PyU4V Documentation

Release 3.1.7

Ciara Stacke

Dec 04, 2019

Contents

CHAPTER 1

Installation Guide

1.1 Using pip

PyU4V is available on pip, so can be installed by simply running this command:

```
$ pip install PyU4V
```

1.2 Get the Source Code

Alternatively, you can clone the public repository (using either SSH or HTTPS):

```
$ git clone git@github.com:MichaelMcAleer/PyU4V.git
$ git clone https://github.com/MichaelMcAleer/PyU4V.git
```

Once you have a copy of the source, you can embed it in your own Python package, or install it into your site-packages easily:

```
$ cd PyU4V
$ pip install .
```

- genindex
- modindex

CHAPTER 2

Quick Start Guide

First, make sure that PyU4V is installed.

Let's get started with some simple examples.

Begin by importing the PyU4V package, and creating a connection to the Unisphere server by instantiating an instance of U4VConn. For now, we will pass in our login credentials on instantiation, but you can also create a configuration file (see *The Configuration File - PyU4V.conf*, below. Also see *Configuring SSL* below to get more information on the 'verify' parameter)

```
>>> import PyU4V

>>> conn = PyU4V.U4VConn(username='my_name', password='password', server_ip='10.0.0.1
↪', port='8443', verify=True)
```

Now we have a connection to the Unisphere server. Next, we must select an array for our queries. First, let's see what arrays we have in our environment:

```
>>> conn.common.get_array_list()
["000194900123", "000195900123", "000196900123", "000197900123", "000197900123"]
```

Let's pick one of these and set it as our array of choice:

```
>>> conn.set_array_id('000194900123')
```

If you wish to query another array without creating a new connection, this function can be called whenever required. The array_id can also be set on initialisation (PyU4V.U4VConn(array_id='000197123456')), or by putting it in the conf file.

Now we are ready to start making some calls! The functions are divided up into categories - common, provisioning, replication, and performance.

- Common covers a few utils and also covers system calls
- Provisioning covers all things provisioning and masking related, and corresponds with the 'sloprovisioning' endpoints

- Replication covers local and remote replication
- Performance covers all the performance related calls.

```
>>> conn.provisioning.get_host_list()
['host1', 'host2', 'host3']
>>> conn.replication.find_expired_snapvx_snapshots()
[{'storagegroup_name': 'my-storagegroup1', 'snapshot_name': 'my-temporary-snap',
↪'generation_number': '0',
  'expiration_time': '14:46:24 Wed, 24 Jan 2018', 'linked_sg_name': 'my-linked-sg',
  'snap_creation_time': '14:46:24 Wed, 23 Jan 2018'}]
```

2.1 The Configuration File - PyU4V.conf

Instead of passing the Unisphere server details in on initialisation, there is also the option to put them in a configuration file, 'PyU4V.conf'. This file also provides the option to setup logging in whatever way suits your project (please see here for further information on logging configuration). The configuration file should be placed in your working directory, or it can be placed in '~/.PyU4V/'. A local PyU4V file (i.e. in the current working directory) will override a conf file in '~/.PyU4V/'. Please see PyU4V.conf.example for an example conf file.

Please note that parameters passed in on initialisation will override those set in any configuration file, i.e. the priority goes

- First, parameters set on initialisation are selected,
- If any of these are unset, next any parameters set in a PyU4V.conf will be used,
- The local PyU4V.conf will be selected first, and if that cannot be found, a global file will be selected

2.2 Configuring SSL

We STRONGLY recommend that you configure the library to verify SSL. If not, you leave yourself open to MITM attacks and other potential security issues. However, you can disable SSL verification by setting 'verify=False' on initialisation, or in the configuration file.

To set:

1. Get the CA certificate of the Unisphere server.

```
# openssl s_client -showcerts -connect {server_hostname}:8443 </dev/null 2>/dev/
↪null|openssl x509 -outform PEM > {server_hostname}.pem
```

(This pulls the CA cert file and saves it as server_hostname.pem e.g. esxi01vm01.pem)

2. Either add the certificate to a ca-certificates bundle, OR add the path to the conf file/ pass it in as a parameter on initialisation:

- Copy the pem file to the system certificate directory:

```
# cp {server_hostname}.pem /usr/share/ca-certificates/{server_hostname}.crt
```

Update CA certificate database with the following commands (Ensure the new cert file is highlighted)

```
# dpkg-reconfigure ca-certificates
# update-ca-certificates
```

If the conf file is being used, ensure that if the 'verify' tag is present, that it is set to True ("verify=True") (If it is not set anywhere, 'verify' defaults to True)

OR

- In the conf file insert the following: verify=/{path-to-file}/{server_hostname}.pem OR pass the value in on initialization.

2.3 Recommendations

It is strongly recommended that you create a volume with a unique volume_name or volume_identifier. When you search for a volume device_id based on it's volume_name, it is preferable to receive a single device id rather than a list of device ids, of which any could be the device that you just created.

- genindex
- modindex

CHAPTER 3

Tools Guide

First, make sure that PyU4V is installed. Then visit the quick_start section to make sure you have secure connectivity to your array.

3.1 OpenStack

Description

This script facilitates the seamless(live) migration of volumes from the SMIS masking view structure to the REST masking view structure introduced in Pike. This is only applicable if you have existing volumes created in Ocata or an earlier release.

Important:

- Running this script is not necessary unless you intend 'Live Migrating' from one compute node to another.

Pre-requisites

1. The OpenStack system must first be successfully upgraded to Pike or a post Pike release.
2. All your existing compute nodes must be online.
3. Avoid executing any cinder operations when running migrate.py python script.
4. Avoid Unisphere for PowerMax upgrades or VMAX / PowerMAX OS upgrades when running migrate.py python script.

Recommendations

1. It is recommended to create a test instance in OpenStack to force a creation of a masking view on the array. When you run the script it should move the volumes to the child storage group associated with that volume type. If it does not and it creates a masking view or storage group with a slightly different name then please file a bug on the Github issues page for this project.

2. It is also recommended to move one volume first and verify it has been moved to the correct storage group within the correct masking view.
3. If in any doubt, please file an issue on the Github issues page for this project.

The script can be run using python 2.7, python3.6 and python 3.7. It is recommended you run from the PyU4V base directory, however you can run from the 'openstack' directory so long as you copy/create PyU4V.conf in that directory.

```
$ alias python3='/usr/bin/python3.7'
$ cd $PYU4V_WORKING_DIR
$ python3 PyU4V/tools/openstack/migrate.py
```

```
$ alias python3='/usr/bin/python3.7'
$ cp ./PyU4V.conf $PYU4V_WORKING_DIR/PyU4V/tools/openstack/.
$ cd $PYU4V_WORKING_DIR/PyU4V/tools/openstack
$ python3 migrate.py
```

Warning: Python 2.7 is nearing EOL and will not be maintained past 2020

Note:

- Only masking views that are eligible for migrating will be presented.
- You have the option to migrate all volume's or a subset of volumes, in a storage group.
- The old masking view and storage group will remain even if all volumes have been migrated, so you can always move them back if in any doubt.
- The new masking view will contain the same port group and initiator group as the original.
- If you find any issues, please open them on the Github issues page for this project.

- genindex
- modindex

CHAPTER 4

API Glossary

4.1 PyU4V package

4.1.1 PyU4V.univmax_conn module

Creates the connection with the Unisphere for VMAX instance.

univmax_conn.py.

class PyU4V.univmax_conn.**U4VConn**(*username=None*, *password=None*, *server_ip=None*, *port=None*, *verify=None*, *u4v_version='84'*, *interval=5*, *retries=200*, *array_id=None*, *application_type=None*)

Bases: object

U4VConn.

close_session()
Close the current rest session.

set_array_id(*array_id*)
Set the array serial number.

Parameters **array_id** – the array serial number

set_requests_timeout(*timeout_value*)
Set the requests timeout.

Parameters **timeout_value** – the new timeout value - int

4.1.2 PyU4V.common module

common.py.

class PyU4V.common.**CommonFunctions**(*request*, *interval*, *retries*, *u4v_version*)

Bases: object

CommonFunctions.

static check_status_code_success(*operation*, *status_code*, *message*)
Check if a status code indicates success.

Parameters

- **operation** – the operation
- **status_code** – the status code
- **message** – the server response

Raises VolumeBackendAPIException

static create_list_from_file(*file_name*)
Given a file, create a list from its contents.

Parameters **file_name** – the path to the file

Returns list of contents

create_resource(**args*, ***kwargs*)
Create a resource.

The args passed in are positional and should be passed in using the order they are listed in below.

Parameters

- **args** – Traditional Method param0 array_id: the array serial number param1 category: the resource category e.g. sloprovisioning param2 resource_type: the resource type e.g. maskingview
- **kwargs** – Traditional Method param version: optional version of Unisphere param resource_name: optional name of a specific resource param payload: optional payload dict
- **kwargs** – New Method param version: the version of Unisphere param no_version: (boolean) if the URI required no version param category: the resource category e.g. sloprovisioning, system param resource_level: the resource level e.g. storagegroup, alert param resource_level_id: the resource level ID param resource_type: the name of a specific resource param resource_type_id: the name of a specific resource param resource: the name of a specific resource param resource_id: the name of a specific resource param object_type: the name of a specific resource param object_type_id: the name of a specific resource param payload: optional payload dict

Returns message – string, server response

delete_resource(**args*, ***kwargs*)
Delete a resource.

The args passed in are positional and should be passed in using the order they are listed in below.

Parameters

- **args** – Traditional Method param0 array_id: the array serial number param1 category: the resource category e.g. sloprovisioning param2 resource_type: the resource type e.g. maskingview
- **kwargs** – Traditional Method param version: optional version of Unisphere param resource_name: optional name of a specific resource param payload: optional payload dict
- **kwargs** – New Method param version: the version of Unisphere param no_version: (boolean) if the URI required no version param category: the resource category e.g. sloprovisioning, system param resource_level: the resource level e.g. storagegroup, alert

param resource_level_id: the resource level ID param resource_type: the name of a specific resource param resource_type_id: the name of a specific resource param resource: the name of a specific resource param resource_id: the name of a specific resource param object_type: the name of a specific resource param object_type_id: the name of a specific resource param payload: optional payload dict

get_array(*array_id*)
Return details on specific array.

Returns server response

get_array_list(*filters=None*)
Return a list of arrays.

Parameters **filters** – optional dict of filters

Returns list

get_headroom(*array_id*, *workload*, *srp='SRP_1'*, *slo='Diamond'*)
Get the Remaining Headroom Capacity.

Get the headroom capacity for a given srp/ slo/ workload combination. Example output: [{'workloadType': 'OLTP', 'headroomCapacity': 29076.34, 'processingDetails': {'lastProcessedSpaTimestamp': 1485302100000, 'nextUpdate': 1670}, 'sloName': 'Diamond', 'srp': 'SRP_1', 'emulation': 'FBA'}]})

Parameters

- **array_id** – the array serial number
- **workload** – the workload type (DSS, OLTP, DSS_REP, OLTP_REP)
- **srp** – the storage resource pool. Default SRP_1.
- **slo** – the service level. Default Diamond.

Returns dict

get_iterator_page_list(*iterator_id*, *start*, *end*)
Get a page of results from an iterator instance.

Parameters

- **iterator_id** – the id of the iterator
- **start** – the start number
- **end** – the end number

Returns list of results

get_job_by_id(*job_id*)
Get details of a specific job.

Parameters **job_id** – the job id

get_request(*target_uri*, *resource_type*, *params=None*)
Send a GET request to the array.

Parameters

- **target_uri** – the target uri
- **resource_type** – the resource type, e.g. maskingview
- **params** – optional dict of filter params

Returns resource_object – dict or None

Raises ResourceNotFoundException

get_resource(**args*, ***kwargs*)
Get resource details from the array.

The args passed in are positional and should be passed in using the order they are listed in below.

Parameters

- **args** – Traditional Method param0 array_id: the array serial number param1 category: the resource category e.g. sloprovisioning param2 resource_type: the resource type e.g. maskingview
- **kwargs** – Traditional Method param version: optional version of Unisphere param resource_name: optional name of a specific resource param params: optional dict of filter params
- **kwargs** – New Method param version: the version of Unisphere param no_version: (boolean) if the URI required no version param category: the resource category e.g. sloprovisioning, system param resource_level: the resource level e.g. storagegroup, alert param resource_level_id: the resource level ID param resource_type: the name of a specific resource param resource_type_id: the name of a specific resource param resource: the name of a specific resource param resource_id: the name of a specific resource param object_type: the name of a specific resource param object_type_id: the name of a specific resource param params: query parameters

Returns resource object – dict

get_uni_version()
Get the unisphere version from the server.

Returns version and major_version(e.g. ("V8.4.0.16", "84"))

get_v3_or_newer_array_list(*filters=None*)
Return a list of V3 or newer arrays in the environment.

Parameters **filters** – optional dict of filters

Returns list of array ids

get_wlp_information(*array_id*)
Get the latest timestamp from WLP for processing New Workloads.

Ezample return: {"processingDetails": { "lastProcessedSpaTimestamp": 1517408700000, "nextUpdate": 1038}, "spaRegistered": True}

Returns dict

modify_resource(**args*, ***kwargs*)
Modify a resource.

The args passed in are positional and should be passed in using the order they are listed in below.

Parameters

- **args** – Traditional Method param0 array_id: the array serial number param1 category: the resource category e.g. sloprovisioning param2 resource_type: the resource type e.g. maskingview
- **kwargs** – Traditional Method param version: optional version of Unisphere param resource_name: optional name of a specific resource param payload: optional payload dict

- **kwargs** – New Method param version: the version of Unisphere param no_version: (boolean) if the URI required no version param category: the resource category e.g. sloprovisioning, system param resource_level: the resource level e.g. storagegroup, alert param resource_level_id: the resource level ID param resource_type: the name of a specific resource param resource_type_id: the name of a specific resource param resource: the name of a specific resource param resource_id: the name of a specific resource param object_type: the name of a specific resource param object_type_id: the name of a specific resource param payload: optional payload dict

Returns message – string (server response)

static read_csv_values(*file_name*)
Read any csv file with headers.

You can extract the multiple lists from the headers in the CSV file. In your own script, call this function and assign to data variable, then extract the lists to the variables. Example: data=ru.read_csv_values(mycsv.csv) sgnamelist = data['sgname'] policylist = data['policy']

Parameters **file_name** – path to CSV file

Returns Dictionary of data parsed from CSV

wait_for_job(*operation*, *status_code*, *job*)
Check if call is async, wait for it to complete.

Parameters

- **operation** – the operation being performed
- **status_code** – the status code
- **job** – the job

Returns task – list of dicts detailing tasks in the job

Raises VolumeBackendAPIException

wait_for_job_complete(*job*)
Given the job wait for it to complete.

Parameters **job** – the job dict

Returns rc – int, result – string, status – string, task – list of dicts detailing tasks in the job

Raises VolumeBackendAPIException

4.1.3 PyU4V.performance module

performance.py.

class PyU4V.performance.**PerformanceFunctions**(*array_id*, *request*, *common*, *provisioning*, *u4v_version*)

Bases: object

PerformanceFunctions.

generate_threshold_settings_csv(*outputcsvname*)
Generate a csv file with threshold settings.

Creates a CSV file with the following headers format containing current alert configuration for the given unisphere instance category,metric,firstthreshold,secondthreshold,notify,kpi array,HostReads,100000,300000,true,true array,HostWrites,100000,300000,true,false

Parameters **outputcsvname** – filename for CSV to be generated

get_all_fe_director_metrics(*start_date*, *end_date*)
Get a list of all Directors.

Calculate start and End Dates for Gathering Performance Stats Last 1 Hour.

Parameters

- **start_date** – start date
- **end_date** – end date

Returns director_results_combined

get_array_metrics(*start_date*, *end_date*)
Get array metrics.

Get all avaliable performance statistics for specified time period return in JSON.

Parameters

- **start_date** – EPOCH Time
- **end_date** – Epoch Time

Returns array_results_combined

get_days_to_full(*category*, *array_id=None*)
Get days to full.

Requests Days to Full Metrics from performance stats requires at least 10 Days of Performance data

Returns Requested stats

get_director_info(*director_id*, *start_date*, *end_date*)
Get director performance information.

Get Director level information and performance metrics for specified time frame, hard coded to average numbers.

Parameters

- **director_id** – Director ID
- **start_date** – start date
- **end_date** – end date

Returns Combined payload

get_fe_director_list()
Get list of all FE Directors.

Returns director list

get_fe_director_metrics(*start_date*, *end_date*, *director*, *dataformat*)
Get one or more metrics for front end directors.

Parameters

- **start_date** – Date EPOCH Time in Milliseconds
- **end_date** – Date EPOCH Time in Milliseconds
- **director** – List of FE Directors
- **dataformat** – Average or Maximum

Returns JSON Payload, and RETURN CODE 200 for success

get_fe_port_list()
Get a list of all front end ports in the array.

Returns List of Directors and Ports

get_fe_port_metrics(*start_date*, *end_date*, *director_id*, *port_id*, *dataformat*, *metriclist*)
Get one or more Metrics for Front end Director ports.

The metric list can contain a list of one or more of PercentBusy, IOs, MBRead, MBWritten, MBs, AvgIOSize, SpeedGBs, MaxSpeedGBs, HostIOLimitIOs, HostIOLimitMBs.

Parameters

- **start_date** – Date EPOCH Time in Milliseconds
- **end_date** – Date EPOCH Time in Milliseconds
- **director_id** – Director id
- **port_id** – port id
- **dataformat** – Average or Maximum
- **metriclist** – list of one or more metrics

Returns JSON Payload, and RETURN CODE 200 for success

get_fe_port_util_last4hrs(*dir_id*, *port_id*)
Get stats for last 4 hours.

Currently only coded for one metric - can be adapted for multiple.

Returns Requested stats

get_host_metrics(*host*, *start_date*, *end_date*)
Get host metrics.

Get all avaliable host performance statiscics for specified time period return in JSON.

Parameters

- **host** – the host name
- **start_date** – EPOCH Time
- **end_date** – Epoch Time

Returns Formatted results

get_perf_category_threshold_settings(*category*)
Get performance threshold category settings.

Will accept valid category (categories listed from get_threshold_categories).

Parameters **category** –

Returns dict, sc

get_perf_threshold_categories()
Get performance threshold categories.

Returns category_list

get_port_group_metrics(*pg_id*, *start_date*, *end_date*)
Get Port Group Performance Metrics.

Parameters

- **pg_id** – port group id

- **start_date** – the start date
- **end_date** – the end date

Returns pg_results_combined

get_storage_group_metrics(*sg_id*, *start_date*, *end_date*)
Get storage group metrics.

Parameters

- **sg_id** – the storage group id
- **start_date** – the start date
- **end_date** – the end date

Returns sg_results_combined

set_perf_threshold_and_alert(*category*, *metric*, *firstthreshold*, *secondthreshold*, *notify*)
Set performance thresholds and alerts.

Function to set performance alerts, suggested use with CSV file to get parameter settings from user template. Default is to check for 3 out of 5 samples before returning alert, users may want to modify as potentially 3 of 5 could mean could take 25 minutes for an alert to be seen as samples are at 5 minute intervals.

Parameters

- **category** – the category name
- **metric** – the required metric
- **firstthreshold** – the first threshold
- **secondthreshold** – the second threshold
- **notify** – Notify user with Alert Boolean

set_perfthresholds_csv(*csvfilename*)
Set performance thresholds using a CSV file.

reads CSV file, and sets perforamnce threshold metrics, should be used with generate_threshold_settings_csv to produce CSV file that can be edited and uploaded. The CSV file should have the following headers format category,metric,firstthreshold,secondthreshold, notify,kpi,array,HostReads,100000,300000,True,True array,HostWrites,100000,300000,True,False Boolean values are case sensitive ensure that when editing file that they are True or False. KPI setting can not be changed with REST API in current implementation, if you change this value it will not be updated in the UI. Only notify alert Boolean can be changed with REST. Only KPI Metrics should be alterted on, note if you are changing default threshold values for metrics used for dashboard views these will also update the numbers used for your dashboards. It's not recommended to alert on every value as this will just create noise.

Parameters **csvfilename** – the path to the csv file

4.1.4 PyU4V.provisioning module

provisioning.py.

class PyU4V.provisioning.**ProvisioningFunctions**(*array_id*, *request*, *common*, *u4v_version*)

Bases: object

ProvisioningFunctions.

add_child_sg_to_parent_sg(*child_sg*, *parent_sg*)
Add a storage group to a parent storage group.

This method adds an existing storage group to another storage group, i.e. cascaded storage groups.

Parameters

- **child_sg** – the name of the child sg
- **parent_sg** – the name of the parent sg

add_existing_vol_to_sg(*sg_id*, *vol_ids*, *_async=False*)
Expand an existing storage group by adding existing volumes.

Parameters

- **sg_id** – the name of the storage group
- **vol_ids** – the device id of the volume - can be list
- **_async** – Flag to indicate if the call should be async

Returns dict

add_new_vol_to_storagegroup(*sg_id*, *num_vols*, *vol_size*, *cap_unit*, *_async=False*, *vol_name=None*, *create_new_volumes=None*)
Expand an existing storage group by adding new volumes.

Parameters

- **sg_id** – the name of the storage group
- **num_vols** – the number of volumes
- **vol_size** – the size of the volumes
- **cap_unit** – the capacity unit
- **_async** – Flag to indicate if call should be async
- **vol_name** – name to give to the volume, optional
- **create_new_volumes** – when true will force create new volumes, optional

Returns dict

create_empty_sg(*srp_id*, *sg_id*, *slo*, *workload*, *disable_compression=False*, *_async=False*)
Create an empty storage group.

Set the disable_compression flag for disabling compression on an All Flash array (where compression is on by default).

Parameters

- **srp_id** – the storage resource pool
- **sg_id** – the name of the new storage group
- **slo** – the service level agreement (e.g. Gold)
- **workload** – the workload (e.g. DSS)
- **disable_compression** – flag for disabling compression (AF only)
- **_async** – Flag to indicate if this call should be asyncronously executed

Returns dict

create_host(*host_name*, *initiator_list=None*, *host_flags=None*, *init_file=None*, *_async=False*)
Create a host with the given initiators.

Accepts either initiator_list, e.g. [10000000ba873cbf, 10000000ba873cba], or file. The initiators must not be associated with another host. An empty host can also be created by not passing any initiator ids.

Parameters

- **host_name** – the name of the new host
- **initiator_list** – list of initiators
- **host_flags** – dictionary of optional host flags to apply
- **init_file** – full path to file that contains initiator names
- **_async** – Flag to indicate if call should be _async

Returns dict

create_hostgroup(*hostgroup_id*, *host_list*, *host_flags=None*, *_async=False*)
Create a hostgroup containing the given hosts.

Parameters

- **hostgroup_id** – the name of the new hostgroup
- **host_list** – list of hosts
- **host_flags** – dictionary of optional host flags to apply
- **_async** – Flag to indicate if call should be async

Returns dict

create_masking_view_existing_components(*port_group_name*, *masking_view_name*, *storage_group_name*, *host_name=None*, *host_group_name=None*, *_async=False*)
Create a new masking view using existing groups.

Must enter either a host name or a host group name, but not both.

Parameters

- **port_group_name** – name of the port group
- **masking_view_name** – name of the new masking view
- **storage_group_name** – name of the storage group
- **host_name** – name of the host (initiator group)
- **host_group_name** – name of host group
- **_async** – flag to indicate if command should be run asynchronously

Returns dict

Raises InvalidInputException

create_multiport_portgroup(*portgroup_id*, *ports*)
Create a new portgroup.

Parameters

- **portgroup_id** – the name of the new port group
- **ports** – list of port dicts - {'directorId': director_id, 'portId': port_id}

Returns dict

create_non_empty_storagegroup(*srp_id*, *sg_id*, *slo*, *workload*, *num_vols*, *vol_size*, *cap_unit*, *disable_compression=False*, *_async=False*)

Create a new storage group with the specified volumes.

Generates a dictionary for json formatting and calls the create_sg function to create a new storage group with the specified volumes. Set the disable_compression flag for disabling compression on an All Flash array (where compression is on by default).

Parameters

- **srp_id** – the storage resource pool
- **sg_id** – the name of the new storage group
- **slo** – the service level agreement (e.g. Gold)
- **workload** – the workload (e.g. DSS)
- **num_vols** – the amount of volumes to be created – int
- **vol_size** – the size of each volume – string
- **cap_unit** – the capacity unit (MB, GB)
- **disable_compression** – Flag for disabling compression (AF only)
- **_async** – Flag to indicate if this call should be async

Returns dict

create_portgroup(*portgroup_id*, *director_id*, *port_id*)

Create a new portgroup.

Parameters

- **portgroup_id** – the name of the new port group
- **director_id** – the directoy id
- **port_id** – the port id

Returns dict

create_portgroup_from_file(*file_name*, *portgroup_id*)

Given a file with director:port pairs, create a portgroup.

Each director:port pair must be on a new line. Example director:port - FA-1D:4.

Parameters

- **file_name** – the path to the file
- **portgroup_id** – the name for the portgroup

Returns dict, status_code

create_storage_group(*srp_id*, *sg_id*, *slo*, *workload=None*, *do_disable_compression=False*, *num_vols=0*, *vol_size='0'*, *cap_unit='GB'*, *allocate_full=False*, *_async=False*, *vol_name=None*)

Create the volume in the specified storage group.

Parameters

- **srp_id** – the SRP (String)
- **sg_id** – the group name (String)
- **slo** – the SLO (String)

- **workload** – the workload (String)
- **do_disable_compression** – flag for disabling compression
- **num_vols** – number of volumes to be created
- **vol_size** – the volume size
- **cap_unit** – the capacity unit (MB, GB, TB, CYL)
- **allocate_full** – boolean to indicate if you want a thick volume
- **async** – Flag to indicate if call should be async
- **vol_name** – name to give to the volume, optional

Returns dict

create_volume_from_sg_return_dev_id(*volume_name*, *storagegroup_name*, *vol_size*, *cap_unit='GB'*)

Create a new volume in the given storage group.

Parameters

- **volume_name** – the volume name (String)
- **storagegroup_name** – the storage group name
- **vol_size** – volume size (String)
- **cap_unit** – the capacity unit, default 'GB'

Returns device_id

deallocate_volume(*device_id*)

Deallocate all tracks on a volume.

Necessary before deletion. Please note that it is not possible to know exactly when a deallocation is complete. This method will return when the array has accepted the request for deallocation; the deallocation itself happens as a background task on the array.

Parameters **device_id** – the device id

Returns dict

delete_host(*host_id*)

Delete a given host.

Cannot delete if associated with a masking view.

Parameters **host_id** – name of the host

delete_hostgroup(*hostgroup_id*)

Delete a given hostgroup.

Cannot delete if associated with a masking view.

Parameters **hostgroup_id** – name of the hostgroup

delete_masking_view(*maskingview_name*)

Delete a masking view.

Parameters **maskingview_name** – the masking view name

delete_portgroup(*portgroup_id*)

Delete a portgroup.

Parameters **portgroup_id** – the name of the portgroup

delete_storagegroup(*storagegroup_id*)
Delete a given storage group.

A storage group cannot be deleted if it is associated with a masking view.

Parameters **storagegroup_id** – the name of the storage group

delete_volume(*device_id*)
Delete a volume.

Parameters **device_id** – volume device id

extend_volume(*device_id*, *new_size*, *_async=False*)
Extend a VMAX volume.

Parameters

- **device_id** – volume device id
- **new_size** – the new required size for the device
- **_async** – flag to indicate if call should be async

find_host_lun_id_for_vol(*maskingview*, *device_id*)
Find the host_lun_id for a volume in a masking view.

Parameters

- **maskingview** – the masking view name
- **device_id** – the device ID

Returns host_lun_id – int

find_low_volume_utilization(*low_utilization_percentage*, *csvname*)
Find volumes under a certain utilization threshold.

Function to find volumes under a specified percentage, (e.g. find volumes with utilization less than 10%) - may be long running as will check all sg on array and all storage group. Only identifies volumes in storage group, note if volume is in more than one sg it may show up more than once.

Parameters

- **low_utilization_percentage** – low utilization watermark percent
- **csvname** – filename for CFV output file

Returns will create csvfile with name passed

find_volume_device_id(*volume_name*)
Given a volume identifier, find the corresponding device_id.

Parameters **volume_name** – the volume name

Returns device_id

find_volume_identifier(*device_id*)
Get the volume identifier of a VMAX volume.

Parameters **device_id** – the device id

Returns the volume identifier – string

get_child_sg_from_parent(*parent_name*)
Get child storage group list.

Parameters **parent_name** – the parent sg name

Returns list

get_common_masking_views(*portgroup_name*, *ig_name*)
Get common masking views for a given portgroup and initiator group.

Parameters

- **portgroup_name** – the port group name
- **ig_name** – the initiator group name

Returns masking view list

get_compressibility_report(*srp_id*)
Get a specified SRP Compressibility Report.

Parameters **srp_id** – the srp id

Returns list of compressibility reports

get_director(*director*)
Query for details of a director for a symmetrix.

Parameters **director** – the director ID e.g. FA-1D

Returns dict

get_director_list()
Query for details of Symmetrix directors for a symmetrix.

Returns director list

get_director_port(*director*, *port_no*)
Get details of the symmetrix director port.

Parameters

- **director** – the director ID e.g. FA-1D
- **port_no** – the port number e.g. 1

Returns dict

get_director_port_list(*director*, *filters=None*)
Get list of the ports on a particular director.

Can be filtered by optional parameters, please see documentation.

Parameters

- **director** – the director ID e.g. FA-1D
- **filters** – optional filters - dict

Returns list of port key dicts

get_element_from_masking_view(*maskingview_name*, *portgroup=False*, *host=False*, *storagegroup=False*)
Return the name of the specified element from a masking view.

Parameters

- **maskingview_name** – the masking view name
- **portgroup** – the port group name - optional
- **host** – the host name - optional
- **storagegroup** – the storage group name - optional

Returns name of the specified element – string

Raises ResourceNotFoundException

get_host(*host_id*)
Get details on a host on the array.

Parameters **host_id** – the name of the host, optional

Returns dict

get_host_from_maskingview(*masking_view_id*)
Given a masking view, get the associated host or host group.

Parameters **masking_view_id** – the name of the masking view

Returns host ID

get_host_list(*filters=None*)
Get list of the hosts on the array.

See documentation for applicable filters.

Parameters **filters** – optional list of filters - dict

Returns list of hosts

get_hostgroup(*hostgroup_id*)
Get details on a hostgroup on the array.

Parameters **hostgroup_id** – the name of the hostgroup

Returns dict

get_hostgroup_list(*filters=None*)
Get list of hostgroup(s) on the array.

See unisphere documentation for applicable filters.

Parameters **filters** – optional list of filters - dict

Returns dict

get_in_use_initiator_list_from_array()
Get the list of initiators which are in-use from the array.

Gets the list of initiators from the array which are in hosts/ initiator groups.

Returns list of in-use initiators

get_initiator(*initiator_id*)
Get details of an initiator.

Parameters **initiator_id** – initiator id, optional

Returns initiator details

get_initiator_group_from_initiator(*initiator*)
Given an initiator, get its corresponding initiator group, if any.

Parameters **initiator** – the initiator id

Returns found_init_group_name – string, or None

get_initiator_ids_from_host(*host_id*)
Get initiator details from a host.

Parameters **host_id** – the name of the host

Returns list of initiator IDs

get_initiator_list (*params=None*)
Retrieve initiator list from the array.

Parameters **params** – dict of optional params

Returns list of initiators

get_iscsi_ip_address_and_iqn (*port_id*)
Get the ip addresses from the director port.

Parameters **port_id** – the director port identifier

Returns (list of ip_addresses, iqn)

get_masking_view (*masking_view_name*)
Get details of a masking view.

Parameters **masking_view_name** – the masking view name

Returns masking view dict

get_masking_view_list (*filters=None*)
Get a masking view or list of masking views.

See unisphere documentation for possible filters.

Parameters **filters** – dictionary of filters

Returns list of masking views

get_masking_views_by_host (*initiatorgroup_name*)
Given a host (initiator group), retrieve the masking view name.

Retrieve the list of masking views associated with the given initiator group.

Parameters **initiatorgroup_name** – the name of the initiator group

Returns list of masking view names

get_masking_views_from_storage_group (*storagegroup*)
Return any masking views associated with a storage group.

Parameters **storagegroup** – the storage group name

Returns masking view list

get_maskingview_connections (*mv_name*, *filters=None*)
Get all connection information for a given masking view.

Parameters

- **mv_name** – the name of the masking view
- **filters** – dict of optional filter parameters

Returns list of masking view connection dicts

get_mv_from_sg (*storage_group*)
Get the associated masking views from a given storage group.

Parameters **storage_group** – the name of the storage group

Returns Masking view list

get_mvs_from_host (*host_id*)
Retrieve masking view information for a specified host.

Parameters **host_id** – the name of the host

Returns list of masking views

get_num_vols_in_sg(*storage_group_name*)
Get the number of volumes in a storage group.

Parameters **storage_group_name** – the storage group name

Returns num_vols – int

get_port_identifier(*director*, *port_no*)
Get the identifier (wwn) of the physical port.

Parameters

- **director** – the ID of the director
- **port_no** – the number of the port

Returns wwn (FC) or iqn (iscsi), or None

get_port_list(*filters=None*)
Query for a list of Symmetrix port keys.

Note a mixture of Front end, back end and RDF port specific values are not allowed. See UniSphere documentation for possible values.

Parameters **filters** – dictionary of filters e.g. {'vnx_attached': 'true'}

Returns list of port key dicts

get_portgroup(*portgroup_id*)
Get portgroup details.

Parameters **portgroup_id** – the name of the portgroup

Returns dict

get_portgroup_from_maskingview(*masking_view_id*)
Given a masking view, get the associated port group.

Parameters **masking_view_id** – the masking view name

Returns the name of the port group

get_portgroup_list(*filters=None*)
Get portgroup details.

Parameters **filters** – dict of optional filters

Returns list of portgroups

get_ports_from_pg(*portgroup*)
Get a list of port identifiers from a port group.

Parameters **portgroup** – the name of the portgroup

Returns list of port ids, e.g. ['FA-3D:35', 'FA-4D:32']

get_size_of_device_on_array(*device_id*)
Get the size of the volume from the array.

Parameters **device_id** – the volume device id

Returns size

get_slo(*slo_id*)
Get details on a specific service level.

Parameters **slo_id** – the service level agreement

Returns dict

get_slo_list(*filters=None*)
Retrieve the list of slo's from the array.

Returns slo_list – list of service level names

get_srp(*srp*)
Get details on a specific SRP.

Parameters **srp** – the storage resource pool

Returns dict

get_srp_list(*filters=None*)
Get a list of available SRP's on a given array.

Parameters **filters** – optional dict of filter parameters

Returns list

get_storage_group(*storage_group_name*)
Given a name, return storage group details.

Parameters **storage_group_name** – the name of the storage group

Returns storage group dict

get_storage_group_demand_report()
Get the storage group demand report.

Get the storage group demand report from Unisphere. Functionality only available in unisphere 9.0 assumes single SRP SRP_1 :returns: returns report

get_storage_group_list(*filters=None*)
Return a list of storage groups.

Parameters **filters** – optional filter parameters

Returns storage group list

get_storagegroup_from_maskingview(*masking_view_id*)
Given a masking view, get the associated storage group.

Parameters **masking_view_id** – the masking view name

Returns the name of the storage group

get_storagegroup_from_vol(*vol_id*)
Retrieve sg information for a specified volume.

Parameters **vol_id** – the device ID of the volume

Returns list of storage groups

get_target_wwns_from_pg(*portgroup_id*)
Get the director ports' wwns.

Parameters **portgroup_id** – the name of the portgroup

Returns target_wwns – the list of target wwns for the pg

get_vol_effective_wwn_details_84(*vol_list*)
Get the effective wwn for a list of vols.

Get volume details for a list of volume device ids, and write results to a csv file.

Parameters **vol_list** – list of device ids

Returns Dictionary

get_vols_from_storagegroup(*storagegroup_id*)
Retrieve volume information associated with a particular sg.

Parameters **storagegroup_id** – the name of the storage group

Returns list of device IDs of associated volumes

get_volume(*device_id*)
Get a VMAX volume from array.

Parameters **device_id** – the volume device id

Returns volume dict

get_volume_list(*filters=None*)
Get list of volumes from array.

Parameters **filters** – optional dictionary of filters

Returns list of device ids

get_workload_settings()
Get valid workload options from array.

Returns workload_setting – list of workload names

is_child_sg_in_parent_sg(*child_name*, *parent_name*)
Check if a child storage group is a member of a parent group.

Parameters

- **child_name** – the child sg name
- **parent_name** – the parent sg name

Returns bool

is_compression_capable()
Check if array is compression capable.

Returns bool

is_initiator_in_host(*initiator*)
Check to see if a given initiator is already assigned to a host.

Parameters **initiator** – the initiator ID

Returns bool

is_volume_in_storagegroup(*device_id*, *storagegroup*)
See if a volume is a member of the given storage group.

Parameters

- **device_id** – the device id
- **storagegroup** – the storage group name

Returns bool

modify_host(*host_id*, *host_flag_dict=None*, *remove_init_list=None*, *add_init_list=None*, *new_name=None*)

Modify an existing host.

Only one parameter can be modified at a time.

Parameters

- **host_id** – the host name
- **host_flag_dict** – dictionary of host flags
- **remove_init_list** – list of initiators to be removed
- **add_init_list** – list of initiators to be added
- **new_name** – new host name

Returns dict

modify_hostgroup(*hostgroup_id*, *host_flag_dict=None*, *remove_host_list=None*, *add_host_list=None*, *new_name=None*)

Modify an existing hostgroup.

Only one parameter can be modified at a time.

Parameters

- **hostgroup_id** – the name of the hostgroup
- **host_flag_dict** – dictionary of host flags
- **remove_host_list** – list of hosts to be removed
- **add_host_list** – list of hosts to be added
- **new_name** – new name of the hostgroup

Returns dict

modify_initiator(*initiator_id*, *remove_masking_entry=None*, *replace_init=None*, *rename_alias=None*, *set_fcid=None*, *initiator_flags=None*)

Modify an initiator.

Only one parameter can be edited at a time.

Parameters

- **initiator_id** – the initiator id
- **remove_masking_entry** – string - “true” or “false”
- **replace_init** – Id of the new initiator
- **rename_alias** – tuple (‘new node name’, ‘new port name’)
- **set_fcid** – set fcid value - string
- **initiator_flags** – dictionary of initiator flags to set

Returns dict

modify_portgroup(*portgroup_id*, *remove_port=None*, *add_port=None*, *rename_portgroup=None*)

Modify an existing portgroup.

Only one parameter can be modified at a time.

Parameters

- **portgroup_id** – the name of the portgroup

- **remove_port** – tuple of port details ($director_id, $portId)
- **add_port** – tuple of port details ($director_id, $portId)
- **rename_portgroup** – new portgroup name

Returns dict

modify_slo(*slo_id*, *new_name*)
Modify an SLO.

Currently, the only modification permitted is renaming.

Parameters

- **slo_id** – the current name of the slo
- **new_name** – the new name for the slo

Returns dict

modify_storage_group(*storagegroup*, *payload*)
Modify a storage group (PUT operation).

Parameters

- **storagegroup** – storage group name
- **payload** – the request payload

Returns message – dict, server response

move_volumes_between_storage_groups(*device_ids*, *source_storagegroup_name*, *target_storagegroup_name*, *force=False*, *_async=False*)
Move volumes to a different storage group.

Note: 8.4.0.7 or later

Parameters

- **source_storagegroup_name** – the originating storage group name
- **target_storagegroup_name** – the destination storage group name
- **device_ids** – the device ids - can be list
- **force** – force flag (necessary if volume is in masking view)
- **_async** – _async flag

remove_child_sg_from_parent_sg(*child_sg*, *parent_sg*)
Remove a storage group from its parent storage group.

This method removes a child storage group from its parent group.

Parameters

- **child_sg** – the name of the child sg
- **parent_sg** – the name of the parent sg

remove_vol_from_storagegroup(*sg_id*, *vol_id*, *_async=False*)
Remove a volume from a given storage group.

Parameters

- **sg_id** – the name of the storage group
- **vol_id** – the device id of the volume

- **_async** – Flag to indicate if call should be async

Returns dict

rename_masking_view(*masking_view_id*, *new_name*)
Rename an existing masking view.

Currently, the only supported modification is "rename".

Parameters

- **masking_view_id** – the current name of the masking view
- **new_name** – the new name of the masking view

Returns dict

rename_volume(*device_id*, *new_name*)
Rename a volume.

Parameters

- **device_id** – the volume device id
- **new_name** – the new name for the volume

set_host_io_limit_iops_or_mbps(*storage_group*, *iops*, *dynamic_distribution*, *mbps=None*)
Set the HOSTIO Limits on an existing storage group.

Parameters

- **storage_group** – String up to 32 Characters
- **dynamic_distribution** – valid values Always, Never, OnFailure
- **iops** – integer value. Min Value 100, must be specified to nearest 100, e.g.202 is not a valid value
- **mbps** – MB per second, integer value. Min Value 100

Returns dict

update_storagegroup_qos(*storage_group_name*, *qos_specs*)
Update the storagegroupinstance with qos details.

If maxIOPS or maxMBPS is in qos_specs, then DistributionType can be modified in addition to maxIOPS or/and maxMBPS If maxIOPS or maxMBPS is NOT in qos_specs, we check to see if either is set in StorageGroup. If so, then DistributionType can be modified. Example qos specs: {'maxIOPS': '4000', 'maxMBPS': '4000', 'DistributionType': 'Dynamic'}.

Parameters

- **storage_group_name** – the storagegroup instance name
- **qos_specs** – the qos specifications

Returns dict

4.1.5 PyU4V.replication module

replication.py.

class PyU4V.replication.**ReplicationFunctions**(*array_id*, *request*, *common*, *provisioning*, *u4v_version*)

Bases: object

ReplicationFunctions.

are_vols_rdf_paired(*remote_array*, *device_id*, *target_device*, *rdf_group*)
Check if a pair of volumes are RDF paired.

Parameters

- **remote_array** – the remote array serial number
- **device_id** – the device id
- **target_device** – the target device id
- **rdf_group** – the rdf group

Returns paired – bool, state – string

choose_snapshot_from_list_in_console(*storagegroup_id*)
Allow a user to select a snapshot from a list.

Parameters **storagegroup_id** – the storagegoup id

create_storagegroup_snap(*sg_name*, *snap_name*, *ttl=None*, *hours=False*)
Create a snapVx snapshot of a storage group.

To establish a new generation of an existing SnapVX snapshot for a source SG, use the same name as the existing snapshot for the new snapshot.

Parameters

- **sg_name** – the source group name
- **snap_name** – the name of the snapshot
- **ttl** – ttl in days, if any - int
- **hours** – Boolean, if set will specify TTL value is hours not days

create_storagegroup_srdf_pairings(*storagegroup_id*, *remote_sid*, *srdfmode*, *establish=None*, *_async=False*, *rdfg_number=None*, *forceNewRdfGroup=False*)
SRDF protect a storage group.

Parameters

- **storagegroup_id** – Unique string up to 32 Characters
- **remote_sid** – Type Integer 12 digit VMAX ID e.g. 000197000008
- **srdfmode** – String, values can be Active, AdaptiveCopyDisk, Synchronous, Asynchronous
- **establish** – default is none. Bool
- **_async** – Flag to indicate if call should be async (NOT to be confused with the SRDF mode)
- **rdfg_number** – the required RDFG number (optional)

Returns message and status Type JSON

delete_storagegroup_snapshot(*storagegroup*, *snap_name*, *gen_num=0*)
Delete the snapshot of a storagegroup.

Parameters

- **storagegroup** – the storage group name
- **snap_name** – the name of the snapshot

- **gen_num** – the generation number

delete_storagegroup_srdf(*storagegroup_id*, *rdfg_num=None*)
Delete srdf pairings for a given storagegroup.

Parameters

- **storagegroup_id** –
- **rdfg_num** – the rdfg number to remove pairings from - can be list

establish_storagegroup_srdf(*storagegroup_id*, *rdfg_no*, *establish_options=None*, *_async=False*)
Establish io on the links for the given storagegroup.

Optional parameters to set are "bypass", "metroBias", "star", "hop2", "force", "symForce", "full" - all true/false.

Parameters

- **storagegroup_id** – the storagegroup id
- **rdfg_no** – the rdf group no
- **establish_options** – Optional dict of establish params
- **_async** – flag to indicate async call

failback_storagegroup_srdf(*storagegroup_id*, *rdfg_no*, *failback_options=None*, *_async=False*)
Failback a given storagegroup.

Optional parameters to set are "bypass", "recoverPoint", "star", "hop2", "force", "symForce", "remote" - all true/false.

Parameters

- **storagegroup_id** – the storagegroup id
- **rdfg_no** – the rdf group no
- **failback_options** – Optional dict of failover params
- **_async** – flag to indicate async call

failover_storagegroup_srdf(*storagegroup_id*, *rdfg_no*, *failover_options=None*, *_async=False*)
Failover a given storagegroup.

Optional parameters to set are "bypass", "star", "restore", "immediate", "hop2", "force", "symForce", "remote", "establish" - all true/false.

Parameters

- **storagegroup_id** – the storagegroup id
- **rdfg_no** – the rdf group no
- **failover_options** – Optional dict of failover params
- **_async** – flag to indicate async call

find_expired_snapvx_snapshots()
Find all expired snapvx snapshots.

Parses through all Snapshots for array and lists those that have snapshots where the expiration date has passed however snapshots have not been deleted as they have links. Example output: [{'storagegroup_name': 'my-storagegroup1', 'snapshot_name': 'my-temporary-snap', 'generation_number': '0', 'expiration_time': '14:46:24 Wed, 24 Jan 2018', 'linked_sg_name': 'my-linked-sg', 'snap_creation_time': '14:46:24 Wed, 23 Jan 2018'}]

Returns list of dicts with expired snap details,

get_array_replication_capabilities()
Check what replication facilities are available.

Returns array_capabilities dict

get_rdf_group(*rdf_number*)
Get specific rdf group details.

Parameters **rdf_number** – the rdf number

get_rdf_group_list()
Get rdf group list from array.

Returns list of rdf group dicts with 'rdfgNumber' and 'label'

get_rdf_group_number(*rdf_group_label*)
Given an rdf_group_label, return the associated group number.

Parameters **rdf_group_label** – the group label

Returns rdf_group_number

get_rdf_group_volume(*rdf_number*, *device_id*)
Get specific volume details, from an RDF group.

Parameters

- **rdf_number** – the rdf group number
- **device_id** – the device id

get_rdf_group_volume_list(*rdf_number*)
Get specific volume details, from an RDF group.

Parameters **rdf_number** – the rdf group number

Returns list of device ids

get_replication_info()
Return replication information for an array.

Returns dict

get_snapshot_generation_details(*sg_id*, *snap_name*, *gen_num*)
Get the details for a particular snapshot generation.

Parameters

- **sg_id** – the storage group id
- **snap_name** – the snapshot name
- **gen_num** – Generation number

Returns dict

get_storage_group_rep(*storage_group_name*)
Given a name, return storage group details wrt replication.

Parameters **storage_group_name** – the name of the storage group

Returns storage group dict

get_storage_group_rep_list(*has_snapshots=False*, *has_srdf=False*)
Return list of storage groups with replication.

Parameters

- **has_snapshots** – return only storagegroups with snapshots
- **has_srdf** – return only storagegroups with SRDF

Returns list of storage groups with associated replication

get_storagegroup_snapshot_generation_list(*storagegroup_id*, *snap_name*)
Get a snapshot and its generation count information for an sg.

The most recent snapshot will have a gen number of 0. The oldest snapshot will have a gen number = genCount - 1 (i.e. if there are 4 generations of particular snapshot, the oldest will have a gen num of 3)

Parameters

- **storagegroup_id** – the name of the storage group
- **snap_name** – the name of the snapshot

Returns list of generation numbers

get_storagegroup_snapshot_list(*storagegroup_id*)
Get a list of snapshots associated with a storagegroup.

Parameters **storagegroup_id** – the storagegroup name

Returns list of snapshot names

get_storagegroup_srdf_details(*storagegroup_id*, *rdfg_num*)
Get the SRDF details for an RDF group on a particular storagegroup.

Parameters

- **storagegroup_id** – name of storage group
- **rdfg_num** – rdf number

Returns dict

get_storagegroup_srdfg_list(*storagegroup_id*)
Get the SRDF numbers for a storage group.

Parameters **storagegroup_id** – Storage Group Name of replicated group

Returns list of RDFG numbers

is_snapvx_licensed()
Check if the snapVx feature is licensed and enabled.

Returns True if licensed and enabled; False otherwise.

is_vol_in_rep_session(*device_id*)
Check if a volume is in a replication session.

Parameters **device_id** – the device id

Returns snapvx_tgt – bool, snapvx_src – bool, rdf_grp – list or None

link_gen_snapshot(*sg_id*, *snap_name*, *link_sg_name*, *_async=False*, *gen_num=0*)
Link a snapshot to another storage group.

Target storage group will be created if it does not exist.

Parameters

- **sg_id** – Source storage group name
- **snap_name** – name of the snapshot
- **link_sg_name** – the target storage group name
- **_async** – flag to indicate if call is async
- **gen_num** – generation number of a snapshot (int)

Returns dict

modify_storagegroup_snap(*source_sg_id*, *target_sg_id*, *snap_name*, *link=False*, *unlink=False*, *restore=False*, *new_name=None*, *gen_num=0*, *_async=False*)
Modify a storage group snapshot.

Please note that only one parameter can be modified at a time. Default action is not to create full copy

Parameters

- **source_sg_id** – the source sg id
- **target_sg_id** – the target sg id (Can be None)
- **snap_name** – the snapshot name
- **link** – Flag to indicate action = Link
- **unlink** – Flag to indicate action = Unlink
- **restore** – Flag to indicate action = Restore
- **new_name** – the new name for the snapshot
- **gen_num** – the generation number
- **_async** – flag to indicate if call should be async

modify_storagegroup_srdf(*storagegroup_id*, *action*, *rdfg*, *options=None*, *_async=False*)
Modify the state of an srdf.

This may be a long running task depending on the size of the SRDF group, can switch to async call if required.

Parameters

- **storagegroup_id** – name of storage group
- **action** – the rdf action e.g. Suspend, Establish, SetMode etc
- **rdfg** – rdf number
- **options** – a dict of possible options - depends on action type, example options={setMode': {'mode': 'Asynchronous'}}
- **_async** – flag to indicate if call should be async

rename_snapshot(*sg_id*, *snap_name*, *new_name*, *gen_num=0*)
Rename an existing storage group snapshot.

Parameters

- **sg_id** – the name of the storage group

- **snap_name** – the name of the snapshot
- **new_name** – the new name of the snapshot
- **gen_num** – generation number of a snapshot (int)

Returns dict

restore_snapshot(*sg_id*, *snap_name*, *gen_num=0*)
Restore a storage group to its snapshot.

Parameters

- **sg_id** – the name of the storage group
- **snap_name** – the name of the snapshot
- **gen_num** – the generation number of the snapshot (int)

Returns dict

suspend_storagegroup_srdf(*storagegroup_id*, *rdfg_no*, *suspend_options=None*, *_async=False*)
Suspend io on the links for the given storagegroup.

Optional parameters to set are "bypass", "metroBias", "star", "immediate", "hop2", "consExempt", "force", "symForce" - all true/false.

Parameters

- **storagegroup_id** – the storagegroup id
- **rdfg_no** – the rdf group no
- **suspend_options** – Optional dict of suspend params
- **_async** – flag to indicate async call

unlink_gen_snapshot(*sg_id*, *snap_name*, *unlink_sg_name*, *_async=False*, *gen_num=0*)
Unink a snapshot from another storage group.

Parameters

- **sg_id** – Source storage group name
- **snap_name** – name of the snapshot
- **unlink_sg_name** – the target storage group name
- **_async** – flag to indicate if call is async
- **gen_num** – generation number of a snapshot (int)

Returns dict

4.1.6 PyU4V.rest_requests module

rest_requests.py.

class PyU4V.rest_requests.**RestRequests**(*username*, *password*, *verify*, *base_url*, *application_type=None*)
Bases: object

RestRequests.

close_session()
Close the current rest session.

establish_rest_session()
establish_rest_session.

rest_request(*target_url*, *method*, *params=None*, *request_object=None*, *timeout=None*)
Send a request (GET, POST, PUT, DELETE) to the target api.

Parameters

- **target_url** – target url (string)
- **method** – The method (GET, POST, PUT, or DELETE)
- **params** – Additional URL parameters
- **request_object** – request payload (dict)
- **timeout** – optional timeout override

Returns server response object (dict), status code

4.1.7 PyU4V.tools.openstack.migrate_utils module

4.1.8 Subpackages

PyU4V.utils package

PyU4V.utils.config_handler module

config_handler.py.

PyU4V.utils.config_handler.**set_logger_and_config**(*file_path=None*)
Set logger and config file.

PyU4V.utils.constants module

Constants.py.

PyU4V.utils.exception module

Exception package.

exception PyU4V.utils.exception.**InvalidInputException**(*message=None*, ***kwargs*)
Bases: *PyU4V.utils.exception.PyU4VException*

InvalidInputException.

message = 'Invalid input received: %(data)s'

exception PyU4V.utils.exception.**MissingConfigurationException**(*message=None*, ***kwargs*)
Bases: *PyU4V.utils.exception.PyU4VException*

MissingConfigurationFileException

message = 'PyU4V settings not be loaded, please check file location or univmax_conn in

exception PyU4V.utils.exception.**PyU4VException**(*message=None*, ***kwargs*)

Bases: exceptions.Exception

PyU4VException.

code = 500

headers = {}

message = 'An unknown exception occurred.'

safe = False

exception PyU4V.utils.exception.**ResourceNotFoundException**(*message=None*, ***kwargs*)

Bases: *PyU4V.utils.exception.PyU4VException*

ResourceNotFoundException.

message = 'The requested resource was not found: %(data)s'

exception PyU4V.utils.exception.**UnauthorizedRequestException**(*message=None*, ***kwargs*)

Bases: *PyU4V.utils.exception.PyU4VException*

UnauthorizedRequestException.

meesage = 'Unauthorized request - please check credentials'

exception PyU4V.utils.exception.**VolumeBackendAPIException**(*message=None*, ***kwargs*)

Bases: *PyU4V.utils.exception.PyU4VException*

VolumeBackendAPIException.

message = 'Bad or unexpected response from the storage volume backend API: %(data)s'

__init__.py. __init__.py.

- genindex

CHAPTER 5

Overview

PyU4V is a python module that simplifies integration with the Unisphere for VMAX RestAPI interface.

It wraps REST calls with simple APIs and abstracts the HTTP request and response handling. For specifics on API arguments, consult the REST API guide for the VMAX release currently running on the target array. To get this documentation, navigate to https://{ip-address}:{port-number}/univmax/restapi/docs, where {ip-address}: the ip of your Unisphere server and {port-number}: the corresponding port to connect through, eg: https://10.0.0.1:8443/univmax/restapi/docs.

CHAPTER 6

Getting Started

Installation Guide How to get the source code, and how to build or install the python package.

Quick Start Guide A quick start guide for PyU4V.

Tools Guide The tools guide for PyU4V

API Glossary A glossary of all available functions.

CHAPTER 7

Supported Versions

PyU4V supports Unisphere for VMAX version 8.4 and higher. VMAX 3 and VMAX All Flash arrays are supported. PyU4V officially supports Python 2.7 & 3.4-3.7 (but do remember that Python 2 will soon be retired.)

Please note that PyU4V V3.0 is NOT BACKWARDS COMPATIBLE with existing scripts written for previous versions of PyU4V, and does not support any Unisphere for VMAX version earlier that 8.4. PyU4V version 2.0.2.6 is still available on Pip, and there is a 'stable/2.0' branch available on Github. Version 3 will be the version maintained going forward, and we do suggest you move to this version when possible.

CHAPTER 8

Feedback, Bug Reporting, Feature Requests

We greatly value your feedback! Please file bugs and issues on the Github issues page for this project. This is to help keep track and document everything related to this repo. The code and documentation are released with no warranties or SLAs and are intended to be supported through a community driven process.

Python Module Index

p

Index

E

F

G

H

I

L

M

P

R

S

U

V

W