 'hh:mm:ss' or a datetime object

start(*start*)

Start time

Parameters
start (*str*) – A military time string 'hh:mm:ss' or a datetime object

terminate_at_endtime()

Terminate at end time, defaults to teminate on completion.

3.1.3 cterasdk.convert package

3.1.3.1 Submodules

cterasdk.convert.exception module

exception cterasdk.convert.exception.**ParseException**

Bases: Exception

cterasdk.convert.format module

cterasdk.convert.format.**CreateElement**(*parent*, *tag*)

cterasdk.convert.format.**tojsonstr**(*obj*, *pretty_print=True*, *no_log=True*)

Convert a Python object to a JSON string.

Parameters

- **obj** (*object*) – the Python object
- **pretty_print** (*bool*) – Whether to format the JSON string, defaults to True
- **no_log** (*book*) – Hide sensitive values in the log messages

Returns
JSON string of the object

Return type
str

cterasdk.convert.format.**toxml**(*obj*)

cterasdk.convert.format.**toxmlstr**(*obj*, *pretty_print=False*)

Convert a Python object to an XML string

Parameters

- **obj** (*object*) – the Python object
- **pretty_print** (*bool*) – whether to format the XML string, defaults to False

Returns
XML string of the object

Return type
str

cterasdk.convert.parse module

cterasdk.convert.parse.**ParseValue**(*data*)

cterasdk.convert.parse.**SetAppendValue**(*item*, *value*)

cterasdk.convert.parse.**fromjsonstr**(*fromstr*)

cterasdk.convert.parse.**fromxmlstr**(*string*)

cterasdk.convert.xml_types module

class cterasdk.convert.xml_types.**XMLTypes**
Bases: object

ATT = 'att'

CLASS = 'class'

DB = 'db'

FIRMWARE = 'firmware'

ID = 'id'

LIST = 'list'

LOCATION = 'xsi:noNamespaceSchemaLocation'

NS = 'xmlns:xsi'

OBJ = 'obj'

UUID = 'uuid'

VAL = 'val'

VERSION = 'version'

3.1.4 cterasdk.core package

3.1.4.1 Subpackages

cterasdk.core.files package

Submodules

cterasdk.core.files.browser module

class cterasdk.core.files.browser.**Backups**(*portal*, *base_path*)

Bases: *FileBrowser*

Backups File Browser APIs

device_config(*device*, *destination=None*)

Download a device configuration file

Parameters

- **device** (*str*) – The device name
- **destination** (*str,optional*) – File destination, if it is a directory, the original filename will be kept, defaults to the default directory

class cterasdk.core.files.browser.**CloudDrive**(*portal*, *base_path*)

Bases: *FileBrowser*

Cloud Drive File Browser APIs

add_share_recipients(*path*, *recipients*)

Add share recipients

Parameters

- **path** (*str*) – The path of the file or folder
- **recipients** (*list[cterasdk.core.types.ShareRecipient]*) – A list of share recipients

Returns
A list of all recipients added

Return type
list[*cterasdk.core.types.ShareRecipient*]

delete(*path*)

Delete a file

Parameters
path (*str*) – Path of the file or directory to delete

delete_multi(**args*)

Delete multiple files and/or directories

Parameters
***args** – Variable lengthed list of paths of files and/or directories to delete

get_share_info(*path*)

Get share settings and recipients

mkdir(*path*, *recurse=False*)

Create a new directory

Parameters

- **path** (*str*) – Path of the directory to create
- **recurse** (*bool,optional*) – Whether to create the path recursivly, defaults to False

move(*src*, *dest*)

Move a file or directory

Parameters

- **src** (*str*) – The source path of the file or directory
- **dst** (*str*) – The destination path of the file or directory

move_multi(*src*, *dest*)

remove_share_recipients(*path*, *accounts*)

Remove share recipients

Parameters

- **path** (*str*) – The path of the file or folder
- **accounts** (*list[cterasdk.core.types.PortalAccount]*) – A list of portal user or group accounts

Returns
A list of all share recipients removed

Return type
list[*cterasdk.core.types.PortalAccount*]

rename(*path*, *name*)

Rename a file

Parameters

- **path** (*str*) – Path of the file or directory to rename
- **name** (*str*) – The name to rename to

share(*path*, *recipients*, *as_project=True*, *allow_reshare=True*, *allow_sync=True*)

Share a file or a folder

Parameters

- **path** (*str*) – The path of the file or folder to share
- **recipients** (*list[cterasdk.core.types.ShareRecipient]*) – A list of share recipients
- **as_project** (*bool,optional*) – Share as a team project, defaults to True when the item is a cloud folder else False
- **allow_reshare** (*bool,optional*) – Allow recipients to re-share this item, defaults to True
- **allow_sync** (*bool,optional*) – Allow recipients to sync this item, defaults to True when the item is a cloud folder else False

Returns
A list of all recipients added to the collaboration share

Return type
list[*cterasdk.core.types.ShareRecipient*]

undelete(*path*)

Restore a previously deleted file or directory

Parameters
path (*str*) – Path of the file or directory to restore

undelete_multi(**args*)

Restore previously deleted multiple files and/or directories

Parameters

***args** – Variable length list of paths of files and/or directories to restore

unshare(*path*)

Unshare a file or a folder

upload(*file_path*, *server_path*)

Upload a file

Parameters

- **file_path** (*str*) – Path to the local file to upload
- **server_path** (*str*) – Path to the directory to upload the file to

class cterasdk.core.files.browser.**FileBrowser**(*portal*, *base_path*)

Bases: *BaseCommand*

Portal File Browser APIs

copy(*src*, *dest*)

Copy a file or directory

Parameters

- **src** (*str*) – The source path of the file or directory
- **dst** (*str*) – The destination path of the file or directory

copy_multi(*src*, *dest*)

download(*path*, *destination=None*)

Download a file

Parameters

- **path** (*str*) – Path of the file to download
- **destination** (*str,optional*) – File destination, if it is a directory, the original filename will be kept, defaults to the default directory

download_as_zip(*cloud_directory*, *files*, *destination=None*)

Download a list of files and/or directories from a cloud folder as a ZIP file

> **Warning:** The list of files is not validated. The ZIP file will include only the existing files and directories

Parameters

- **cloud_directory** (*str*) – Path to the cloud directory
- **files** (*list[str]*) – List of files and/or directories in the cloud folder to download
- **destination** (*str,optional*) – File destination, if it is a directory, the original filename will be kept, defaults to the default directory

ls(*path*, *include_deleted=False*)

Execute ls on the provided path

Parameters

- **path** (*str*) – Path to list
- **include_deleted** (*bool,optional*) – Include deleted files, defaults to False

mklink(*path*, *access='RO'*, *expire_in=30*)

Create a link to a file

Parameters

- **path** (*str*) – The path of the file to create a link to
- **access** (*str,optional*) – Access policy of the link, defaults to 'RO'
- **expire_in** (*int,optional*) – Number of days until the link expires, defaults to 30

mkpath(*array*)

walk(*path*, *include_deleted=False*)

Perform walk on the provided path

Parameters

- **path** (*str*) – Path to perform walk on
- **include_deleted** (*bool,optional*) – Include deleted files, defaults to False

cterasdk.core.files.collaboration module

cterasdk.core.files.collaboration.**add_share_recipients**(*ctera_host*, *path*, *recipients*)

cterasdk.core.files.collaboration.**get_share_info**(*ctera_host*, *path*)

cterasdk.core.files.collaboration.**remove_share_recipients**(*ctera_host*, *path*, *accounts*)

cterasdk.core.files.collaboration.**share**(*ctera_host*, *path*, *recipients*, *as_project*, *allow_reshare*, *allow_sync*)

cterasdk.core.files.collaboration.**unshare**(*ctera_host*, *path*)

cterasdk.core.files.common module

class cterasdk.core.files.common.**ActionResourcesParam**

Bases: *Object*

add(*param*)

static instance()

class cterasdk.core.files.common.**CreateShareParam**(*path*, *access*, *expire_on*)

Bases: *Object*

static instance(*path*, *access*, *expire_on*)

class cterasdk.core.files.common.**SrcDstParam**(*src*, *dest=None*)

Bases: *Object*

static instance(*src*, *dest=None*)

cterasdk.core.files.common.**get_resource_info**(*ctera_host*, *path*)

cterasdk.core.files.cp module

cterasdk.core.files.cp.**copy**(*ctera_host*, *src*, *dest*)

cterasdk.core.files.cp.**copy_multi**(*ctera_host*, *src*, *dest*)

cterasdk.core.files.directory module

exception cterasdk.core.files.directory.**InvalidName**(*message=None*, *instance=None*, ***kwargs*)

Bases: *CTERAException*

exception cterasdk.core.files.directory.**InvalidPath**(*message=None*, *instance=None*, ***kwargs*)

Bases: *CTERAException*

exception cterasdk.core.files.directory.**ItemExists**(*message=None*, *instance=None*, ***kwargs*)

Bases: *CTERAException*

exception cterasdk.core.files.directory.**ReservedName**(*message=None*, *instance=None*, ***kwargs*)

Bases: *CTERAException*

cterasdk.core.files.directory.**mkdir**(*ctera_host*, *path*, *recurse=False*)

cterasdk.core.files.file_access module

class cterasdk.core.files.file_access.**FileAccess**(*ctera_host*)

Bases: *FileAccessBase*

cterasdk.core.files.ln module

cterasdk.core.files.ln.**mklink**(*ctera_host*, *path*, *access*, *expire_in*)

cterasdk.core.files.ls module

cterasdk.core.files.ls.**fetch_resources**(*ctera_host*, *param*)

cterasdk.core.files.ls.**list_dir**(*ctera_host*, *param*)

cterasdk.core.files.ls.**ls**(*ctera_host*, *path*, *depth=1*, *include_deleted=False*)

cterasdk.core.files.mv module

cterasdk.core.files.mv.**move**(*ctera_host*, *src*, *dest*)

cterasdk.core.files.mv.**move_multi**(*ctera_host*, *src*, *dest*)

cterasdk.core.files.path module

class cterasdk.core.files.path.**CTERAPath**(*item*, *basepath*)

Bases: object

encoded_fullpath()

encoded_parent()

fullpath()

joinpath(*path*)

name()

parent()

parts()

cterasdk.core.files.recover module

cterasdk.core.files.recover.**undelete**(*ctera_host*, *path*)

cterasdk.core.files.recover.**undelete_multi**(*ctera_host*, **paths*)

cterasdk.core.files.rename module

cterasdk.core.files.rename.**rename**(*ctera_host*, *path*, *name*)

cterasdk.core.files.rm module

cterasdk.core.files.rm.**delete**(*ctera_host*, *path*)

cterasdk.core.files.rm.**delete_multi**(*ctera_host*, **paths*)

3.1.4.2 Submodules

cterasdk.core.activation module

class cterasdk.core.activation.**Activation**(*portal*)

Bases: *BaseCommand*

Portal activation

generate_code(*username=None*, *tenant=None*)

Generate device activation code

Parameters

- **username** (*str,optional*) – User name used for activation, defaults to None
- **tenant** (*str,optional*) – Tenant name used for activation, defaults to None

Returns
Portal Activation Code

Return type
str

cterasdk.core.antivirus module

class cterasdk.core.antivirus.**Antivirus**(*portal*)

Bases: *BaseCommand*

Portal Antivirus APIs

Variables
servers (cterasdk.core.antivirus.AntivirusServers) – Object holding the Portal antivirus server APIs

default = ['name', 'type']

list_servers(*include=None*)

List the antivirus servers

Parameters
include (*list[str],optional*) – List of fields to retrieve, defaults to ['name', 'type']

rescan()

Scan all files using the latest antivirus update. This may take a while

status()

Get antivirus service status

suspend()

Suspend antivirus scanning

unsuspend()

Unsuspend antivirus scanning

class cterasdk.core.antivirus.**AntivirusServers**(*portal*)

Bases: *BaseCommand*

add(*name*, *vendor*, *url*, *connection_timeout=5*)

Add an antivirus server

Parameters

- **name** (*str*) – Server name
- **vendor** (cterasdk.core.enum.AntivirusType) – Server type
- **url** (*str*) – Server URL (example: http://your-antivirus.server.local:1234/signature)

- **connection_timeout** (*int,optional*) – Server connection timeout (in seconds), defaults to 5 seconds

delete(*name*)

Remove an antivirus server

get(*name*)

Get an antivirus server's configuration

Parameters
name (*str*) – Server name

suspend(*name*)

Suspend an antivirus server

unsuspend(*name*)

Unsuspend antivirus scanning

cterasdk.core.base_command module

class cterasdk.core.base_command.**BaseCommand**(*portal*)

Bases: object

Base class for all Portal API classes

session()

cterasdk.core.buckets module

class cterasdk.core.buckets.**Buckets**(*portal*)

Bases: *BaseCommand*

Portal Storage Node APIs

add(*name*, *bucket*, *read_only=False*, *dedicated_to=None*)

Add a Bucket

Parameters

- **name** (*str*) – Name of the bucket
- **bucket** (cterasdk.core.types.Bucket) – Storage bucket to add
- **read_only** (*bool,optional*) – Set bucket to read-delete only, defaults to False
- **dedicated_to** (*str,optional*) – Name of a tenant, defaults to None

default = ['name']

delete(*name*)

Delete a Bucket

Parameters
name (*str*) – Name of the bucket

get(*name*, *include=None*)

Get a Bucket

Parameters

- **name** (*str*) – Name of the bucket
- **include** (*list[str]*) – List of fields to retrieve, defaults to `['name']`

list_buckets(*include=None*)

List Buckets.

To retrieve buckets, you must first browse the Global Administration Portal, using: *GlobalAdmin.portals.browse_global_admin()*

Parameters

include (*list[str],optional*) – List of fields to retrieve, defaults to `['name']`

modify(*current_name*, *new_name=None*, *read_only=None*, *dedicated_to=None*)

Modify a Bucket

Parameters

- **current_name** (*str*) – The current bucket name
- **new_name** (*str,optional*) – New name
- **read_only** (*bool,optional*) – Set bucket to read-delete only
- **dedicated** (*bool,optional*) – Dedicate bucket to a tenant
- **dedicated_to** (*str,optional*) – Tenant name

read_only(*name*)

Set bucket to Read Only

Parameters

name (*str*) – Name of the bucket

read_write(*name*)

Set bucket to Read Write

Parameters

name (*str*) – Name of the bucket

cterasdk.core.cli module

class cterasdk.core.cli.**CLI**(*portal*)

Bases: *BaseCommand*

Portal CLI APIs

run_command(*cli_command*)

Run a CLI command

Parameters

cli_command (*str*) – The CLI command to run on the gateway

Return str

The response of the Portal

cterasdk.core.cloudfs module

class cterasdk.core.cloudfs.**CloudFS**(*portal*)

Bases: *BaseCommand*

CloudFS APIs

default = ['name', 'group', 'owner']

delete(*name*, *owner*)

Delete a Cloud Drive Folder

Parameters

- **name** (*str*) – Name of the Cloud Drive Folder to delete
- **owner** (cterasdk.core.types.UserAccount) – User account, the owner of the Cloud Drive Folder to delete

find(*name*, *owner*, *include=None*)

Find a Cloud Drive Folder

Parameters

- **name** (*str*) – Name of the Cloud Drive Folder to find
- **owner** (cterasdk.core.types.UserAccount) – User account of the folder group owner
- **include** (*list[str]*) – List of metadata fields to include in the response

Returns

A Cloud Drive Folder

get(*name*, *include=None*)

Get folder group

Parameters

- **name** (*str*) – Name of the Folder Group to find
- **include** (*str,optional*) – List of fields to retrieve, defaults to [‘name’, ‘owner’]

list_folder_groups(*include=None*, *user=None*)

List folder groups

Parameters

- **include** (*str,optional*) – List of fields to retrieve, defaults to [‘name’, ‘owner’]
- **user** (cterasdk.core.types.UserAccount) – User account of the folder group owner

Returns

Iterator for all folder groups

list_folders(*include=None*, *list_filter='NonDeleted'*, *user=None*)

List Cloud Drive folders.

Parameters

- **include** (*str,optional*) – List of fields to retrieve, defaults to [‘name’, ‘group’, ‘owner’]
- **list_filter** (cterasdk.core.enum.ListFilter) – Filter the list of Cloud Drive folders, defaults to non-deleted folders
- **user** (cterasdk.core.types.UserAccount) – User account of the cloud folder owner

Returns
Iterator for all Cloud Drive folders

mkdir(*name*, *group*, *owner*, *winacls=True*, *description=None*)

Create a new directory

Parameters

- **name** (*str*) – Name of the new directory
- **group** (*str*) – The Folder Group to which the directory belongs
- **owner** (cterasdk.core.types.UserAccount) – User account, the owner of the new directory
- **winacls** (*bool,optional*) – Use Windows ACLs, defaults to True
- **description** (*str,optional*) – Cloud drive folder description

mkfg(*name*, *user=None*, *deduplication_method_type=None*, *storage_class=None*)

Create a new Folder Group

Parameters

- **name** (*str*) – Name of the new folder group
- **user** (cterasdk.core.types.UserAccount) – User account, the user directory and name of the new folder group owner (default to None)
- **deduplication_method_type** (cterasdk.core.enum.DeduplicationMethodType) – Deduplication-Method
- **storage_class** (*str,optional*) – Storage class, defaults to the Default storage class

rmfg(*name*)

Remove a Folder Group

Parameters
name (*str*) – Name of the folder group to remove

set_folders_acl(*folders_path*, *sddl_string*, *is_recursive=False*)

Changing the file or Folder ACLs

Parameters

- **folders_path** (*list*) – A list of paths
- **sddl_string** (*str*) – The SDDL string with the ACL permissions
- **is_recursive** (*bool*) – For path that is not a file but a folder

Returns
execution response

set_owner_acl(*folders_path*, *owner_sid*, *is_recursive=False*)

Changing the File or Folder Owner SID or ACLs

Parameters

- **folders_path** (*list*) – A list of paths
- **owner_sid** (*str*) – The SID string that identifies the object's owner.
- **is_recursive** (*bool*) – If the path is not a file but a folder

Returns
execution response

undelete(*name*, *owner*)

Un-Delete a Cloud Drive Folder

Parameters

- **name** (*str*) – Name of the Cloud Drive Folder to un-delete
- **owner** (cterasdk.core.types.UserAccount) – User account, the owner of the Cloud Drive Folder to delete

cterasdk.core.connection module

cterasdk.core.connection.**test**(*CTERAHost*)

cterasdk.core.connection.**test_network**(*CTERAHost*)

cterasdk.core.decorator module

cterasdk.core.decorator.**update_current_tenant**(*function*)

cterasdk.core.devices module

class cterasdk.core.devices.**Devices**(*portal*)

Bases: *BaseCommand*

Portal Devices APIs

agents(*include=None*, *allPortals=False*)

Get Agents

Parameters

- **include** (*list[str],optional*) – List of fields to retrieve, defaults to [‘name’, ‘portal’, ‘deviceType’]
- **allPortals** (*bool,optional*) – Search in all portals, defaults to False

Returns
Iterator for all matching Agents

Return type
cterasdk.lib.iterator.Iterator[*cterasdk.object.Agent.Agent*]

by_name(*names*, *include=None*)

Get Devices by their names

Parameters

- **names** (*list[str],optional*) – List of names of devices
- **include** (*list[str],optional*) – List of fields to retrieve, defaults to [‘name’, ‘portal’, ‘deviceType’]

Returns
Iterator for all matching Devices

Return type
cterasdk.lib.iterator.Iterator

`default = ['name', 'portal', 'deviceType']`

`desktops`(*include=None*, *allPortals=False*)

Get Desktops

Parameters

- **include** (*list[str],optional*) – List of fields to retrieve, defaults to ['name', 'portal', 'deviceType']
- **allPortals** (*bool,optional*) – Search in all portals, defaults to False

Returns
Iterator for all matching Desktops

Return type
cterasdk.lib.iterator.Iterator

`device`(*device_name*, *tenant=None*, *include=None*)

Get a Device by its name

Parameters

- **device_name** (*str*) – Name of the device to retrieve
- **tenant** (*str,optional*) – Tenant of the device, defaults to the tenant in the current session
- **include** (*list[str],optional*) – List of fields to retrieve, defaults to ['name', 'portal', 'deviceType']

Returns
Managed Device

Return type
ctera.object.Gateway.Gateway or ctera.object.Agent.Agent

`devices`(*include=None*, *allPortals=False*, *filters=None*, *user=None*)

Get Devices

Parameters

- **include** (*list[str],optional*) – List of fields to retrieve, defaults to ['name', 'portal', 'deviceType']
- **allPortals** (*bool,optional*) – Search in all portals, defaults to False
- **filters** (*list[],optional*) – List of additional filters, defaults to None
- **user** (`cterasdk.core.types.UserAccount`) – User account of the device owner

Returns
Iterator for all matching Devices

Return type
cterasdk.lib.iterator.Iterator

`filers`(*include=None*, *allPortals=False*, *deviceTypes=None*)

Get Filers

Parameters

- **include** (*list[str],optional*) – List of fields to retrieve, defaults to ['name', 'portal', 'deviceType']
- **allPortals** (*bool,optional*) – Search in all portals, defaults to False
- **deviceTypes** (*list[cterasdk.core.enum.DeviceType.Gateways]*) – Types of Filers, defaults to all Filer types

Returns
Iterator for all matching Filers

Return type
cterasdk.lib.iterator.Iterator[*cterasdk.object.Gateway.Gateway*]

get_comment(*device_name*, *tenant=None*)

Get Portal device comment

Parameters
device (*str*) – Device name

Returns
Comment

Return type
str

name_attr = 'name'

servers(*include=None*, *allPortals=False*)

Get Servers

Parameters

- **include** (*list[str],optional*) – List of fields to retrieve, defaults to ['name', 'portal', 'deviceType']
- **allPortals** (*bool,optional*) – Search in all portals, defaults to False

Returns
Iterator for all matching Servers

Return type
cterasdk.lib.iterator.Iterator

set_comment(*device_name*, *comment*, *tenant=None*)

Set a comment to a Portal device

Parameters

- **device** (*str*) – Device name
- **comment** (*str*) – Comment

type_attr = 'deviceType'

cterasdk.core.directoryservice module

class cterasdk.core.directoryservice.**DirectoryService**(*portal*)

Bases: *BaseCommand*

Portal Active Directory APIs

connect(*domain*, *username*, *password*, *directory='ActiveDirectory'*, *domain_controllers=None*, *ou=None*, *ssl=False*, *krb=False*, *mapping=None*, *acl=None*, *default='Disabled'*, *fetch='Lazy'*)

Connect a Portal tenant to directory services

Parameters

- **domain** (*str*) – The directory services domain to connect to
- **username** (*str*) – The user name to use when connecting to the active directory services
- **password** (*str*) – The password to use when connecting to the active directory services
- **ou** (*str,optional*) – The OU path to use when connecting to the active directory services, defaults to *None*
- **directory** (cterasdk.core.enum.DirectoryServiceType,*optional*) – The directory service type, deafults to *'ActiveDirectory'*
- **domain_controllers** (cterasdk.core.types.DomainControllers,*optional*) – Connect to a primary and a secondary domain controllers, defaults to *None*
- **ssl** (*bool,optional*) – Connect using SSL, defaults to *False*
- **krb** (*bool,optional*) – Connect using Kerberos, defaults to *False*
- **list[cterasdk.common.types.ADDomainIDMapping],optional** – The directory services UID/GID mapping
- **acl** (*list[*cterasdk.core.types.AccessControlEntry*],optional*) – List of access control entries and their associated roles
- **default** (cterasdk.core.enum.Role) – Default role if no match applies, defaults to *None*
- **fetch** (*str,optional*) – Configure identity fetching mode, defaults to *'Lazy'*

connected()

disconnect()

Disconnect a Portal tenant from directory services

domains()

Get domains

Return list(str)
List of names of all discovered domains

fetch(*active_directory_accounts*)

Instruct the Portal to fetch the provided Active Directory Accounts

Parameters
active_directory_accounts (*list[*cterasdk.core.types.PortalAccount*]*) – List of Active Directory Accounts to fetch

Returns
Response Code

get_access_control()

Retrieve directory services access control list

Returns
List of access control entries

Return type
list[*cterasdk.core.types.AccessControlEntry*]

get_advanced_mapping()

Retrieve directory services advanced mapping configuration

Returns
A dictionary of domain mapping objects

Return type
dict

get_connected_domain()

Get the connected domain information. Returns *None* if the Portal tenant is not connected to a domain

Return str
The connected domain

get_default_role()

Retrieve the default role assigned when no access control entry match was found

set_access_control(*acl=None*, *default=None*)

Configure directory services access control

Parameters

- **acl** (*list[cterasdk.core.types.AccessControlEntry],optional*) – List of access control entries and their associated roles
- **default** (*cterasdk.core.enum.Role*) – Default role if no match applies, defaults to *None*

set_advanced_mapping(*mapping*)

Configure advanced mapping

Parameters
mapping (*list[cterasdk.common.types.ADDomainIDMapping]*) – List of domains and their UID/GID mapping range

cterasdk.core.enum module

class cterasdk.core.enum.**AntivirusType**

Bases: object

Antivirus Type

Variables

- **McAfeeWG** (*str*) – McAfee Web Gateway
- **Symantec** (*str*) – Symantec Protection Engine
- **ESET** (*str*) – ESET Gateway Security
- **Sophos** (*str*) – Sophos AV

- **McAfeeVSES** (*str*) – McAfee VirusScan Enterprise for Storage
- **TrendMicro** (*str*) – Trend Micro InterScan

ESET = 'Eset'

McAfeeVSES = 'McAfeeVSES'

McAfeeWG = 'McAfee'

Sophos = 'Sophos'

Symantec = 'Symantec'

TrendMicro = 'TrendMicro'

class cterasdk.core.enum.**BucketType**

Bases: object

Bucket Type

Variables

- **Azure** (*str*) – Azure
- **Scality** (*str*) – Scality
- **AWS** (*str*) – Amazon Web Services S3
- **ICOS** (*str*) – IBM Cloud Object Storage
- **GenericS3** (*str*) – Generic S3
- **Nutanix** (*str*) – Nutanix S3
- **Wasabi** (*str*) – Wasabi S3
- **Google** (*str*) – Google S3
- **NetAppStorageGRID** (*str*) – NetApp StorageGRID WebScale (S3)

AWS = 'S3'

Azure = 'Azure'

GenericS3 = 'GenericS3'

Google = 'GoogleS3'

ICOS = 'CleverSafeS3'

NetAppStorageGRID = 'NTAP'

Nutanix = 'Nutanix'

Scality = 'ScalityS3'

Wasabi = 'WasabiS3'

class cterasdk.core.enum.**CollaboratorType**

Bases: object

Collaborator Type

Variables

- **LU** (*str*) – Local User
- **DU** (*str*) – Domain User
- **LG** (*str*) – Local Group
- **DG** (*str*) – Domain Group
- **EXT** (*str*) – External

DG = 'adGroup'

DU = 'adUser'

EXT = 'external'

LG = 'localGroup'

LU = 'localUser'

class cterasdk.core.enum.**Context**

Bases: object

Portal connection context

Variables

- **admin** (*str*) – Global admin context
- **ServicesPortal** (*str*) – Services Portal context

ServicesPortal = 'ServicesPortal'

admin = 'admin'

class cterasdk.core.enum.**DeduplicationMethodType**

Bases: object

Folder Group Deduplication Method Type

Variables

- **AverageBlockSize** (*str*) – AverageBlockSize
- **FixedBlockSize** (*str*) – FixedBlockSize

AverageBlockSize = 'AverageBlockSize'

FixedBlockSize = 'FixedBlockSize'

class cterasdk.core.enum.**DeviceType**

Bases: object

Device type

Variables

- **CloudPlug** (*str*) – Cloud Plug device
- **C200** (*str*) – C200 device
- **C400** (*str*) – C400 device
- **C800** (*str*) – C800 device
- **C800P** (*str*) – C800P device

- **vGateway** (*str*) – vGateway device
- **ServerAgent** (*str*) – Server Agent device
- **WorkstationAgent** (*str*) – Workstation Agent Agent device
- **Gateways** (*list[str]*) – List of all the Gateway DeviceTypes
- **Agents** (*list[str]*) – List of all the Agents DeviceTypes

Agents = ['Server Agent', 'Workstation Agent']

C200 = 'C200'

C400 = 'C400'

C800 = 'C800'

C800P = 'C800P'

CloudPlug = 'CloudPlug'

Gateways = ['CloudPlug', 'C200', 'C400', 'C800', 'C800P', 'vGateway']

ServerAgent = 'Server Agent'

WorkstationAgent = 'Workstation Agent'

vGateway = 'vGateway'

class cterasdk.core.enum.**DirectorySearchEntityType**

Bases: object

Directory Search Entity Type

Variables

- **User** (*str*) – User
- **Group** (*str*) – Group

Group = 'group'

User = 'user'

class cterasdk.core.enum.**DirectoryServiceFetchMode**

Bases: object

Directory Service Fetch Mode

Variables

- **Eager** (*str*) – Eager
- **Lazy** (*str*) – Lazy

Eager = 'Eager'

Lazy = 'Lazy'

class cterasdk.core.enum.**DirectoryServiceType**

Bases: object

Directory Service Type

Variables

- **Microsoft** (*str*) – Active Directory
- **LDAP** (*str*) – LDAP
- **Apple** (*str*) – Apple Open Directory

Apple = 'AppleOpenDirectory'

LDAP = 'LDAP'

Microsoft = 'ActiveDirectory'

class cterasdk.core.enum.**EnvironmentVariables**

Bases: object

Environment Variables.

Some environment variables are applicable across platforms (i.e. Windows, Linux), while others are limited to a designated platform

Variables

- **ALLUSERSPROFILE** (*str*) – All users profile
- **WINDIR** (*str*) – Windows directory
- **TEMP** (*str*) – Temp directory
- **SYSTEMDRIVE** (*str*) – System drive
- **PROGRAMFILES** (*str*) – Program files
- **APPDATA** (*str*) – Application data
- **USERPROFILE** (*str*) – Current user profile
- **PRIMARYUSER** (*str*) – Primary user
- **USERS** (*str*) – Users directory (CTERA Edge Filer)
- **AGENTS** (*str*) – Agents directory (CTERA Edge Filer)
- **SYNCS** (*str*) – Syncs directory (CTERA Edge Filer)
- **PROJECTS** (*str*) – Projects directory (CTERA Edge Filer)

AGENTS = '$AGENTS'

ALLUSERSPROFILE = '$ALLUSERSPROFILE'

APPDATA = '$APPDATA'

PRIMARYUSER = '$PRIMARYUSER'

PROGRAMFILES = '$PROGRAMFILES'

PROJECTS = '$PROJECTS'

SYNCS = '$SYNCS'

SYSTEMDRIVE = '$SYSTEMDRIVE'

TEMP = '$TEMP'

USERPROFILE = '$USERPROFILE'

USERS = '$USERS'

WINDIR = '$WINDIR'

class cterasdk.core.enum.FileAccessMode

Bases: object

Share Access Mode

Variables

- **RW** (*str*) – Read Write
- **RO** (*str*) – Read Only
- **PO** (*str*) – Preview Only
- **NA** (*str*) – None

NA = 'None'

PO = 'PreviewOnly'

RO = 'ReadOnly'

RW = 'ReadWrite'

class cterasdk.core.enum.ICAPServices

Bases: object

ICAP Services

Variables

- **Antivirus** (*str*) – Antivirus
- **DLP** (*str*) – Data Loss Prevention

Antivirus = 'Antivirus'

DLP = 'DLP'

class cterasdk.core.enum.IPProtocol

Bases: object

IP Protocol

Variables

- **TCP** (*str*) – TCP Protocol
- **UDP** (*str*) – UDP Protocol

TCP = 'TCP'

UDP = 'UDP'

class cterasdk.core.enum.**ListFilter**

Bases: object

Cloud Drive Folder List Filter

Variables

- **All** (*str*) – All
- **Deleted** (*str*) – Deleted
- **NonDeleted** (*str*) – NonDeleted

All = 'All'

Deleted = 'Deleted'

NonDeleted = 'NonDeleted'

class cterasdk.core.enum.**LocationType**

Bases: object

Location Type

Variables

- **Azure** (*str*) – Azure Blob Storage
- **S3** (*str*) – Amazon Web Services S3
- **S3Compatible** (*str*) – S3 Compatible
- **NetAppStorageGRID** (*str*) – NetApp StorageGRID WebScale (S3)

Azure = 'AzureLocation'

NetAppStorageGRID = 'NetAppLocation'

S3 = 'S3Location'

S3Compatible = 'S3Compatible'

class cterasdk.core.enum.**LogTopic**

Bases: object

Portal Log Topic

Variables

- **System** (*str*) – System log topic
- **CloudBackup** (*str*) – Cloud Backup log topic
- **CloudSync** (*str*) – Cloud Sync log topic
- **Access** (*str*) – Access log topic
- **Audit** (*str*) – Audit log topic

Access = 'access'

Audit = 'audit'

CloudBackup = 'backup'

CloudSync = 'cloudsync'

System = 'system'

class cterasdk.core.enum.Mode

Bases: object

Enum for operational mode

Variables

- **Enabled** (*str*) – Operational mode enabled
- **Disabled** (*str*) – Operational mode diabled

Disabled = 'disabled'

Enabled = 'enabled'

class cterasdk.core.enum.OriginType

Bases: object

Log Origin Type

Variables

- **Portal** (*str*) – Portal originated logs
- **Device** (*str*) – Device originated logs

Device = 'Device'

Portal = 'Portal'

class cterasdk.core.enum.PlanCriteria

Bases: object

Subscription Plan Auto Assignment Rule Builder Criterias

Variables

- **Username** (*str*) – Username
- **Groups** (*str*) – User groups
- **Role** (*str*) – User role
- **First** (*str*) – User first name
- **Last** (*str*) – User last name
- **Company** (*str*) – User company
- **BillingId** (*str*) – User billing id
- **Comment** (*str*) – User comment

BillingId = 'billingId'

Comment = 'comment'

Company = 'company'

First = 'firstName'

Groups = 'userGroups'

Last = 'lastName'

Role = 'role'

Username = 'username'

class cterasdk.core.enum.**PlanItem**

Bases: object

Portal plan item

Variables

- **EV4** (*str*) – EV4
- **EV8** (*str*) – EV8
- **EV16** (*str*) – EV16
- **EV32** (*str*) – EV32
- **EV64** (*str*) – EV64
- **EV128** (*str*) – EV128
- **WA** (*str*) – Workstation Agent
- **SA** (*str*) – Server Agent
- **Share** (*str*) – Cloud Drive
- **Connect** (*str*) – Cloud Drive Connect

Connect = 'Connect'

EV128 = 'EV128'

EV16 = 'EV16'

EV32 = 'EV32'

EV4 = 'EV4'

EV64 = 'EV64'

EV8 = 'EV8'

SA = 'SA'

Share = 'Share'

Storage = 'Storage'

WA = 'WA'

class cterasdk.core.enum.**PlanRetention**

Bases: object

Portal plan retention policy

Variables

- **All** (*str*) – All versions

- **Hourly** (*str*) – Hourly versions
- **Daily** (*str*) – Daily versions
- **Weekly** (*str*) – Weekly versions
- **Monthly** (*str*) – Monthly versions
- **Quarterly** (*str*) – Quarterly versions
- **Yearly** (*str*) – Yearly versions
- **Deleted** (*str*) – Recycle bin

All = 'retainAll'

Daily = 'daily'

Deleted = 'retainDeleted'

Hourly = 'hourly'

Monthly = 'monthly'

Quarterly = 'quarterly'

Weekly = 'weekly'

Yearly = 'yearly'

class cterasdk.core.enum.**Platform**

Bases: object

CTERA Edge Platform Type.

Variables

- **C200_Orion** (*str*) – All users profile
- **C200_ARM** (*str*) – Windows directory
- **C200_Kirkwood** (*str*) – Temp directory
- **C400_C800** (*str*) – System drive
- **Edge_6** (*str*) – CTERA 6.0 Edge Filer
- **Edge_7** (*str*) – CTERA 7.0 Edge Filer
- **Windows** (*str*) – Windows Agent (Drive App)
- **Linux** (*str*) – Linux Agent (Drive App)
- **OSX** (*str*) – Mac Agent (Drive App)

C200_ARM = 'ARM'

C200_Kirkwood = 'Kirkwood'

C200_Orion = 'Orion'

C400_C800 = 'X86'

Edge_6 = 'VBox'

Edge_7 = 'Genesis'

Linux = 'LinuxX86'

OSX = 'OSxX86'

Windows = 'WindowsX86'

class cterasdk.core.enum.**PolicyType**

Bases: object

Zone Policy Type

Variables

- **ALL** (*str*) – All folders
- **SELECT** (*str*) – Selected Folders
- **NONE** (*str*) – No Folders

ALL = 'allFolders'

NONE = 'noFolders'

SELECT = 'selectedFolders'

class cterasdk.core.enum.**PortalAccountType**

Bases: object

Portal Account Type

Variables

- **User** (*str*) – User account type
- **Group** (*str*) – Group account type

Group = 'group'

User = 'user'

class cterasdk.core.enum.**PortalType**

Bases: object

Portal Type

Variables

- **Team** (*str*) – Team Portal
- **Reseller** (*str*) – Reseller Portal

Reseller = 'reseller'

Team = 'team'

class cterasdk.core.enum.**ProtectionLevel**

Bases: object

External Share Protection Level

Variables

- **publicLink** (*str*) – No authentication
- **email** (*str*) – 2FA via email

Email = 'email'

Public = 'publicLink'

class cterasdk.core.enum.**Role**

Bases: object

Portal User Role

Variables

- **Disabled** (*str*) – Disabled user role
- **EndUser** (*str*) – EndUser user role
- **ReadWriteAdmin** (*str*) – ReadWriteAdmin user role
- **ReadOnlyAdmin** (*str*) – ReadOnlyAdmin user role
- **Support** (*str*) – Support user role

Disabled = 'Disabled'

EndUser = 'EndUser'

ReadOnlyAdmin = 'ReadOnlyAdmin'

ReadWriteAdmin = 'ReadWriteAdmin'

Support = 'Support'

class cterasdk.core.enum.**SearchType**

Bases: object

Search Type

Variables

- **User** (*str*) – User search type
- **Group** (*str*) – Group search type

Groups = 'groups'

Users = 'users'

class cterasdk.core.enum.**ServerMode**

Bases: object

Portal Server Mode

Variables

- **Master** (*str*) – Master
- **Slave** (*str*) – Slave

Master = 'master'

Slave = 'slave'

class cterasdk.core.enum.**SetupWizardStage**

Bases: object

Portal Setup Wizard Stage

Variables

- **Server** (*str*) – Initializing Server
- **Portal** (*str*) – Initializing Portal
- **Replication** (*str*) – Setting Database Replication
- **Restart** (*str*) – Restarting Server
- **Finish** (*str*) – Finished

Finish = 'finish'

Portal = 'initPortal'

Replication = 'setReplication'

Restart = 'restartingServer'

Server = 'initServer'

class cterasdk.core.enum.**SetupWizardStatus**

Bases: object

Portal Setup Wizard Status

Variables

- **NA** (*str*) – Not Relevant
- **Running** (*str*) – In Progress
- **Failed** (*str*) – Failed
- **Completed** (*str*) – Completed

Completed = 'completed'

Failed = 'failed'

NA = 'notRelevant'

Running = 'inProgress'

class cterasdk.core.enum.**Severity**

Bases: object

Portal Log Severity

Variables

- **EMERGENCY** (*str*) – Emergency log severity
- **ALERT** (*str*) – Alert log severity
- **CRITICAL** (*str*) – Critical log severity
- **ERROR** (*str*) – Error log severity
- **WARNING** (*str*) – Warning log severity

- **NOTICE** (*str*) – Notice log severity
- **INFO** (*str*) – Info log severity
- **DEBUG** (*str*) – Debug log severity

ALERT = 'alert'

CRITICAL = 'critical'

DEBUG = 'debug'

EMERGENCY = 'emergency'

ERROR = 'error'

INFO = 'info'

NOTICE = 'notice'

WARNING = 'warning'

class cterasdk.core.enum.**SlaveAuthenticaionMethod**

Bases: object

Secondary Portal server authentication mode

Variables

- **Password** (*str*) – Password
- **PrivateKey** (*str*) – Private Key

Password = 'Password'

PrivateKey = 'Key'

class cterasdk.core.enum.**TemplateCriteria**

Bases: object

Configuration Template Auto Assignment Rule Builder Criterias

Variables

- **Type** (*str*) – Device type
- **OperatingSystem** (*str*) – Operating system
- **Version** (*str*) – Installed software version
- **Hostname** (*str*) – Hostname
- **Name** (*str*) – Device name
- **Owner** (*str*) – Device owner username
- **Plan** (*str*) – Plan name
- **Groups** (*str*) – Device owner local or domain groups

Groups = 'ownerGroups'

Hostname = 'Hostname'

Name = 'DeviceName'

OperatingSystem = 'OperatingSystem'

Owner = 'OwnerUsername'

Plan = 'Plan'

Type = 'DeviceType'

Version = 'InstalledSoftwareVersion'

cterasdk.core.login module

class cterasdk.core.login.**Login**(*portal*)

Bases: *BaseCommand*

Portal Login APIs

login(*username*, *password*)

Log into the portal

Parameters

- **username** (*str*) – User name to log in
- **password** (*str*) – User password

logout()

Log out of the portal

cterasdk.core.logs module

class cterasdk.core.logs.**Alerts**(*portal*)

Bases: *BaseCommand*

Portal Log Based Alerts APIs

add(*name*, *description=None*, *topic=None*, *log=None*, *min_severity=None*, *origin_type=None*, *content=None*)

Add a Log Based Alert

Parameters

- **name** (*str*) – Alert name
- **description** (*str,optional*) – Alert description
- **topic** (cterasdk.core.enum.LogTopic,*optional*) – Log topic to get, defaults to any topic
- **log** (*str,optional*) – Class name of the log
- **min_severity** (cterasdk.core.enum.Severity,*optional*) – Minimun severity for triggering an alert, defaults to any severity
- **origin_type** (cterasdk.core.enum.OriginType,*optional*) – Origin type of the log, defaults to any origin
- **content** (*str*) – Content of the log message

Returns
A list of alerts

Return type
list[*cterasdk.common.object.Object*]

`delete`(*name*)

Remove a Log Based Alert

Parameters
name (*str*) – Alert name

`get`()

Get a List of Log Based Alerts

Returns
A list of alerts

Return type
list[*cterasdk.common.object.Object*]

`put`(*alerts*)

Set Log Based Alerts
Use *cterasdk.core.types.AlertBuilder()* to build log based alerts`

Parameters
alerts (*list[cterasdk.core.types.Alert]*) – List of alerts

`class cterasdk.core.logs.Logs`(*portal*)

Bases: *BaseCommand*

Portal Logs APIs

Variables
alerts (cterasdk.core.logs.Alerts) – Object holding the Portal Log Based Alerts APIs

`device`(*name*, *topic='system'*, *min_severity='info'*, *before=None*, *after=None*, *filters=None*)

Get device logs from the Portal

Parameters

- **name** (*str*) – Name of a device
- **topic** (cterasdk.core.enum.LogTopic,*optional*) – Log topic to get, defaults to cterasdk.core.enum.LogTopic.System
- **min_severity** (cterasdk.core.enum.Severity,*optional*) – Minimun severity of logs to get, defaults to cterasdk.core.enum.Severity.INFO
- **before** (*str,optional*) – Get logs before this date (in format "%m/%d/%Y %H:%M:%S"), defaults to None
- **after** (*str,optional*) – Get logs after this date (in format "%m/%d/%Y %H:%M:%S"), defaults to None
- **filters** (*list[*cterasdk.core.query.FilterBuilder*],optional*) – List of additional filters, defaults to None

Returns
Iterator for all matching logs

Return type
cterasdk.lib.iterator.Iterator[cterasdk.object.Object]

get(*topic='system'*, *min_severity='info'*, *origin_type='Portal'*, *origin=None*, *before=None*, *after=None*, *filters=None*)

Get logs from the Portal

Parameters

- **topic** (*cterasdk.core.enum.LogTopic,optional*) – Log topic to get, defaults to cterasdk.core.enum.LogTopic.System
- **min_severity** (*cterasdk.core.enum.Severity,optional*) – Minimun severity of logs to get, defaults to cterasdk.core.enum.Severity.INFO
- **origin_type** (*cterasdk.core.enum.OriginType,optional*) – Origin type of the logs to get, defaults to cterasdk.core.enum.OriginType.Portal
- **origin** (*str,optional*) – Log origin (e.g. device name, Portal server name), defaults to None
- **before** (*str,optional*) – Get logs before this date (in format "%m/%d/%Y %H:%M:%S"), defaults to None
- **after** (*str,optional*) – Get logs after this date (in format "%m/%d/%Y %H:%M:%S"), defaults to None
- **filters** (*list[cterasdk.core.query.FilterBuilder],optional*) – List of additional filters, defaults to None

Returns
Iterator for all matching logs

Return type
cterasdk.lib.iterator.Iterator[cterasdk.object.Object]

cterasdk.core.messaging module

class cterasdk.core.messaging.**Messaging**(*portal*)

Bases: *BaseCommand*

Portal Messaging Service Management APIs

add(*servers*)

Add messaging servers to cluster

Parameters
servers (*list[str]*) – Server names (number of allowed servers: 1 or 3)

get_servers_status()

Retrieve the status of the messaging servers

get_status()

Retrieve the global status of messaging service

is_active()

Check if messaging service is Active

cterasdk.core.portals module

class cterasdk.core.portals.**Portals**(*portal*)

Bases: *BaseCommand*

Global Admin Portals APIs

add(*name*, *display_name=None*, *billing_id=None*, *company=None*, *plan=None*, *comment=None*)

Add a new tenant

Parameters

- **name** (*str*) – Name of the new tenant
- **display_name** (*str,optional*) – Display Name of the new tenant, defaults to None
- **billing_id** (*str,optional*) – Billing ID of the new tenant, defaults to None
- **company** (*str,optional*) – Company Name of the new tenant, defaults to None
- **plan** (*str,optional*) – Subscription plan name to assign to the new tenant, defaults to None
- **comment** (*str,optional*) – Assign a comment to the new tenant, defaults to None

Return str
A relative url path to the Team Portal

apply_changes(*wait=False*)

Apply provisioning changes.

Parameters
wait (*bool,optional*) – Wait for all changes to apply

browse(*tenant*)

Browse a tenant

Parameters
tenant (*str*) – Name of the tenant to browse

browse_global_admin()

Browse the Global Admin

default = ['name']

delete(*name*)

Delete an existing tenant

Parameters
name (*str*) – Name of the tenant to delete

get(*name*, *include=None*)

Get a tenant

Parameters

- **name** (*str*) – Name of the tenant
- **include** (*list[str]*) – List of fields to retrieve, defaults to ['name']

list_tenants(*include=None*, *portal_type=None*)

List tenants.

To retrieve tenants, you must first browse the Global Administration Portal, using: *GlobalAdmin.portals.browse_global_admin()*

Parameters

- **include** (*list[str],optional*) – List of fields to retrieve, defaults to ['name']
- **portal_type** (cterasdk.core.enum.PortalType) – The Portal type

subscribe(*tenant*, *plan*)

Subscribe a tenant to a plan

Parameters

- **name** (*str*) – Name of the tenant
- **str,plan** – Name of the subscription plan

tenants(*include_deleted=False*)

Get all tenants

Parameters
include_deleted (*bool,optional*) – Include deleted tenants, defaults to False

undelete(*name*)

Undelete a previously deleted tenant

Parameters
name (*str*) – Name of the tenant to undelete

cterasdk.core.query module

class cterasdk.core.query.**Filter**(*field*)

Bases: *Object*

class cterasdk.core.query.**FilterBuilder**(*name*, *reference=False*)

Bases: *Object*

after(*value*)

before(*value*)

eq(*value*)

ge(*value*)

gt(*value*)

le(*value*)

like(*value*)

lt(*value*)

ne(*value*)

notLike(*value*)

static ref(*name*)

setValue(*value*)

class cterasdk.core.query.**FilterType**

Bases: object

Boolean = 'BooleanFilter'

BooleanRefFilter = 'BooleanRefFilter'

DateTime = 'DateTimeFilter'

IntRefFilter = 'IntRefFilter'

Integer = 'IntFilter'

RefFilter = 'RefFilter'

String = 'StringFilter'

static fromValue(*value*, *ref*)

class cterasdk.core.query.**QueryParamBuilder**

Bases: object

addFilter(*query_filter*)

allPortals(*allPortals*)

build()

countLimit(*countLimit*)

include(*include*)

include_classname()

orFilter(*orFilter*)

ownedBy(*ownedBy*)

put(*key*, *value*)

sortBy(*sortBy*)

startFrom(*startFrom*)

class cterasdk.core.query.**QueryParams**

Bases: *Object*

include_classname()

increment()

class cterasdk.core.query.**Restriction**

Bases: object

EQUALS = 'eq'

GREATER_EQUALS = 'ge'

GREATER_THAN = 'gt'

LESS_EQUALS = 'le'

LESS_THAN = 'lt'

LIKE = 'like'

NOT_EQUALS = 'ne'

UNLIKE = 'notLike'

cterasdk.core.query.**iterator**(*CTERAHost*, *path*, *param*, *name=None*)

cterasdk.core.query.**query**(*CTERAHost*, *path*, *name*, *param*)

cterasdk.core.query.**show**(*CTERAHost*, *path*, *name*, *param*)

cterasdk.core.reports module

class cterasdk.core.reports.**Reports**(*portal*)

Bases: *BaseCommand*

Reports APIs

folder_groups()

Retrieve the folder groups statistics report.

To retrieve this report, you must first browse the Virtual Portal that contains the report, using: *GlobalAdmin.portals.browse()*

folders()

Retrieve the cloud folders statistics report.

To retrieve this report, you must first browse the Virtual Portal that contains the report, using: *GlobalAdmin.portals.browse()*

portals()

Retrieve the storage statistics report.

To retrieve this report, you must first browse the Global Administration Portal, using: *GlobalAdmin.portals.browse_global_admin()*

storage()

Retrieve the portals statistics report.

To retrieve this report, you must first browse the Global Administration Portal, using: *GlobalAdmin.portals.browse_global_admin()*

cterasdk.core.plans module

class cterasdk.core.plans.**PlanAutoAssignPolicy**(*portal*)

Bases: *BaseCommand*

get_policy()

Get plans auto assignment policy

set_policy(*rules*, *apply_default=None*, *default=None*, *apply_changes=True*)

Set plans auto assignment policy

Parameters

- **rules** (*list[cterasdk.common.types.PolicyRule]*) – List of policy rules
- **apply_default** (*bool,optional*) – If no match found, apply default plan. If not passed, the current config will be kept
- **default** (*str,optional*) – Name of a plan to assign if no match found. Ignored unless the `apply_default` is set to `True`
- **apply_changes** (*bool,optional*) – Apply provisioning changes upon update, defaults to `True`

class cterasdk.core.plans.**Plans**(*portal*)

Bases: *BaseCommand*

Portal Plan APIs

Variables
auto_assign (cterasdk.core.plans.PlanAutoAssignPolicy) – Object holding the Portal subscription plan auto assignment rules APIs

add(*name*, *retention=None*, *quotas=None*)

Add a subscription plan

Parameters

- **retention** (*dict,optional*) – The data retention policy
- **quotas** (*dict,optional*) – The items included in the plan and their respective quota

by_name(*names*, *include=None*)

Get Plans by their names

Parameters

- **names** (*list[str],optional*) – List of names of plans
- **include** (*list[str],optional*) – List of fields to retrieve, defaults to [‘name’]
- **filters** (*list[cterasdk.core.query.FilterBuilder],optional*) – List of additional filters, defaults to None

Returns
Iterator for all matching Plans

Return type
cterasdk.lib.iterator.Iterator

default = ['name']

delete(*name*)

Delete a subscription plan

Parameters
username (*str*) – The name of the subscription plan

get(*name*, *include=None*)

Retrieve subscription plan properties

Parameters

- **name** (*str*) – Name of the subscription plan
- **include** (*list[str]*) – List of fields to retrieve, defaults to ['name']

Returns
The subscription plan, including the requested fields

list_plans(*include=None*, *filters=None*)

List Plans

Parameters

- **include** (*list[str],optional*) – List of fields to retrieve, defaults to ['name']
- **filters** (*list[],optional*) – List of additional filters, defaults to None

Returns
Iterator for all matching Plans

Return type
cterasdk.lib.iterator.Iterator

modify(*name*, *retention=None*, *quotas=None*, *apply_changes=True*)

Modify a subscription plan

Parameters

- **retention** (*dict,optional*) – The data retention policy
- **quotas** (*dict,optional*) – The items included in the plan and their respective quota
- **apply_changes** (*bool,optional*) – Apply provisioning changes immediately

cterasdk.core.remote module

cterasdk.core.remote.**remote_command**(*Portal*, *device*)

cterasdk.core.servers module

class cterasdk.core.servers.**Servers**(*portal*)

Bases: *BaseCommand*

Global Admin Servers APIs

default = ['name']

get(*name*, *include=None*)

Retrieve server properties

Parameters

- **name** (*str*) – Name of the server
- **include** (*list[str]*) – List of fields to retrieve, defaults to ['name']

Returns
The server, including the requested fields

list_servers(*include=None*)

Retrieve the servers that comprise CTERA Portal.

To retrieve servers, you must first browse the Global Administration Portal, using: *GlobalAdmin.portals.browse_global_admin()*

Parameters
include (*list[str],optional*) – List of fields to retrieve, defaults to ['name']

modify(*name*, *server_name=None*, *app=None*, *preview=None*, *enable_public_ip=None*, *public_ip=None*, *allow_user_login=None*, *enable_replication=None*, *replica_of=None*)

Modify a Portal server

Parameters

- **name** (*str*) – The current server name
- **server_name** (*str,optional*) – New server name
- **app** (*bool,optional*) – Application server
- **preview** (*bool,optional*) – Preview server
- **enable_public_ip** (*bool,optional*) – Enable or disable public NAT address
- **public_ip** (*str,optional*) – Public NAT address
- **allow_user_login** (*bool,optional*) – Allow or disallow logins to this server
- **enable_replication** (*bool,optional*) – Enable or disable database replication
- **replica_of** (*str,optional*) – Configure as a replicate of another Portal server. *enable_replication* must be set to *True*

class cterasdk.core.servers.**Tasks**(*portal*)

Bases: *BaseCommand*

background(*name*)

Get all background tasks

Parameters
name (*str*) – Name of the server

Returns
List of tasks

scheduled(*name*)

Get all scheduled tasks

Parameters
name (*str*) – Name of the server

Returns
List of tasks

cterasdk.core.session module

class cterasdk.core.session.**Session**(*host*, *context*)

Bases: *SessionBase*

Administration = 'Administration'

in_tenant_context()

is_global_admin()

update_tenant(*current_tenant*)

cterasdk.core.setup module

class cterasdk.core.setup.**Setup**(*portal*)

Bases: *BaseCommand*

Global Admin Setup APIs

static default_settings()

get_replication_candidates()

get_setup_status()

init_application_server(*ipaddr*, *secret*)

Initialize a CTERA Portal Application Server.

Parameters

- **ipaddr** (*str*) – The CTERA Portal master server IP address
- **secret** (*str*) – A password or a PEM-encoded private key

init_master(*name*, *email*, *first_name*, *last_name*, *password*, *domain*)

Initialize the CTERA Portal master server.

Parameters

- **name** (*str*) – User name for the new user
- **email** (*str*) – E-mail address of the new user
- **first_name** (*str*) – The first name of the new user
- **last_name** (*str*) – The last name of the new user
- **password** (*str*) – Password for the new user
- **domain** (*str*) – The domain suffix for CTERA Portal

init_replication_server(*ipaddr*, *secret*, *replicate_from=None*)

Initialize a CTERA Portal Database Replication Server.

Parameters

- **ipaddr** (*str*) – The CTERA Portal master server IP address

- **secret** (*str*) – A password or a PEM-encoded private key
- **replicate_from** (*str*) – The name of a CTERA Portal server to replicate from

class cterasdk.core.setup.**SetupWizardStatusMonitor**(*portal*, *retries=60*, *seconds=5*)

Bases: object

wait(*stage*)

cterasdk.core.ssl module

class cterasdk.core.ssl.**SSL**(*portal*)

Bases: *BaseCommand*

Portal SSL Certificate APIs

create_zip_archive(*private_key*, **certificates*)

Create a ZIP archive that can be imported to CTERA Portal

Parameters

- **private_key** (*str*) – The PEM-encoded private key, or a path to the PEM-encoded private key file
- **certificates** (*list[str]*) – The PEM-encoded certificates, or a list of paths of the PEM-encoded certificate files

export(*destination=None*)

Export the Portal SSL Certificate to a ZIP archive

Parameters
destination (*str,optional*) – File destination, defaults to the default directory

get()

Retrieve details of the current installed SSL certificate

Return cterasdk.common.object.Object
An object including the SSL certificate details

import_from_chain(*private_key*, **certificates*)

Import an SSL Certificate to CTERA Portal from a chain

Parameters

- **private_key** (*str*) – The PEM-encoded private key, or a path to the PEM-encoded private key file
- **certificates** (*list[str]*) – The PEM-encoded certificates, or a list of paths of the PEM-encoded certificate files

import_from_zip(*zipfile*)

Import an SSL Certificate to CTERA Portal from a ZIP archive

Parameters
zipfile (*str*) – A zip archive including the private key and SSL certificate chain

property thumbprint

Get the SHA1 thumbprint of the Portal SSL certificate

cterasdk.core.startup module

class cterasdk.core.startup.**Startup**(*portal*)

Bases: *BaseCommand*

Server Startup APIs

Started = 'Started'

status()

Get the server startup status

wait(*retries=120*, *seconds=5*)

Wait for server startup

cterasdk.core.syslog module

class cterasdk.core.syslog.**Syslog**(*portal*)

Bases: *BaseCommand*

Portal Syslog Management APIs

disable()

enable(*server*, *port=514*, *protocol='UDP'*, *min_severity='info'*)

Enable Syslog

Parameters

- **server** (*str*) – Syslog server address
- **port** (*int*, *optional*) – Syslog server port
- **protocol** (cterasdk.core.enum.IPProtocol, *optional*) – Syslog server IP protocol
- **min_severity** (cterasdk.core.enum.Severity, *optional*) – Minimum log severity to forward

get_configuration()

Retrieve the syslog server configuration

is_enabled()

Check if forwarding log messages over syslog is enabled

modify(*server=None*, *port=None*, *protocol=None*, *min_severity=None*)

Modify Syslog log forwarding configuration

Parameters

- **server** (*str*) – Syslog server address
- **port** (*int*, *optional*) – Syslog server port
- **protocol** (cterasdk.core.enum.IPProtocol, *optional*) – Syslog server IP protocol
- **min_severity** (cterasdk.core.enum.Severity, *optional*) – Minimum log severity to forward

cterasdk.core.taskmgr module

class cterasdk.core.taskmgr.**Task**(*CTERAHost*, *ref*, *retries=10*, *seconds=1*)

Bases: TaskBase

get_task_status()

class cterasdk.core.taskmgr.**Tasks**(*portal*)

Bases: *BaseCommand*

Portal Background Task APIs

status(*ref*)

Get background task status

Parameters
ref (*str*) – Task reference

wait(*ref*, *retries=100*, *seconds=1*)

Wait for background task to complete

Parameters

- **ref** (*str*) – Task reference
- **retries** (*int,optional*) – Number of retries when sampling the task status, defaults to 100
- **seconds** (*int,optional*) – Number of seconds to wait between retries, defaults to 1

cterasdk.core.templates module

class cterasdk.core.templates.**TemplateAutoAssignPolicy**(*portal*)

Bases: *BaseCommand*

apply_changes(*wait=False*)

Apply provisioning changes.

Parameters
wait (*bool,optional*) – Wait for all changes to apply, defaults to *False*

get_policy()

Get templates auto assignment policy

set_policy(*rules*, *apply_default=None*, *default=None*, *apply_changes=True*)

Set templates auto assignment policy

Parameters

- **rules** (*list[cterasdk.common.types.PolicyRule]*) – List of policy rules
- **apply_default** (*bool,optional*) – If no match found, apply default template. If not passed, the current config will be kept
- **default** (*str,optional*) – Name of a template to assign if no match found. Ignored unless the `apply_default` is set to `True`
- **apply_changes** (*bool,optional*) – Apply changes upon update, defaults to `True`

class cterasdk.core.templates.**Templates**(*portal*)

Bases: *BaseCommand*

Portal Configuration Template APIs

add(*name*, *description=None*, *include_sets=None*, *exclude_sets=None*, *apps=None*, *backup_schedule=None*, *versions=None*, *scripts=None*, *cli_commands=None*)

Add a Configuration Template

Parameters

- **name** (*str*) – Name of the template
- **description** (*str,optional*) – Template description
- **include_sets** (*list[cterasdk.common.types.FilterBackupSet],optional*) – List of backup sets to include
- **exclude_sets** (*list[cterasdk.common.types.FilterBackupSet],optional*) – List of backup sets to exclude
- **apps** (*list[cterasdk.common.enum.Application],optional*) – List of applications to back up
- **backup_schedule** (*cterasdk.common.types.TaskSchedule,optional*) – Backup schedule
- **versions** (*list[cterasdk.core.types.PlatformVersion],optional*) – List of platforms and their associated versions. Pass *None* to inehrit the default settings from the Global Administration Portal
- **scripts** (*list[cterasdk.core.types.TemplateScript],optional*) – Scripts to execute after logon, before or after backup
- **cli_commands** (*list[str],optional*) – Template CLI commands to execute

by_name(*names*, *include=None*)

Get Templates by their names

Parameters

- **names** (*list[str],optional*) – List of names of templates
- **include** (*list[str],optional*) – List of fields to retrieve, defaults to ['name']
- **filters** (*list[cterasdk.core.query.FilterBuilder],optional*) – List of additional filters, defaults to None

Returns

Iterator for all matching Templates

Return type

cterasdk.lib.iterator.Iterator

default = ['name']

delete(*name*)

Delete a Configuration Template

Parameters

name (*str*) – Name of the template

get(*name*, *include=None*)

Get a Configuration Template

Parameters

- **name** (*str*) – Name of the template
- **include** (*list[str]*) – List of fields to retrieve, defaults to `['name']`

list_templates(*include=None*, *filters=None*)

List Configuration Templates.

To retrieve templates, you must first browse the tenant, using: *GlobalAdmin.portals.browse()*

Parameters

- **include** (*list[str],optional*) – List of fields to retrieve, defaults to `['name']`
- **filters** (*list[],optional*) – List of additional filters, defaults to None

remove_default(*name*, *wait=False*)

Set a Configuration Template not to be the default template

Parameters

- **name** (*str*) – Name of the template
- **wait** (*bool,optional*) – Wait for all changes to apply, defaults to *False*

set_default(*name*, *wait=False*)

Set a Configuration Template as the default template

Parameters

- **name** (*str*) – Name of the template
- **wait** (*bool,optional*) – Wait for all changes to apply, defaults to *False*

cterasdk.core.types module

class cterasdk.core.types.**AccessControlEntry**(*account*, *role*)

Bases: `tuple`

Tuple holding a Portal account and its respective permission

property account

The Portal group or user account

property role

The group or user role

class cterasdk.core.types.**AccessControlRule**(*group*, *role*)

Bases: *Object*

class cterasdk.core.types.**AlertBuilder**(*name*)

Bases: `object`

build()

Build the alert

Returns

Alert object

Return type
cterasdk.common.object.Object

`content`(*content*)

Set alert log message content

Parameters
content (*str*) – Log content

Returns
Alert Builder

Return type
cterasdk.core.types.AlertBuilder

`description`(*description*)

Set alert description

Parameters
description (*str*) – Alert description

Returns
Alert Builder

Return type
cterasdk.core.types.AlertBuilder

`log`(*log*)

Set alert log class name

Parameters
log (*str*) – Log class name

Returns
Alert Builder

Return type
cterasdk.core.types.AlertBuilder

`min_severity`(*min_severity*)

Set alert log minimum severity

Parameters
min_severity (`cterasdk.core.enum.Severity`) – Minimum severity

Returns
Alert Builder

Return type
cterasdk.core.types.AlertBuilder

`static name`(*name*)

Create an Alert Builder

Parameters
name (*str*) – Alert name

Returns
Alert Builder

Return type
cterasdk.core.types.AlertBuilder

origin_type(*origin_type*)

Set alert origin type

Parameters
origin_type (cterasdk.core.enum.OriginType) – Log origin type

Returns
Alert Builder

Return type
cterasdk.core.types.AlertBuilder

topic(*topic*)

Set alert log topic

Parameters
topic (cterasdk.core.enum.LogTopic) – Log topic

Returns
Alert Builder

Return type
cterasdk.core.types.AlertBuilder

class cterasdk.core.types.**AmazonS3**(*bucket*, *access_key*, *secret_key*, *endpoint='s3.amazonaws.com'*, *https=True*, *direct=True*)

Bases: *HTTPBucket*

to_server_object()

class cterasdk.core.types.**AzureBlob**(*bucket*, *access_key*, *secret_key*, *endpoint='core.windows.net'*, *https=True*, *direct=True*)

Bases: *HTTPBucket*

to_server_object()

class cterasdk.core.types.**BackgroundTask**(*task_id*, *name*, *start_time*, *end_time*, *elapsed_time*, *status*, *message*, *ref*)

Bases: *Task*

static from_server_object(*server_object*, *ref*)

class cterasdk.core.types.**Bucket**(*bucket*, *driver*)

Bases: object

to_server_object()

class cterasdk.core.types.**CloudFSFolderFindingHelper**(*name*, *owner*)

Bases: tuple

Tuple holding the name and owner couple to search for folders

property name

The name of the CloudFS folder

property owner

The name of the owner of the CloudFS folder

class cterasdk.core.types.**DomainControllers**(*primary=None*, *secondary=None*)

Bases: object

property primary

property secondary

class cterasdk.core.types.**GenericS3**(*bucket*, *access_key*, *secret_key*, *endpoint*, *https=False*, *direct=False*)

Bases: *S3Compatible*

class cterasdk.core.types.**Google**(*bucket*, *access_key*, *secret_key*, *endpoint*, *https=False*, *direct=False*)

Bases: *S3Compatible*

class cterasdk.core.types.**GroupAccount**(*name*, *directory=None*)

Bases: *PortalAccount*

property account_type

The Portal Account Type

Return cterasdk.core.enum.PortalAccountType
The Portal Account Type

class cterasdk.core.types.**HTTPBucket**(*bucket*, *driver*, *access_key*, *secret_key*, *endpoint*, *https*, *direct=False*)

Bases: *Bucket*

class cterasdk.core.types.**ICOS**(*bucket*, *access_key*, *secret_key*, *endpoint*, *https=False*, *direct=False*)

Bases: *S3Compatible*

class cterasdk.core.types.**NetAppStorageGRID**(*bucket*, *access_key*, *secret_key*, *endpoint*, *https=False*, *direct=False*, *tags=False*)

Bases: *S3Compatible*

to_server_object()

class cterasdk.core.types.**Nutanix**(*bucket*, *access_key*, *secret_key*, *endpoint*, *https=False*, *direct=False*)

Bases: *S3Compatible*

class cterasdk.core.types.**PlanCriteriaBuilder**

Bases: object

Type = 'PlanCriteria'

static billing_id()

static comment()

static company()

static first_name()

static last_name()

static role()

static user_groups()

static username()

class cterasdk.core.types.**PlatformVersion**(*name*, *version*)

Bases: tuple

Tuple holding the platform name and version

property name

The name of the platform

property version

The version identifier

class cterasdk.core.types.PortalAccount(*name*, *directory=None*)

Bases: ABC

Base Class for Portal Account

Variables

- **name** (*str*) – The user name
- **directory** (*str*) – The fully-qualified name of the user directory, defaults to None

property account_type

The Portal Account Type

Return cterasdk.core.enum.PortalAccountType
The Portal Account Type

static from_collaborator(*collaborator*)

property is_local

Is the account local

Return bool
True if the account if local, otherwise False

class cterasdk.core.types.S3Compatible(*bucket*, *driver*, *access_key*, *secret_key*, *endpoint*, *https*, *direct*)

Bases: *HTTPBucket*

to_server_object()

class cterasdk.core.types.Scality(*bucket*, *access_key*, *secret_key*, *endpoint*, *https=False*, *direct=False*)

Bases: *S3Compatible*

class cterasdk.core.types.ScheduledTask(*task_id*, *name*, *start_time*)

Bases: *Task*

static from_server_object(*server_object*)

class cterasdk.core.types.ShareRecipient(*account*, *account_type*, *two_factor=False*)

Bases: object

Class Representing a Collboration Share Recipient

static domain_group(*group_account*)

Share with a domain group

Parameters
group_account (GroupAccount) – A domain group account

static domain_user(*user_account*)

Share with a domain user

Parameters
user_account (UserAccount) – A domain user account

expire_in(*days*)

Set share to expire after (days)

Parameters
days (*int*) – The number of days the share will remain accessible

expire_on(*expiration_date*)

Set the share expiration date

Parameters
expire_on (*str*) – The expiration date (%Y-%m-%d)

static external(*email*, *two_factor=False*)

Share with an external user

Parameters

- **email** (*str*) – The email address of the recipient
- **two_factor** (*bool*) – Require two factor authentication over e-mail

static local_group(*group_account*)

Share with a local group

Parameters
group_account (GroupAccount) – A local group account

static local_user(*user_account*)

Share with a local user

Parameters
user_account (UserAccount) – A local user account

no_access()

Deny access

preview_only()

Grant preview only access

read_only()

Grant read only access

read_write()

Grant read write access

class cterasdk.core.types.**Task**(*task_id*, *name*)

Bases: *Object*

class cterasdk.core.types.**TemplateCriteriaBuilder**

Bases: object

Type = 'DeviceCriteria'

static groups()

static hostname()

static name()

static os()

static owner()

static plan()

static type()

static version()

class cterasdk.core.types.TemplateScript(*platform*)

Bases: object

after_backup(*after_backup*)

Set the post backup script

Parameters
after_backup (*str*) – A string or path to the script file

after_logon(*after_logon*)

Set the post logon script

Parameters
after_logon (*str*) – A string or path to the script file

before_backup(*before_backup*)

Set the pre backup script

Parameters
before_backup (*str*) – A string or path to the script file

static linux()

Configure Windows Scripts

static mac()

Configure Windows Scripts

property platform

to_server_object()

static windows()

Configure Windows Scripts

class cterasdk.core.types.UserAccount(*name*, *directory=None*)

Bases: *PortalAccount*

property account_type

The Portal Account Type

Return cterasdk.core.enum.PortalAccountType
The Portal Account Type

class cterasdk.core.types.Wasabi(*bucket*, *access_key*, *secret_key*, *endpoint*, *https=False*, *direct=False*)

Bases: *S3Compatible*

cterasdk.core.admins module

class cterasdk.core.admins.**Administrators**(*portal*)

Bases: *BaseCommand*

Portal Global Administrators User Management APIs

add(*name*, *email*, *first_name*, *last_name*, *password*, *role*, *company=None*, *comment=None*, *password_change=False*)

Create a Global Administrator

Parameters

- **name** (*str*) – User name for the new GlobalAdmin
- **email** (*str*) – E-mail address of the new GlobalAdmin
- **first_name** (*str*) – The first name of the new GlobalAdmin
- **last_name** (*str*) – The last name of the new GlobalAdmin
- **password** (*str*) – Password for the new GlobalAdmin
- **role** (*cterasdk.core.enum.Role*) – User role of the new GlobalAdmin
- **company** (*str,optional*) – The name of the company of the new GlobalAdmin, defaults to None
- **comment** (*str,optional*) – Additional comment for the new GlobalAdmin, defaults to None
- **password_change** (*variable,optional*) – Require the user to change the password on the first login. Pass datetime.date for a specific date, integer for days from creation, or True for immediate , defaults to False

default = ['name']

delete(*name*)

Delete a Global Administrator

Parameters
username (*str*) – Global administrator username

get(*name*, *include=None*)

Get a Global Administrator user account

Parameters

- **name** (*str*) – Global administrator username
- **include** (*list[str]*) – List of fields to retrieve, defaults to ['name']

Returns
The user account, including the requested fields

list_admins(*include=None*)

List local administrators

Parameters
include (*list[str]*) – List of fields to retrieve, defaults to ['name']

Returns
Iterator for local administrators

Return type
cterasdk.lib.iterator.Iterator

modify(*current_username*, *new_username=None*, *email=None*, *first_name=None*, *last_name=None*, *password=None*, *role=None*, *company=None*, *comment=None*)

Modify a Global Administrator user account

Parameters

- **current_username** (*str*) – The current GlobalAdmin username
- **new_username** (*str,optional*) – New name
- **email** (*str,optional*) – E-mail address
- **first_name** (*str,optional*) – First name
- **last_name** (*str,optional*) – Last name
- **password** (*str,optional*) – Password
- **role** (cterasdk.core.enum.Role,*optional*) – User role
- **company** (*str,optional*) – Company name
- **comment** (*str,optional*) – Comment

cterasdk.core.users module

class cterasdk.core.users.**Users**(*portal*)

Bases: *BaseCommand*

Portal User Management APIs

add(*name*, *email*, *first_name*, *last_name*, *password*, *role*, *company=None*, *comment=None*, *password_change=False*)

Create a local user account

Parameters

- **name** (*str*) – User name for the new user
- **email** (*str*) – E-mail address of the new user
- **first_name** (*str*) – The first name of the new user
- **last_name** (*str*) – The last name of the new user
- **password** (*str*) – Password for the new user
- **role** (cterasdk.core.enum.Role) – User role of the new user
- **company** (*str,optional*) – The name of the company of the new user, defaults to None
- **comment** (*str,optional*) – Additional comment for the new user, defaults to None
- **password_change** (*variable,optional*) – Require the user to change the password on the first login. Pass datetime.date for a specific date, integer for days from creation, or True for immediate , defaults to False

apply_changes(*wait=False*)

Apply provisioning changes.

Parameters
wait (*bool, optional*) – Wait for all changes to apply

default = ['name']

delete(*user*)

Delete a user

Parameters
user (cterasdk.core.types.UserAccount) – the user account

get(*user_account*, *include=None*)

Get a user account

Parameters

- **user_account** (cterasdk.core.types.UserAccount) – User account, including the user directory and user name
- **include** (*list[str]*) – List of fields to retrieve, defaults to ['name']

Returns
The user account, including the requested fields

list_domain_users(*domain*, *include=None*)

List all the users in the domain

Parameters
include (*list[str]*) – List of fields to retrieve, defaults to ['name']

Returns
Iterator for all the domain users

Return type
cterasdk.lib.iterator.Iterator

list_domains()

List all domains

Return list
List of all domains

list_local_users(*include=None*)

List all local users

Parameters
include (*list[str]*) – List of fields to retrieve, defaults to ['name']

Returns
Iterator for all local users

Return type
cterasdk.lib.iterator.Iterator

modify(*current_username*, *new_username=None*, *email=None*, *first_name=None*, *last_name=None*, *password=None*, *role=None*, *company=None*, *comment=None*)

Modify a local user account

Parameters

- **current_username** (*str*) – The current user name
- **new_username** (*str, optional*) – New name

- **email** (*str, optional*) – E-mail address
- **first_name** (*str, optional*) – First name
- **last_name** (*str, optional*) – Last name
- **password** (*str, optional*) – Password
- **role** (*cterasdk.core.enum.Role, optional*) – User role
- **company** (*str, optional*) – Company name
- **comment** (*str, optional*) – Comment

cterasdk.core.zones module

class cterasdk.core.zones.**Zones**(*portal*)

Bases: *BaseCommand*

Portal Zones APIs

add(*name*, *policy_type='selectedFolders'*, *description=None*)

Add a new zone

Parameters

- **name** (*str*) – The name of the new zone
- **policy_type** (*cterasdk.core.enum.PolicyType, optional*) – Policy type of the new zone, defaults to cterasdk.core.enum.PolicyType.SELECT
- **description** (*str, optional*) – The description of the new zone

add_devices(*name*, *device_names*)

Add devices to a zone

Parameters

- **name** (*str*) – The name of the zone to add devices to
- **device_names** (*list[str]*) – The names of the devices to add to the zone

add_folders(*name*, *folder_finding_helpers*)

Add the folders to the zone

Parameters

- **name** (*str*) – The name of the zone
- **folder_finding_helpers** (*list[cterasdk.core.types.CloudFSFolderFindingHelper]*) – List of folder names and owners

delete(*name*)

Delete a zone

Parameters

name (*str*) – The name of the zone to delete

get(*name*)

Get zone by name

Parameters

name (*str*) – The name of the zone to get

Returns
The requested zone

`list_zones`(*filters=None*)

List Zones :param list[],optional filters: List of additional filters, defaults to None

Returns
Iterator for all Zones

Return type
cterasdk.lib.iterator.Iterator

`name_attr = 'name'`

`search`(*name*)

Search for Zones by name :param str name: Search query

Returns
Iterator for all matching Zones

Return type
cterasdk.lib.iterator.Iterator

3.1.5 cterasdk.edge package

3.1.5.1 Subpackages

cterasdk.edge.files package

Submodules

cterasdk.edge.files.browser module

`class cterasdk.edge.files.browser.FileBrowser`(*Gateway*)

Bases: `object`

Gateway File Browser APIs

`copy`(*src*, *dest*, *overwrite=False*)

Copy a file or a folder

Parameters

- **src** (*str*) – Source file or folder path
- **dest** (*str*) – Destination folder path
- **overwrite** (*bool,optional*) – Overwrite on conflict, defaults to False

`delete`(*path*)

Delete a file

Parameters
path (*str*) – The file's path on the gateway

download(*path*, *destination=None*)

Download a file

Parameters

- **path** (*str*) – The file path on the Edge Filer
- **destination** (*str,optional*) – File destination, if it is a directory, the original filename will be kept, defaults to the default directory

download_as_zip(*cloud_directory*, *files*, *destination=None*)

Download a list of files and/or directories from a cloud folder as a ZIP file

> **Warning:** The list of files is not validated. The ZIP file will include only the existing files and directories

Parameters

- **cloud_directory** (*str*) – Path to the cloud directory
- **files** (*list[str]*) – List of files and/or directories in the cloud folder to download
- **destination** (*str,optional*) – File destination, if it is a directory, the filename will be calculated, defaults to the default directory

static ls(*_path*)

mkdir(*path*, *recurse=False*)

Create a new directory

Parameters

- **path** (*str*) – The path of the new directory
- **recurse** (*bool,optional*) – Create subdirectories if missing, defaults to False

static mkpath(*path*)

move(*src*, *dest*, *overwrite=False*)

Move a file or a folder

Parameters

- **src** (*str*) – Source file or folder path
- **dest** (*str*) – Destination folder path
- **overwrite** (*bool,optional*) – Overwrite on conflict, defaults to False

openfile(*path*)

Obtain a file handle

Parameters
path (*str*) – The file path on the Edge Filer

upload(*file_path*, *server_path*)

Upload a file

Parameters

- **file_path** (*str*) – Path to the local file to upload
- **server_path** (*str*) – Path to the directory to upload the file to

cterasdk.edge.files.copy module

cterasdk.edge.files.copy.**copy**(*CTERAHost*, *src*, *dest*, *overwrite*)

cterasdk.edge.files.file_access module

class cterasdk.edge.files.file_access.**FileAccess**(*ctera_host*)

Bases: *FileAccessBase*

cterasdk.edge.files.mkdir module

exception cterasdk.edge.files.mkdir.**Forbidden**(*message=None*, *instance=None*, ***kwargs*)

Bases: *CTERAException*

exception cterasdk.edge.files.mkdir.**ItemExists**(*message=None*, *instance=None*, ***kwargs*)

Bases: *CTERAException*

cterasdk.edge.files.mkdir.**mkdir**(*ctera_host*, *path*, *recurse=False*)

cterasdk.edge.files.move module

cterasdk.edge.files.move.**move**(*CTERAHost*, *src*, *dest*, *overwrite*)

cterasdk.edge.files.path module

class cterasdk.edge.files.path.**CTERAPath**(*relativepath*, *basepath*)

Bases: object

encoded_fullpath()

encoded_parent()

fullpath()

joinpath(*path*)

name()

parent()

parts()

cterasdk.edge.files.rm module

cterasdk.edge.files.rm.**delete**(*ctera_host*, *path*)

3.1.5.2 Submodules

cterasdk.edge.afp module

class cterasdk.edge.afp.**AFP**(*gateway*)

Bases: *BaseCommand*

Gateway AFP APIs

disable()

Disable AFP

is_disabled()

Check if the AFP server is disabled

cterasdk.edge.aio module

class cterasdk.edge.aio.**AIO**(*gateway*)

Bases: *BaseCommand*

Gateway AIO APIs

disable()

Disable AIO

enable()

Enable AIO

is_enabled()

Is AIO enabled

Returns

True is AIO is enabled, else False

Return type

bool

cterasdk.edge.array module

class cterasdk.edge.array.**Array**(*gateway*)

Bases: *BaseCommand*

Gateway Array APIs

add(*array_name*, *level*, *members=None*)

Add a new array

Parameters

- **array_name** (*str*) – Name for the new array

- **level** (RAIDLevel) – RAID level
- **members** (*list(str)*) – Members of the array. If not specified, the system will try to create an array using all available drives

delete(*array_name*)

Delete an array

Parameters
name (*str*) – The name of the array to delete

delete_all()

Delete all arrays

get(*name=None*)

Get Array. If an array name was not passed as an argument, a list of all arrays will be retrieved

Parameters
name (*str,optional*) – Name of the array

cterasdk.edge.audit module

class cterasdk.edge.audit.**Audit**(*gateway*)

Bases: *BaseCommand*

Gateway Audit configuration APIs

Variables
defaultAuditEvents (*list[cterasdk.edge.enum.AuditEvents]*) – Default audit events

defaultAuditEvents = ['WD', 'AD', 'WEA', 'DC', 'WA', 'DE', 'WDAC', 'WO']

disable()

Disable Gateway Audit log

enable(*path*, *auditEvents=None*, *logKeepPeriod=30*, *maxLogKBSize=102400*, *maxRotateTime=1440*, *includeAuditLogTag=True*, *humanReadableAuditLog=False*)

Enable Gateway Audit log

Parameters

- **path** (*str*) – Path to save the audit log
- **auditEvents** (*list[cterasdk.edge.enum.AuditEvents],optional*) – List of audit event types to save, defaults to Audit.defaultAuditEvents
- **logKeepPeriod** (*int,optional*) – Period to key the logs in days, defaults to 30
- **maxLogKBSize** (*int,optional*) – The maximum size of the log file in KB, defaults to 102400 (100 MB)
- **maxRotateTime** (*int,optional*) – The maximal time before rotating the log file in Minutes, defaults to 1440 (24 hours)
- **includeAuditLogTag** (*bool,optional*) – Include audit log tag, defaults to True
- **humanReadableAuditLog** (*bool,optional*) – Human readable audit log, defaults to False

cterasdk.edge.backup module

exception cterasdk.edge.backup.**AttachEncrypted**(*encryptionMode*, *encryptedFolderKey*, *passPhraseSalt*)

Bases: *CTERAException*

Attach Encrypted exception

class cterasdk.edge.backup.**AttachRC**

Bases: object

CheckCodeInCorrect = 'CheckCodeInCorrect'

ClocksOutOfSync = 'ClocksOutOfSync'

InternalServerError = 'InternalServerError'

IsEncrypted = 'IsEncrypted'

NotFound = 'NotFound'

OK = 'OK'

PermissionDenied = 'PermissionDenied'

class cterasdk.edge.backup.**Backup**(*gateway*)

Bases: *BaseCommand*

Gateway backup configuration APIs

configure(*passphrase=None*)

Gateway backup configuration

Parameters
passphrase (*str,optional*) – Passphrase for the backup, defaults to None

is_configured()

Is Backup configured

Return bool
True if backup is configured, else False

start()

Start backup

suspend()

Suspend backup

unsuspend()

Unsuspend backup

class cterasdk.edge.backup.**BackupFiles**(*gateway*)

Bases: *BaseCommand*

ALL_FILES = 'All File Types'

unselect_all()

Unselect all files from backup

exception cterasdk.edge.backup.**ClocksOutOfSync**

Bases: *CTERAException*

Clocks Out of Sync exception

class cterasdk.edge.backup.**CreateFolderRC**

Bases: object

FolderAlreadyExists = 'FolderAlreadyExists'

InternalServerError = 'InternalServerError'

OK = 'OK'

PermissionDenied = 'PermissionDenied'

class cterasdk.edge.backup.**EncryptionMode**

Bases: object

Encryption mode types

Variables

- **Recoverable** (*str*) – Recoverable key encryption mode
- **Secret** (*str*) – Secret key encryption mode

Recoverable = 'RecoverableKeyEncryption'

Secret = 'SecretKeyEncryption'

exception cterasdk.edge.backup.**IncorrectPassphrase**

Bases: *CTERAException*

Incorrect Passphrase exception

exception cterasdk.edge.backup.**NotFound**(*message=None*, *instance=None*, ***kwargs*)

Bases: *CTERAException*

Not found exception

cterasdk.edge.base_command module

class cterasdk.edge.base_command.**BaseCommand**(*gateway*)

Bases: object

Base class for all Gateway API classes

session()

cterasdk.edge.cache module

class cterasdk.edge.cache.**Cache**(*gateway*)

Bases: *BaseCommand*

Gateway cache configuration

disable()

Disable caching

enable()

Enable caching

force_eviction()

Force eviction

is_enabled()

pin(*path*)

Pin a folder

Parameters
path (*str*) – Directory path

pin_all()

Pin all folders

pin_exclude(*path*)

Exclude a sub-folder from a pinned folder

Parameters
path (*str*) – Directory path

remove_pin(*path*)

Remove a pin from a previously pinned folder

Parameters
path (*str*) – Directory path

unpin_all()

Remove all folder pins

cterasdk.edge.cli module

class cterasdk.edge.cli.**CLI**(*gateway*)

Bases: *BaseCommand*

Gateway CLI APIs

run_command(*cli_command*)

Run a CLI command

Parameters
cli_command (*str*) – The CLI command to run on the gateway

Return str
The response of the gateway

cterasdk.edge.config module

class cterasdk.edge.config.**Config**(*gateway*)

Bases: *BaseCommand*

General gateway configuraion

disable_wizard()

Disable the first time wizard

enable_wizard()

Enable the first time wizard

export(*destination=None*)

Export the Edge Filer configuration

Parameters
destination (*str,optional*) – File destination, defaults to the default directory

get_hostname()

Get the hostname of the gateway

Return str
The hostname of the gateway

get_location()

Get the location of the gateway

Return str
The location of the gateway

import_config(*config*, *exclude=None*)

Import the Edge Filer configuration

Parameters

- **config** (*str*) – A string or a path to the Edge Filer configuration file
- **delete_attrs** (*list[str],optional*) – List of configuration properties to exclude from import

is_wizard_enabled()

Get the current configuration of the first time wizard

Return bool
True if the first time wizard is enabled, else False

load_config(*config*)

Load the Edge Filer configuration

Parameters
config (*str*) – A string or a path to the Edge Filer configuration file

set_hostname(*hostname*)

Set the hostname of the gateway

Parameters
hostname (*str*) – New hostname to set

Return str
The new hostname

set_location(*location*)

Set the location of the gateway

Parameters
location (*str*) – New location to set

Return str
The new location

cterasdk.edge.connection module

cterasdk.edge.connection.**test**(*CTERAHost*)

cterasdk.edge.connection.**test_application**(*CTERAHost*)

cterasdk.edge.connection.**test_network**(*CTERAHost*)

cterasdk.edge.decorator module

cterasdk.edge.decorator.**authenticated**(*function*)

cterasdk.edge.decorator.**is_nosession**(*function*, *path*)

cterasdk.edge.dedup module

class cterasdk.edge.dedup.**Dedup**(*gateway*)

Bases: *BaseCommand*

Edge Filer Local Deduplication APIs

disable(*reboot=False*, *wait=False*)

Disable local deduplication

Parameters

- **reboot** (*bool*) – Reboot, defaults to False
- **wait** (*bool,optional*) – Wait for reboot to complete, defaults to False

enable(*reboot=False*, *wait=False*)

Enable local deduplication

Parameters

- **reboot** (*bool*) – Reboot, defaults to False
- **wait** (*bool,optional*) – Wait for reboot to complete, defaults to False

status()

Get the de-duplication status

Returns
An object including the deduplication status

Return type
cterasdk.edge.types.DeduplicationStatus

class cterasdk.edge.dedup.**Regeneration**(*gateway*)

Bases: *BaseCommand*

Edge Filer Local Deduplication Regeneration APIs

run()

Run the regeneration process

status()

Get the regeneration process statistics

cterasdk.edge.directoryservice module

class cterasdk.edge.directoryservice.**DirectoryService**(*gateway*)

Bases: *BaseCommand*

Gateway Active Directory configuration APIs

connect(*domain*, *username*, *password*, *ou=None*, *check_connection=False*)

Connect the Gateway to an Active Directory

Parameters

- **domain** (*str*) – The active directory domain to connect to
- **username** (*str*) – The user name to use when connecting to the active directory services
- **password** (*str*) – The password to use when connecting to the active directory services
- **ou** (*str,optional*) – The OU path to use when connecting to the active directory services, defaults to None
- **check_connection** (*bool,optional*) – Check connectivity before attempting to connect to directory services, defaults to *False*

connected()

Get the Active Directory join status

disconnect()

Disconnect from Active Directory Service

domains()

Get all domains

Return list(str)
List of names of all discovered domains

get_advanced_mapping()

Retrieve directory services advanced mapping configuration

Returns
A dictionary of domain mapping objects

Return type
dict

get_connected_domain()

Get the connected domain information

Return cterasdk.common.object.Object

get_static_domain_controller()

Retrieve the static domain controller configuration

Returns
A FQDN, hostname or ip address of the domain controller

Return type
str

remove_static_domain_controller()

Delete the static domain controller configuration

set_advanced_mapping(*mappings*)

Configure advanced mapping

Parameters
mappings (*list[cterasdk.common.types.ADDomainIDMapping]*) – List of domains and their UID/GID mapping range

set_static_domain_controller(*dc*)

Configure the Gateway to use a static domain controller

Parameters
dc (*str*) – The FQDN, hostname or ip address of the domain controller

Returns
The FQDN, hostname or ip address of the domain controller

Return type
str

cterasdk.edge.drive module

class cterasdk.edge.drive.**Drive**(*gateway*)

Bases: *BaseCommand*

Gateway Drive APIs

format(*name*)

Format a drive

Parameters
name (*str*) – The name of the drive to format

format_all()

Format all drives

get(*name=None*)

Get Drive. If a drive name was not passed as an argument, a list of all drives will be retrieved

Parameters
name (*str,optional*) – Name of the drive

get_status(*name=None*)

Get drive status. If a drive name was not passed as an argument, a list of all drives will be retrieved

Parameters
name (*str*) – Name of the drive

cterasdk.edge.enum module

class cterasdk.edge.enum.**Acl**

Bases: object

ACL types

Variables

- **WindowsNT** (*str*) – Windows NT ACL Mode
- **OnlyAuthenticatedUsers** (*str*) – Authenticated Users ACL Mode

OnlyAuthenticatedUsers = 'authenticated'

WindowsNT = 'winAclMode'

class cterasdk.edge.enum.**AuditEvents**

Bases: object

Audit log event types

Variables

- **ListFolderReadData** (*str*) – List Folder Read Data
- **CreateFilesWriteData** (*str*) – Create Files Write Data
- **CreateFoldersAppendData** (*str*) – Create Folders Append Data
- **ReadExtendedAttributes** (*str*) – Read Extended Attributes
- **WriteExtendedAttributes** (*str*) – Write Extended Attributes
- **TraverseFolderExecuteFile** (*str*) – Traverse Folder Execute File
- **DeleteSubfoldersAndFiles** (*str*) – Delete Subfolders And Files
- **WriteAttributes** (*str*) – Write Attributes
- **Delete** (*str*) – Delete
- **ChangePermissions** (*str*) – Change Permissions
- **ChangeOwner** (*str*) – Change Owner

ChangeOwner = 'WO'

ChangePermissions = 'WDAC'

CreateFilesWriteData = 'WD'

CreateFoldersAppendData = 'AD'

Delete = 'DE'

DeleteSubfoldersAndFiles = 'DC'

ListFolderReadData = 'RD'

ReadExtendedAttributes = 'REA'

TraverseFolderExecuteFile = 'X'

WriteAttributes = 'WA'

WriteExtendedAttributes = 'WEA'

class cterasdk.edge.enum.**BackupConfStatusID**

Bases: object

Status of backup configuration

Variables

- **NotInitialized** (*str*) – Backup configuration was not initialized
- **Configuring** (*str*) – Backup is being configured
- **Attaching** (*str*) – Backup configuration is Attaching
- **Attached** (*str*) – Backup configuration is Attached
- **NoFolder** (*str*) – No Folder for backup
- **WrongPassword** (*str*) – Wrong password used
- **Failed** (*str*) – Backup configuration failed
- **Unsubscribed** (*str*) – Unsubscribed to backup
- **Unlicensed** (*str*) – Unlicensed" to backup
- **ClocksOutOfSync** (*str*) – Clocks are out of sync
- **GetFoldersList** (*str*) – Get folders list

Attached = 'Attached'

Attaching = 'Attaching'

ClocksOutOfSync = 'ClocksOutOfSync'

Configuring = 'Configuring'

Failed = 'Failed'

GetFoldersList = 'GetFoldersList'

NoFolder = 'NoFolder'

NotInitialized = 'NotInitialized'

Unlicensed = 'Unlicensed'

Unsubscribed = 'Unsubscribed'

WrongPassword = 'WrongPassword'

class cterasdk.edge.enum.**CIFSPacketSigning**

Bases: object

CIFS Packet signing options

Variables

- **Disabled** (*str*) – CIFS Packet signing is disabled
- **IfClientAgrees** (*str*) – Use CIFS Packet signing is client agrees
- **Required** (*str*) – Require CIFS Packet signing

Disabled = 'Disabled'

IfClientAgrees = 'If client agrees'

Required = 'Required'

class cterasdk.edge.enum.**ClientSideCaching**

Bases: object

Client side caching types

Variables

- **Manual** (*str*) – Manual client side caching
- **Documents** (*str*) – Documents client side caching
- **Disabled** (*str*) – Client side caching disabled

Disabled = 'disable'

Documents = 'documents'

Manual = 'manual'

class cterasdk.edge.enum.**FileAccessMode**

Bases: object

File Access Mode

Variables

- **RW** (*str*) – Read Write
- **RO** (*str*) – Read Only
- **NA** (*str*) – None

NA = 'None'

RO = 'ReadOnly'

RW = 'ReadWrite'

class cterasdk.edge.enum.**IPProtocol**

Bases: object

IP Protocol

Variables

- **TCP** (*str*) – TCP Protocol
- **UDP** (*str*) – UDP Protocol

TCP = 'TCP'

UDP = 'UDP'

class cterasdk.edge.enum.**License**

Bases: object

Gateway license types

Variables

- **EV4** (*str*) – EV4 license
- **EV8** (*str*) – EV8 license
- **EV16** (*str*) – EV16 license
- **EV32** (*str*) – EV32 license
- **EV64** (*str*) – EV64 license
- **EV128** (*str*) – EV128 license

EV128 = 'EV128'

EV16 = 'EV16'

EV32 = 'EV32'

EV4 = 'EV4'

EV64 = 'EV64'

EV8 = 'EV8'

class cterasdk.edge.enum.**LocalGroup**

Bases: object

Local Group types

Variables

- **Administrators** (*str*) – Administrators
- **ReadOnlyAdministrators** (*str*) – Read Only Administrators
- **Everyone** (*str*) – Everyone

Administrators = 'Administrators'

Everyone = 'Everyone'

ReadOnlyAdministrators = 'Read Only Administrators'

class cterasdk.edge.enum.**Mode**

Bases: object

Enum for operational mode

Variables

- **Enabled** (*str*) – Operational mode enabled
- **Disabled** (*str*) – Operational mode diabled

Disabled = 'disabled'

Enabled = 'enabled'

class cterasdk.edge.enum.**OperationMode**

Bases: object

Gateway operation mode

Variables

- **Disabled** (*str*) – Gateway is Disabled

- **CachingGateway** (*str*) – Gateway is in Caching mode

CachingGateway = 'CachingGateway'

Disabled = 'Disabled'

class cterasdk.edge.enum.**PrincipalType**

Bases: object

ACL Principal Type

Variables

- **LU** (*str*) – Local User
- **LG** (*str*) – Local Group
- **DU** (*str*) – Domain User
- **DG** (*str*) – Domain Group

DG = 'DomainGroup'

DU = 'DomainUser'

LG = 'LocalGroup'

LU = 'LocalUser'

class cterasdk.edge.enum.**RAIDLevel**

Bases: object

RAID Levels

Variables

- **JBOD** (*str*) – Linear concatenation
- **RAID_0** (*str*) – Stripe set
- **RAID_1** (*str*) – Mirror
- **RAID_5** (*str*) – Distributed parity
- **RAID_6** (*str*) – Dual parity

JBOD = 'linear'

LVM = 'LVM'

RAID_0 = '0'

RAID_1 = '1'

RAID_5 = '5'

RAID_6 = '6'

class cterasdk.edge.enum.**SMBProtocol**

Bases: object

SMB Protocol

Variables

- **SMB1** (*str*) – SMB v1

- **NT1** (*str*) – SMB v1
- **SMB2_02** (*str*) – Vista, Server 2008
- **SMB2_10** (*str*) – Windows 7, Server 2008 R2
- **SMB3_00** (*str*) – Windows 8, Server 2012
- **SMB3_02** (*str*) – Windows 8.1, Server 2012 R2
- **SMB3_11** (*str*) – Windows 10, Server 2016
- **SMB2** (*str*) – Windows 7, Server 2008
- **SMB3** (*str*) – SMB 3.1.1

NT1 = 'NT1'

SMB1 = 'NT1'

SMB2 = 'SMB2'

SMB2_02 = 'SMB2_02'

SMB2_10 = 'SMB2_10'

SMB3 = 'SMB3'

SMB3_00 = 'SMB3_00'

SMB3_02 = 'SMB3_02'

SMB3_11 = 'SMB3_11'

class cterasdk.edge.enum.**ServicesConnectionState**

Bases: object

Gateway connection status

Variables

- **ResolvingServers** (*str*) – The Edge Filer is resolving CTERA Portal servers
- **Connecting** (*str*) – The Edge Filer is in connecting to CTERA Portal
- **Attaching** (*str*) – The Edge Filer is attaching to CTERA Portal
- **Authenticating** (*str*) – The Edge Filer is authenticating to CTERA Portal
- **Disconnected** (*str*) – The Edge Filer is disconnected from CTERA Portal
- **Connected** (*str*) – The Edge Filer is connected to CTERA Portal

Attaching = 'Attaching'

Authenticating = 'Authenticating'

Connected = 'Connected'

Connecting = 'Connecting'

Disconnected = 'Disconnected'

ResolvingServers = 'ResolvingServers'

class cterasdk.edge.enum.**Severity**

Bases: object

Log severity levels

Variables

- **EMERGENCY** (*str*) – Emergency log level
- **ALERT** (*str*) – Alert log level
- **CRITICAL** (*str*) – Critical log level
- **ERROR** (*str*) – Error log level
- **WARNING** (*str*) – Warning log level
- **NOTICE** (*str*) – Notice log level
- **INFO** (*str*) – Info log level
- **DEBUG** (*str*) – Debug log level

ALERT = 'alert'

CRITICAL = 'critical'

DEBUG = 'debug'

EMERGENCY = 'emergency'

ERROR = 'error'

INFO = 'info'

NOTICE = 'notice'

WARNING = 'warning'

class cterasdk.edge.enum.**SourceType**

Bases: object

Source Host Type

Variables

- **Edge** (*str*) – This Edge Filer
- **Windows** (*str*) – Windows Server
- **ONTAP** (*str*) – NetApp ONTAP
- **OneFS** (*str*) – Isilon OneFS
- **Panzura** (*str*) – Panzura Freedom Filer
- **SGRID9_SMB** (*str*) – NetApp StorageGRID 9
- **SGRID11_SMB** (*str*) – NetApp StorageGRID 11
- **StorSimple** (*str*) – Microsoft Azure StorSimple

Edge = 'currentDevice'

ONTAP = 'netapp'

OneFS = 'isilon'

Panzura = 'panzura'

SGRID11_SMB = 'storageGrid11'

SGRID9_SMB = 'storageGrid9'

StorSimple = 'azureStorSimple'

Windows = 'windowsServer'

class cterasdk.edge.enum.**SyncStatus**

Bases: object

Gateway sync status

Variables

- **Off** (*str*) – Off
- **NotInitialized** (*str*) – Not Initialized
- **InitializingConnection** (*str*) – Initializing Connection
- **ConnectingFolders** (*str*) – Connecting Folders
- **Connected** (*str*) – Connected
- **ClocksOutOfSync** (*str*) – Clocks is Out Of Sync
- **ConnectionFailed** (*str*) – Connection Failed
- **InternalError** (*str*) – Internal Error
- **InvalidConfiguration** (*str*) – Invalid Configuration
- **VolumeUnavailable** (*str*) – Volume Unavailable
- **NoFolder** (*str*) – No Folder
- **DisconnectedPortal** (*str*) – Disconnected from Portal
- **ServiceUnavailable** (*str*) – Service is Unavailable
- **Unlicensed** (*str*) – Unlicensed
- **Synced** (*str*) – Synced
- **Syncing** (*str*) – Syncing
- **Scanning** (*str*) – Scanning
- **UpgradingDataBase** (*str*) – Upgrading Database
- **OutOfQuota** (*str*) – Out of Quota
- **RejectedByPolicy** (*str*) – Rejected by Policy
- **FailedFilesInReadOnlyFolder** (*str*) – Failed Files in Read Only Folder
- **ShouldSupportWinNtAcl** (*str*) – Should Support WinNt Acl
- **TakingSnapshot** (*str*) – Taking Snapshot
- **CatalogReadOnlyMode** (*str*) – Catalog ReadOnly Mode
- **InvalidAverageBlockSize** (*str*) – Invalid Average Block Size

CatalogReadOnlyMode = 'CatalogReadOnlyMode'

ClocksOutOfSync = 'ClocksOutOfSync'

Connected = 'Connected'

ConnectingFolders = 'ConnectingFolders'

ConnectionFailed = 'ConnectionFailed'

DisconnectedPortal = 'DisconnectedPortal'

FailedFilesInReadOnlyFolder = 'FailedFilesInReadOnlyFolder'

InitializingConnection = 'InitializingConnection'

InternalError = 'InternalError'

InvalidAverageBlockSize = 'InvalidAverageBlockSize'

InvalidConfiguration = 'InvalidConfiguration'

NoFolder = 'NoFolder'

NotInitialized = 'NotInitialized'

Off = 'Off'

OutOfQuota = 'OutOfQuota'

RejectedByPolicy = 'RejectedByPolicy'

Scanning = 'Scanning'

ServiceUnavailable = 'ServiceUnavailable'

ShouldSupportWinNtAcl = 'ShouldSupportWinNtAcl'

Synced = 'Synced'

Syncing = 'Syncing'

TakingSnapshot = 'TakingSnapshot'

Unlicensed = 'Unlicensed'

UpgradingDataBase = 'UpgradingDataBase'

VolumeUnavailable = 'VolumeUnavailable'

class cterasdk.edge.enum.TCPConnectRC

Bases: object

Open = 'Open'

class cterasdk.edge.enum.TaskStatus

Bases: object

Gateway task status

Variables

- **Failed** (*str*) – The task has failed
- **Running** (*str*) – The task is running
- **Completed** (*str*) – The task has completed

Completed = 'completed'

Failed = 'failed'

Running = 'running'

class cterasdk.edge.enum.**TaskType**

Bases: object

Migration Tool Task Type

Variables

- **Discovery** (*str*) – Discovery
- **Migration** (*str*) – Migration

Discovery = 0

Migration = 1

class cterasdk.edge.enum.**Traffic**

Bases: object

Traffic type

Variables

- **Upload** (*str*) – Upload
- **Download** (*str*) – Download

Download = 'Download'

Upload = 'Upload'

class cterasdk.edge.enum.**VolumeStatus**

Bases: object

Gateway volume status

Variables

- **Ok** (*str*) – Volume is ok
- **ContainsErrors** (*str*) – Volume contains errors
- **ReadOnly** (*str*) – Voilume is read only
- **Corrupted** (*str*) – Volume is corrupted”
- **Unknown** (*str*) – Volume status is unknown
- **Recovering** (*str*) – Volume is recovering
- **Mounting** (*str*) – Volume is mounting
- **Unmounted** (*str*) – Volume is unmounted
- **Formatting** (*str*) – Volume is formatting
- **Converting** (*str*) – Volume is converting

- **Resizing** (*str*) – Volume is resizing
- **Repairing** (*str*) – Volume is repairing
- **Checking** (*str*) – Volume is checking
- **KeyRequired** (*str*) – Volume required key
- **CheckingQuota** (*str*) – Checking volume quota

Checking = 'checking'

CheckingQuota = 'checkingQuota'

ContainsErrors = 'containsErrors'

Converting = 'converting'

Corrupted = 'corrupted'

Formatting = 'formatting'

KeyRequired = 'keyRequired'

Mounting = 'mounting'

Ok = 'ok'

ReadOnly = 'readOnly'

Recovering = 'recovering'

Repairing = 'repairing'

Resizing = 'resizing'

Unknown = 'unknown'

Unmounted = 'unmounted'

cterasdk.edge.ftp module

class cterasdk.edge.ftp.**FTP**(*gateway*)

Bases: *BaseCommand*

Gateway FTP configuration APIs

disable()

Disable FTP

enable()

Enable FTP

get_configuration()

Get the current FTP configuration

Return cterasdk.common.object.Object

is_disabled()

Check if the FTP server is disabled

modify(*allow_anonymous_ftp=None*, *anonymous_download_limit=None*, *anonymous_ftp_folder=None*, *banner_message=None*, *max_connections_per_ip=None*, *require_ssl=None*)

Modify the FTP Configuration. Parameters that are not passed will not be affected

Parameters

- **allow_anonymous_ftp** (*bool,optional*) – Enable/Disable anonymous FTP downloads
- **anonymous_download_limit** (*int,optional*) – Limit download bandwidth of anonymous connection in KB/sec per connection. 0 for unlimited
- **anonymous_ftp_folder** (*str,optional*) – Anonymous FTP Directory
- **banner_message** (*str,optional*) – FTP Banner Message
- **max_connections_per_ip** (*int,optional*) – Maximum Connections per Client
- **require_ssl** (*bool,optional*) – If Ture, allow only SSL/TLS connections

cterasdk.edge.firmware module

class cterasdk.edge.firmware.**Firmware**(*gateway*)

Bases: *BaseCommand*

Gateway Firmware upgrade API

upgrade(*file_path*, *reboot=True*, *wait_for_reboot=True*)

Upgrade the Filer firmware with the provided file

Parameters

- **file_path** (*str*) – Path to the local file to upload
- **reboot** (*bool,optional*) – Perform reboot after uploading the new firmware, defaults to True
- **wait_for_reboot** (*bool,optional*) – Wait for reboot to complete (if reboot is performed), defaults to True

class cterasdk.edge.firmware.**UploadTaskStatus**

Bases: object

COMPLETE = 1

FAIL = -1

IN_PROGRESS = 0

cterasdk.edge.groups module

class cterasdk.edge.groups.**Groups**(*gateway*)

Bases: *BaseCommand*

add_members(*group*, *members*)

Add members to a group

Parameters

- **group** (*str*) – Name of the group

- **members** (*list[cterasdk.edge.types.UserGroupEntry]*) – List of users and groups to add to the group

get(*name=None*)

Get Group. If a group name was not passed as an argument, a list of all local groups will be retrieved

Parameters
name (*str,optional*) – Name of the group

remove_members(*group*, *members*)

Remove members from a group

Parameters

- **group** (*str*) – Name of the group
- **members** (*list[cterasdk.edge.types.UserGroupEntry]*) – List of users and groups to remove from the group

cterasdk.edge.licenses module

class cterasdk.edge.licenses.**Licenses**(*gateway*)

Bases: *BaseCommand*

Edge Filer License Configuration APIs

apply(*ctera_license*)

Apply a license

Parameters
ctera_license (cterasdk.edge.enum.License) – License type

get()

Get the current Gateway License

static infer(*ctera_license*)

class cterasdk.edge.licenses.**LocalLicenses**(*gateway*)

Bases: *BaseCommand*

Edge Filer Local License Configuration APIs

add(*code*)

Install a local license. Use this option when running the Edge Filer as a standalone appliance.

Parameters
code (*str*) – License code

clear()

Remove all local licenses

get()

Retrieve a list of local licenses installed

cterasdk.edge.login module

class cterasdk.edge.login.**Login**(*gateway*)

Bases: *BaseCommand*

info()

Get login info

login(*username*, *password*)

logout()

cterasdk.edge.logs module

class cterasdk.edge.logs.**Logs**(*gateway*)

Bases: *BaseCommand*

Gateway Logs APIs

Variables
default_include (*list[str]*) – Default log fields - 'severity', 'time', 'msg', 'more'

default_include = ['severity', 'time', 'msg', 'more']

logs(*topic*, *include=None*, *minSeverity='info'*)

Fetch Gateway logs

Parameters

- **topic** (*str*) – Log Topic to fetch
- **include** (*list[str],optional*) – List of fields to include in the response, defailts to Logs.default_include
- **minSeverity** (cterasdk.edge.enum.Severity,*optional*) – Minimal log severity to fetch, defaults to cterasdk.edge.enum.Severity.INFO

Returns
Log lines

Return type
cterasdk.lib.iterator.Iterator

settings(*retention*, *min_severity=None*)

Configure log settings

Parameters

- **retention** (*int*) – Log retention period in days
- **min_severity** (cterasdk.edge.enum.Severity,*optional*) – Minimal log severity

cterasdk.edge.mail module

class cterasdk.edge.mail.**Mail**(*gateway*)

Bases: *BaseCommand*

Gateway Mail Server configuration APIs

disable()

Disable e-mail delivery using a custom SMTP server

enable(*smtp_server*, *port=25*, *username=None*, *password=None*, *use_tls=True*)

Enable e-mail delivery using a custom SMTP server

Parameters

- **smtp_server** (*str*) – Address of the SMTP Server
- **port** (*int,optional*) – The listening port of the SMTP Server, defaults to 25
- **username** (*str,optional*) – The user name of the SMTP Server, defaults to None
- **password** (*str,optional*) – The password of the SMTP Server, defaults to None
- **use_tls** (*bool,optional*) – Use TLS when connecting to the SMTP Server, defaults to True

cterasdk.edge.migration_tool module

class cterasdk.edge.migration_tool.**Discovery**(*migration_tool*)

Bases: *TaskManager*

add(*name*, *credentials*, *shares*, *auto_start=False*, *log_every_file=False*, *notes=None*)

Create a discovery task

Parameters

- **name** (*str*) – Task name
- **credentials** (cterasdk.edge.types.HostCredentials) – Target host credentials
- **auto_start** (*bool,optional*) – Start task after creation, defaults to False
- **log_every_file** (*bool,optional*) – Log every file, defaults to False
- **notes** (*str,optional*) – Task notes

Returns
Task

Return type
cterasdk.common.object.Object

list_tasks(*deleted=False*)

update(*task*, *name=None*, *notes=None*)

Update a discovery task

Parameters

- **name** (*str,optional*) – Task name
- **notes** (*str,optional*) – Task notes

class cterasdk.edge.migration_tool.**DiscoveryTask**(*task_id*, *task_type*, *name*, *created_at*, *source*, *source_type*, *last_status*, *shares*, *notes*, *log_every_file*)

Bases: *Task*

Class representing a migration tool discovery task

class cterasdk.edge.migration_tool.**Jobs**(*jobs*)

Bases: object

Class representing task jobs

property all

Get all jobs of a task

property latest

Get the latest job of a task

class cterasdk.edge.migration_tool.**Migration**(*migration_tool*)

Bases: *TaskManager*

add(*name*, *credentials*, *shares*, *auto_start=False*, *access_time=None*, *winacls=True*, *cloud_folder=None*, *create_cloud_folder_per_share=False*, *compute_checksum=False*, *exclude=None*, *include=None*, *notes=None*)

Create a discovery task

Parameters

- **name** (*str*) – Task name
- **credentials** (cterasdk.edge.types.HostCredentials) – Target host credentials
- **auto_start** (*bool,optional*) – Start task after creation, defaults to False
- **access_time** (*bool,optional*) – Copy last access time, defaults to None
- **winacls** (*bool,optional*) – Copy NTFS ACL's, defaults to True
- **cloud_folder** (*str,optional*) – Target cloud folder, if create_cloud_folder_per_share was set to True then this attribute serves as the cloud folder name prefix
- **create_cloud_folder_per_share** (*bool,optional*) – Create cloud folder per share, defaults to False
- **compute_checksum** (*bool,optional*) – Validate and report checksums post-migration, defaults to False
- **exclude** (*list(str),optional*) – List of patterns to exclude, defaults to None
- **include** (*list(str),optional*) – List of patterns to include, defaults to None
- **notes** (*str,optional*) – Task notes

Returns
Task

Return type
cterasdk.common.object.Object

list_tasks(*deleted=False*)

class cterasdk.edge.migration_tool.**MigrationTask**(*task_id*, *task_type*, *name*, *created_at*, *source*, *source_type*, *last_status*, *shares*, *notes*, *winacls*, *cloud_folder*, *create_cloud_folder_per_share*, *compute_checksum*, *exclude*, *include*, *access_time*, *schedule*, *throttling*)

Bases: *Task*

Class representing a migration tool migration task

class cterasdk.edge.migration_tool.**MigrationTool**(*gateway*)

Bases: *BaseCommand*

Edge Filer Migration Tool APIs

delete(*tasks*)

Delete tasks

Parameters
tasks (*list(cterasdk.common.object.Object)*) – List of tasks

details(*task*)

Get task details

list_shares(*credentials*)

Log in

Parameters
credentials (cterasdk.edge.types.HostCredentials) – Target host credentials

list_tasks(*deleted=False*)

List tasks

Parameters
deleted (*bool,optional*) – List deleted tasks, defaults to False

Returns
List of tasks

Return type
list(*cterasdk.common.object.Object*)

login()

Login to CTERA Migrate

restore(*tasks*)

Recover tasks

Parameters
tasks (*list(cterasdk.common.object.Object)*) – List of tasks

results(*task*)

start(*task*)

Start a task

Parameters
task (cterasdk.common.object.Object) – Task object

stop(*task*)

Stop a task

Parameters
task (cterasdk.common.object.Object) – Task object

class cterasdk.edge.migration_tool.**Task**(*task_id*, *task_type*, *name*, *created_at=None*, *source=None*, *source_type=None*, *last_status=None*, *shares=None*, *notes=None*)

Bases: *Object*

Class representing a migration tool task

static from_server_object(*server_object*)

class cterasdk.edge.migration_tool.**TaskManager**(*migration_tool*)

Bases: object

Class representing a migration tool task

cterasdk.edge.network module

class cterasdk.edge.network.**Network**(*gateway*)

Bases: *BaseCommand*

Gateway Network configuration APIs

add_static_route(*source_ip*, *destination_ip_mask*)

Set a Static Route

Parameters

- **source_ip** (*str*) – The source IP (192.168.15.55)
- **destination_ip_mask** (*str*) – The destination IP and CIDR block (10.5.0.1/32)

clean_all_static_routes()

Clean all Static routes

diagnose(*services*)

Test a TCP connection to a host over a designated port

Parameters
services (*list[cterasdk.edge.types.TCPService]*) – List of services, identified by a host and a port

Returns
A list of named-tuples including the host, port and a boolean value indicating whether TCP connection can be established

Return type
list[*cterasdk.edge.types.TCPConnectResult*]

enable_dhcp()

Enable DHCP

get_static_routes()

Get all Static Routes

get_status()

Retrieve the network interface status

ifconfig()

Retrieve the ip configuration

ipconfig()

Retrieve the ip configuration

iperf(*address*, *port=5201*, *threads=1*, *protocol='TCP'*, *direction='Upload'*, *retries=120*, *seconds=1*)

Invoke a network throughput test

Parameters

- **address** (*str*) – The host running the iperf server
- **port** (*int,optional*) – The iperf server port, defaults to 5201
- **threads** (*int,optional*) – The number of threads, defaults to 1
- **protocol** (cterasdk.edge.enum.IPProtocol,*optional*) – IP protocol, defaults to *'TCP'*
- **direction** (cterasdk.edge.enum.Traffic,*optional*) – Traffic direction, defaults to *'Upload'*
- **retries** (*int,optional*) – Number of retries when sampling the iperf task status, defaults to 120
- **seconds** (*int,optional*) – Number of seconds to wait between retries, defaults to 1

Returns

A string containing the iperf output

Return type

str

remove_static_route(*destination_ip_mask*)

Delete a Static Route

Parameters

destination_ip_mask (*str*) – The destination IP and CIDR block (10.5.0.1/32)

reset_mtu()

Set the default maximum transmission unit (MTU) settings

set_mtu(*mtu*)

Set a custom network maximum transmission unit (MTU)

Parameters

mtu (*int*) – Maximum transmission unit

set_static_ipaddr(*address*, *subnet*, *gateway*, *primary_dns_server*, *secondary_dns_server=None*)

Set a Static IP Address

Parameters

- **address** (*str*) – The static address
- **subnet** (*str*) – The subnet for the static address
- **gateway** (*str*) – The default gateway
- **primary_dns_server** (*str*) – The primary DNS server

- **secondary_dns_server** (*str,optinal*) – The secondary DNS server, defaults to None

set_static_nameserver(*primary_dns_server*, *secondary_dns_server=None*)

Set the DNS Server addresses statically

Parameters

- **primary_dns_server** (*str*) – The primary DNS server
- **secondary_dns_server** (*str,optinal*) – The secondary DNS server, defaults to None

tcp_connect(*service*)

Test a TCP connection between the Gateway and the provided host address

Parameters
service (cterasdk.edge.types.TCPService) – A service, identified by a host and a port

Returns
A named-tuple including the host, port and a boolean value indicating whether TCP connection can be established

Return type
cterasdk.edge.types.TCPConnectResult

cterasdk.edge.nfs module

class cterasdk.edge.nfs.**NFS**(*gateway*)

Bases: *BaseCommand*

Gateway NFS configuration

disable()

Disable NFS

enable()

Enable NFS

get_configuration()

Get the current NFS configuration

Return cterasdk.common.object.Object

is_disabled()

Check if the NFS server is disabled

modify(*async_write=None*, *aggregate_writes=None*, *mountd_port=None*, *statd_port=None*, *nfsv4_enabled=None*, *krb5_enabled=None*, *nfsd_host=None*)

Modify the FTP Configuration. Parameters that are not passed will not be affected

Parameters

- **async_write** (*bool,optional*) – If True, use asynchronous writes
- **aggregate_writes** (*bool,optional*) – If True, aggregate write requests
- **mountd_port** (*int,optional*) – Instruct mountd to bind to a specific port
- **statd_port** (*int,optional*) – Instruct statd to bind to a specific port
- **nfsv4_enabled** (*bool,optional*) – Enable NFSv4

- **krb5_enabled** (*bool, optional*) – Enable Kerberos. Note that NFS4V must be enabled to enable Kerberos
- **nfsd_host** (*str, optional*) – Instruct nfsd to bind to a specific network interface. Set to an empty string to clear

cterasdk.edge.ntp module

class cterasdk.edge.ntp.**NTP**(*gateway*)

Bases: *BaseCommand*

Gateway NTP configuration

disable()

Disable NTP

enable(*servers=None*)

Enable NTP

Parameters

servers (*list[str]*) – List of NTP servers address

get_configuration()

property servers

cterasdk.edge.ssh module

class cterasdk.edge.ssh.**SSH**(*gateway*)

Bases: *BaseCommand*

Edge Filer SSH daemon APIs

disable()

enable(*public_key=None*, *public_key_file=None*, *exponent=65537*, *key_size=2048*)

Enable the Edge Filer's SSH daemon

Parameters

- **public_key** (*str, optional*) – A PEM-encoded public key in OpenSSH format. If neither a public key nor public key file were specified, an RSA key pair will be generated automatically. The PEM-encoded private key will be saved to the default Downloads directory
- **public_key_file** (*str, optional*) – A path to the public key file
- **exponent** (*int, optional*) – The public exponent of the new key, defaults to *65537*
- **key_size** (*int, optional*) – The length of the modulus in bits, defaults to *2048*

cterasdk.edge.ssl module

class cterasdk.edge.ssl.**SSL**(*gateway*)

Bases: *BaseCommand*

Edge Filer SSL APIs

disable_http()

Disable HTTP access

enable_http()

Enable HTTP access

get_storage_ca()

Get object storage trusted CA certificate

import_certificate(*private_key*, **certificates*)

Import the Edge Filer's web server's SSL certificate

Parameters

- **private_key** (*str*) – The PEM-encoded private key, or a path to the PEM-encoded private key file
- **certificates** (*list[str]*) – The PEM-encoded certificates, or a list of paths to the PEM-encoded certificates

import_storage_ca(*certificate*)

Import the object storage trusted CA certificate

Parameters

certificate (*str*) – The PEM-encoded certificate or a path to the PEM-encoded server certificate file

is_http_disabled()

Check if HTTP access is disabled

is_http_enabled()

Check if HTTP access is enabled

remove_storage_ca()

Remove object storage trusted CA certificate

cterasdk.edge.power module

class cterasdk.edge.power.**Boot**(*gateway*, *retries=60*, *seconds=5*)

Bases: `object`

wait()

class cterasdk.edge.power.**Power**(*gateway*)

Bases: *BaseCommand*

Gateway Power APIs

reboot(*wait=False*)

Reboot the Gateway

Parameters
wait (*bool, optional*) – Wait for reboot to complete, defaults to False

reset(*wait=False*)

Reset the Gateway setting

Parameters
wait (*bool, optional*) – Wait for reset to complete, defaults to False

shutdown()

Shutdown the Gateway

cterasdk.edge.query module

class cterasdk.edge.query.**QueryParam**

Bases: *Object*

include_classname()

increment()

class cterasdk.edge.query.**QueryParamBuilder**

Bases: object

build()

countLimit(*countLimit*)

include(*include*)

put(*key*, *value*)

startFrom(*startFrom*)

cterasdk.edge.query.**query**(*CTERAHost*, *path*, *key*, *value*)

cterasdk.edge.query.**show**(*CTERAHost*, *path*, *key*, *value*)

cterasdk.edge.remote module

cterasdk.edge.remote.**login**(*Gateway*, *ticket*)

cterasdk.edge.remote.**obtain_ticket**(*Portal*, *device_name*)

cterasdk.edge.remote.**remote_access**(*Gateway*, *Portal*)

cterasdk.edge.rsync module

class cterasdk.edge.rsync.**RSync**(*gateway*)

Bases: *BaseCommand*

Gateway RSync configuration

disable()

Disable FTP

enable()

Enable FTP

get_configuration()

Get the current RSync configuration

Return cterasdk.common.object.Object

is_disabled()

Check if the Rsync server is disabled

modify(*port=None*, *max_connections=None*)

Modify the RSync Configuration. Parameters that are not passed will not be affected

Parameters

- **port** (*int*,*optional*) – RSync Port
- **max_connections** (*int*,*optional*) – Maximum Connections

cterasdk.edge.services module

class cterasdk.edge.services.**Services**(*gateway*)

Bases: *BaseCommand*

Gateway Cloud Services configuration APIs

activate(*server*, *user*, *code*, *ctera_license='EV16'*)

Activate the gateway using an activation code

Parameters

- **server** (*str*) – Address of the Portal
- **user** (*str*) – User for the Portal connection
- **code** (*str*) – Activation code for the Portal connection
- **ctera_license** (cterasdk.edge.enum.License,*optional*) – CTERA License, defaults to cterasdk.edge.enum.License.EV16

connect(*server*, *user*, *password*, *ctera_license='EV16'*)

Connect to a Portal.

The connect method will first validate the *license* argument,
ensure the Gateway can establish a TCP connection over port 995 to *server* using `Gateway.tcp_connect()` and verify the Portal does not require device activation via code

Parameters

- **server** (*str*) – Address of the Portal

- **user** (*str*) – User for the Portal connection
- **password** (*str*) – Password for the Portal connection
- **ctera_license** (cterasdk.edge.enum.License, *optional*) – CTERA License, defaults to cterasdk.edge.enum.License.EV16

connected()

Check if the Edge Filer is connected to CTERA Portal

disable_sso()

Disable SSO connection

disconnect()

Disconnect from the Portal

enable_sso()

Enable SSO connection

get_status()

Retrieve the cloud services connection status

reconnect()

Reconnect to the Portal

sso_enabled()

Is SSO connection enabled

Return bool

True if SSO connection is enabled, else False

cterasdk.edge.session module

class cterasdk.edge.session.**Session**(*host*)

Bases: *SessionBase*

disable_remote_access()

enable_remote_access()

local()

remote()

remote_access()

remote_from()

start_remote_session(*remote_session*)

class cterasdk.edge.session.**SessionConnection**(*session_type*, *remote_from=None*)

Bases: *Object*

class cterasdk.edge.session.**SessionType**

Bases: object

Local = 'local'

Remote = 'remote'

cterasdk.edge.shares module

class cterasdk.edge.shares.**Shares**(*gateway*)

Bases: *BaseCommand*

add(*name*, *directory*, *acl=None*, *access='winAclMode'*, *csc='manual'*, *dir_permissions=777*, *comment=None*, *export_to_afp=False*, *export_to_ftp=False*, *export_to_nfs=False*, *export_to_pc_agent=False*, *export_to_rsync=False*, *indexed=False*, *trusted_nfs_clients=None*, *uuid=None*)

Add a network share.

Parameters

- **name** (*str*) – The share name
- **directory** (*str*) – Full directory path
- **acl** (*list[cterasdk.edge.types.ShareAccessControlEntry]*) – List of access control entries
- **access** (*cterasdk.edge.enum.Acl*) – The Windows File Sharing authentication mode, defaults to `winAclMode`
- **csc** (*cterasdk.edge.enum.ClientSideCaching*) – The client side caching (offline files) configuration, defaults to `manual`
- **dir_permissions** (*int*) – Directory Permission, defaults to 777
- **comment** (*str*) – Comment
- **export_to_afp** (*bool*) – Whether to enable AFP access, defaults to `False`
- **export_to_ftp** (*bool*) – Whether to enable FTP access, defaults to `False`
- **export_to_nfs** (*bool*) – Whether to enable NFS access, defaults to `False`
- **export_to_pc_agent** (*bool*) – Whether to allow as a destination share for CTERA Backup Agents, defaults to `False`
- **export_to_rsync** (*bool*) – Whether to enable access over rsync, defaults to `False`
- **indexed** (*bool*) – Whether to enable indexing for search, defaults to `False`
- **trusted_nfs_clients** (*list[cterasdk.edge.types.NFSv3AccessControlEntry]*) – Trusted NFS v3 clients, defaults to `None`

add_acl(*name*, *acl*)

Add one or more access control entries to an existing share.

Parameters

- **name** (*str*) – The share name
- **acl** (*list[cterasdk.edge.types.ShareAccessControlEntry]*) – List of access control entries to add

add_screened_file_types(*name*, *extensions*)

Add extensions to the share's current list of blocked file extensions

Parameters

- **name** (*str*) – The share name
- **extensions** (*list[str]*) – List of file extensions to add

add_trusted_nfs_clients(*name*, *trusted_nfs_clients*)

Add one or more trusted NFS client entries to an existing share.

Parameters

- **name** (*str*) – The share name
- **trusted_nfs_clients** (*list[cterasdk.edge.types.NFSv3AccessControlEntry]*) – Trusted NFS v3 clients

block_files(*name*, *extensions*)

Configure a share to block one or more file extensions

Parameters

- **name** (*str*) – The share name
- **extensions** (*list[str]*) – List of file extensions to block

delete(*name*)

Delete a share.

Parameters
name (*str*) – The share name

get(*name=None*)

Get Share. If a share name was not passed as an argument, a list of all shares will be retrieved :param str,optional name: Name of the share

get_access_type(*name*)

Get the network share Windows File Sharing authentication mode

Parameters
name (*str*) – The share name

get_acl(*name*)

Get the current access control entries from an existing share.

Parameters
name (*str*) – The share name

get_screened_file_types(*name*)

Get the share's current list of blocked file extensions

Parameters
name (*str*) – The share name

get_trusted_nfs_clients(*name*)

Get the current trusted NFS client entries from an existing share.

Parameters
name (*str*) – The share name

modify(*name*, *directory=None*, *acl=None*, *access=None*, *csc=None*, *dir_permissions=None*, *comment=None*, *export_to_afp=None*, *export_to_ftp=None*, *export_to_nfs=None*, *export_to_pc_agent=None*, *export_to_rsync=None*, *indexed=None*, *trusted_nfs_clients=None*)

Modify an existing network share. All parameters but name are optional and default to None

Parameters

- **name** (*str*) – The share name
- **directory** (*str,optional*) – Full directory path

- **acl** (*list[cterasdk.edge.types.ShareAccessControlEntry],optional*) – List of access control entries
- **access** (*cterasdk.edge.enum.Acl,optional*) – The Windows File Sharing authentication mode
- **csc** (*cterasdk.edge.enum.ClientSideCaching,optional*) – The client side caching (offline files) configuration
- **dir_permissions** (*int,optional*) – Directory Permission
- **comment** (*str,optional*) – Comment
- **export_to_afp** (*bool,optional*) – Whether to enable AFP access
- **export_to_ftp** (*bool,optional*) – Whether to enable FTP access
- **export_to_nfs** (*bool,optional*) – Whether to enable NFS access
- **export_to_pc_agent** (*bool,optional*) – Whether to allow as a destination share for CTERA Backup Agents
- **export_to_rsync** (*bool,optional*) – Whether to enable access over rsync
- **indexed** (*bool,optional*) – Whether to enable indexing for search
- **trusted_nfs_clients** (*list[cterasdk.edge.types.NFSv3AccessControlEntry]*) – Trusted NFS v3 clients, defaults to None

remove_acl(*name*, *acl*)

Remove one or more access control entries from an existing share.

Parameters

- **name** (*str*) – The share name
- **acl** (*list[cterasdk.edge.types.RemoveShareAccessControlEntry]*) – List of access control entries to remove

remove_screened_file_types(*name*, *extensions*)

Remove extensions from the share's current list of blocked file extensions

Parameters

- **name** (*str*) – The share name
- **extensions** (*list[str]*) – List of file extensions to remove

remove_trusted_nfs_clients(*name*, *trusted_nfs_clients*)

Remove one or more trusted NFS client entries from an existing share.

Parameters

- **name** (*str*) – The share name
- **trusted_nfs_clients** (*list[cterasdk.edge.types.RemoveNFSv3AccessControlEntry]*) – Trusted NFS v3 clients

set_access_type(*name*, *access*)

Set the network share Windows File Sharing authentication mode

Parameters

- **name** (*str*) – The share name
- **access** (*cterasdk.edge.enum.Acl*) – The Windows File Sharing authentication mode

set_acl(*name*, *acl*)

Set a network share's access control entries.

Parameters

- **name** (*str*) – The share name
- **acl** (*list[cterasdk.edge.types.ShareAccessControlEntry]*) – List of access control entries

> **Warning:** this method will override the existing access control entries

set_screened_file_types(*name*, *extensions*)

Set the share's current list of blocked file extensions (override the current list)

Parameters

- **name** (*str*) – The share name
- **extensions** (*list[str]*) – List of file extensions to block

set_share_winacls(*name*)

Set a network share to use Windows ACL Emulation Mode

Parameters

name (*str*) – The share name

set_trusted_nfs_clients(*name*, *trusted_nfs_clients*)

Set a network share's trusted NFS client entries.

Parameters

- **name** (*str*) – The share name
- **trusted_nfs_clients** (*list[cterasdk.edge.types.NFSv3AccessControlEntry]*) – Trusted NFS v3 clients

> **Warning:** this method will override the existing access control entries

cterasdk.edge.shell module

class cterasdk.edge.shell.**Shell**(*gateway*)

Bases: *BaseCommand*

Gateway Shell command

run_command(*shell_command*, *wait=True*)

Execute a shell command on the gateway

Parameters

- **shell_command** (*str*) – The shell command to execute
- **wait** (*bool,optional*) – Wait for the command to execute, defaults to `True`

Returns

The command result, or the task url path when wait equals `False`

cterasdk.edge.smb module

class cterasdk.edge.smb.**SMB**(*gateway*)

Bases: *BaseCommand*

Gateway SMB configuration APIs

disable()

Disable SMB

disable_abe()

Disable ABE

enable()

Enable SMB

enable_abe()

Enable ABE

get_configuration()

Get current SMB Configuration

Return cterasdk.common.object.Object
SMB configuration

modify(*packet_signing=None*, *idle_disconnect_time=None*, *compatibility_mode=None*, *unix_extensions=None*, *abe_enabled=None*, *min_client_protocol=None*, *max_client_protocol=None*, *min_server_protocol=None*, *max_server_protocol=None*)

Modify the current SMB Configuration. Parameters that are not passed will not be affected

Parameters

- **packet_signing,optional** (cterasdk.edge.enum.CIFSPacketSigning) – Packet signing type
- **idle_disconnect_time** (*int,optional*) – Client idle disconnect timeout
- **compatibility_mode** (*bool,optional*) – Enable/Disable compatibility mode
- **unix_extensions** (*bool,optional*) – Enable/Disable unix extensions
- **abe_enabled** (*bool,optional*) – Enable/Disable ABE
- **min_client_protocol** (cterasdk.edge.enum.SMBProtocol,*optional*) – Minimum client protocol version
- **max_client_protocol** (cterasdk.edge.enum.SMBProtocol,*optional*) – Maximum client protocol version
- **min_server_protocol** (cterasdk.edge.enum.SMBProtocol,*optional*) – Minimum server protocol version
- **max_server_protocol** (cterasdk.edge.enum.SMBProtocol,*optional*) – Maximum server protocol version

restart()

set_packet_signing(*packet_signing*)

Set Packet signing

Parameters
packet_signing (cterasdk.edge.enum.CIFSPacketSigning) – Packet signing type

cterasdk.edge.support module

class cterasdk.edge.support.**DebugLevel**

Bases: object

aapi = 'aapi'

alert = 'alert'

apps = 'apps'

auth = 'auth'

av = 'av'

backup = 'backup'

caching = 'caching'

cbck = 'cbck'

cloud_extender = 'cloud_extender'

collaboration = 'collaboration'

cttp = 'cttp'

cttp_data = 'cttp_data'

db = 'db'

debug = 'debug'

dns = 'dns'

error = 'error'

error_abort = 'error_abort'

evictor = 'evictor'

evictor_verbose = 'evictor_verbose'

files = 'files'

http = 'http'

index = 'index'

info = 'info'

license = 'license'

none = 'none'

ntp = 'ntp'

process = 'process'

rsync = 'rsync'

samba = 'samba'

storage = 'storage'

upload = 'upload'

warning = 'warning'

class cterasdk.edge.support.**Support**(*gateway*)

Bases: *BaseCommand*

Gateway Support APIs

get_support_report()

Download support report

set_debug_level(**levels*)

Set the debug level

cterasdk.edge.sync module

class cterasdk.edge.sync.**CloudSyncBandwidthThrottling**(*gateway*)

Bases: *BaseCommand*

Edge Filer Cloud Sync Bandwidth Throttling APIs

get_policy()

Get the bandwidth throttling policy

Returns
a list of bandwidth throttling rules

Return type
list[*cterasdk.common.types.ThrottlingRule*]

set_policy(*rules*)

Set the bandwidth throttling policy

Parameters
rules (*list[*cterasdk.common.types.ThrottlingRule*]*) – List of bandwidth throttling rules

class cterasdk.edge.sync.**Sync**(*portal*)

Bases: *BaseCommand*

Edge Filer Cloud Sync APIs

Variables
throttling (cterasdk.edge.sync.CloudSyncBandwidthThrottling) – Object holding the Edge Filer's bandwidth throttling APIs

evict(*path*, *wait=False*)

Evict a directory from the Edge Filer

Parameters

- **path** (*str*) – Directory path
- **wait** (*bool*) – Wait for eviction task to complete, defaults to False

Returns
A reference to the background task

Return type
str

exclude_files(*extensions=None*, *filenames=None*, *paths=None*, *custom_exclusion_rules=None*)

Exclude files from Cloud Sync. This method will override any existing file exclusion rules Use *cterasdk.common.types.FileFilterBuilder()* to build custom file exclusion rules`

Parameters

- **extensions** (*list[str]*) – List of file extensions
- **filenames** (*list[str]*) – List of file names
- **paths** (*list[str]*) – List of file paths
- **rules** (*list[cterasdk.common.types.FilterBackupSet]*) – Set of custom exclusion rules

get_linux_avoid_using_fanotify()

get_status()

Retrieve the Cloud Sync status

is_disabled()

Check if Cloud Sync is disabled

is_enabled()

Check if Cloud Sync is enabled

refresh()

Refresh Cloud Folders

remove_file_exclusion_rules()

Remove previously configured sync exclusion rules

set_linux_avoid_using_fanotify(*avoid*)

suspend(*wait=True*)

Suspend Cloud Sync

Parameters
wait (*bool*) – Wait for synchronization to stop

unsuspend()

Unsuspend Cloud Sync

cterasdk.edge.syslog module

class cterasdk.edge.syslog.**Syslog**(*gateway*)

Bases: *BaseCommand*

Gateway Syslog configuration APIs

disable()

Disable Syslog

enable(*server*, *port=514*, *proto='UDP'*, *min_severity='info'*)

Enable Syslog

Parameters

- **server** (*str*) – Server address to send syslog logs
- **port** (*int,optional*) – Syslog server communication port, defaults to 514
- **proto** (cterasdk.edge.enum.IPProtocol,*optional*) – Syslog server communication protocol, defaults to cterasdk.edge.enum.IPProtocol.UDP
- **min_severity** (cterasdk.edge.enum.Severity,*optional*) – Minimal log severity to fetch, defaults to cterasdk.edge.enum.Severity.INFO

get_configuration()

modify(*server=None*, *port=None*, *proto=None*, *min_severity=None*)

Modify current Syslog configuration. Only configurations that are not None will be changed. Syslog must be enabled

Parameters

- **server** (*str,optional*) – Server address to send syslog logs
- **port** (*int,optional*) – Syslog server communication port
- **proto** (cterasdk.edge.enum.IPProtocol,*optional*) – Syslog server communication protocol
- **min_severity** (cterasdk.edge.enum.Severity,*optional*) – Minimal log severity to fetch

cterasdk.edge.taskmgr module

class cterasdk.edge.taskmgr.**Task**(*CTERAHost*, *ref*, *retries=10*, *seconds=1*)

Bases: TaskBase

get_task_status()

class cterasdk.edge.taskmgr.**Tasks**(*gateway*)

Bases: *BaseCommand*

Gateway Background Task APIs

by_name(*name*)

Get background tasks by name

Parameters
name (*str*) – Task name

running()

Get all running background tasks

status(*ref*)

Get background task status

Parameters
ref (*str*) – Task reference

wait(*ref*, *retries=100*, *seconds=1*)

Wait for background task to complete

Parameters

- **ref** (*str*) – Task reference
- **retries** (*int,optional*) – Number of retries when sampling the task status, defaults to 100
- **seconds** (*int,optional*) – Number of seconds to wait between retries, defaults to 1

cterasdk.edge.telnet module

class cterasdk.edge.telnet.**Telnet**(*gateway*)

Bases: *BaseCommand*

Gateway Telnet configuration APIs

disable()

Disable Telnet

enable(*code*)

Enable Telnet

cterasdk.edge.timezone module

class cterasdk.edge.timezone.**Timezone**(*gateway*)

Bases: *BaseCommand*

Gateway Timezone configuration

get_timezone()

Get the timezone of the gateway

Return str
The timezone of the gateway

set_timezone(*timezone*)

Set Timezone

Parameters
timezone (*str*) – New timezone to set

cterasdk.edge.types module

class cterasdk.edge.types.**AccessControlEntryValidator**

Bases: object

static validate_permission(*permission*)

class cterasdk.edge.types.**DeduplicationStatus**(*size*, *usage*)

Bases: *Object*

Edge Filer Local Deduplication Status Object

Variables

- **size** (*int*) – Logical Size in Bytes
- **usage** (*int*) – Actual Size in Bytes

class cterasdk.edge.types.**HostCredentials**(*host*, *username*, *password*, *host_type=None*)

Bases: object

Source Host Credential Object

Variables

- **host_type** (cterasdk.edge.enum.SourceType) – Host type
- **host** (*str*) – Fully qualified domain name, hostname or IP address
- **username** (*str*) – Source host account username
- **password** (*str*) – Source host account password

property host

property host_type

static localhost()

property password

property username

class cterasdk.edge.types.**NFSv3AccessControlEntry**(*address*, *netmask*, *perm*, *insecure=False*)

Bases: object

NFS v3 export access control entry :ivar str address: IP address, hostname or fully qualified domain name of client machine :ivar str netmask: Subnet mask :ivar cterasdk.edge.enum.FileAccessMode perm: File access permission :ivar bool insecure: Allow insecure NFS connections

property address

static from_server_object(*server_object*)

property insecure

property netmask

property perm

to_server_object()

class cterasdk.edge.types.**RemoveNFSv3AccessControlEntry**(*address*, *netmask*)

Bases: object

Object holding address and netmasak for NFS v3 export access control entry :ivar str address: IP address, hostname or fully qualified domain name of client machine :ivar str netmask: Subnet mask

property address

property netmask

class cterasdk.edge.types.**RemoveShareAccessControlEntry**(*principal_type*, *name*)

Bases: *UserGroupEntry*

Object holding share access control principal type and name

Variables

- **principal_type** (cterasdk.edge.enum.PrincipalType) – Principal type of the ACL
- **name** (*str*) – The name of the user or group

class cterasdk.edge.types.**ShareAccessControlEntry**(*principal_type*, *name*, *perm*)

Bases: object

Share access control entry for filer shares

Variables

- **principal_type** (cterasdk.edge.enum.PrincipalType) – Principal type of the ACL
- **name** (*str*) – The name of the user or group
- **perm** (cterasdk.edge.enum.FileAccessMode) – The file access permission

static from_server_object(*server_object*)

property name

property perm

property principal_type

to_server_object()

class cterasdk.edge.types.**TCPConnectResult**(*host*, *port*, *is_open*)

Bases: tuple

Tuple holding the host and port to connect over TCP

property host

The ip address, hostname or fully qualified domain name of the host

property is_open

Boolean, indicating whether a TCP connection can be successfully established to the target host over the specified port

property port

The port number

class cterasdk.edge.types.**TCPService**(*host*, *port*)

Bases: tuple

Tuple holding the host and port to connect over TCP

property host

The ip address, hostname or fully qualified domain name of the host

property port

The port number

class cterasdk.edge.types.**UserGroupEntry**(*principal_type*, *name*)

Bases: object

User or Group Entry

Variables

- **principal_type** (cterasdk.edge.enum.PrincipalType) – Principal type of the ACL
- **name** (*str*) – The name of the user or group

static from_server_object(*server_object*)

property principal_type

to_server_object()

cterasdk.edge.uri module

cterasdk.edge.uri.**api**(*Gateway*)

cterasdk.edge.uri.**files**(*Gateway*)

cterasdk.edge.uri.**local**(*baseurl*)

cterasdk.edge.uri.**remote**(*baseurl*, *tenant*, *device*)

cterasdk.edge.uri.**remote_access**(*baseurl*, *device*)

cterasdk.edge.users module

class cterasdk.edge.users.**Users**(*gateway*)

Bases: *BaseCommand*

Gateway Users configuration APIs

add(*username*, *password*, *full_name=None*, *email=None*, *uid=None*)

Add a user of the Gateway

Parameters

- **username** (*str*) – User name for the new user
- **password** (*str*) – Password for the new user
- **full_name** (*str,optional*) – The full name of the new user, defaults to None
- **email** (*str,optional*) – E-mail address of the new user, defaults to None
- **uid** (*str,optional*) – The uid of the new user, defaults to None

add_first_user(*username*, *password*, *email=''*)

Add the first user of the Gateway and login

Parameters

- **username** (*str*) – User name for the new user
- **password** (*str*) – Password for the new user
- **email** (*str,optional*) – E-mail address of the new user, defaults to an empty string

delete(*username*)

Delete an existing user

Parameters

username (*str*) – User name of the user to delete

get(*name=None*)

Get User. If a user name was not passed as an argument, a list of all local users will be retrieved

Parameters
name (*str,optional*) – Name of the user

modify(*username*, *password=None*, *full_name=None*, *email=None*, *uid=None*)

Modify an existing user of the Gateway

Parameters

- **username** (*str*) – User name to modify
- **password** (*str,optional*) – New password, defaults to None
- **full_name** (*str,optional*) – The full name of the user, defaults to None
- **email** (*str,optional*) – E-mail address of the user, defaults to None
- **uid** (*str,optional*) – The uid of the user, defaults to None

cterasdk.edge.volumes module

class cterasdk.edge.volumes.**Volumes**(*gateway*)

Bases: *BaseCommand*

Gateway Volumes configuration APIs

add(*name*, *size=None*, *filesystem='xfs'*, *device=None*, *passphrase=None*)

Add a new volume to the gateway

Parameters

- **name** (*str*) – Name of the new volume
- **size** (*int,optional*) – Size of the new volume, defaults to the device's size
- **filesystem** (*str,optional*) – Filesystem to use, defaults to xfs
- **device** (*str,optional*) – Name of the device to use for the new volume, can be left as None if there the gateway has only one
- **passphrase** (*str,optional*) – Passphrase for the volume

Returns
Gateway response

delete(*name*)

Delete a volume

Parameters
name (*str*) – Name of the volume to delete

delete_all()

Delete all volumes

get(*name=None*)

Get Volume. If a volume name was not passed as an argument, a list of all storage volumes will be retrieved
:param str,optional name: Name of the volume

modify(*name*, *size=None*)

Modify an existing volume

Parameters
size (*int,optional*) – New size of the volume, if not set, the size will not change

Returns
Gateway response

3.1.6 cterasdk.lib package

3.1.6.1 Submodules

cterasdk.lib.cmd module

class cterasdk.lib.cmd.**Command**(*cmd*, **args*)

Bases: object

cterasdk.lib.consent module

cterasdk.lib.consent.**ask**(*question*)

cterasdk.lib.filesystem module

class cterasdk.lib.filesystem.**FileSystem**

Bases: object

static compute_zip_file_name(*cloud_directory*, *files*)

copyfile(*src*, *dst*)

static exists(*filepath*)

static expanduser(*path*)

get_dirpath()

static get_local_file_info(*local_file*)

static instance()

static join(**paths*)

rename(*dirpath*, *src*, *dst*)

save(*dirpath*, *filename*, *handle*)

static split_file_directory(*path*)

static split_file_directory_with_defaults(*path=None*, *default_filename=None*)

Compute the destination file path.

Parameters

- **path** (*str*) – A path to a file or a folder

- **default_filename** (*str*) – The default file name to use unless *path* argument specifies a file path

Returns
A tuple including the destination directory and filename

Return type
(string, string)

validate_directory(*dirpath*)

static version(*filename*, *version*)

static write(*filepath*, *handle*)

cterasdk.lib.file_access_base module

class cterasdk.lib.file_access_base.**FileAccessBase**(*ctera_host*)

Bases: ABC

download(*path*, *destination=None*)

download_as_zip(*cloud_directory*, *files*, *destination=None*)

upload(*local_file*, *dest_path*)

cterasdk.lib.iterator module

class cterasdk.lib.iterator.**Iterator**(*function*, *param*)

Bases: object

Objects Iterator

cterasdk.lib.platform module

class cterasdk.lib.platform.**Platform**

Bases: object

arch()

static instance()

os()

python_version()

cterasdk.lib.registry module

class cterasdk.lib.registry.**Registry**

Bases: object

get(*key*)

static instance()

register(*key*, *value*)

remove(*key*)

cterasdk.lib.session_base module

class cterasdk.lib.session_base.**SessionBase**(*host*)

Bases: *Object*

property active

authenticated()

initializing()

is_local_auth()

start_local_session(*ctera_host*)

tenant()

terminate()

whoami()

class cterasdk.lib.session_base.**SessionStatus**

Bases: object

Active = 'Active'

Inactive = 'Inactive'

Initializing = 'Initializing'

class cterasdk.lib.session_base.**SessionUser**(*name*, *tenant=None*, *role=None*)

Bases: *Object*

cterasdk.lib.tempfile module

class cterasdk.lib.tempfile.**TempfileServices**

Bases: object

static mkdir()

static mkfile(*prefix*, *suffix*)

static rmdir()

cterasdk.lib.tracker module

exception cterasdk.lib.tracker.**ErrorStatus**(*status*)

Bases: *CTERAException*

class cterasdk.lib.tracker.**StatusTracker**(*CTERAHost*, *ref*, *success*, *progress*, *transient*, *failure*, *retries*, *seconds*)

Bases: object

failed()

increment()

resolve()

running()

successful()

track()

cterasdk.lib.tracker.**track**(*CTERAHost*, *ref*, *success*, *progress*, *transient*, *failure*, *retries=300*, *seconds=1*)

cterasdk.lib.version module

class cterasdk.lib.version.**Version**

Bases: object

as_header()

static instance()

3.1.7 cterasdk.object package

3.1.7.1 Submodules

cterasdk.object.Agent module

class cterasdk.object.Agent.**Agent**(*host*, *port=80*, *https=False*, *Portal=None*)

Bases: *CTERAHost*

Main class operating on a Agent

Variables

- **backup** (cterasdk.edge.backup.Backup) – Object holding the Agent Backup APIs
- **cli** (cterasdk.edge.cli.CLI) – Object holding the Agent CLI APIs
- **logs** (cterasdk.edge.logs.Logs) – Object holding the Agent Logs APIs
- **services** (cterasdk.edge.services.Services) – Object holding the Agent Services APIs
- **support** (cterasdk.edge.support.Support) – Object holding the Agent Support APIs
- **sync** (cterasdk.edge.sync.Sync) – Object holding the Agent Sync APIs

property base_api_url

cterasdk.object.Gateway module

class cterasdk.object.Gateway.**Gateway**(*host*, *port=None*, *https=False*, *Portal=None*)

Bases: *CTERAHost*

Main class operating on a Gateway

Variables

- **config** (cterasdk.edge.config.Config) – Object holding the Gateway Configuration APIs
- **network** (cterasdk.edge.network.Network) – Object holding the Gateway Network APIs
- **licenses** (cterasdk.edge.licenses.Licenses) – Object holding the Gateway Licenses APIs
- **services** (cterasdk.edge.services.Services) – Object holding the Gateway Services APIs
- **directoryservice** (cterasdk.edge.directoryservice.DirectoryService) – Object holding the Gateway Active Directory APIs
- **telnet** (cterasdk.edge.telnet.Telnet) – Object holding the Gateway Telnet APIs
- **syslog** (cterasdk.edge.syslog.Syslog) – Object holding the Gateway Syslog APIs
- **tasks** (cterasdk.edge.taskmgr.Tasks) – Object holding the Gateway Background Tasks APIs
- **audit** (cterasdk.edge.audit.Audit) – Object holding the Gateway Audit APIs
- **mail** (cterasdk.edge.mail.Mail) – Object holding the Gateway Mail APIs
- **backup** (cterasdk.edge.backup.Backup) – Object holding the Gateway Backup APIs
- **sync** (cterasdk.edge.sync.Sync) – Object holding the Gateway Sync APIs
- **cache** (cterasdk.edge.cache.Cache) – Object holding the Gateway Cache APIs
- **snmp** (*cterasdk.edge.snmp.SNMP*) – Object holding the Gateway SNMP APIs
- **ssl** (cterasdk.edge.ssl.SSL) – Object holding the Gateway SSL APIs
- **ssh** (*cterasdk.edge.ssl.SSH*) – Object holding the Gateway SSH APIs
- **power** (cterasdk.edge.power.Power) – Object holding the Gateway Power APIs
- **users** (cterasdk.edge.users.Users) – Object holding the Gateway Users APIs
- **groups** (cterasdk.edge.groups.Groups) – Object holding the Gateway Groups APIs
- **mtool** (cterasdk.edge.migration_tool.MigrationTool) – Object holding the Edge Filer's Migration Tool APIs
- **drive** (cterasdk.edge.drive.Drive) – Object holding the Gateway Drive APIs
- **volumes** (cterasdk.edge.volumes.Volumes) – Object holding the Gateway Volumes APIs
- **array** (cterasdk.edge.array.Array) – Object holding the Gateway Array APIs

- **shares** (cterasdk.edge.shares.Shares) – Object holding the Gateway Shares APIs
- **smb** (cterasdk.edge.smb.SMB) – Object holding the Gateway SMB APIs
- **aio** (cterasdk.edge.aio.AIO) – Object holding the Gateway AIO APIs
- **ftp** (cterasdk.edge.ftp.FTP) – Object holding the Gateway FTP APIs
- **afp** (cterasdk.edge.afp.AFP) – Object holding the Gateway AFP APIs
- **nfs** (cterasdk.edge.nfs.NFS) – Object holding the Gateway NFS APIs
- **rsync** (cterasdk.edge.rsync.RSync) – Object holding the Gateway RSync APIs
- **timezone** (cterasdk.edge.timezone.Timezone) – Object holding the Gateway Timezone APIs
- **logs** (cterasdk.edge.logs.Logs) – Object holding the Gateway Logs APIs
- **ntp** (cterasdk.edge.ntp.NTP) – Object holding the Gateway NTP APIs
- **shell** (cterasdk.edge.shell.Shell) – Object holding the Gateway Shell APIs
- **cli** (cterasdk.edge.cli.CLI) – Object holding the Gateway CLI APIs
- **dedup** (cterasdk.edge.dedup.Dedup) – Object holding the Gateway Local Deduplication APIs
- **support** (cterasdk.edge.support.Support) – Object holding the Gateway Support APIs
- **files** (*cterasdk.edge.files.FileBrowser*) – Object holding the Gateway File Browsing APIs
- **firmware** (cterasdk.edge.firmware.Firmware) – Object holding the Gateway Firmware APIs

property base_api_url

property base_file_url

property initialized

static make_local_files_dir(*full_path*)

query(*path*, *key*, *value*)

remote_access()

rm(*path*)

show_query(*path*, *key*, *value*)

test()

Verification check to ensure the target host is a Gateway.

cterasdk.object.Portal module

class cterasdk.object.Portal.**GlobalAdmin**(*host*, *port=None*, *https=True*)

Bases: *Portal*

Main class for Global Admin operations on a Portal

Variables

- **portals** (cterasdk.core.portals.Portals) – Object holding the Portals Management APIs
- **cli** (cterasdk.core.cli.CLI) – Object holding the Portal GlobalAdmin CLI APIs
- **servers** (cterasdk.core.servers.Servers) – Object holding the Servers Management APIs
- **setup** (cterasdk.core.setup.Setup) – Object holding the Portal setup APIs
- **ssl** (cterasdk.core.ssl.SSL) – Object holding the Portal SSL Certificate APIs
- **startup** (cterasdk.core.startup.Startup) – Object holding the Portal startup APIs
- **syslog** (cterasdk.core.syslog.Syslog) – Object holding the Portal syslog APIs
- **antivirus** (cterasdk.core.antivirus.Antivirus) – Object holding the Portal Antivirus APIs
- **buckets** (cterasdk.core.buckets.Buckets) – Object holding the Portal Storage Node APIs
- **messaging** (cterasdk.core.messaging.Messaging) – Object holding the Portal Messaging Service Management APIs

property **backups_base_path**

property **cloud_drive_base_path**

property **context**

class cterasdk.object.Portal.**Portal**(*host*, *port*, *https*)

Bases: *CTERAHost*

Parent class for communicating with the Portal through either GlobalAdmin or ServicesPortal

Variables

- **users** (cterasdk.core.users.Users) – Object holding the Portal user APIs
- **admins** (cterasdk.core.admins.Administrators) – Object holding the Portal GlobalAdmin users APIs
- **plans** (cterasdk.core.plans.Plans) – Object holding the Plan APIs
- **reports** (cterasdk.core.reports.Reports) – Object holding the Portal reports APIs
- **devices** (cterasdk.core.devices.Devices) – Object holding the Portal devices APIs
- **directoryservice** (cterasdk.core.directoryservice.DirectoryService) – Object holding the Portal Active Directory Service APIs
- **zones** (cterasdk.core.zones.Zones) – Object holding the Portal zones APIs
- **activation** (cterasdk.core.activation.Activation) – Object holding the Portal activation APIs

- **logs** (cterasdk.core.logs.Logs) – Object holding the Portal logs APIs
- **cloudfs** (cterasdk.core.cloudfs.CloudFS) – Object holding the Portal CloudFS APIs
- **settings** (*cterasdk.core.settings.Settings*) – Object holding the Portal Settings APIs
- **storage_classes** (*cterasdk.core.settings.StorageClasses*) – Object holding the Portal Storage Classes APIs
- **tasks** (cterasdk.core.taskmgr.Tasks) – Object holding the Portal Background Tasks APIs
- **templates** (cterasdk.core.templates.Templates) – Object holding the Portal Configuration Templates APIs
- **firmwares** (*cterasdk.core.firmwares.Firmwares*) – Object holding the Portal Firmware Repository APIs
- **files** (cterasdk.core.files.browser.FileBrowser) – Object holding the Portal File Browsing APIs

property backups_base_path

property base_api_url

property base_file_url

property base_portal_url

property cloud_drive_base_path

property context

iterator(*path*, *param*)

public_info()

Obtain the Portal's public info.

put(*path*, *value*, *use_file_url=False*)

Update a schema object or attribute.

query(*path*, *name*, *param*)

show_query(*path*, *name*, *param*)

test()

Verification check to ensure the target host is a Portal.

class cterasdk.object.Portal.**ServicesPortal**(*host*, *port=None*, *https=True*)

Bases: *Portal*

Main class for Service operations on a Portal

property backups_base_path

property cloud_drive_base_path

property context

3.1.8 cterasdk.transcript package

3.1.8.1 Submodules

cterasdk.transcript.transcribe module

class cterasdk.transcript.transcribe.**Transcribe**

Bases: object

COMMENT = '<!--TEMPLATE-->'

RECORDED_HEADERS = ['content-type', 'cookie']

SECTION_END = ' '

SECTION_START = ' '

transcribe(*request*, *response=None*)

cterasdk.transcript.transcribe.**transcribe**(*request*, *response=None*)

3.2 Submodules

3.2.1 cterasdk.config module

class cterasdk.config.**Logging**

Bases: object

static df()

static disable()

static enable()

static fmt()

static get()

static setLevel(*level*)

3.2.2 cterasdk.exception module

exception cterasdk.exception.**CTERAClientException**(*message=None*, *instance=None*, ***kwargs*)

Bases: *CTERAException*

exception cterasdk.exception.**CTERAConnectionError**(*message*, *instance*, *host*, *port*, *protocol*, ***kwargs*)

Bases: *CTERAException*

exception cterasdk.exception.**CTERAException**(*message=None*, *instance=None*, ***kwargs*)

Bases: Exception

join(*instance=None*)

put(***kwargs*)

exception cterasdk.exception.**ConnectionTimeout**(*message*, *seconds*, ***kwargs*)

Bases: *CTERAException*

exception cterasdk.exception.**ConsentException**

Bases: *CTERAException*

exception cterasdk.exception.**ExhaustedException**(*retries*, *timeout*)

Bases: *ConnectionTimeout*

exception cterasdk.exception.**FileSystemException**(*message=None*, *instance=None*, ***kwargs*)

Bases: *CTERAException*

exception cterasdk.exception.**HostUnreachable**(*instance*, *host*, *port*, *protocol*)

Bases: *CTERAConnectionError*

exception cterasdk.exception.**InputError**(*message*, *expression*, *options*)

Bases: *CTERAException*

exception cterasdk.exception.**LocalDirectoryNotFound**(*path*)

Bases: *FileSystemException*

exception cterasdk.exception.**LocalFileNotFound**(*path*)

Bases: *FileSystemException*

exception cterasdk.exception.**LocalPathNotFound**(*path*)

Bases: *FileSystemException*

exception cterasdk.exception.**ObjectNotFoundException**(*message*, *object_ref*, ***kwargs*)

Bases: *CTERAException*

exception cterasdk.exception.**PythonVersionException**(*version*)

Bases: *CTERAException*

exception cterasdk.exception.**RemoteDirectoryNotFound**(*path*)

Bases: *RemoteFileSystemException*

exception cterasdk.exception.**RemoteFileSystemException**(*message=None*, *instance=None*, ***kwargs*)

Bases: *CTERAException*

exception cterasdk.exception.**RenameException**(*dirpath*, *src*, *dst*)

Bases: *FileSystemException*

exception cterasdk.exception.**SSLException**(*host*, *port*, *reason*)

Bases: *CTERAConnectionError*

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

FIVE

HELP US IMPROVE THE DOCS <3

If you'd like to contribute an improvement to the site, its source is available on GitHub. Simply fork the repository and submit a pull request. Thank you!

PYTHON MODULE INDEX

C

INDEX

A

B

C

D

E

F

G

H

I

J

K

L

M

P

Q

R

S

T

U

V

W

X

Y

Z