PyU4V Documentation

Release 10.1.0.1

Dell

Dec 14, 2023

CONTENTS

CHAPTER

ONE

REQUIREMENTS

PyU4V Version	10.1.0.1
Minimum Unisphere Version	10.1
Array Model	VMAX-3, VMAX AFA, PowerMax
Array uCode	HyperMax OS, PowerMax OS
Platforms	Linux, Windows
Python	3.6, 3.7, 3.8, 3.9, 3.10
Requirements	Requests, Six, urllib3, prettytable
Test Requirements	TestTools, Tox

Note: If you want to continue to use Unisphere 8.4.x or 9.0.x with PyU4V you will need to remain on PyU4V 3.1.x. There is no support for PyU4V 9.x with any version of Unisphere older than 9.1.x

Note: PyU4V officially supports Python 3.6, 3.7, 3.8, 3.9 & 3.10 Python 2.x support has been dropped since January 1st 2020.

CHAPTER

TWO

PYU4V VERSION COMPATIBILITY

PyU4V version 10.x is compatible with scripts written for PyU4V versions >= 9.x, there is **zero** support or compatibility for PyU4V 2.x or earlier scripts in later versions of PyU4V. If you have scripts written which specifically target Unisphere REST 8.4, 9.0 or 9.1 endpoints these are still accessible via PyU4V 10.x however you will need to ensure you are passing the version required when performing these calls as PyU4V 9.2 will default to using 9.2 endpoints exclusively. You will also need to pay special attention to any REST JSON payloads in custom scripts as payloads are subject to change between major Unisphere REST releases.

CHAPTER

THREE

INSTALLATION

PyU4V can be installed from source, via `pip`, or run directly from the source directory. To clone PyU4V and install from source use `git` and `pip`:

```
$ git clone https://github.com/dell/PyU4V
$ cd PyU4V/
$ pip install .
```

Installing via `pip` without cloning from source can be achieved by specifying `PyU4V` as the install package for `pip`:

```
$ pip install PyU4V
# Install a specific version
$ pip install PyU4V==10.0.0.7
```

CHAPTER

FOUR

CONFIGURATION

Once PyU4V is installed the next step is to configure it through `PyU4V.conf`. There is a sample version of `PyU4V.conf` provided in PyU4V which has default configuration options for logging but will require environment configuration setting changes for PyU4V to work.

Copy the sample `PyU4V.conf` provided with PyU4V to either your working directory or within a directory named `.PyU4V` in your current users home directory. The `.sample` suffix has to be removed for the configuration file to become valid for loading by PyU4V:

```
$ mkdir ~/.PyU4V
$ cp PyU4V/PyU4V.conf.sample ~/.PyU4V/PyU4V.conf
```

Note: The ~ symbol is used here to represent the users home directory regardless of operating system, however, ~ is not a valid shortcut in Windows command prompt so direct path to users home directory should be used instead `C:\> mkdir C:\Users\{user}\.PyU4V` where `{user}` is the current user.

Note: If `PyU4V.conf` is present in both the current working directory and the current user's home directory, the version of `PyU4V.conf` in the current working directory will take precedence. See the section below on PyU4V settings precedence.

Edit PyU4V configuration settings in `PyU4V.conf` under the `[setup]` heading, these setting will need to reflect your environment configuration:

```
[setup]
username=pyu4v-user
password=secret-pass
server_ip=10.0.0.75
port=8443
array=00012345678
verify=/path-to-file/server_hostname.pem
```

Where…

Key	Description
username	Unisphere REST login username
password	Unisphere REST login password
server_ip	Unisphere server IP address
port	Unisphere server port number
array	12 digit array serial number
verify	True - Load SSL cert from CA certificate bundle /path/to/file - Load SSL cert from file location False - Disable SSL verification

4.1 Alternative PyU4V.conf Load Using U4VConn().file_path

It is also possible to override PyU4V.conf in both the working directory and home directory by specifying univmax_conn.file_path='/path/to/PyU4V.conf' before initialising PyU4V.

```
import PyU4V

PyU4V.univmax_conn.file_path = '~/path/to/PyU4V.conf'
# Instantiate U4VConn() using the PyU4V config file specified in file_path
conn = U4VConn()
```

If you specify a file_path whilst having a copy of PyU4V.conf in both your working directory and home directory in ~/.PyU4V, the instance of PyU4V.conf as specified in file_path will take precedence. See the section below on PyU4V settings precedence.

4.2 Passing Environment Configuration to U4VConn() on Initialisation

Instead of specifying PyU4V configuration options within PyU4V.conf it is possible to pass these values directly to U4VConn() on initialisation. The key/values expected are the same as those specified in PyU4V.conf.

```
>>> import PyU4V
>>> conn = U4VConn(
        username='pyu4v-user', password='secret-pass',
        server_ip='10.0.0.75', port='8443', verify=True,
        array_id='00012345678')
>>> conn.common.get_unisphere_version()
{'version': 'V9.1.0.5'}
```

If you pass configuration into U4VConn() directly in the code, these settings will override any that are defined in PyU4V.conf at any location.

4.3 PyU4V Configuration Loading Precedence

There are a number of ways to initialise PyU4V with your environment settings through `PyU4V.conf` or passing the values directly. These various methods of setting PyU4V environment configuration have a load precedence, these are listed in order with number 1 being the first load precedent:

1. Configuration key/values passed directly to `U4VConn()`
2. `PyU4V.conf` as specified in `univmax_conn.file_path`
3. `PyU4V.conf` in current working directory
4. `PyU4V.conf` in current users home directory
5. If none of the above or missing mandatory options raise `MissingConfigurationException`

4.4 PyU4V Logger Configuration

Logger options in PyU4V have been streamlined since the previous 3.1.x version, all options are now consolidated to save on duplicate options being presented. All logger configuration options in PyU4V.conf can be found under the comment `; log configuration` in the sections `[loggers*]`, `[handlers*]`, and `[formatters*]`. There are a number of configuration options which you can change to suit your needs, the most relevant of those for the installation and configuration process are outlined in the table below.

Section	Config Option	Description
`[logger_PyU4V]`	`level=INFO`	Sets the PyU4V log level, this defaults to INFO but can be changed to any logger LOG level
`[handler_consoleHandler]`	`args=(sys.stdout,)`	Control how log messages are output to console
`[handler_fileHandler]`	`args=('PyU4V.log', 'a', 10485760, 10)`	Control how log messages are written to log files and where the log file is located
`[formatter_simpleFormatter]`	`format=%(asctime)s - %(name)s - %(levelname)s - %(message)s`	Set the format for the log prefix output in PyU4V.log

Note: PyU4V log functionality is run on top of Python's great inbuilt logger. If you require in depth descriptions of the PyU4V logger configuration options, the logger sections, or input arguments for the handlers, please see the official

Python Logger documentation here.

CHAPTER

FIVE

QUICK START GUIDE

5.1 Initialise PyU4V

First, make sure that PyU4V is installed as directed in the *Installation Guide*:

```
$ pip show PyU4V
```

To initialise the connection with Unisphere use `U4VConn`, it will load the configuration provided in `PyU4V.conf` configured during installation:

```
import PyU4V
conn = PyU4V.U4VConn()
```

With a connection to the Unisphere server you can start to run some test queries to validate the successful connection.

```
conn.common.get_unisphere_version()
('V10.1.0.0', '101')
conn.common.get_array_list()
["000197900123", "000197900124", "000197900125", "000197900126"]
```

If you want to query another array, you can set the `array_id` to a value of your choice which overrides what is set in `PyU4V.conf`. Alternatively you can initialise a new PyU4V connection and pass in the `array_id` but load all other configuration settings from `PyU4V.conf`.

```
# Option 1 - Set new array ID in current PyU4V connection
conn.set_array_id('000197900126')
# Option 2 - Create new PyU4V connection using PyU4V.conf settings
new_conn = PyU4V.U4VConn(array_id='000197900126')
```

5.2 PyU4V Unisphere REST Coverage

The functions in PyU4V have been divided into logical categories which reflect the various categories provided by the Unisphere REST API:

- Common - REST methods and assistive utilities
- Migration - all migration related calls
- Performance - all performance and threshold calls
- Real Time - all performance real-time calls

- Provisioning - all provisioning and masking related calls
- Replication - all local and remote replication calls
- Clone - Create and manage clone replicas of storage groups.
- Metro DR - all Metro DR calls
- Snapshot Policy - all Snapshot Policy calls
- System - all system level calls
- Workload Planner - all workload planner calls
- Utils - assistive functions to aid with PyU4V usage
- Enhanced API - Array & Performance, Bulk API calls to simplify and reduce the number of API calls needed to get information for Data Mining.

There are plans to further increase the coverage of Unisphere REST calls in future releases. All changes are reflected in the PyU4V change log (link).

5.3 Perform a Custom REST Call in PyU4V

At its core PyU4V is your typical REST client. It creates a request header, defines any request parameters in JSON, sends the request with an associated method which dictates the action being performed (`GET`, `POST`, etc.), and expects a response payload in most instances (`DELETE` calls tend to return a status instead of a response payload).

There are functions created in `PyU4V.common` which provide the ability to perform `GET`, `POST` and other request calls directly with the Unisphere REST API to any supported endpoint.

- `PyU4V.common.get_resource()` - `GET`
- `PyU4V.common.create_resource()` - `POST`
- `PyU4V.common.modify_resource()` - `PUT`
- `PyU4V.common.delete_resource()` - `DELETE`

If there is any functionality that is provided by the Unisphere REST API that is not yet implemented in PyU4V, it is possible to create a custom function which use the above functions to make use of that functionality. For information on the Unisphere REST API please consult its related documentation.

The Unisphere for PowerMax REST documentation is accessible directly from the help menu in Unipshere or you can Navigate to `https://developer.dell.com/apis/4458/versions/10.1/`

```
"""docs/source/programmers_guide_src/code/custom_call.py."""
import PyU4V

# Initialise PyU4V connection to Unisphere
conn = PyU4V.U4VConn()

# Create a Clone Copy of a Stoage group, The Target Storage Group will be
# created with all required volumes for the clone copy if it does not exist
# already.


def my_custom_api_Call(
        storage_group_id, target_storage_group_id,
        consistent=True, establish_terminate=False,
```

(continues on next page)

(continued from previous page)

```
        array_id=None, force=False, star=False, skip=False):
    """Create Clone.
    :param storage_group_id: The Storage Group ID -- string
    :param consistent: creates the clone crash consistent using ECA
                       technology -- bool
    :param establish_terminate: creates the clone and immediately
                                terminates, very useful if you want to
                                make an independent copy available
                                immediately but don't intend to use for
                                restore purposes -- bool
    :param array_id: The storage array ID -- string
    :param target_storage_group_id: name of storage group to contain
                                    clone devices -- string
    :param force: Attempts to force the operation even though one or more
                  volumes may not be in the normal, expected state(s) for
                  the specified operation -- bool
    :param star: Acknowledge the volumes are in an SRDF/Star
                 configuration -- bool
    :param skip: Skips the source locks action -- bool

    """
    array_id = array_id if array_id else conn.array_id
    payload = {
        "target_storage_group_name": target_storage_group_id,
        "establish_terminate": establish_terminate,
        "consistent": consistent,
        "force": force,
        "star": star,
        "skip": skip
    }
    return conn.common.create_resource(
        target_uri=f"/100/replication/symmetrix"
                   f"/{array_id}/storagegroup/{storage_group_id}"
                   f"/clone/storagegroup", payload=payload)


# Close the session
conn.close_session()
```

To find out more information on the any PyU4V calls refer to the supporting function documentation in the *API Glossary* , there are also programmers guide examples provided with this documentation which demonstrate a range of functions using PyU4V.

CHAPTER

SIX

PYU4V USAGE RECOMMENDATIONS

In this section we aim to highlight various topics that we hope will improve your overall experience when using PyU4V. If you have any recommendations that you would like to make please let us know by opening an issue and we will review it for addition here.

6.1 Volume Naming Conventions

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 its 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.

6.2 Performance Monitoring

When using PyU4V for performance metrics collection there are a number of best practices that you should follow:

- After enabling Unisphere for performance metrics collection allow Unisphere 30 minutes to gather enough data before making any calls.
- The most granular time available with PyU4V performance metrics collection is 5 minutes, so querying for data more frequently than 5 minutes is a wasteful use of resources.
- If you want to ensure that your performance metric collection is querying the most recent performance data available set a recency window value on your calls.
- If the performance timestamp is not recent as of 5-10 minutes ago there is a strong likelihood that your instance of Unisphere has gone into catchup mode and is processing a backlog of performance data. It will resume normal operations once this backlog processing is complete and be indicated by performance timestamps with a window of 5-10 minutes from the current time.
- When querying a single instance of Unisphere for performance metrics across a number of arrays be careful on the load placed on Unisphere and try to determine if it is possible to query that amount of data in the given time frame you have set.
- If querying for data at regular intervals, examine your calls to see if you can create the interval time. If information is not likely to change over the course of 24 hours then querying once a day would be sufficient.
- If querying for real-time performance data it not possible to query for less than one minute of data at a time, this is to try negate the possibility of users querying for real-time data every 5 seconds and overloading Unisphere. With one minute intervals users can query for 12 deltas of data at 5 second intervals for a specified minute range.
- It is not possible to query for more than one hour of real-time performance data at a time. This is a hard restriction enforced by Unisphere. If you need more than one hour of data then consider if diagnostic performance data is more suitable.

Lastly, and most importantly, with great power comes great responsibility, PyU4V provides you with the ability to query every performance metric for every performance category. Instead of gathering everything possible, be resourceful with your calls and only query what is needed. This will provide improvements in PyU4V performance, network load, and Unisphere REST performance. If you are only interested in querying for KPIs, you can specify that only KPI metrics are returned, but better still only query for a subset of metrics that you are interested in.

CHAPTER

SEVEN

CONTRIBUTE TO PYU4V

Please do! Create a fork of the project into your own repository. Make all your necessary changes and create a pull request with a description on what was added or removed and details explaining the changes in lines of code.

The following tests must all run cleanly:

```
$ cd PyU4V
$ tox -e py36
$ tox -e py37
$ tox -e py38
$ tox -e py39
$ tox -e pep8
$ tox -e pylint
```

Note: If you do not have all the versions of Python installed, just run tox on the versions you have.

Once the above tests all run clean and a pull request has been opened, PyU4V core reviewers will review the code to ensure it conforms to all conventions outlined in the section below. If the code looks good we will run CI on it internally to ensure that it does not break any current functionality. If all the tests run cleanly and two approvals from core reviewers is achieved we will merge it to PyU4V in the target branch.

7.1 Conventions

For neatness and readability we will enforce the following conventions going forward on all code in PyU4V.

1. Single quotes ' unless double quotes " necessary.
2. Use `.format()` for string concatenation.

```
my_string = '{variable1}, thanks for contributing to {variable2}'.format(
    variable1='Hello', variable2='PyU4V')
```

3. Use the following format for doc strings, the return description uses `:returns:` instead of the docstring default `:return:`. Pep8 will guide you with all the docstring conventions.

```
def my_test_func(input_1, input_2):
    """The is my summary of the method with full stop.

    This is a brief description of what the method does.  Keep
    it as simple as possible.
```

(continues on next page)

(continued from previous page)

```
    :param input_1: brief description of input parameter 1, if it goes over
                    one line it must be indented with the start of the
                    previous lines -- str
    :param input_2: brief description of input parameter 2, you must also
                    provide the input parameter type after the description
                    after double dash -- int
    :returns: what gets returned from method, omit if none, type must also
              be specified, in this case it is a boolean -- bool
    :raises: Exceptions raised, omit if none
    """
    return True if input_1 or input_2 else raise Exception
```

5. Class names are mixed case with no underscores _.

```
class ClassFunctions(object):
    """Collection of functions ClassFunctions."""
```

6. Public Methods are separated by underscores _. Make the name as meaningful as possible.

```
def public_function_does_exactly_what_it_says_it_does(self):
    """Function does exactly what it says on the tin."""
```

7. Private Methods are prefixed and separated by underscores _. Make the name as meaningful as possible.

```
def _private_function_does_exactly_what_it_says_it_does(self):
    """Function does exactly what it says on the tin."""
```

8. If functions seems to big or too complicated then consider breaking them into smaller functions.

9. If a line of code must extend over more than one line, use parenthesis () around the code instead of \ at the end of the line.

```
my_multi_line_string = ('This is an example of a string '
                        'that extends over more than one line.')

my_multi_line_function = (
    this_is_a_very_long_function_call_that_cannot_meet_79_char_limit())
```

10. Each new function must be unit tested.

11. Each bug fix must be unit tested.

12. Unix and OS X format only. If in doubt run

```
$ sudo apt-get install dos2unix
$ dos2unix myfile.txt
```

or in PyCharm:

```
File -> Line Separators -> LF- Unix and OS X (\n)
```

CHAPTER

EIGHT

ISSUES & SUPPORT

We greatly value your feedback! Please file bugs and issues on the GitHub issues page for this project.

We aim to track and document everything related to this repo via the issues page. The code and documentation are released with no warranties or SLAs and are intended to be supported through a community driven process.

When opening an issue please include the following information to help us debug:

- PyU4V version
- Unisphere version
- Description of the problem
- Impacted functions
- PyU4V logs showing issue

Note: We will support N-2 releases from the current master release which includes bug and security fixes. If an issue appears in a code base older than N-2 we will try to assist as best possible but ultimately upgrading to a newer version of PyU4V will be the ideal outcome. As new releases of PyU4V are made available, anything older than N-2 will be marked as End of Life (EOL).

CHAPTER

NINE

PROGRAMMERS GUIDE

In this section a number of examples demonstrating various PyU4V functionality are provided. If you have an example which you believe would make a good addition here let us know be opening a support issue and we will review it for addition!

Although the scope of these programmers examples is limited, it is worth pointing out that if you want to see a working example of any function you can do so by looking at that function's associated continuous integration test.

9.1 Basic Operations

9.1.1 First connection to PyU4V

In this basic example we initialise a connection with Unisphere, retrieving version and array information, and finishing by closing the REST session.

```
"""docs/source/programmers_guide_src/code/basic-unisphere_connect.py."""

import PyU4V

# Initialise PyU4V connection to Unisphere
conn = PyU4V.U4VConn(
    server_ip='10.0.0.75', port=8443, verify='~/.PyU4V/Unisphere91.pem',
    username='pyu4v-user', password='secret-pass')

# Get the Unisphere version
version = conn.common.get_uni_version()

# Retrieve a list of arrays managed by your instance of Unisphere
array_list = conn.common.get_array_list()

# Output results to screen
print('Congratulations on your first connection to Unisphere, your '
      'version is: {ver}'.format(ver=version[0]))
print('This instance of Unisphere instance manages the following arrays: '
      '{arr_list}'.format(arr_list=array_list))

# GET those arrays which are local to this instance of Unisphere
local_array_list = list()
for array_id in array_list:
    array_details = conn.common.get_array(array_id)
```

(continues on next page)

(continued from previous page)

```
    if array_details['local']:
        local_array_list.append(array_id)

# Output results to screen
print('The following arrays are local to this Unisphere instance: '
      '{arr_list}'.format(arr_list=local_array_list))

# Close the session
conn.close_session()
```

9.2 Provisioning

9.2.1 A-Synchronous Provisioning & Creating Storage for a Host

This example demonstrates checking an array SRP and service level to determine if there is enough headroom to provision storage of a set size, if so, proceed to creating a storage group with volume. Create a host, port group, and masking view to tie all the elements together, close the session when done.

```
"""docs/source/programmers_guide_src/code/provision-async_create_storage.py"""

import PyU4V

# Initialise PyU4V connection to Unisphere
conn = PyU4V.U4VConn()

# Before provisioning storage we are going to check that there is enough
# headroom left on the array for our provisioning operations
REQUESTED_CAPACITY = 10

# Get the available headroom for the SRP and service level required
headroom_info = conn.wlp.get_headroom(array_id=conn.array_id,
                                      srp='SRP_1', slo='Diamond')

# Extract the capacity value from the headroom_info REST response
headroom_capacity = headroom_info[
    'OLTP'][0]['headroom'][0]['headroomCapacity']

# If the requested capacity of 10GB is less than or equal to the available
# capacity proceed with the provisioning operations
if REQUESTED_CAPACITY <= int(headroom_capacity):
    # Create a non-empty storage group using asynchronous request - we can
    # wait until the job completes or proceed with operations and check
    # back at a later time
    storage_group_async_job = (
        conn.provisioning.create_non_empty_storage_group(
            srp_id='SRP_1', storage_group_id='example-sg',
            service_level='Diamond', num_vols=1,
            vol_size=REQUESTED_CAPACITY, cap_unit='GB', _async=True))

    # We will wait this time on the results of the storage group create
```

(continues on next page)

(continued from previous page)

```
    # request
    conn.common.wait_for_job(operation='Create SG with volume',
                             job=storage_group_async_job)
    print('Storage Group created successfully...')

    # Get information on our new storage group
    storage_group_info = conn.provisioning.get_storage_group(
        storage_group_name='Example-SG')
    print('Storage Group details: {details}'.format(
        details=storage_group_info))

    # Create a Host using supplied initiator IDs, these can be also be
    # retrieved via the call conn.provisioning.get_available_initiator()
    initiator_list = ['iqn:2020-test1', 'iqn:2020-test1']
    host_info = conn.provisioning.create_host(
        host_name='Example-Host', initiator_list=initiator_list)
    print('Host created successfully...')
    print('New Host details: {details}'.format(details=host_info))

    # Create a Port Group using supplied ports, these could be also be
    # retrieved via the call conn.provisioning.get_port_list()
    port_group_info = conn.provisioning.create_port_group(
        port_group_id='Example-PG', director_id='SE-01', port_id='1')
    print('Port Group created successfully...')
    print('Port Group details: {details}'.format(details=port_group_info))

    # Create a Masking View and tie all the elements we have created
    # together
    masking_view_info = (
        conn.provisioning.create_masking_view_existing_components(
            port_group_name='Example-PG', masking_view_name='Example-MV',
            storage_group_name='Example-SG', host_name='Example-Host'))
    print('Masking View created...')
    print('Masking View details: {details}'.format(
        details=masking_view_info))

# Close the session
conn.close_session()
```

9.3 Replication

This section gives examples of the following replication functionality:

- *Local Replication with SnapVX*
- *Snapshot Policies*
- *Remote Replication with SRDF*
- *Remote Replication with SRDF Metro Smart DR*

9.3.1 Local Replication with SnapVX

In this example a new storage group is created with a single 1GB volume. A snapshot name is generated using the current time so it can be easily identified, and the storage group snapshot is created. The operation is verified by querying for a list of snapshots for a given storage group and confirming the snapshot we created is present in that list.

```
"""docs/source/programmers_guide_src/code/replication-snapshot_create.py"""

import PyU4V
import time

# Initialise PyU4V connection to Unisphere
conn = PyU4V.U4VConn()

# Create storage Group with one volume using settings specified for
# service level and capacity
storage_group = conn.provisioning.create_non_empty_storage_group(
    srp_id='SRP_1', storage_group_id='PyU4V_SG', service_level='Diamond',
    workload=None, num_vols=1, vol_size=1, cap_unit='GB')

# Define a Name for the Snapshot, in this case the name auto appends
# the host
# time for when it was taken for ease of identification
snap_name = 'PyU4V_Snap_' + time.strftime('%d%m%Y%H%M%S')

# Create the snapshot of the storage group containing the volume and
# storage group created in the previous step
snapshot = conn.replication.create_storage_group_snapshot(
    storage_group_id=storage_group['storageGroupId'], snap_name=snap_name)

# Confirm the snapshot was created successfully, get a list of storage
# group snapshots
snap_list = conn.replication.get_storage_group_snapshot_list(
    storage_group_id=storage_group['storageGroupId'])

# Assert the snapshot name is in the list of storage group snapshots
assert snapshot['name'] in snap_list

# Close the session
conn.close_session()
```

This example will create a storage group with a volume, create a snapshot of that storage group and link the snapshot to a new storage group. This is a typical workflow for provisioning a dev environment and making a copy available.

```
"""docs/source/programmers_guide_src/code/replication-snapshot_link.py"""

import PyU4V
import time

# Set up connection to Unisphere for PowerMax Server, details collected
# from configuration file in working directory where script is stored.
conn = PyU4V.U4VConn()

# Create storage Group with one volume
```

(continues on next page)

(continued from previous page)

```
storage_group = conn.provisioning.create_non_empty_storage_group(
    srp_id='SRP_1', storage_group_id='PyU4V_SG', service_level='Diamond',
    workload=None, num_vols=1, vol_size=1, cap_unit='GB')

# Define a Name for the Snapshot, in this case the name auto appends the
# host time for when it was taken for ease of identification
snap_name = 'PyU4V_Snap_' + time.strftime('%d%m%Y%H%M%S')

# Create the snapshot of the storage group containing the volume and
# storage group created in the previous step
snapshot = conn.replication.create_storage_group_snapshot(
    storage_group_id=storage_group['storageGroupId'], snap_name=snap_name)
snapshot_details = conn.replication.get_storage_group_snapshot_snap_id_list(
    storage_group['storageGroupId'], snap_name)
snap_id = snapshot_details[0]

# Link The Snapshot to a new storage group, the API will automatically
# create the link storage group with the right number of volumes if one
# with that name doesn't already exist
conn.replication.modify_storage_group_snapshot_by_snap_id(
    src_storage_grp_id=storage_group['storageGroupId'],
    tgt_storage_grp_id='PyU4V_LNK_SG',
    snap_name=snap_name, snap_id=snap_id, link=True,)
```

9.3.2 Snapshot Policies

The Snapshot policy feature provides snapshot orchestration at scale (1,024 snaps per storage group). The feature simplifies snapshot management for standard and cloud snapshots.

Snapshots can to be used to recover from data corruption, accidental deletion or other damage, offering continuous data protection. A large number of snapshots can be difficult to manage. The Snapshot policy feature provides an end to end solution to create, schedule and manage standard (local) and cloud snapshots.

For full detailed information on snapshot policies in Unisphere for PowerMax please consult the official Unisphere for PowerMax online help guide.

In the example below a new snapshot policy is created, modified, then deleted.

```
"""docs/source/programmers_guide_src/code/replication-snapshot_policy.py"""

import PyU4V

# Set up connection to Unisphere for PowerMax Server, details collected
# from configuration file in working directory where script is stored.
conn = PyU4V.U4VConn()

# Create storage Group with one volume
storage_group = conn.provisioning.create_non_empty_storage_group(
    srp_id='SRP_1', storage_group_id='PyU4V_SG', service_level='Diamond',
    workload=None, num_vols=1, vol_size=1, cap_unit='GB')

# Create a snapshot policy for the new storage group
```

(continues on next page)

(continued from previous page)

```
policy_name = 'PyU4V-Test_Policy'
snap_policy = conn.snapshot_policy.create_snapshot_policy(
    snapshot_policy_name=policy_name, interval='1 Day',
    cloud_retention_days=7, cloud_provider_name='Generic_Provider',
    local_snapshot_policy_snapshot_count=5)

# Confirm the snapshot policy was created successfully
assert policy_name in conn.snapshot_policy.get_snapshot_policy_list()

# Get snapshot policy detailed info
snap_policy_details = conn.snapshot_policy.get_snapshot_policy(
    snapshot_policy_name=policy_name)

# Modify the snapshot policy
new_policy_name = 'PyU4V-Test_Policy-5mins'
new_snap_policy = conn.snapshot_policy.modify_snapshot_policy(
    snapshot_policy_name=policy_name, action='Modify',
    new_snapshot_policy_name=new_policy_name, interval_mins=5)

# Confirm the snapshot policy was renamed successfully
assert new_policy_name in conn.snapshot_policy.get_snapshot_policy_list()
assert policy_name not in conn.snapshot_policy.get_snapshot_policy_list()

# Delete the snapshot policy
conn.snapshot_policy.delete_snapshot_policy(
    snapshot_policy_name=new_policy_name)

# Close the session
conn.close_session()
```

9.3.3 Snapshot Policy Compliance

This allows the user to query snapshot policy compliance over a period of time. Last week, last four weeks, epoch to/from, human readable to/from are supported.

For full detailed information on snapshot policies in Unisphere for PowerMax please consult the official Unisphere for PowerMax online help guide.

In the example below, snapshot policy compliance over the last week is queried.

```
"""docs/source/programmers_guide_src/code/
replication-snapshot-policy-compliance.py"""

import PyU4V

# Set up connection to Unisphere for PowerMax Server, details collected
# from configuration file in working directory where script is stored.
conn = PyU4V.U4VConn()

# Create a snapshot policy
snapshot_policy_name = 'PyU4V_Compliance_Policy'
conn.snapshot_policy.create_snapshot_policy(
```

(continues on next page)

(continued from previous page)

```
    snapshot_policy_name=snapshot_policy_name, interval='1 Day',
    local_snapshot_policy_snapshot_count=5)

# Get the snapshot policy
snapshot_policy_details = (
    conn.snapshot_policy.get_snapshot_policy(snapshot_policy_name))

# Check that snapshot policy exists
assert snapshot_policy_details and snapshot_policy_details.get(
    'snapshot_policy_name')

# Create storage Group with one volume and associate with snapshot
# policy.
storage_group_name = 'PyU4V_compliance_SG'
storage_group = conn.provisioning.create_non_empty_storage_group(
    srp_id='SRP_1', storage_group_id=storage_group_name,
    service_level='Diamond', workload=None,
    num_vols=1, vol_size=1, cap_unit='GB',
    snapshot_policy_ids=[snapshot_policy_name])

# Get the storage group
storage_group_details = conn.provisioning.get_storage_group(
    storage_group_name)

# Check that storage group exists
assert storage_group_details and storage_group_details.get('storageGroupId')

# Get the compliance details
compliance_details = (
    conn.snapshot_policy.get_snapshot_policy_compliance_last_week(
        storage_group_name))

# Check details have been return
assert compliance_details

# Disassociate from snapshot policy
conn.snapshot_policy.modify_snapshot_policy(
    snapshot_policy_name, 'DisassociateFromStorageGroups',
    storage_group_names=[storage_group_name])

# Delete the snapshot policy
conn.snapshot_policy.delete_snapshot_policy(snapshot_policy_name)

# Get volumes from the storage group
volume_list = (conn.provisioning.get_volumes_from_storage_group(
    storage_group_name))

# Delete the storage group
conn.provisioning.delete_storage_group(storage_group_id=storage_group_name)

# Delete each volume from storage group
for volume in volume_list:
```

(continues on next page)

(continued from previous page)

```
    conn.provisioning.delete_volume(volume)

# Close the session
conn.close_session()
```

9.3.4 Remote Replication with SRDF

This example will create a storage group on the PowerMax array with some volumes. Once the storage group has been created it will protect the volumes in the storage group to a remote array using SRDF/Metro, providing Active/Active business continuity via Symmetrix Remote Data Facility (SRDF).

```
"""docs/source/programmers_guide_src/code/replication-srdf_protection.py"""

import PyU4V

# Initialise PyU4V connection to Unisphere
conn = PyU4V.U4VConn()

# Create storage Group with one volume using settings specified for
# service level and capacity
storage_group = conn.provisioning.create_non_empty_storage_group(
    srp_id='SRP_1', storage_group_id='PyU4V_SG', service_level='Diamond',
    workload=None, num_vols=1, vol_size=1, cap_unit='GB')

# Make a call to setup the remote replication, this will automatically
# create a storage group with the same name on the remote array with the
# correct volume count and size, the example here is executed
# asynchronously and a wait is added to poll the async job id until
# complete
srdf_job_id = conn.replication.create_storage_group_srdf_pairings(
    storage_group_id=storage_group['storageGroupId'],
    remote_sid=conn.remote_array, srdf_mode="Active", _async=True)

# Wait until the previous create SRDF pairing job has completed before
# proceeding
conn.common.wait_for_job_complete(job=srdf_job_id)

# The now protected storage group will have an RDFG associated with it,
# using the function conn.replication.get_storage_group_rdfg() function we
# can retrieve a list of RDFGs associated with the storage group, in this
# case there will only be one
rdfg_list = conn.replication.get_storage_group_srdf_group_list(
    storage_group_id=storage_group['storageGroupId'])

# Extract the (only) RDFG number from the retrieved list
rdfg_number = rdfg_list[0]

# Finally the details of the protected storage group can be output to the
# user.
storage_group_srdf_info = conn.replication.get_storage_group_srdf_details(
    storage_group_id=storage_group['storageGroupId'],
```

(continues on next page)

(continued from previous page)

```
    rdfg_num=rdfg_number)

# Close the session
conn.close_session()
```

9.3.5 Remote Replication with SRDF Metro Smart DR

SRDF/Metro Smart DR is a two region High-Availability (HA) Disaster Recovery (DR) solution. It integrates SRDF/Metro and SRDF/A, enabling HA DR for an SRDF/Metro session. By closely coupling the SRDF/A sessions on both sides of an SRDF/Metro pair, SRDF/Metro Smart DR can replicate to a single DR device.

SRDF/Metro Smart DR environments are identified by a unique name and contain three arrays (MetroR1, MetroR2, and DR). For SRDF/Metro Smart DR, arrays must be running PowerMaxOS 5978.669.669 or higher.

This example will create a storage group on the PowerMax array with some volumes. Once the storage group has been created it will protect the volumes in the storage group to a remote array using SRDF/Metro DR.

Note once Metro DR is setup, the environment is controlled exclusively by the environment name. SRDF Metro and SRDF/A replication can not be controlled by standard Replication Calls without first deleting the Metro DR environment.

For more information on SRDF/Metro Smart DR, please refer to Dell EMC Solutions Enabler SRDF Family CLI user guide available on https://support.dell.com

```
"""docs/source/programmers_guide_src/code/replication-metro_dr.py"""

import PyU4V

# Initialise PyU4V connection to Unisphere
conn = PyU4V.U4VConn()

metro_r1_array_id = '000297600111'
metro_r2_array_id = '000297600112'
dr_array_id = '000297600113'

sg_name = 'PyU4V_Test_MetroDR'
environment_name = 'PyU4VMetro'

# Create a storage Group with some volumes
sg_details = conn.provisioning.create_non_empty_storage_group(
    storage_group_id=sg_name, service_level='Diamond',
    num_vols=5, vol_size=6, cap_unit='GB', srp_id='SRP_1',
    workload=None)

"""
The next call section creates the metro dr environment, this includes all
Necessary SRDF setup creating the SRDF metro pairings and remote devices
and storage group at the Metro R2 array.

The API also creates all the necessary SRDF groups between R11, R21 and R2
for the DR leg, this includes a recovery SRDF group.  The example is
using async execution of the REST calls,  this task can take many
minutes to complete and depending on the number of devices.
```

(continues on next page)

(continued from previous page)

```
"""
job = conn.metro_dr.create_metrodr_environment(
    storage_group_name=sg_name, environment_name=environment_name,
    metro_r1_array_id=metro_r1_array_id, metro_r2_array_id=metro_r2_array_id,
    dr_array_id=dr_array_id, dr_replication_mode='adaptivecopydisk')

conn.common.wait_for_job_complete(job=job)

metro_dr_env_details = conn.metro_dr.get_metrodr_environment_details(
    environment_name, array_id=metro_r1_array_id)

# Close the session
conn.close_session()
```

9.4 Clone

This section gives examples of the following system functionality:

- *Create and Manage Clones*

9.4.1 Create and Manage Clones

This example of replication calls demonstrates how to create independent copies of your data using clone technology, for more detail on working with TimeFinder Clone please see Dell Solutions Enabler 10.0.0 TimeFinder/ Clone CLI User Guide available on https://support.dell.com.

```
"""docs/source/programmers_guide_src/code/clone.py."""

import PyU4V

# Initialise PyU4V connection to Unisphere
conn = PyU4V.U4VConn()

# Create a Clone Copy of a Storage group, The Target Storage Group will be
# created with all required volumes for the clone copy if it does not exist
# already.
# Using the Establish & Terminate Boolean value in the example removes the
# clone session leaving the source and target devices indepenent of
# each other, the default value of false will allow you to create
# differential copies, updating the data on target storage group volumes.

conn.clone.create_clone(
    storage_group_id="clonesrc", target_storage_group_id="clonetgt",
    establish_terminate=True)

# Close the session
conn.close_session()
```

```
conn.clone.split_clone(
    storage_group_id="clonesrc", target_storage_group_id="clonetgt")

# Close the session
conn.close_session()
```

9.5 Performance

This section gives examples of the following performance functionality:

- *Registering an Array For Performance Data Collection*
- *Diagnostic Metrics Gathering*
- *Real-Time Metrics Gathering*
- *Backup Performance Database*
- *Thresholds*

9.5.1 Registering an Array For Performance Data Collection

Arrays can be enabled for both diagnostic and real-time performance data collection using PyU4V. The example below demonstrates enabling both varieties and confirming the registration has been successful.

```
"""docs/source/programmers_guide_src/code/performance-registration.py"""

import PyU4V

# Initialise PyU4V Unisphere connection
conn = PyU4V.U4VConn()

# Check if the primary array is registered for diagnostic performance data
# collection
if not conn.performance.is_array_diagnostic_performance_registered():
    # If not, enable diagnostic data collection
    conn.performance.enable_diagnostic_data_collection()
    # Confirm the enable operation was successful
    assert conn.performance.is_array_diagnostic_performance_registered()

# Check if the primary array is registered for real-time performance data
# collection
if not conn.performance.is_array_real_time_performance_registered():
    # If not, enable real-time data collection
    conn.performance.enable_real_time_data_collection()
    # Confirm the enable operation was successful
    assert conn.performance.is_array_real_time_performance_registered()

# Close the session
conn.close_session()
```

9.5.2 Diagnostic Metrics Gathering

This example demonstrates a range of diagnostics (5 minute interval) performance functionality such as getting performance categories and metrics, timestamps from Unisphere for an array, get recent only performance information, and getting ResponseTime for all SRPs in an array.

```
"""docs/source/programmers_guide_src/code/performance-diagnostic_calls.py"""

import PyU4V

# Initialise PyU4V Unisphere connection
conn = PyU4V.U4VConn()

# Get a list of performance categories
category_list = conn.performance.get_performance_categories_list()

# Get a list of supported metrics for the category 'FEDirector'
fe_dir_metrics = conn.performance.get_performance_metrics_list(
    category='FEDirector')

# Get a list of KPI only metrics for the category 'StorageGroup'
storage_group_metrics = conn.performance.get_performance_metrics_list(
    category='StorageGroup', kpi_only=True)

# Get array KPI performance metrics for the most recent timestamp only, set
# recency so timestamp has to be less than 5 minutes old
array_performance_data = conn.performance.get_array_stats(metrics='KPI',
                                                          recency=5)

# Get ResponseTime for each SRP for the last 4 hours
# Firstly get the most recent performance timestamp for your array
recent_timestamp = conn.performance.get_last_available_timestamp()
# Set the performance recency value to 10 minutes and check if the most
# recent timestamp meets that recency value
conn.performance.recency = 10
is_recent_ten = conn.performance.is_timestamp_current(recent_timestamp)
# Recency can also be passed to is_timestamp_current
is_recent_five = conn.performance.is_timestamp_current(recent_timestamp,
                                                       minutes=5)

# Get the start and end times by providing the most recent timestamp and
# specifying a 4 hour difference
start_time, end_time = conn.performance.get_timestamp_by_hour(
    end_time=recent_timestamp, hours_difference=4)
# Get the list of SRPs
srp_keys = conn.performance.get_storage_resource_pool_keys()
srp_list = list()
for key in srp_keys:
    srp_list.append(key.get('srpId'))
# Get the performance data for each of the SRPs in the list
for srp in srp_list:
    srp_data = conn.performance.get_storage_resource_pool_stats(
        srp_id=srp, metrics='ResponseTime', start_time=start_time,
```

(continues on next page)

(continued from previous page)

```
        end_time=end_time)

# Close the session
conn.close_session()
```

9.5.3 Real-Time Metrics Gathering

With the PyU4V 9.2 release it is possible to retrieve real-time performance data from Unisphere for arrays which have been registered for real-time data. This example demonstrates getting supported real-time peformance metrics for a given category, getting the keys for that category (the assets which we can obtain real-time data for), and retrieving real-time data for the last hour.

Note: The maximum amount of real-time data that can be retrieved from Unisphere via PyU4V in one call is one hour. If more than one hour of data is required then more calls can be run, however if you find yourself doing this often it is a good idea to save data locally for easy retrieval and reduce the amount of REST calls required.

Note: Storage Groups are supported for real-time metrics but these **must** be enabled in Unisphere via `Settings > Performance > System Registrations`. Under the table heading `Storage Group Real Time` and for the array you want to enable real-time for, select the option `Real Time Storage Groups` and add up to 100 Storage Groups for real-time data collection.

```
"""docs/source/programmers_guide_src/code/performance-real_time_calls.py"""

import random
import time

import PyU4V

# Initialise PyU4V Unisphere connection
conn = PyU4V.U4VConn()

# Get a list of real-time performance categories
category_list = conn.performance.real_time.get_categories()

# Get the first and last available timestamps for my primary array
array_timestamps = conn.performance.real_time.get_timestamps(
    array_id=conn.array_id)
# Get the first and last available timestamps for all registered arrays
all_timestamps = conn.performance.real_time.get_timestamps()

# Get storage groups enabled for real-time data on primary array
rt_sg_list = conn.performance.real_time.get_storage_group_keys()
# Get the real-time metrics for storage groups
rt_sg_metrics = conn.performance.real_time.get_storage_group_metrics()

# Get the current time in milliseconds since epoch
current_time = int(time.time()) * 1000
# Set the start time as one hour previous to current time, one hour is the
```

(continues on next page)

(continued from previous page)

```
# maximum amount of data that can be retrieved in one call for real-time
start_time = current_time - (60 * 60 * 1000)

# Get real-time storage group data for a random group and random selection of
# metrics
sg_metrics = random.choices(rt_sg_metrics, k=5)
sg_name = random.choice(rt_sg_list)

# Retrieve the performance data from Unisphere
rt_sg_data = conn.performance.real_time.get_storage_group_stats(
    start_date=start_time, end_date=current_time, metrics=sg_metrics,
    instance_id=sg_name)

# Close the session
conn.close_session()
```

9.5.4 Backup Performance Database

Backup of a performance database is a recommended practice. The backup performance database option is available for one or more storage systems, regardless of their registration status.

By default, only Trending & Planning (Historical) data is backed up. The performance databases backups should be stored in a safe location. Performance database backups can be restored. For more information on restoring backups please see Unisphere for PowerMax official documentation, for now only performing backups is supported via REST.

To create a backup of a performance data simply call `performance.backup_performance_database()`, both the `array_id` and `filename` are optional values, if no `filename` is provided then the client hostname will be used.

Note: Underscores will be stripped from any filename provided, this is due to Unisphere restricting the length of the filename string when underscores are provided. The backup filename format will be as follows when viewed in Unisphere: `{array_id}_{date}{time}_{TZ}_{filename}_SPABackup.dat`

```
"""docs/source/programmers_guide_src/code/performance-db_backup.py"""

import PyU4V

# Initialise PyU4V connection to Unisphere
conn = PyU4V.U4VConn()

# Backup the performance database
conn.performance.backup_performance_database(
    array_id=conn.array_id, filename='PyU4V_Example')

# Close the session
conn.close_session()
```

To view this performance database backup in Unisphere navigate to `Settings` > `Unisphere Databases` > `Performance Databases`. Use the performance databases list view to view and manage databases.

9.5.5 Thresholds

In this example both performance threshold calls and CSV file handling with PyU4V are demonstrated. A call is made to retrieve a full list of performance threshold settings and output the results to a CSV file at a path specified by the user. That CSV file is read into a Python dictionary and the respective values within are updated. Once complete the updated threshold settings are uploaded to Unisphere to take immediate effect.

```
"""docs/source/programmers_guide_src/code/performance-thresholds.py."""

import os
import PyU4V

# Initialise PyU4V connection to Unisphere
conn = PyU4V.U4VConn()

# Set the CSV file name and path
current_directory = os.getcwd()
output_csv_name = 'thresholds-test.csv'
output_csv_path = os.path.join(current_directory, output_csv_name)

# Generate a CSV file with all of the thresholds and corresponding values
conn.performance.generate_threshold_settings_csv(
    output_csv_path=output_csv_path)

# Read the CSV values into a dictionary, cast all string booleans and
# numbers to their proper types
threshold_dict = PyU4V.utils.file_handler.read_csv_values(output_csv_path,
                                                          convert=True)

# Increase all of the first threshold values by 5 and second threshold
# values by 10, alert only on the KPIs
for i in range(0, len(threshold_dict.get('metric'))):
    threshold_dict['firstThreshold'][i] += 5
    threshold_dict['secondThreshold'][i] += 5
    if threshold_dict['kpi'][i] is True:
        threshold_dict['alertError'][i] = True

# Process the CSV file and update the thresholds with their corresponding
# values, we are only going to set the threshold value if it is a KPI
conn.performance.set_thresholds_from_csv(csv_file_path=output_csv_path,
                                         kpi_only=True)

# It is also possible to set a threshold value without editing the values
# in a CSV, the threshold metric and be edited directly
threshold_settings = conn.performance.update_threshold_settings(
    category='Array', metric='PercentCacheWP', first_threshold=60,
    first_threshold_occurrences=2, first_threshold_samples=6,
    first_threshold_severity='INFORMATION', second_threshold=90,
    second_threshold_occurrences=1, second_threshold_samples=3,
    second_threshold_severity='CRITICAL', alert=True)

# Close the session
conn.close_session()
```

9.6 System

This section gives examples of the following system functionality:

- *Health Checks*
- *Audit Logs*
- *Download & Upload Settings*

9.6.1 Health Checks

This example of system calls demonstrates performing a system health check, retrieving information from the last health check, querying for all installed disk IDs in an array and outputting information about each.

```
"""docs/source/programmers_guide_src/code/system-health_check.py."""

import PyU4V

# Initialise PyU4V connection to Unisphere
conn = PyU4V.U4VConn()

# Perform a system health check, this call can take 15-20 minutes to
# complete in Unisphere due to the nature of the checks performed
conn.system.perform_health_check(description='test-hc-dec19')

# Get details of the last system health check
health_check = conn.system.get_system_health()

# Get a list of physical disks installed in the array
disk_list = conn.system.get_disk_id_list()
# Get disk information for each disk installed
for disk in disk_list.get('disk_ids'):
    disk_info = conn.system.get_disk_details(disk_id=disk)

# Close the session
conn.close_session()
```

9.6.2 Audit Logs

The storage system audit records include all actions that are taken on that storage system. The storage system audit records come from the SYMAPI database and include all actions that are taken on that storage system. The audit log resides on the storage system and currently has a maximum size of 40 MB. Once the 40 MB limit is reached, the log begins to overwrite itself. Beginning in Unisphere release 9.2, operations that are executed through Unisphere UI have been added to the audit log.

> **Warning:** Audit logs can be very large. You are warned that the operation may take a long time if you attempt to retrieved audit log records for a time period of greater than 7 days.

In the example below a query is made to return audit logs for the last hour which match a provided user name and client host name. A sample audit log record is then queried to return more detailed information.

```
"""docs/source/programmers_guide_src/code/system-audit_log_query.py."""

import random
import time

import PyU4V

# Initialise PyU4V connection to Unisphere
conn = PyU4V.U4VConn()

# Get a list of audit logs for the last ten minutes where the user name and
# host name are defined as filters to return only matching results. Note, the
# time interval here, similar to performance calls, is also in milliseconds
# since epoch
current_time = int(time.time()) * 1000
start_time = current_time - (60 * 10 * 1000)
audit_log_list = conn.system.get_audit_log_list(
    start_time=start_time, end_time=current_time, user_name='PyU4V',
    host_name='PyU4V-Host')

# Select an audit log record from the list of audit logs and get the
# associated audit log record ID
audit_log_record = random.choice(audit_log_list)
audit_log_record_id = audit_log_record.get('record_id')

# Retrieve detailed information on a record returned in the audit log list
audit_log_detailed = conn.system.get_audit_log_record(
    record_id=audit_log_record_id)

# Close the session
conn.close_session()
```

In addition to querying audit logs via REST it is possible to download a PDF audit log record for the previous week. The example below demonstrates this.

Note: When providing a `file_name` there is no need to add the `.pdf` extension, PyU4V will do this automatically.

Note: When downloading an audit log record, it is possible to return the binary data instead of writing to pdf automatically, this allows you to handle the file writing process yourself. To do so just set `return_binary=True`. The data will be stored in the response dict under the key `binary_data`.

```
"""docs/source/programmers_guide_src/code/system-audit_log_download.py."""

import PyU4V

# Initialise PyU4V connection to Unisphere
conn = PyU4V.U4VConn()

# Download PDF audit log record for the last week for primary array
audit_log_details = conn.system.download_audit_log_record(
```

(continues on next page)

(continued from previous page)

```
    return_binary=False, dir_path='~/datastore/audit_records',
    file_name='audit-log-sept2020')

# Assert the download operation was successful
assert audit_log_details.get('success') is True
print('The audit log has been downloaded to: {loc}'.format(
    loc=audit_log_details.get('audit_record_path')))

# Close the session
conn.close_session()
```

9.6.3 Download & Upload Settings

Unisphere allows an authorized user with appropriate access to manage system settings. Settings can also be cloned, that is, settings can be copied from one array and applied to one or more target arrays subject to compatibility checks. In addition to cloning system settings from one array to another, it is also possible to clone Unisphere settings from one instance to another.

The exported settings have a generic format and do not contain any specific information regarding particular storage array or Unisphere instance, thus making it applicable in any environment.

The intention is to help users to port the system wide settings to another instance of Unisphere, and also to capture single array settings so that they can be applied to another storage array within single instance or another instance of Unisphere at any point of time.

The example below downloading system settings for the primary array and applying those same settings to another array registered to the same instance of Unisphere.

Note: A file_password is required not to protect the file when it is downloaded to a local computer, it is to verify the settings when they are being applied to another system or instance of Unisphere. This prevents unauthorised settings changes by people who have access to the data file.

```
"""docs/source/programmers_guide_src/code/system-settings_management.py."""

import PyU4V

# Initialise PyU4V connection to Unisphere
conn = PyU4V.U4VConn()

# Download system settings for primary array defined in PyU4V.conf, exclude the
# system thresholds settings
settings_info = conn.system.download_system_settings(
    file_password='PyU4V', dir_path='~/datastore/system_settings',
    file_name='settings_gold_master', exclude_system_threshold_settings=True)

# Assert the download operation was successful and print the location of the
# downloaded file
assert settings_info.get('success') is True
print('The system settings for {array} are saved to: {loc}'.format(
    array=conn.array_id, loc=settings_info.get('settings_path')))

```

(continues on next page)

(continued from previous page)

```
# Upload the settings and apply to target array
upload_details = conn.system.upload_settings(
    file_password='PyU4V', file_path=settings_info.get('settings_path'),
    array_id='000111222333')

# Close the session
conn.close_session()
```

CHAPTER

TEN

TOOLS GUIDE

First, make sure that PyU4V is installed as directed in the *Installation Guide*. Then visit the *Quick Start Guide* to make sure you have secure connectivity to your array.

10.1 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 issues.

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 has reached EOL however is referenced here due to the existence old environments which may still be running Python 2.7.

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 issues.

CHAPTER

ELEVEN

API GLOSSARY

11.1 PyU4V.univmax_conn

Creates the connection with the Unisphere for PowerMax instance. univmax_conn.py.

class PyU4V.univmax_conn.**U4VConn**(*username=None*, *password=None*, *server_ip=None*, *port=None*, *verify=None*, *u4v_version='101'*, *interval=5*, *retries=200*, *array_id=None*, *application_type='PyU4V-10.1.0.1'*, *remote_array=None*, *remote_array_2=None*, *proxies=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 – str

set_requests_timeout(*timeout_value*)

Set the requests timeout.

Parameters
timeout_value – the new timeout value – int

validate_unisphere()

Check that the minimum version of Unisphere is in-use.

If the version of Unisphere used does not meet minimum requirements the application will exit gracefully.

Raises
SystemExit

11.2 PyU4V.common

common.py.

class PyU4V.common.**CommonFunctions**(*rest_client*)

Bases: object

CommonFunctions.

build_target_uri(***kwargs*)

Build the target URI.

This function calls into _build_uri() for access outside this class.

Key version
Unisphere version – int

Key no_version
if versionless uri – bool

Key category
resource category e.g. sloprovisioning – str

Key resource_level
resource level e.g. storagegroup – str

Key resource_level_id
resource level id – str

Key resource_type
optional name of resource – str

Key resource_type_id
optional name of resource – str

Key resource
optional name of resource – str

Key resource_id
optional name of resource – str

Key object_type
optional name of resource – str

Key object_type_id
optional name of resource – str

Returns
target URI – str

static check_epoch_timestamp(*in_epoch_timestamp*)

Check that the timestamp is in the correct format

Parameters
in_epoch_timestamp – timestamp e.g. 1554332400 – str

static check_ipv4(*ipv4*)

Check if a given string is a valid ipv6 address

Parameters
ipv4 – ipv4 address – str

Returns
string is valid ipv4 address – bool

`static check_ipv6(ipv6)`

Check if a given string is a valid ipv6 address

Parameters
ipv6 – ipv6 address – str

Returns
string is valid ipv6 address – bool

`static check_status_code_success(operation, status_code, message)`

Check if a status code indicates success.

Parameters

- **operation** – operation being performed – str
- **status_code** – status code – int
- **message** – server response – str

Raises
VolumeBackendAPIException

`static check_timestamp(in_timestamp)`

Check that the timestamp is in the correct format

Parameters
in_timestamp – timestamp e.g. 2020-11-24 15:00 – str

`static convert_to_snake_case(camel_case_string)`

Convert a string from camel case to snake case.

Parameters
camel_case_string – string for formatting – str

Returns
snake case variant – str

`create_resource(*args, **kwargs)`

Create a resource.

Key version
Unisphere version – int

Key no_version
if versionless uri – bool

Key category
resource category e.g. sloprovisioning – str

Key resource_level
resource level e.g. storagegroup – str

Key resource_level_id
resource level id – str

Key resource_type
optional name of resource – str

Key resource_type_id
optional name of resource – str

Key resource
optional name of resource – str

Key resource_id
optional name of resource – str

Key object_type
optional name of resource – str

Key object_type_id
optional name of resource – str

Key payload
query parameters – dict

Returns
resource object – dict

`delete_resource`(**args*, ***kwargs*)

Delete a resource.

Key version
Unisphere version – int

Key no_version
if versionless uri – bool

Key category
resource category e.g. sloprovisioning – str

Key resource_level
resource level e.g. storagegroup – str

Key resource_level_id
resource level id – str

Key resource_type
optional name of resource – str

Key resource_type_id
optional name of resource – str

Key resource
optional name of resource – str

Key resource_id
optional name of resource – str

Key object_type
optional name of resource – str

Key object_type_id
optional name of resource – str

Key payload
query parameters

`download_file`(***kwargs*)

Download a file.

Key version
Unisphere version – int

Key no_version
if versionless uri – bool

Key category
resource category e.g. sloprovisioning – str

Key resource_level
resource level e.g. storagegroup – str

Key resource_level_id
resource level id – str

Key resource_type
optional name of resource – str

Key resource_type_id
optional name of resource – str

Key resource
optional name of resource – str

Key resource_id
optional name of resource – str

Key object_type
optional name of resource – str

Key object_type_id
optional name of resource – str

Key payload
query parameters – dict

Returns
file info including binary data – dict

Raises
ValueError

`get_array`(*array_id*)

Get array details.

Parameters
array_id – array id – str

Returns
array details – dict

`get_array_list`(*filters=None*)

Return a list of arrays.

Parameters
filters – optional filters – dict

Returns
arrays ids – list

`get_iterator_page_list`(*iterator_id*, *start*, *end*)

Get a page of results from an iterator instance.

Parameters

- **iterator_id** – iterator id – str

- **start** – the start number – int
- **end** – the end number – int

Returns
iterator page results – dict

get_iterator_results(*rest_response*)

Get all results from all pages of an iterator if count > 1000.

Parameters
rest_response – response JSON from REST API – dict

Returns
all results – dict

get_job_by_id(*job_id*)

Get details of a specific job.

Parameters
job_id – job id – str

Returns
job details – dict

get_request(*target_uri*, *resource_type*, *params=None*)

Send a GET request to the array.

Parameters

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

Returns
resource_object – dict

Raises
ResourceNotFoundException

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

Get resource details from the array.

Key version
Unisphere version – int

Key no_version
if versionless uri – bool

Key category
resource category e.g. sloprovisioning – str

Key resource_level
resource level e.g. storagegroup – str

Key resource_level_id
resource level id – str

Key resource_type
optional name of resource – str

Key resource_type_id
optional name of resource – str

Key resource
optional name of resource – str

Key resource_id
optional name of resource – str

Key object_type
optional name of resource – str

Key object_type_id
optional name of resource – str

Key params
query parameters – dict

Returns
resource object – dict

`get_uni_version()`

Get the unisphere version from the server.

Returns
version and major_version e.g. "V10.0.0.0", "100" – str, str

`get_uni_version_info()`

Get the unisphere version from the server.

Returns
{'version': 'T10.1.0.468', 'api_version': '101', 'supported_api_versions': ['101', '100', '92']} – dict

`get_v3_or_newer_array_list(filters=None)`

Return a list of V3 or newer arrays in the environment.

Parameters
filters – optional filters – dict

Returns
arrays ids – list

`is_array_v4(array_id)`

Check to see if array is a v4

Parameters
array_id – the array serial number

Returns
bool

`modify_resource(*args, **kwargs)`

Modify a resource.

Key version
Unisphere version – int

Key no_version
if versionless uri – bool

Key category
resource category e.g. sloprovisioning – str

Key resource_level
resource level e.g. storagegroup – str

Key resource_level_id
resource level id – str

Key resource_type
optional name of resource – str

Key resource_type_id
optional name of resource – str

Key resource
optional name of resource – str

Key resource_id
optional name of resource – str

Key object_type
optional name of resource – str

Key object_type_id
optional name of resource – str

Key payload
query parameters

Returns
resource object – dict

`upload_file`(***kwargs*)

Upload a file.

Key version
Unisphere version – int

Key no_version
if versionless uri – bool

Key category
resource category e.g. sloprovisioning – str

Key resource_level
resource level e.g. storagegroup – str

Key resource_level_id
resource level id – str

Key resource_type
optional name of resource – str

Key resource_type_id
optional name of resource – str

Key resource
optional name of resource – str

Key resource_id
optional name of resource – str

Key object_type
optional name of resource – str

Key object_type_id
optional name of resource – str

Key form_data
multipart form data – dict

Returns
response success details – dict

`wait_for_job`(*operation*, *status_code*, *job*)

Check if call is async, wait for it to complete.

Parameters

- **operation** – operation being performed – str
- **status_code** – status code – int
- **job** – job id – str

Returns
task details – list

Raises
VolumeBackendAPIException

`wait_for_job_complete`(*job*)

Given the job wait for it to complete.

Parameters
job – job details – dict

Returns
response code, result, status, task details – int, str, str, list

Raises
VolumeBackendAPIException

11.3 PyU4V.metro_dr

metro_dr.py.

`class PyU4V.metro_dr.MetroDRFunctions`(*array_id*, *rest_client*)

Bases: `object`

Metro DR Functions.

`convert_to_metrodr_environment`(*storage_group_name*, *environment_name*, *metro_r1_array_id=None*, *_async=True*)

Converts existing R2–Async–R11–Metro–R2 to Metro DR Environment.

Automatically adds recovery RDFG between Metro R2 and Async R2.

Parameters

- **storage_group_name** – storage group name containing source devices – str
- **environment_name** – name of Metro Dr Environment up to 16 characters– str
- **metro_r1_array_id** – 12 Digit Serial Number of R1 Array for SRDF Metro Source Array, optional – int
- **_async** – if call should be executed asynchronously or synchronously – bool

Returns
details of newly created metro dr environment – dict

create_metrodr_environment(*storage_group_name*, *environment_name*, *metro_r1_array_id*, *metro_r2_array_id*, *dr_array_id*, *dr_replication_mode*, *metro_r2_storage_group_name=None*, *dr_storage_group_name=None*, *force_new_metro_r1_dr_rdfg=True*, *force_new_metro_r2_dr_rdfg=True*, *_async=True*)

Protects Non-SRDF Storage Group with Metro and DR legs.

Note: This function is set to run as an Asynchronous job on the server by default as there is the potential for this task to take a few minutes. Storage Groups and SRDF groups are created automatically for the user end result is R2–SRDF/A–R11–Metro–R2.

Parameters

- **storage_group_name** – name of storage group containing devices to be replicated in Metro DR environment – str
- **environment_name** – name of Metro Dr Environment up to 16 characters– str
- **metro_r1_array_id** – 12 Digit Serial Number of R1 Array for SRDF Metro Source Array – int
- **metro_r2_array_id** – 12 Digit Serial Number of SRDF Metro R2 Array – int
- **dr_array_id** – 12 Digit Serial Number of Disaster Recovery Array, replication – int
- **dr_replication_mode** – Asynchronous or AdaptiveCopyDisk – str
- **metro_r2_storage_group_name** – Name for R2 Storage Group of metro SRDF pairing, only used if R2 group naming is required to be different from source - str
- **dr_storage_group_name** – Name for Storage Group at DR, only used if group naming is required to be different from source - str
- **force_new_metro_r1_dr_rdfg** – whether or not to create a new RDFG to be created for Metro R1 array to DR array, or will autoselect from existing – bool
- **force_new_metro_r2_dr_rdfg** – whether or not to create a new RDFG to be created for Metro R2 array to DR array, or will autoselect from existing – bool
- **_async** – if call should be executed asynchronously or synchronously – bool

Returns
details of newly created metro dr environment– dict

delete_metrodr_environment(*environment_name*, *remove_r1_dr_rdfg=False*, *force=False*, *metro_r1_array_id=None*)

Deletes Metro DR Environment.

SRDF replication will remain in place for Metro and SRDF/A configuration, be default this function simply removes the standby SRDF group for Metro DR environment on R2 site. If you intend to remove SRDF replication completely, suspend and delete SRDF pairings using suspend_storage_group_srdf and suspend_storage_group_srdf functions.

Parameters

- **environment_name** – name of Metro Dr Environment up to 16 characters– str
- **remove_r1_dr_rdfg** – override default behavior and delete R11-R2 RDFG from metro R1 side – bool
- **force** – required True if deleting R1 DR group – bool

- **metro_r1_array_id** – 12 Digit Serial of Metro R1 source array – str

get_metrodr_environment_details(*environment_name*, *array_id=None*, *config=True*)

Get details for metro DR environment.

Parameters

- **environment_name** – Unique name to identify Metro DR environment – str
- **array_id** – 12 Digit Serial Number of Array – int
- **config** – return full environment config details or summary – bool

Returns
details of the metro dr environments – dict

get_metrodr_environment_list(*array_id=None*)

Gets a list of metro dr environments.

Parameters
array_id – 12 Digit Serial Number of Array – int

Returns
list of metro dr environments –list

modify_metrodr_environment(*environment_name*, *action*, *metro=False*, *dr=False*, *keep_r2=False*, *force=False*, *symforce=False*, *_async=False*, *dr_replication_mode=None*)

Performs Functions to modify state of MetroDR environment.

Parameters

- **environment_name** – name of Metro Dr Environment up to 16 characters– str
- **action** – action to be performed on Environment, Establish, Failover, Failback, Restore, SetMode, Split, UpdateR1 –str
- **metro** – directs action towards R11–R21 Metro Device leg of Metro DR environment – bool
- **dr** – directs action towards Device Pairs on Disaster Recovery leg of Metro DR environment – bool
- **keep_r2** – Used with Suspend Option, Ensures that the R2 data on Metro remains available to host – bool
- **force** – forces operation to complete, used with caution, not recommended as part of fully automated workflow –bool
- **symforce** – forces operation to complete, used with caution, requires ALLOW_SRDF_SYMFORCE parameter to be set in solutions enabler options file, default is not enabled,not recommended as part of fully automated workflow – bool
- **_async** – if call should be executed asynchronously or synchronously – bool
- **dr_replication_mode** – set mode of DR link, AdaptiveCopyDisk or Asynchronous – str

Returns
details of metro dr environment and state – dict

11.4 PyU4V.migration

migration.py.

class PyU4V.migration.**MigrationFunctions**(*array_id*, *rest_client*)

Bases: object

MigrationFunctions.

create_migration_environment(*target_array_id*)

Create a new migration environment between two arrays.

Creates a new migration environment between two arrays for use with non disruptive migrations

Parameters
target_array_id – target array id – str

Returns
migration environment info – dict

create_storage_group_migration(*storage_group_name*, *target_array_id*, *srp_id=None*, *port_group_id=None*, *no_compression=False*, *pre_copy=False*, *validate=False*)

Create a migration session for a storage group.

Parameters

- **storage_group_name** – storage group id – str
- **target_array_id** – target array id – str
- **srp_id** – storage resource pool id – str
- **port_group_id** – port group id – str
- **no_compression** – dont use compression – bool
- **pre_copy** – use pre copy – bool
- **validate** – validate – bool

Returns
new storage group – dict

delete_migration_environment(*target_array_id*)

Delete migration environment.

Given a target array will delete migration environment, used once all migrations are complete

Parameters
target_array_id – target array id – str

delete_storage_group_migration(*storage_group_name*)

Given a name, delete the storage group migration session.

Parameters
storage_group_name – storage group id – str

get_array_migration_capabilities(*array_id=None*)

Check what migration facilities are available.

Returns
array capabilities – dict

get_environment(*target_array_id*)

Given a name, return migration environment details.

Parameters
target_array_id – target array id – str

Returns
environment details – dict

get_environment_list()

Get list of all migration environments.

Returns
environments – list

get_migration_info(*array_id=None*)

Return migration information for an array.

Returns
migration info – dict

get_storage_group(*storage_group_name*)

Given a name, return storage group migrations details.

Parameters
storage_group_name – storage group id – str

Returns
storage group details – dict

get_storage_group_list(*include_migrations=False*)

Get list of all storage groups or migrating storage groups.

Parameters
include_migrations – return only SGs with migration sessions – bool

Returns
storage groups or migrating storage groups – list

get_storage_groups()

Get all storage groups and migrating storage groups.

Returns
storage groups and migrating storage groups – dict

modify_storage_group_migration(*storage_group_name*, *action*, *options=None*, *_async=False*)

Modify the state of a storage group's migration session.

Valid migrations options are 'Cutover', 'Sync', 'Commit', 'Recover', and 'ReadyTgt'.

Parameters

- **storage_group_name** – storage group id – str
- **action** – migration action – str
- **options** – migration options, example: {cutover': {'force': True}} – dict
- **_async** – if call should be async – bool

Returns
modified storage group info – dict

11.5 PyU4V.performance

performance.py.

class PyU4V.performance.**PerformanceFunctions**(*array_id*, *rest_client*)

Bases: object

PerformanceFunctions.

backup_performance_database(*array_id=None*, *filename=''*, *last_day_of_diagnostic=False*, *named_real_time_traces=False*)

Backup an array performance database.

Backup of a performance database is a recommended practice. The backup performance database option is available for one or more storage systems, regardless of their registration status.

By default, only Trending & Planning (Historical) data is backed up. The performance databases backups should be stored in a safe location. Performance database backups can be restored. For more information on restoring backups please see Unisphere for PowerMax official documentation, for now only performing backups is supported via REST.

Note: Underscores will be stripped from any filename provided, this is due to Unisphere restricting the length of the filename string when underscores are provided.

The backup filename format will be as follows when viewed in Unisphere:

{array_id}_{date}{time}_{TZ}_{filename}_SPABackup.dat

Parameters

- **array_id** – array id – str
- **filename** – performance backup file name – str
- **last_day_of_diagnostic** – Last day of Diagnostics, this option is not recommended for recurring backups – bool
- **named_real_time_traces** – Named Real Time Traces, this option is not recommended for recurring backups – bool

Raises
exception.VolumeBackendAPIException

disable_diagnostic_data_collection(*array_id=None*)

Disable an array from diagnostic performance data gathering.

Note: Disabling diagnostic performance data gathering will also disable real-time data gathering.

Parameters
array_id – array id – str

Raises
VolumeBackendAPIException

disable_real_time_data_collection(*array_id=None*)

Disable an array from real-time performance data gathering.

Parameters
array_id – array_id – str

Raises
VolumeBackendAPIException

enable_diagnostic_data_collection(*array_id=None*)

Register an array for diagnostic performance data gathering.

Parameters
array_id – array id – str

Raises
VolumeBackendAPIException

enable_real_time_data_collection(*array_id=None*, *storage_group_list=None*, *file=False*)

Register an array for real-time performance data gathering.

Note: Real-time performance data is not supported for arrays running HyperMax OS.

Parameters

- **array_id** – array id – str
- **storage_group_list** – comma separated list of storage groups to be registered for real time stats collection e.g. 'sg1, sg2' – str
- **file** – register file collection for performance –bool

Raises
VolumeBackendAPIException

extract_timestamp_keys(*array_id=None*, *category=None*, *director_id=None*, *key_tgt_id=None*)

Retrieve the timestamp keys for a given performance asset.

Note: If a director key timestamp is required, set this as the key_tgt_id, the input parameter director_id is only required for port key extraction.

Parameters

- **array_id** – array id – str
- **category** – performance category – str
- **director_id** – director id – str
- **key_tgt_id** – object id for the timestamp required – str

Returns
timestamp in milliseconds since epoch – str

static format_metrics(*metrics*)

Format metrics input for inclusion in REST request.

Take metric parameters and format them correctly to be used in REST request body. Valid input types are string and list.

Parameters
metrics – metric(s) – str or list

Returns
metrics – list

Raises
InvalidInputException

format_time_input(*array_id=None*, *category=None*, *director_id=None*, *key_tgt_id=None*, *start_time=None*, *end_time=None*)

Format time range for use in the request object.

Use cases are: 1. If start is set but not end, set end to most recent timestamp 2. If end is set but not start, set start time to first available 3. If neither are set, use most recent timestamp 4. If both are set, skip if conditions and check input is valid

Note: If a director key timestamp is required, set this as the key_tgt_id, the input parameter director_id is only required for port key extraction. A category is only required when timestamp extraction is required.

Parameters

- **array_id** – array id – str
- **category** – performance category – str
- **director_id** – director id (for port key extraction only) – str
- **key_tgt_id** – object id for the timestamp required – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str

Returns
start time, end time (tuple) – str, str

Raises
InvalidInputException

generate_threshold_settings_csv(*output_csv_path*, *category=None*)

Generate a csv file with threshold settings.

Creates a CSV file with current alert configuration for the given unisphere instance category, metric, first_threshold, second_threshold, alert_user, kpi.

Parameters

- **output_csv_path** – filename for CSV to be generated – str
- **category** – threshold specific category – str

get_array_keys()

List Arrays registered for performance data collection.

Returns
Arrays with first and last available dates – list

get_array_registration_details(*array_id=None*)

Get array performance registration details.

This call will return information about both diagnostic and real-time performance registration along with the diagnostic collection interval in minutes.

Parameters
array_id – array id – str

Returns
array performance registration details – dict

get_array_stats(*metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List performance data for specified array for giving time range.

Parameters

- **metrics** – performance metrics to retrieve – str or list

- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_backend_director_keys(*array_id=None*)

List BE directors for the given array.

Parameters
array_id – array id – str

Returns
BE directors with first and last available dates – list

get_backend_director_stats(*director_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given BE director.

Parameters

- **director_id** – director id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_backend_emulation_keys(*array_id=None*)

List BE emulations for the given array.

Parameters
array_id – array id – str

Returns
BE emulation info with first and last available dates – list

get_backend_emulation_stats(*emulation_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given BE emulation.

Parameters

- **emulation_id** – emulation id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str

- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_backend_port_keys(*director_id*, *array_id=None*)

List BE ports for the given array.

Parameters

- **director_id** – array id – str
- **array_id** – director id – str

Returns
BE port info with first and last available dates – list

get_backend_port_stats(*director_id*, *port_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given BE port.

Parameters

- **director_id** – director id – str
- **port_id** – port id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_board_keys(*array_id=None*)

List boards for the given array.

Parameters
array_id – array id – str

Returns
board info with first and last available dates – list

get_board_stats(*board_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given board.

Parameters

- **board_id** – board id – str
- **metrics** – performance metrics to retrieve – str or list

- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_cache_partition_keys(*array_id=None*)

List cache partitions for the given array.

Parameters
array_id – array id – str

Returns
cache partition info with first and last available dates – list

get_cache_partition_perf_stats(*cache_partition_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given cache partition.

Parameters

- **cache_partition_id** – cache partition id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_cloud_provider_keys(*array_id=None*)

List cache partitions for the given array.

Parameters
array_id – array id – str

Returns
cache partition info with first and last available dates – list

get_cloud_provider_stats(*cloud_provider_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given cache partition.

Parameters

- **cloud_provider_id** – cache partition id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str

- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_days_to_full(*array_id=None*, *array_to_full=False*, *srp_to_full=False*, *thin_pool_to_full=False*)

Get days to full information.

Requires at least 10 Days of Performance data, available categories are 'Array', 'SRP', and 'ThinPool'.

Parameters

- **array_id** – array id – str
- **array_to_full** – get array days to full info – bool
- **srp_to_full** – get storage resource pool days to full info – bool
- **thin_pool_to_full** – get thin pool days to full info – bool

Returns
days to full information – list

get_device_group_keys(*array_id=None*)

List device groups for the given array.

Parameters
array_id – array id – str

Returns
device group info with first and last available dates – list

get_device_group_stats(*device_group_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given device group.

Parameters

- **device_group_id** – device group id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_disk_group_keys(*array_id=None*)

List disk groups for the given array.

Parameters
array_id – array id – str

Returns
disk info with first and last available dates – list

`get_disk_group_stats`(*disk_group_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given disk group.

Parameters

- **disk_group_id** – disk group id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

`get_eds_director_keys`(*array_id=None*)

List EDS directors for the given array.

Parameters
array_id – array id – str

Returns
EDS director info with first and last available dates – list

`get_eds_director_stats`(*director_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given EDS director.

Parameters

- **director_id** – director id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

`get_eds_emulation_keys`(*array_id=None*)

List EDS emulations for the given array.

Parameters
array_id – array id – str

Returns
EDS emulation info with first and last available dates – list

`get_eds_emulation_stats`(*emulation_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given EDS emulation.

Parameters

- **emulation_id** – emulation id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

`get_em_director_keys`(*array_id=None*)

List EM directors for the given array.

Parameters
array_id – array id – str

Returns
EDS director info with first and last available dates – list

`get_em_director_stats`(*director_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given EM director.

Parameters

- **director_id** – director id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

`get_endpoint_keys`(*array_id=None*)

List endpoints for the given array.

Parameters
array_id – array_id: array id – str

Returns
Endpoint info with first and last available dates – list

get_endpoint_stats(*endpoint_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given iSCSI target.

Parameters

- **endpoint_id** – iSCSI target id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_external_disk_keys(*array_id=None*)

List external disks for the given array.

Parameters
array_id – array id – str

Returns
external disks with first and last available dates – list

get_external_disk_stats(*disk_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given external disk.

Parameters

- **disk_id** – disk id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_ficon_emulation_keys(*array_id=None*)

List FICON emulations for the given array.

Parameters
array_id – array id – str

Returns
FICON emulation info with first and last available dates – list

get_ficon_emulation_stats(*ficon_emulation_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given FICON emulation.

Parameters

- **ficon_emulation_id** – FICON emulation id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_ficon_emulation_thread_keys(*array_id=None*)

List FICON emulation threads for the given array.

Parameters
array_id – array id – str

Returns
FICON emulation thread info with first and last available dates – list

get_ficon_emulation_thread_stats(*ficon_emulation_thread_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given FICON emulation thread.

Parameters

- **ficon_emulation_thread_id** – FICON emulation thread id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_ficon_port_thread_keys(*array_id=None*)

List FICON port threads for the given array.

Parameters
array_id – array id – str

Returns
FICON port info with first and last available dates – list

get_ficon_port_thread_stats(*ficon_port_thread_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given FICON port thread.

Parameters

- **ficon_port_thread_id** – FICON port thread id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_frontend_director_keys(*array_id=None*)

List FE directors for the given array.

Parameters
array_id – array id – str

Returns
FE directors with first and last available dates – list

get_frontend_director_stats(*director_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given FE director.

Parameters

- **director_id** – director id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_frontend_emulation_keys(*array_id=None*)

List FE emulations for the given array.

Parameters
array_id – array id – str

Returns
BE emulation info with first and last available dates – list

get_frontend_emulation_stats(*emulation_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given FE emulation.

Parameters

- **emulation_id** – emulation id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_frontend_port_keys(*director_id*, *array_id=None*)

List FE ports for the given array.

Parameters

- **director_id** – array id – str
- **array_id** – director id – str

Returns
FE port info with first and last available dates – list

get_frontend_port_stats(*director_id*, *port_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given FE port.

Parameters

- **director_id** – director id – str
- **port_id** – port id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_host_keys(*array_id=None*, *start_time=None*, *end_time=None*)

List active hosts for the given array by time range.

Only active hosts from within the specified time range are returned. If no time range is provided, start and end times from array level are used.

Parameters

- **array_id** – array id – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str

Returns
host info with first and last available dates – list

get_host_stats(*host_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given host.

Performance details will only be returned if the host was active during the specified time range. If no time range is provided, start and end times from array level are used.

Parameters

- **host_id** – host id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_im_director_keys(*array_id=None*)

List IM directors for the given array.

Parameters
array_id – array id – str

Returns
IM directors with first and last available dates – list

get_im_director_stats(*director_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given IM director.

Parameters

- **director_id** – director id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_im_emulation_keys(*array_id=None*)

List IM emulations for the given array.

Parameters
array_id – array id – str

Returns
IM emulation info with first and last available dates – list

get_im_emulation_stats(*emulation_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given IM emulation.

Parameters

- **emulation_id** – emulation id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_initiator_perf_keys(*array_id=None*, *start_time=None*, *end_time=None*)

List active initiators for the given array by time range

Only active initiators from within the specified time range are returned. If no time range is provided, start and end times from array level are used.

Parameters

- **array_id** – array id – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str

Returns
host info with first and last available dates – list

get_initiator_stats(*initiator_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given initiator.

Performance details will only be returned if the initiator was active during the specified time range. If no time range is provided, start and end times from array level are used.

Parameters

- **initiator_id** – initiator id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str

- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_ip_interface_keys(*array_id=None*)

List IP interfaces for the given array.

Parameters
array_id – array id – str

Returns
IP interface info with first and last available dates – list

get_ip_interface_stats(*ip_interface_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given IP interface.

Parameters

- **ip_interface_id** – IP interface id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_iscsi_target_keys(***kwargs*)

get_iscsi_target_stats(***kwargs*)

get_last_available_timestamp(*array_id=None*)

Get the last recorded performance timestamp.

Parameters
array_id – array_id: array id – str

Returns
timestamp – int

Raises
ResourceNotFoundException

get_masking_view_keys(*array_id=None*)

List masking views for the given array.

Parameters
array_id – array id – str

Returns
masking view info with first and last available dates – list

get_masking_view_stats(*masking_view_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given masking view.

Parameters

- **masking_view_id** – masking view id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_performance_categories_list(*array_id=None*)

Get the list of supported performance categories.

Parameters
array_id – array id – str

Returns
categories – list

get_performance_key_list(*category*, *array_id=None*, *director_id=None*, *storage_group_id=None*, *storage_container_id=None*, *storage_resource_id=None*, *start_time=None*, *end_time=None*)

Get performance key list for a given performance category.

Parameters

- **category** – performance category – str
- **array_id** – array id – str
- **director_id** – director id – str
- **storage_group_id** – storage group id – str
- **storage_container_id** – storage container id – str
- **storage_resource_id** – storage resource id – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str

Returns
category performance keys – list

Raises
InvalidInputException

get_performance_metrics_list(*category*, *kpi_only=False*, *array_id=None*)

For a given category, return the list of valid metrics.

Parameters

- **category** – performance category – str
- **kpi_only** – if only KPI metrics should be returned – bool
- **array_id** – array id – str

Returns
metrics – list

get_performance_stats(*category*, *metrics*, *data_format='Average'*, *array_id=None*, *request_body=None*, *start_time=None*, *end_time=None*, *recency=None*)

Retrieve the performance statistics for a given category and object.

Parameters

- **category** – category id – str
- **array_id** – array id – str
- **metrics** – performance metrics, options are individual metrics, a list of metrics, 'KPI' for KPI metrics only, and 'ALL' for all metrics – str/list
- **data_format** – response data format 'Average' or 'Maximum' – str
- **request_body** – request params and object IDs – dict
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

Raises
VolumeBackendAPIException, InvalidInputException

get_port_group_keys(*array_id=None*)

List port group for the given array.

Parameters
array_id – array_id: array id – str

Returns
port group info with first and last available

get_port_group_stats(*port_group_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given port group.

Parameters

- **port_group_id** – port group id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

`get_rdf_director_keys`(*array_id=None*)

List RDF directors for the given array.

Parameters
array_id – array id – str

Returns
RDF directors with first and last available dates – list

`get_rdf_director_stats`(*director_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given RDF director.

Parameters

- **director_id** – director id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

`get_rdf_emulation_keys`(*array_id=None*)

List RDF emulations for the given array.

Parameters
array_id – array id – str

Returns
RDF emulation info with first and last available dates – list

`get_rdf_emulation_stats`(*emulation_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given RDF emulation.

Parameters

- **emulation_id** – emulation id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_rdf_port_keys(*director_id*, *array_id=None*)

List RDF ports for the given array.

Parameters

- **director_id** – array id – str
- **array_id** – director id – str

Returns
RDF port info with first and last available dates – list

get_rdf_port_stats(*director_id*, *port_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given RDF port.

Parameters

- **director_id** – director id – str
- **port_id** – port id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_rdfa_keys(*array_id=None*)

List RDFA groups for the given array.

Parameters
array_id – array_id: array id – str

Returns
RDFA info with first and last available dates – list

get_rdfa_stats(*rdfa_group_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given RDFA group.

Parameters

- **rdfa_group_id** – RDFA group id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

`get_rdfs_keys`(*array_id=None*)

List RDFS groups for the given array.

Parameters
array_id – array_id: array id – str

Returns
RDFS info with first and last available dates – list

`get_rdfs_stats`(*rdfs_group_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given RDFS group.

Parameters

- **rdfs_group_id** – RDFS group id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

`get_sdnas_filesystem_keys`(*array_id=None*)

List SDNAS Filesystems for the given array.

Parameters
array_id – array id – str

Returns
SDNAS Filesystem info with first and last available dates – list

`get_sdnas_filesystem_stats`(*sdnas_filesystem_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given SDNAS Filesystem.

Parameters

- **sdnas_filesystem_id** – SDNAS Filesystem id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_sdnas_interface_keys(*array_id=None*)

List SDNAS Interfaces for the given array.

Parameters
array_id – array id – str

Returns
SDNAS Interface info with first and last available dates – list

get_sdnas_interface_stats(*sdnas_interface_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given SDNAS Interface.

Parameters

- **sdnas_interface_id** – SDNAS Interface id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_sdnas_node_keys(*array_id=None*)

List SDNAS Nodes for the given array.

Parameters
array_id – array id – str

Returns
SDNAS Interface info with first and last available dates – list

get_sdnas_node_stats(*sdnas_node_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given SDNAS Node.

Parameters

- **sdnas_node_id** – SDNAS Node id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_sdnas_server_keys(*array_id=None*)

List SDNAS Servers for the given array.

Parameters
array_id – array id – str

Returns
SDNAS Interface info with first and last available dates – list

get_sdnas_server_stats(*sdnas_server_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given SDNAS Server.

Parameters

- **sdnas_server_id** – SDNAS Server id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_storage_container_keys(*array_id=None*)

List storage containers for the given array.

Parameters
array_id – array id – str

Returns
storage container info with first and last available dates – list

get_storage_container_stats(*storage_container_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given storage container.

Parameters

- **storage_container_id** – storage container id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

`get_storage_group_keys`(*array_id=None*)

List storage groups for the given array.

Parameters
`array_id` – array id – str

Returns
storage container info with first and last available dates – list

`get_storage_group_stats`(*storage_group_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given storage group.

Parameters

- `storage_group_id` – storage group id – str
- `metrics` – performance metrics to retrieve – str or list
- `array_id` – array id – str
- `data_format` – response data format 'Average' or 'Maximum' – str
- `start_time` – timestamp in milliseconds since epoch – str
- `end_time` – timestamp in milliseconds since epoch – str
- `recency` – check recency of timestamp in minutes – int

Returns
performance metrics – dict

`get_storage_resource_keys`(*array_id=None*)

List storage resources for the given array.

Parameters
`array_id` – array id – str

Returns
storage resource info with first and last available dates – list

`get_storage_resource_pool_keys`(*array_id=None*)

List storage resource pools for the given array.

Parameters
`array_id` – array id – str

Returns
storage resource pool info with first and last available dates – list

`get_storage_resource_pool_stats`(*srp_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given storage resource pools.

Parameters

- `srp_id` – storage resource pool id – str
- `metrics` – performance metrics to retrieve – str or list
- `array_id` – array id – str
- `data_format` – response data format 'Average' or 'Maximum' – str
- `start_time` – timestamp in milliseconds since epoch – str

- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_storage_resource_stats(*storage_container_id*, *storage_resource_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given storage resource.

Parameters

- **storage_container_id** – storage container id – str
- **storage_resource_id** – storage resource id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_thin_pool_keys(*array_id=None*)

List thin pools for the given array.

Parameters
array_id – array id – str

Returns
thin pools with first and last available dates – list

get_thin_pool_stats(*thin_pool_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given thin pool.

Parameters

- **thin_pool_id** – thin pool id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_threshold_categories(*array_id=None*)

Get a list of performance threshold categories.

Parameters
array_id – array serial number – str

Returns
performance threshold categories – list

get_threshold_category_settings(*category*, *array_id=None*)

Get performance threshold category settings.

Parameters

- **category** – category id – str
- **array_id** – array serial number – str

Returns
category settings – dict

static get_timestamp_by_hour(*start_time=None*, *end_time=None*, *hours_difference=None*)

Get timestamp difference in hours from supplied time.

If start time is provided but not end time, the time difference will be after the start time.

If end time is provided but not start time, the time difference will be before the end time.

If neither start or end time are provided, or both are incorrectly provided, the time difference is from the current time.

Parameters

- **start_time** – timestamp in milliseconds since epoch – int
- **end_time** – timestamp in milliseconds since epoch – int
- **hours_difference** – difference in hours – int

Returns
timestamp in milliseconds since epoch – str

get_volume_stats(*array_id=None*, *volume_range_start=None*, *volume_range_end=None*, *storage_group_list=None*, *metrics=None*, *start_time=None*, *end_time=None*, *recency=None*, *data_format=None*)

List Performance data for volume level statistics.

Note: This function can gather statistics for up to 10,000 volumes or 100 Storage groups per call, time range can not exceed 1 hour/60 minutes .

Parameters

- **volume_range_start** – 5 digit device id of first device in range – str
- **volume_range_end** – 5 digit device id of last device in range – str
- **storage_group_list** – list of up to 100 storage groups – str or list
- **metrics** – performance metrics to retrieve, if not specified all available metrics will return by default– str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str

- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

get_zhyperlink_port_keys(*array_id=None*)

List zHyperLink Ports for the given array.

Parameters
array_id – array id – str

Returns
thin pools with first and last available dates – list

get_zhyperlink_port_stats(*zhyperlink_port_id*, *metrics*, *array_id=None*, *data_format='Average'*, *start_time=None*, *end_time=None*, *recency=None*)

List time range performance data for given zHyperLink Port.

Parameters

- **zhyperlink_port_id** – zHyperLink Port id – str
- **metrics** – performance metrics to retrieve – str or list
- **array_id** – array id – str
- **data_format** – response data format 'Average' or 'Maximum' – str
- **start_time** – timestamp in milliseconds since epoch – str
- **end_time** – timestamp in milliseconds since epoch – str
- **recency** – check recency of timestamp in minutes – int

Returns
performance metrics – dict

is_array_diagnostic_performance_registered(*array_id=None*)

Check if an array is registered for diagnostic performance data.

Parameters
array_id – array id – str

Returns
is diagnostic registered – bool

is_array_performance_registered(***kwargs*)

is_array_real_time_performance_registered(*array_id=None*)

Check if an array is registered for real-time performance data.

Parameters
array_id – array id – str

Returns
is real-time registered – bool

is_timestamp_current(*timestamp*, *minutes=None*)

Check if the timestamp is less than a user specified set of minutes.

If no minutes value is provided, self.recency is used. Seven minutes is recommended to provide a small amount of time for the STP daemon to record the next set of metrics in five minute intervals.

Parameters

- **timestamp** – timestamp in milliseconds since epoch – int
- **minutes** – timestamp recency in minutes – int

Returns
if timestamp is less than recency value – bool

set_array_id(*array_id*)

Set the array id.

Parameters
array_id – array id – str

set_recency(*minutes*)

Set the recency value in minutes.

Parameters
minutes – recency minutes – int

set_thresholds_from_csv(*csv_file_path*, *kpi_only=True*)

Set performance thresholds using a CSV file.

Reads CSV file and sets performance threshold metrics on the values contained within. The following headers are required: category, metric, firstthreshold, secondthreshold, notify, kpi

It is advisable to generate the CSV file from the function performance.generate_threshold_settings_csv() and edit those values within that you would like to change.

Parameters

- **csv_file_path** – path to CSV file – str
- **kpi_only** – set only KPI thresholds – bool

set_timestamp(*timestamp*)

Set the performance timestamp.

Parameters
timestamp – the performance timestamp – str

update_threshold_settings(*category*, *metric*, *first_threshold*, *second_threshold*, *alert=True*, *first_threshold_occurrences=3*, *first_threshold_samples=5*, *first_threshold_severity='WARNING'*, *second_threshold_occurrences=3*, *second_threshold_samples=5*, *second_threshold_severity='CRITICAL'*)

Edit an existing global threshold across all arrays.

Parameters

- **category** – category id – str
- **metric** – performance metric – str
- **first_threshold** – first threshold value – int
- **second_threshold** – second threshold value – int
- **alert** – alert on/off – bool
- **first_threshold_occurrences** – error occurrences – int
- **first_threshold_samples** – error samples – int

- **first_threshold_severity** – error severity, valid values are 'INFORMATION', 'WARNING', and 'CRITICAL' – str
- **second_threshold_occurrences** – error occurrences – int
- **second_threshold_samples** – error samples – int
- **second_threshold_severity** – error severity, valid values are 'INFORMATION', 'WARNING', and 'CRITICAL' – str

Returns
operation success details – dict

validate_category(*category*, *array_id=None*)

Check that a supplied category is valid.

Parameters

- **category** – performance category – str
- **array_id** – array id – str

Raises
InvalidInputException

11.6 PyU4V.provisioning

provisioning.py.

class PyU4V.provisioning.**ProvisioningFunctions**(*array_id*, *rest_client*)

Bases: object

ProvisioningFunctions.

add_child_storage_group_to_parent_group(*child_storage_group*, *parent_storage_group*)

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_storage_group** – child storage group id – str
- **parent_storage_group** – parent storage group id – str

Returns
storage group details – dict

add_existing_volume_to_storage_group(*storage_group_id*, *vol_ids*, *_async=False*, *remote_array_1_id=None*, *remote_array_1_sgs=None*, *remote_array_2_id=None*, *remote_array_2_sgs=None*, *starting_lun_address=None*)

Expand an existing storage group by adding existing volumes.

Parameters

- **storage_group_id** – storage group id – str
- **vol_ids** – volume device id(s) – str or list
- **_async** – if call should be async – bool
- **remote_array_1_id** – 12 digit serial number of remote array, optional – str

- **remote_array_1_sgs** – list of storage groups on remote array to add Remote device, Unisphere instance must be local to R1 storage group otherwise volumes will only be added to the local group – str or list
- **remote_array_2_id** – optional digit serial number of remote array, only used in multihop SRDF, e.g. R11, or R1 - R21 - R2 optional – str
- **remote_array_2_sgs** – storage groups on remote array, optional – str or list
- **starting_lun_address** – HLU address of starting lun for volumes – int

Returns
storage group details – dict

add_new_volume_to_storage_group(*storage_group_id*, *num_vols*, *vol_size*, *cap_unit*, *_async=False*, *vol_name=None*, *create_new_volumes=None*, *remote_array_1_id=None*, *remote_array_1_sgs=None*, *remote_array_2_id=None*, *remote_array_2_sgs=None*, *enable_mobility_id=False*, *emulation_type='FBA'*, *append_vol_id=False*, *starting_lun_address=None*)

Expand an existing storage group by adding new volumes.

Parameters

- **storage_group_id** – storage group id – str
- **num_vols** – number of volumes to be created – int
- **vol_size** – the volume size – str
- **cap_unit** – capacity unit (MB, GB, TB, CYL) – str
- **_async** – if call should be async – bool
- **vol_name** – name to give to the volume, optional – str
- **create_new_volumes** – new volumes only, no ro-use – bool
- **remote_array_1_id** – 12 digit serial number of remote array, optional – str
- **remote_array_1_sgs** – list of storage groups on remote array to add Remote device, Unisphere instance must be local to R1 storage group otherwise volumes will only be added to the local group – str or list
- **remote_array_2_id** – optional digit serial number of remote array, only used in multihop SRDF, e.g. R11, or R1 - R21 - R2 optional – str
- **remote_array_2_sgs** – storage groups on remote array, optional – str or list
- **enable_mobility_id** – enables unique volume WWN not tied to array serial number – bool
- **emulation_type** – device emulation type (CKD, FBA) – str
- **append_vol_id** – append volume id to the volume name, optional – bool
- **starting_lun_address** – HLU address of starting lun for volumes – int

Returns
storage group details – dict

create_cu_image(*split_id: str*, *cu_number: str*, *cu_ssid: str*, *cu_base_address: str*, *vol_id: str*)

Creates a new CU image under the specified split. :param split_id: split id – str :param cu_number: cu image number – str :param cu_ssid: cu image ssid – str :param cu_base_address: cu image ssid – str :param vol_id volume device id be mapped to the cu – str :returns: None

create_empty_port_group(*port_group_id*, *port_group_protocol=None*, *_async=None*)

Create an empty port group.

Parameters

- **port_group_id** – name of the new port group – str
- **port_group_protocol** – required for V4 only one of [SCSI_FC, iSCSI , NVMe_TCP] – str
- **_async** – if call should be async – bool

Returns
new port group details – dict

create_empty_storage_group(*srp_id*, *storage_group_id*, *service_level*, *workload*, *disable_compression=False*, *_async=False*, *snapshot_policy_ids=None*, *emulation_type='FBA'*)

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** – SRP id – str
- **storage_group_id** – storage group id – str
- **service_level** – service level id – str
- **workload** – workload id – str
- **disable_compression** – disable compression – bool
- **_async** – if call should be async – bool
- **snapshot_policy_ids** – list of one or more snapshot policies to associate with storage group – list
- **emulation_type** – device emulation type (CKD, FBA) – str

Returns
storage group details – 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** – name of the new host – str
- **initiator_list** – list of initiators – list
- **host_flags** – optional host flags to apply – dict
- **init_file** – path to file containing initiator names – str
- **_async** – if call should be _async – bool

Returns
new host details – dict

create_host_group(*host_group_id*, *host_list*, *host_flags=None*, *_async=False*)

Create a host group containing the given hosts.

Parameters

- **host_group_id** – name of the new host group – str
- **host_list** – hosts – list
- **host_flags** – optional host flags to apply – dict
- **_async** – if call should be async – bool

Returns
new host group details – dict

create_masking_view_existing_components(*port_group_name*, *masking_view_name*, *storage_group_name*, *host_name=None*, *host_group_name=None*, *_async=False*, *starting_lun_address=None*)

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 – str
- **masking_view_name** – name of the new masking view – str
- **storage_group_name** – name of the storage group – str
- **host_name** – name of the host (initiator group) – str
- **host_group_name** – name of host group – str
- **_async** – if command should be run asynchronously – bool
- **starting_lun_address** – HLU address of starting lun for volumes – int

Returns
masking view details – dict

Raises
InvalidInputException

create_multiport_port_group(***kwargs*)

create_new_port_group(*port_group_id*, *dir_port_list*, *port_group_protocol=None*)

Create a new port group with one or more ports.

Parameters

- **port_group_id** – name of the new port group – str
- **dir_port_list** – port dicts Example: [{'directorId': director_id, 'portId': port_id}] – list
- **port_group_protocol** – required for V4 only. one of [SCSI_FC, iSCSI, NVMe_TCP]

Returns
new port group details – dict

create_non_empty_storage_group(*srp_id*, *storage_group_id*, *service_level*, *workload*, *num_vols*, *vol_size*, *cap_unit*, *disable_compression=False*, *_async=False*, *vol_name=None*, *snapshot_policy_ids=None*, *enable_mobility_id=False*, *emulation_type='FBA'*)

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** – SRP id – str
- **storage_group_id** – storage group id – str
- **service_level** – service level id – str
- **workload** – workload id – str
- **num_vols** – number of volumes to be created – int
- **vol_size** – the volume size – str
- **cap_unit** – capacity unit (MB, GB, TB, CYL) – str
- **disable_compression** – disable compression – bool
- **_async** – if call should be async – bool
- **vol_name** – name to give to the volume, optional – str
- **snapshot_policy_ids** – list of one or more snapshot policies to associate with storage group – list
- **enable_mobility_id** – enables unique volume WWN not tied to array serial number – bool
- **emulation_type** – device emulation type (CKD, FBA) – str

Returns
storage group details – dict

create_port_group(***kwargs*)

create_port_group_from_file(*file_name*, *port_group_id*, *port_group_protocol=None*)

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** – path to the file – str
- **port_group_id** – name for the port group – str
- **port_group_protocol** – required for V4 only. one of [SCSI_FC, iSCSI, NVMe_TCP] – str

Returns
new port group details – dict

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

Create a storage group with optional volumes on create operation.

Parameters

- **srp_id** – SRP id – str
- **sg_id** – storage group id – str
- **slo** – service level id – str
- **workload** – workload id – str
- **do_disable_compression** – disable compression – bool
- **num_vols** – number of volumes to be created – int
- **vol_size** – the volume size – int
- **cap_unit** – capacity unit (MB, GB, TB, CYL) – str
- **allocate_full** – allocate full capacity – bool
- **_async** – if call should be async – bool
- **vol_name** – name to give to the volume, optional – str
- **snapshot_policy_ids** – list of one or more snapshot policies to associate with storage group – list
- **enable_mobility_id** – enables unique volume WWN not tied to array serial number – bool
- **emulation_type** – device emulation type (CKD, FBA) – str
- **append_vol_id** – append volume id to the volume name, optional – bool

Returns

storage group details – dict

create_volume_from_storage_group_return_id(*volume_name*, *storage_group_id*, *vol_size*, *cap_unit='GB'*, *enable_mobility_id=False*, *emulation_type='FBA'*)

Create a new volume in the given storage group.

Parameters

- **volume_name** – volume name – str
- **storage_group_id** – storage group id – str
- **vol_size** – volume size – str
- **cap_unit** – capacity unit (MB, GB, TB, CYL) – str
- **enable_mobility_id** – enables unique volume WWN not tied to array serial number – bool
- **emulation_type** – device emulation type (CKD, FBA) – str

Returns

device id – str

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 de-allocation is complete. This method will return when the array has accepted the request for de-allocation; the de-allocation itself happens as a background task on the array.

Parameters
device_id – device id – str

Returns
volume details – dict

delete_host(*host_id*)

Delete a given host.

Cannot delete if associated with a masking view.

Parameters
host_id – name of the host – str

delete_host_group(*host_group_id*)

Delete a given host group.

Cannot delete if associated with a masking view.

Parameters
host_group_id – name of the hostgroup – str

delete_masking_view(*maskingview_name*)

Delete a masking view.

Parameters
maskingview_name – masking view name – str

delete_port_group(*port_group_id*)

Delete a port group.

Parameters
port_group_id – name of the port group – str

delete_storage_group(*storage_group_id*)

Delete a given storage group.

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

Parameters
storage_group_id – storage group id – str

delete_volume(*device_id*)

Delete a volume.

Parameters
device_id – device id – str

extend_volume(*device_id*, *new_size*, *_async=False*, *rdf_group_num=None*)

Extend a volume.

Parameters

- **device_id** – device id – str
- **new_size** – the new size for the device – int

- **_async** – if call should be async – bool
- **rdf_group_num** – RDF group number to extend R2 device in same operation – int

Returns
volume details – dict

find_host_lun_id_for_volume(*masking_view_id*, *device_id*)

Find the host_lun_id for a volume in a masking view.

Parameters

- **masking_view_id** – masking view id – str
- **device_id** – the device id – str

Returns
host lun id – str

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 percent – int
- **csvname** – filename for CSV output file – str

find_volume_device_id(*volume_name*)

Given a volume identifier, find the corresponding device_id.

Parameters
volume_name – the volume name – str

Returns
device id – str

find_volume_identifier(*device_id*)

Get the volume identifier of a volume.

Parameters
device_id – device id – str

Returns
volume identifier – str

static format_director_port(*director*, *port*)

Format separate director port into single string.

Parameters

- **director** – director e.g. FA-2D – str
- **port** – port e.g. 4 – str

Returns
formatted director:port string –str

get_active_masking_view_connections()

Get list of active connections from any masking view.

Returns
masking view name, connection details – str, list

get_any_director_port(***kwargs*)

get_array(*array_id=None*)

Query for details of an array from SLOPROVISIONING endpoint.

Parameters
array_id – array serial number – str

Returns
array details – dict

get_available_initiator(*director_type=None*)

Get an available initiator.

Parameters
director_type – director type filter – str

Returns
single available initiator – str

get_available_initiator_list(*director_type=None*)

Get list of available initiators.

Parameters
director_type – director type filter – str

Returns
single available initiator – str

get_available_initiator_wwn_as_list()

Get an available initiator wwn string in a list.

Returns
single available initiator wwn – list

get_child_storage_groups_from_parent(*parent_name*)

Get child storage group list from parent storage group.

Parameters
parent_name – parent sg name – str

Returns
child sg details – list

get_compressibility_report(*srp_id*)

Get a specified SRP Compressibility Report.

Parameters
srp_id – srp id – str

Returns
compressibility reports – list

get_cu_image(*split_id: str*, *cu_ssid: str*)

Get details of a specified CU Image. :param split_id: split id – str :param cu_ssid: cu image ssid – str :returns: CU Image details – dict

get_cu_image_list(*split_id: str*)

Get list of CU Image SSIDs within a specific FICON Split. :param split_id: split id – str :returns: CU Image ssids – list

get_cu_image_volume(*split_id: str*, *cu_ssid: str*, *vol_id: str*)

Get details of a volume mapped to a specified CU Image. :param split_id: split id – str :param cu_ssid: cu image ssid – str :pamam vol_id volume device id to be mapped to the cu – str :returns: volume details – dict

get_cu_image_volumes(*split_id: str*, *cu_ssid: str*)

Get list of Volumes from a specified CU Image. :param split_id: split id – str :param cu_ssid: cu image ssid – str :returns: Volume ids – list

get_director(***kwargs*)

get_director_list(***kwargs*)

get_director_port(***kwargs*)

get_director_port_list(***kwargs*)

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

Returns
specified element name – str

Raises
ResourceNotFoundException

get_fa_directors(***kwargs*)

get_host(*host_id*)

Get details on a host on the array.

Parameters
host_id – the name of the host – str

Returns
host details – dict

get_host_from_masking_view(*masking_view_id*)

Given a masking view, get the associated host or host group.

Parameters
masking_view_id – name of the masking view – str

Returns
host id – str

get_host_group(*host_group_id*)

Get details on a host group on the array.

Parameters
host_group_id – name of the host group – str

Returns
host group details – dict

get_host_group_list(*filters=None*)

Get list of host group(s) on the array.

See unisphere documentation for applicable filters.

Parameters
filters – optional list of filters – dict

Returns
host group list – list

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
hosts – list

get_in_use_initiator(*director_type=None*)

Get an initiator that is in use.

Parameters
director_type – director type filter – str

Returns
single in-use initiator – str

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
in-use initiators – list

get_initiator(*initiator_id*)

Get details of an initiator.

Parameters
initiator_id – initiator id – str

Returns
initiator details – dict

get_initiator_group_from_initiator(*initiator*)

Given an initiator, get its corresponding initiator group, if any.

Parameters
initiator – the initiator id – str

Returns
found initiator group name – str or None

`get_initiator_ids_from_host`(*host_id*)

Get initiator details from a host.

Parameters
host_id – name of the host – str

Returns
initiator IDs – list

`get_initiator_list`(*params=None*)

Retrieve initiator list from the array.

Parameters
params – optional params – dict

Returns
initiators – list

`get_iscsi_ip_address_and_iqn`(***kwargs*)

`get_masking_view`(*masking_view_name*)

Get details of a masking view.

Parameters
masking_view_name – the masking view name – str

Returns
masking view details – dict

`get_masking_view_connections`(*masking_view_id*, *filters=None*)

Get all connection information for a given masking view.

Parameters

- **masking_view_id** – masking view id – str
- **filters** – optional filter parameters – dict

Returns
masking view connection dicts – list

`get_masking_view_from_storage_group`(*storage_group*)

Get the associated masking views from a given storage group.

Parameters
storage_group – name of the storage group – str

Returns
Masking views – list

`get_masking_view_list`(*filters=None*)

Get a masking view or list of masking views.

See unisphere documentation for possible filters.

Parameters
filters – filters – dict

Returns
masking views – list

get_masking_views_by_initiator_group(*initiator_group_name*)

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

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

Parameters
initiator_group_name – name of the initiator group – str

Returns
masking view names – list

get_masking_views_from_host(*host_id*)

Retrieve masking view information for a specified host.

Parameters
host_id – name of the host – str

Returns
masking views – list

get_masking_views_from_host_group(*host_group_id*)

Retrieve masking view information for a specified host group.

Parameters
host_group_id – name of the host – str

Returns
masking views – list

get_masking_views_from_storage_group(*storagegroup*)

Return any masking views associated with a storage group.

Parameters
storagegroup – storage group name – str

Returns
masking view list – list

get_num_vols_in_storage_group(*storage_group_name*)

Get the number of volumes in a storage group.

Parameters
storage_group_name – storage group name – str

Returns
number of volumes – int

get_port_group(*port_group_id*)

Get port group details.

Parameters
port_group_id – name of the portgroup – str

Returns
port group details – dict

get_port_group_common_masking_views(*port_group_name*, *initiator_group_name*)

Get common masking views for a given port group and initiator group.

Parameters

- **port_group_name** – port group name – str

- **initiator_group_name** – initiator group name – str

Returns
masking views - list

get_port_group_from_masking_view(*masking_view_id*)

Given a masking view, get the associated port group.

Parameters
masking_view_id – masking view name – str

Returns
name of the port group – str

get_port_group_list(*filters=None*)

Get port group details.

Parameters
filters – optional filters – dict

Returns
port groups – list

get_port_identifier(***kwargs*)

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 – optional filters e.g. {'vnx_attached': 'true'} – dict

Returns
port key dicts – list

get_ports_from_port_group(*port_group*)

Get a list of port identifiers from a port group.

Parameters
port_group – name of the portgroup – list

Returns
port ids – list

get_service_level(*service_level_id*)

Get details on a specific service level.

Parameters
service_level_id – service level agreement – str

Returns
service level details – dict

get_service_level_list(*filters=None*)

Retrieve the list of service levels from the array.

Parameters
filters – optional filters – dict

Returns
service level names – list

get_size_of_device_on_array(*device_id*)

Get the size of the volume from the array.

Parameters
device_id – device id – str

Returns
size – float

get_split(*split_id: str*)

Get details of a specified FICON split. :param split_id: split id – str :returns: split details – dict

get_split_list()

Get list of FICON splits from array. :returns: split ids – list

get_srp(*srp*)

Get details on a specific SRP.

Parameters
srp – storage resource pool id – str

Returns
srp details – dict

get_srp_list(*filters=None*)

Get a list of available SRPs on a given array.

Parameters
filters – filter parameters – dict

Returns
SRPs – list

get_storage_group(*storage_group_name*)

Given a name, return storage group details.

Parameters
storage_group_name – name of the storage group – str

Returns
storage group details – dict

get_storage_group_demand_report(*srp_id=None*)

Get the storage group demand report.

Get the storage group demand report from Unisphere.

Parameters
srp_id – id of the Storage Resource Pool – str

Returns
demand report – dict

get_storage_group_from_masking_view(*masking_view_id*)

Given a masking view, get the associated storage group.

Parameters
masking_view_id – masking view name – str

Returns
name of the storage group – str

get_storage_group_from_volume(*volume_id*)

Retrieve storage group information for a specified volume.

Parameters
volume_id – device id – str

Returns
storage groups – list

get_storage_group_list(*filters=None*)

Return a list of storage groups.

Parameters
filters – filter parameters e.g. {'is_link_target': True} see API documentation https://developer.dell.com/ for full list of available filters– dict

Returns
storage groups – list

get_target_wwns_from_port_group(***kwargs*)

get_volume(*device_id*)

Get a volume from array.

Parameters
device_id – device id – str

Returns
volume details – dict

get_volume_effective_wwn_details(*vol_list*, *output_file_name=None*)

Get the effective wwn for a list of vols.

Get volume details for a list of volume device ids.

Parameters

- **vol_list** – device id(s) – list
- **output_file_name** – name of the output file – str

Returns
volume details list (nested) – list

get_volume_list(*filters=None*)

Get list of volumes from array.

Parameters
filters – filters parameters – dict

Returns
device ids – list

get_volumes_from_storage_group(*storage_group_id*)

Retrieve volume information associated with a given storage group.

Parameters
storage_group_id – storage group id – name

Returns
device ids – list

get_workload_settings()

Get valid workload options from array.

Returns
workload settings – list

is_child_storage_group_in_parent_storage_group(*child_name*, *parent_name*)

Check if a child storage group is a member of a parent group.

Parameters

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

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 – str

Returns
if initiator is assigned – bool

is_volume_in_storage_group(*device_id*, *storage_group_id*)

See if a volume is a member of the given storage group.

Parameters

- **device_id** – device id – str
- **storage_group_id** – storage group id – name

Returns
bool

modify_cu_image(*split_id: str*, *cu_ssid: str*, *assign_alias_dict=None*, *remove_alias_dict=None*, *map_start_address=None*, *map_volume_list=None*, *unmap_volume_list=None*)

Modify an existing cu image. :param split_id: split id – str :param cu_ssid: cu image ssid – str :param assign_alias_dict: alias range to be assigned – dict :param remove_alias_dict: alias range to be removed – dict :param map_start_address: :param map_volume_list: volumes to be mapped – list :param unmap_volume_list: volumes to be unmapped – list

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** – host name – str
- **host_flag_dict** – host flags – dict
- **remove_init_list** – initiators to be removed – list

- **add_init_list** – initiators to be added – list
- **new_name** – new host name – str

Returns
modified host details – dict

modify_host_group(*host_group_id*, *host_flag_dict=None*, *remove_host_list=None*, *add_host_list=None*, *new_name=None*)

Modify an existing host group.

Only one parameter can be modified at a time.

Parameters

- **host_group_id** – name of the host group – str
- **host_flag_dict** – host flags – dict
- **remove_host_list** – hosts to be removed – list
- **add_host_list** – hosts to be added – list
- **new_name** – new name of the host group – str

Returns
modified host group details – 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** – initiator id – str
- **remove_masking_entry** – 'true' or 'false' – str
- **replace_init** – new initiator id – str
- **rename_alias** – ('new node name', 'new port name') – tuple
- **set_fcid** – fcid – str
- **initiator_flags** – initiator flags to set – dict

Returns
modified initiator details – dict

modify_port_group(*port_group_id*, *remove_port=None*, *add_port=None*, *rename_port_group=None*)

Modify an existing port group.

Only one parameter can be modified at a time.

Parameters

- **port_group_id** – name of the port group – str
- **remove_port** – port details (director_id, port_id) – tuple
- **add_port** – port details (director_id, port_id) – tuple
- **rename_port_group** – new port group name – str

Returns
modified port group details – dict

modify_service_level(*service_level_id*, *new_name*)

Modify an SLO.

Currently, the only modification permitted is renaming.

Parameters

- **service_level_id** – current name of the service level – str
- **new_name** – new name for the – str

Returns
modified service level details – dict

modify_storage_group(*storage_group_id*, *payload*)

Modify a storage group.

Parameters

- **storage_group_id** – storage group id – str
- **payload** – request payload – dict

Returns
modified storage group details – dict

move_volumes_between_storage_groups(*device_ids*, *source_storagegroup_name*, *target_storagegroup_name*, *force=False*, *_async=False*)

Move volumes to a different storage group.

Requires force set to True if volume is in a masking view.

Parameters

- **device_ids** – volume device id(s) – str or list
- **source_storagegroup_name** – originating storage group name – str
- **target_storagegroup_name** – destination storage group name – str
- **force** – force flag – bool
- **_async** – if call should be async – bool

Returns
storage group details – dict

remove_child_storage_group_from_parent_group(*child_storage_group*, *parent_storage_group*)

Remove a storage group from its parent storage group.

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

Parameters

- **child_storage_group** – child storage group id – str
- **parent_storage_group** – parent storage group id – str

Returns
storage group details – dict

remove_volume_from_storage_group(*storage_group_id*, *vol_id*, *_async=False*, *remote_array_1_id=None*, *remote_array_1_sgs=None*, *remote_array_2_id=None*, *remote_array_2_sgs=None*, *terminate_snapshots=False*)

Remove a volume from a given storage group.

Parameters

- **storage_group_id** – storage group id – str
- **vol_id** – device id – str
- **_async** – if call should be async – bool
- **remote_array_1_id** – 12 digit serial number of remote array, optional – str
- **remote_array_1_sgs** – list of storage groups on remote array to add Remote device, Unisphere instance must be local to R1 storage group otherwise volumes will only be added to the local group – str or list
- **remote_array_2_id** – optional digit serial number of remote array, only used in multihop SRDF, e.g. R11, or R1 - R21 - R2 optional – str
- **remote_array_2_sgs** – storage groups on remote array, optional – str or list
- **terminate_snapshots** – terminate any snapshots on volume when removing from storage group – bool

Returns
storage group details – 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** – current name of the masking view – str
- **new_name** – new name of the masking view – str

Returns
modified masking view details – dict

rename_volume(*device_id*, *new_name*, *append_vol_id=False*)

Rename a volume.

Parameters

- **device_id** – device id – str
- **new_name** – new name for the volume – str
- **append_vol_id** – append volume id to the volume name, optional – bool

set_host_io_limit_iops_or_mbps(*storage_group*, *iops*, *dynamic_distribution*, *mbps=None*)

Set the Host IO Limits on an existing storage group.

Parameters

- **storage_group** – storage group id – str
- **iops** – IO per second, min Value 100, must be specified as multiple of 100 – int
- **dynamic_distribution** – 'Always', 'Never', 'OnFailure' – str
- **mbps** – MB per second, min Value 100, must be specified as multiple of 100 – int

Returns
storage group details – dict

update_storage_group_qos(*storage_group_id*, *qos_specs*)

Update the storage group instance 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 Storage Group. If so, then DistributionType can be modified. Example qos specs: {'maxIOPS': '4000', 'maxMBPS': '4000', 'DistributionType': 'Dynamic'}

Parameters

- **storage_group_id** – storage group id – str
- **qos_specs** – qos specifications – dict

Returns
storage group details – dict

11.7 PyU4V.real_time

real_time.py.

class PyU4V.real_time.**RealTimeFunctions**(*array_id*, *rest_client*)

Bases: object

PerformanceFunctions.

static format_metrics(*metrics*)

Format metrics input for inclusion in REST request.

Take metric parameters and format them correctly to be used in REST request body. Valid input types are string and list.

Parameters
metrics – metric(s) – str or list

Returns
metrics – list

Raises
InvalidInputException

get_array_keys()

Get array IDs which are registered for real-time data.

Returns
array IDs – list

get_array_metrics()

Get array real-time performance metrics.

Returns
metrics – list

get_array_stats(*start_date*, *end_date*, *metrics*, *array_id=None*)

List real-time data for specified array.

Parameters

- **start_date** – timestamp in milliseconds since epoch – int
- **end_date** – timestamp in milliseconds since epoch – int

- **metrics** – performance metrics, options are individual metrics, a list of metrics, or 'ALL' for all metrics – str/list
- **array_id** – array serial number – str

Returns
real-time performance data – dict

get_backend_director_keys(*array_id=None*)

Get backend director IDs which are registered for real-time data.

Parameters
array_id – array serial number – str

Returns
backend director IDs – list

get_backend_director_metrics()

Get backend director real-time performance metrics.

Returns
metrics – list

get_backend_director_stats(*start_date*, *end_date*, *metrics*, *instance_id*, *array_id=None*)

List real-time data for specified backend director.

Parameters

- **start_date** – timestamp in milliseconds since epoch – int
- **end_date** – timestamp in milliseconds since epoch – int
- **metrics** – performance metrics, options are individual metrics, a list of metrics, or 'ALL' for all metrics – str/list
- **instance_id** – backend director id – str
- **array_id** – array serial number – str

Returns
real-time performance data – dict

get_backend_port_keys(*array_id=None*)

Get backend dir/port IDs which are registered for real-time data.

Parameters
array_id – array serial number – str

Returns
backend port IDs – list

get_backend_port_metrics()

Get backend port real-time performance metrics.

Returns
metrics – list

get_backend_port_stats(*start_date*, *end_date*, *metrics*, *instance_id*, *array_id=None*)

List real-time data for specified backend port.

Parameters

- **start_date** – timestamp in milliseconds since epoch – int
- **end_date** – timestamp in milliseconds since epoch – int

- **metrics** – performance metrics, options are individual metrics, a list of metrics, or 'ALL' for all metrics – str/list
- **instance_id** – backend dir/port id – str
- **array_id** – array serial number – str

Returns
real-time performance data – dict

get_categories(*array_id=None*)

Get a list of real-time supported performance categories.

Parameters
array_id – array serial number – str

Returns
categories – list

get_category_keys(*category*, *array_id=None*)

Get category keys valid for real-time metrics collection.

Parameters

- **category** – real-time performance category – str
- **array_id** – array serial number – str

Returns
category keys – list

get_category_metrics(*category*, *array_id=None*)

Get metrics available for a real-time performance category.

Parameters

- **category** – real-time performance category – str
- **array_id** – array serial number – str

Returns
metrics – list

get_external_director_keys(*array_id=None*)

Get external director IDs which are registered for real-time data.

Parameters
array_id – array serial number – str

Returns
external director IDs – list

get_external_director_metrics()

Get external director real-time performance metrics.

Returns
metrics – list

get_external_director_stats(*start_date*, *end_date*, *metrics*, *instance_id*, *array_id=None*)

List real-time data for specified external director.

Parameters

- **start_date** – timestamp in milliseconds since epoch – int

- **end_date** – timestamp in milliseconds since epoch – int
- **metrics** – performance metrics, options are individual metrics, a list of metrics, or 'ALL' for all metrics – str/list
- **instance_id** – external director id – str
- **array_id** – array serial number – str

Returns
real-time performance data – dict

get_frontend_director_keys(*array_id=None*)

Get frontend director IDs which are registered for real-time data.

Parameters
array_id – array serial number – str

Returns
frontend director IDs – list

get_frontend_director_metrics()

Get frontend director real-time performance metrics.

Returns
metrics – list

get_frontend_director_stats(*start_date*, *end_date*, *metrics*, *instance_id*, *array_id=None*)

List real-time data for specified frontend director.

Parameters

- **start_date** – timestamp in milliseconds since epoch – int
- **end_date** – timestamp in milliseconds since epoch – int
- **metrics** – performance metrics, options are individual metrics, a list of metrics, or 'ALL' for all metrics – str/list
- **instance_id** – backend director id – str
- **array_id** – array serial number – str

Returns
real-time performance data – dict

get_frontend_port_keys(*array_id=None*)

Get frontend dir/port IDs which are registered for real-time data.

Parameters
array_id – array serial number – str

Returns
frontend port IDs – list

get_frontend_port_metrics()

Get frontend port real-time performance metrics.

Returns
metrics – list

get_frontend_port_stats(*start_date*, *end_date*, *metrics*, *instance_id*, *array_id=None*)

List real-time data for specified frontend port.

Parameters

- **start_date** – timestamp in milliseconds since epoch – int
- **end_date** – timestamp in milliseconds since epoch – int
- **metrics** – performance metrics, options are individual metrics, a list of metrics, or 'ALL' for all metrics – str/list
- **instance_id** – backend dir/port id – str
- **array_id** – array serial number – str

Returns
real-time performance data – dict

get_performance_data(*start_date*, *end_date*, *category*, *metrics*, *array_id=None*, *instance_id=None*)

Retrieve real-time performance statistics for a given category.

Parameters

- **start_date** – timestamp in milliseconds since epoch – int
- **end_date** – timestamp in milliseconds since epoch – int
- **category** – category id – str
- **metrics** – performance metrics, options are individual metrics, a list of metrics, or 'ALL' for all metrics – str/list
- **array_id** – array serial number – str
- **instance_id** – instance id – str

Returns
real-time performance data – dict

get_rdf_director_keys(*array_id=None*)

Get rdf director IDs which are registered for real-time data.

Parameters
array_id – array serial number – str

Returns
rdf director IDs – list

get_rdf_director_metrics()

Get rdf director real-time performance metrics.

Returns
metrics – list

get_rdf_director_stats(*start_date*, *end_date*, *metrics*, *instance_id*, *array_id=None*)

List real-time data for specified backend director.

Parameters

- **start_date** – timestamp in milliseconds since epoch – int
- **end_date** – timestamp in milliseconds since epoch – int
- **metrics** – performance metrics, options are individual metrics, a list of metrics, or 'ALL' for all metrics – str/list
- **instance_id** – rdf director id – str
- **array_id** – array serial number – str

Returns
real-time performance data – dict

`get_rdf_port_keys`(*array_id=None*)

Get rdf dir/port IDs which are registered for real-time data.

Parameters
array_id – array serial number – str

Returns
rdf port IDs – list

`get_rdf_port_metrics`()

Get rdf port real-time performance metrics.

Returns
metrics – list

`get_rdf_port_stats`(*start_date*, *end_date*, *metrics*, *instance_id*, *array_id=None*)

List real-time data for specified rdf port.

Parameters

- **start_date** – timestamp in milliseconds since epoch – int
- **end_date** – timestamp in milliseconds since epoch – int
- **metrics** – performance metrics, options are individual metrics, a list of metrics, or 'ALL' for all metrics – str/list
- **instance_id** – rdf dir/port id – str
- **array_id** – array serial number – str

Returns
real-time performance data – dict

`get_storage_group_keys`(*array_id=None*)

Get storage group IDs which are registered for real-time data.

Parameters
array_id – array serial number – str

Returns
backend director IDs – list

`get_storage_group_metrics`()

Get storage group real-time performance metrics.

Returns
metrics – list

`get_storage_group_stats`(*start_date*, *end_date*, *metrics*, *instance_id*, *array_id=None*)

List real-time data for specified storage group.

Parameters

- **start_date** – timestamp in milliseconds since epoch – int
- **end_date** – timestamp in milliseconds since epoch – int
- **metrics** – performance metrics, options are individual metrics, a list of metrics, or 'ALL' for all metrics – str/list
- **instance_id** – storage group id – str

- **array_id** – array serial number – str

Returns
real-time performance data – dict

`get_timestamps`(*array_id=None*)

Get real-time performance timestamps for array(s).

Parameters
array_id – array serial number – str

Returns
array timestamp info – list

`is_timestamp_current`(*timestamp*, *minutes=None*)

Check if the timestamp is less than a user specified set of minutes.

If no minutes value is provided, self.recency is used. Seven minutes is recommended to provide a small amount of time for the STP daemon to record the next set of metrics in five minute intervals.

Parameters

- **timestamp** – timestamp in milliseconds since epoch – int
- **minutes** – timestamp recency in minutes – int

Returns
if timestamp is less than recency value – bool

`set_array_id`(*array_id*)

Set the array id.

Parameters
array_id – array id – str

`set_recency`(*minutes*)

Set the recency value in minutes.

Parameters
minutes – recency minutes – int

11.8 PyU4V.replication

replication.py.

`class PyU4V.replication.ReplicationFunctions`(*array_id*, *rest_client*)

Bases: `object`

ReplicationFunctions.

`are_volumes_rdf_paired`(*remote_array*, *device_id*, *target_device*, *rdf_group*)

Check if a pair of volumes are RDF paired.

Parameters

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

Returns
paired, state – bool, string

bulk_terminate_snapshots(*storage_group_id*, *snap_name*, *keep_count=None*, *terminate_all_snapshots=False*, *snapset_id_and_older=None*, *force=False*, *array_id=None*)

Terminate Multiple snapshots in a bulk operation.

Parameters

- **storage_group_id** – Name of the storage group with snapshots –str
- **snap_name** – optional name of snapshot to be terminated – str
- **terminate_all_snapshots** – Value that terminates all snapshots – bool
- **keep_count** – Value that terminates a number of snapshots so that there is only the specified number of most recent snapshots retained – int
- **snapset_id_and_older** – Value that terminates all snapshots with the snapset id or older –str
- **force** – Value that sets the force flag – bool
- **array_id** – 12 digit array id – int

create_rdf_group(*local_director_port_list*, *remote_array_id*, *label*, *local_rdfg_number*, *remote_rdfg_number*, *remote_director_port_list*, *array_id=None*)

Create a new RDF group between directors and ports on two PowerMax.

If this is the first connection between 2 arrays please ensure that you run get_rdf_remote_port_details and create the initial connection group using only the first source port you can extract details for one or more remote ports for the desired target array and feed into this function. Additional Ports can be added with modify_rdf_group function.

Parameters

- **local_director_port_list** – list of local directors and ports for group e.g [RF-1E:1, RF-2E:1] – list
- **remote_array_id** – 12 digit serial number of remote array – str
- **label** – label for group up to 10 characters – str
- **local_rdfg_number** – rdfg for the local array – int
- **remote_rdfg_number** – rdfg for the remote array – int
- **remote_director_port_list** – list of remote directors and ports to group e.g [RF-1E:1, RF-2E:1] – list
- **array_id** – 12 digit serial number of Source (R1) array, if no array is specified the array in config file or api connection will be default – str

create_storage_group_from_rdfg(*storage_group_name*, *srdf_group_number*, *array_id=None*, *rdf_type=None*, *remote_storage_group_name=None*)

Creates management storage group from all devices in SRDF group.

Note SRDF management storage group will be created without a service level, it is assumed that this group is created solely for the purpose of managing SRDF device and replication state, devices in an SRDF group may span multiple applications and storage groups.

Parameters

- **storage_group_name** – Name of storage group – str
- **srdf_group_number** – number of RDF group volumes are in – int
- **array_id** – number of RDF group volumes are in – int
- **rdf_type** – The SRDF type of the volumes in the SRDF group to be added to the Storage Group. Only needs to be populated if the SRDF group contains both RDF1 and RDF2 volumes valid values RDF1 or RDF2 – str
- **remote_storage_group_name** – Name of remote storage group – str

create_storage_group_snapshot(*storage_group_id*, *snap_name*, *ttl=None*, *hours=False*, *secure=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

- **storage_group_id** – source storage group id – str
- **snap_name** – snapshot name – str
- **ttl** – Time To Live – str
- **hours** – if TTL is in hours instead of days – bool
- **secure** – sets secure snapshot, snapshot created with secure option can not be deleted before ttl expires – bool

Returns
snapshot details – dict

create_storage_group_srdf_pairings(*storage_group_id*, *remote_sid*, *srdf_mode*, *establish=None*, *_async=False*, *rdfg_number=None*, *force_new_rdf_group=False*)

SRDF protect a storage group.

Valid modes are 'Active', 'AdaptiveCopyDisk', 'Synchronous', and 'Asynchronous'.

Parameters

- **storage_group_id** – storage group id – str
- **remote_sid** – remote array id – str
- **srdf_mode** – replication mode – str
- **establish** – establish srdf – bool
- **_async** – if call should be async – bool
- **rdfg_number** – rdf group number – int
- **force_new_rdf_group** – if force command should be applied – bool

Returns
storage group rdf details – dict

delete_rdf_group(*srdf_group_number*, *array_id=None*)

Function to Delete SRDF groups between VMAX or PowerMax arrays.

Parameters

- **srdf_group_number** – number of RDF group to be deleted – int

- **array_id** – 12 digit serial of array – str

delete_storage_group_snapshot(*storage_group_id*, *snap_name*, *gen=0*)

Delete the snapshot of a storage group.

This is for arrays with microcode less than 5978.669.669. Please use delete_storage_group_snapshot_by_snap_id for microcode greater than 5978.669.669.

Parameters

- **storage_group_id** – storage group id – str
- **snap_name** – snapshot name – str
- **gen** – snapshot generation number – int

delete_storage_group_snapshot_by_snap_id(*storage_group_id*, *snap_name*, *snap_id*, *force=False*, *array_id=None*, *symforce=False*)

Delete the snapshot of a storage group using the snap id.

Snap ids are only available on microcode 5978.669.669 and greater.

Parameters

- **storage_group_id** – storage group id – str
- **snap_name** – snapshot name – str
- **snap_id** – snapshot snap id – int
- **force** – sets force flag – bool
- **array_id** – 12 digit array id – int
- **symforce** – sets symforce flag – bool

Returns
dict

delete_storage_group_srdf(*storage_group_id*, *srdf_group_number*, *filters=None*)

Delete srdf pairings for a given storage group.

Parameters

- **storage_group_id** – storage group id – str
- **srdf_group_number** – srdf group number – int
- **filters** – optional boolean filters, half_delete,hop2, force, symforce, star, bypass, keepR1, keepR2, usage example filters={'keepR1': 'true'} – dict

Returns
storage group rdf details – dict

establish_storage_group_srdf(*storage_group_id*, *srdf_group_number*, *establish_options=None*, *_async=False*)

Establish io on the links for the given storage group.

Optional boolean parameters to set are 'bypass', 'metroBias', 'star', 'hop2', 'force', 'symForce', 'full'.

Parameters

- **storage_group_id** – storage group id – str
- **srdf_group_number** – srdf group number – int
- **establish_options** – establish parameters – dict

- **_async** – if call should be async – bool

Returns
storage group rdf details – dict

failback_storage_group_srdf(*storage_group_id*, *srdf_group_number*, *failback_options=None*, *_async=False*)

Failback a given storage group.

Optional boolean parameters to set are 'bypass', 'recoverPoint', 'star', 'hop2', 'force', 'symForce', 'remote'.

Parameters

- **storage_group_id** – storage group id – str
- **srdf_group_number** – srdf group number – int
- **failback_options** – failback parameters – dict
- **_async** – if call should be async – bool

Returns
storage group rdf details – dict

failover_storage_group_srdf(*storage_group_id*, *srdf_group_number*, *failover_options=None*, *_async=False*)

Failover a given storage group.

Optional boolean parameters to set are 'bypass', 'star', 'restore', 'immediate', 'hop2', 'force', 'symForce', 'remote', 'establish'.

Parameters

- **storage_group_id** – storage group id – str
- **srdf_group_number** – srdf group number – int
- **failover_options** – failover parameters – dict
- **_async** – if call should be async – bool

Returns
storage group rdf details – dict

find_expired_snapvx_snapshots()

Find all expired snapvx snapshots.

This is for arrays with microcode less than 5978.669.669. Please use find_expired_snapvx_snapshots_by_snap_ids for microcode greater than 5978.669.669.

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.

Returns
expired snapshot details – list

find_expired_snapvx_snapshots_by_snap_ids()

Find all expired snapvx snapshots using snap id.

Snap ids are only available on microcode 5978.669.669 and greater.

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.

Returns
expired snapshot details – list

`get_array_replication_capabilities`(*array_id=None*)

Check what replication facilities are available.

Returns
replication capability details – dict

`get_rdf_director_detail`(*director_id*, *array_id=None*)

Retrieves details for specified RDF_director.

Parameters

- **director_id** – identifier for director e.g. RF-1F – str
- **array_id** – 12 digit serial number for PowerMax array – str

Returns
director details –dict

`get_rdf_director_list`(*array_id=None*, *filters=None*)

Finds out directors configured for SRDF on the specified array.

Parameters

- **array_id** – 12 digit serial number for PowerMax array – str
- **filters** – optional filters - dict

Returns
list of directors – list

`get_rdf_director_port_details`(*director_id*, *port_id*, *array_id=None*)

Retrieves details of specified RDF ports.

Parameters

- **director_id** – identifier for director e.g. RF-1F – str
- **port_id** – port number – int
- **array_id** – 12 digit serial number for PowerMax array – str

Returns
port details – dict

`get_rdf_director_port_list`(*director_id*, *array_id=None*, *filters=None*)

Retrieves list of ports available on RDF_director.

Parameters

- **director_id** – identifier for director e.g. RF-1F – str
- **array_id** – 12 digit serial number for PowerMax array – str
- **filters** – optional filters – dict

Returns
list of RDF ports – list

`get_rdf_group`(*rdf_number*, *array_id=None*)

Get specific rdf group details.

Parameters

- **rdf_number** – rdf group number – int
- **array_id** – array serial number – str

Returns
rdf group details – dict

get_rdf_group_list(*array_id=None*, *remote_symmetrix_id=None*, *rdf_mode=None*, *volume_count=None*, *group_type=None*)

Get rdf group list from array.

Parameters

- **array_id** – local array serial number – str
- **remote_symmetrix_id** – Optional value that filters returned list to display only SRDF Groups between the local array and specified remote array, 12 Digit array_id e.g. 000297900330 – str
- **rdf_mode** – Optional value that filters returned list to display only SRDF Groups of a specified RDF mode, Synchronous, Asynchronous, Active, or AdaptiveCopy – str
- **volume_count** – Optional value that filters returned list to display only SRDF Groups that have a specified volume count – int
- **group_type** – Optional value that filters returned list to display only SRDF Groups of a specified group type valid options are Witness, Global_Mirror, Data_Migration, VVOL_ASYNC, MetroDR_Metro, MetroDR_DR, Metro, Star_Normal, Star_Recovery, Sqar_Normal, Sqar_Recovery, PPRC, FILE_SYNC, FILE_ASYNC, Dynamic – str

Returns
rdf group list – list

get_rdf_group_number(*rdf_group_label*)

Given a group label, return the associated group number.

Parameters
rdf_group_label – rdf group label – str

Returns
rdf group number – int

get_rdf_group_volume(*rdf_number*, *device_id*)

Get specific volume details, from an RDF group.

Parameters

- **rdf_number** – rdf group number – int
- **device_id** – device id – str

Returns
rdf group volume details – dict

get_rdf_group_volume_list(*rdf_number*)

Get specific volume details, from an RDF group.

Parameters
rdf_number – rdf group number – int

Returns
device ids – list

get_rdf_port_remote_connections(*director_id*, *port_id*, *array_id=None*)

Performs a scan of the RDF environment via specified port.

This function should be run on initial SRDF configuration once zoning or IP routing is configured, prior to first RDF group being configured.

Parameters

- **director_id** – identifier for director e.g. RF-1F – str
- **port_id** – port number – int
- **array_id** – 12 digit serial number for Source – str

Returns
remote port details – dict

get_replication_enabled_storage_groups(*array_id=None*, *has_srdf=None*, *has_snapshots=None*, *has_cloud_snapshots=None*, *is_link_target=None*, *has_snap_policies=None*, *has_clones=None*)

Return list of storage groups with replication.

Parameters

- **has_snapshots** – return only storage groups with snapshots
- **has_srdf** – return only storage groups with SRDF
- **has_cloud_snapshots** – Value that filters returned list to display Storage Groups that have Cloud Snapshots – bool
- **is_link_target** – Value that filters returned list to display Storage Groups that are Link Targets – bool
- **has_snap_policies** – Value that filters returned list to display Storage Groups that have Snapshot Policies – bool
- **has_clones** – Value that filters returned list to display Storage Groups that have Clones – bool

Returns
list of storage groups with associated replication

get_replication_info()

Return replication information for an array.

Returns
replication details – dict

get_snapshot_generation_details(*sg_id*, *snap_name*, *gen_num*)

Get the details for a particular snapshot generation.

This is for arrays with microcode less than 5978.669.669. Please use get_snapshot_snap_id_details for microcode greater than 5978.669.669.

Parameters

- **sg_id** – storage group id – str
- **snap_name** – snapshot name – str
- **gen_num** – generation number – int

Returns
snapshot generation details – dict

get_snapshot_snap_id_details(*sg_id*, *snap_name*, *snap_id*)

Get the details for a particular snapshot snap_id.

Snap ids are only available on microcode 5978.669.669 and greater.

Parameters

- **sg_id** – storage group id – str
- **snap_name** – snapshot name – str
- **snap_id** – unique snap id – int

Returns
snapshot snap id details – dict

get_storage_group_replication_details(*storage_group_id*, *return_remote_sg_info=False*, *exclude_sl_snaps=False*, *exclude_manual_snaps=False*)

Given a storage group id, return storage group srdf info and snapshot information.

Parameters

- **storage_group_id** – storage group id – str
- **return_remote_sg_info** – returns additional information for keys remote_storage_groups – bool
- **exclude_sl_snaps** – excludes details of any service level snaps from the return – bool
- **exclude_manual_snaps** – excludes details of any manual snaps from the return – bool

Returns
storage group replication details – dict

get_storage_group_snapshot_generation_list(*storage_group_id*, *snap_name*)

Get a snapshot and its generation count information for an sg.

This is for arrays with microcode less than 5978.669.669. Please use get_storage_group_snapshot_snap_id_list for microcode greater than 5978.669.669.

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

- **storage_group_id** – name of the storage group – str
- **snap_name** – the name of the snapshot – str

Returns
generation numbers – list

get_storage_group_snapshot_list(*storage_group_id*)

Get a list of snapshots associated with a storage group.

Parameters
storage_group_id – storage group id – str

Returns
snapshot ids – list

get_storage_group_snapshot_snap_id_list(*storage_group_id*, *snap_name*)

Get a snapshot and its snap id information for an sg.

Each snapid is unique. These have replaced generations starting at U4P9.2. Snap ids are only available on microcode 5978.669.669 and greater.

Parameters

- **storage_group_id** – name of the storage group – str
- **snap_name** – the name of the snapshot – str

Returns
snapids – list

get_storage_group_srdf_details(*storage_group_id*, *rdfg_num*)

Get the details for an rdf group on a particular storage group.

Parameters

- **storage_group_id** – replicated storage group id – str
- **rdfg_num** – rdf group number – int

Returns
storage group rdf details – dict

get_storage_group_srdf_group_list(*storage_group_id*, *array_id=None*)

Get the rdf group numbers for a storage group.

Parameters

- **storage_group_id** – replicated storage group id – str
- **array_id** – array serial number – str

Returns
rdf group numbers – list

is_snapvx_licensed()

Check if the snapVx feature is licensed and enabled.

Returns
bool

is_volume_in_replication_session(*device_id*)

Check if a volume is in a replication session.

NOTE: There may be a time delay between the snapshot creation and the object model update. See the following test for usage: tests.ci_tests.test_pyu4v_ci_replication.CITestReplication.test_is_volume_in_replication_session

Parameters
device_id – device id – str

Returns
snap vx target, snap vx source, rdf group – bool, bool, list

link_gen_snapshot(*sg_id*, *snap_name*, *link_sg_name*, *_async=False*, *gen_num=0*)

Link a snapshot to another storage group.

This is for arrays with microcode less than 5978.669.669. Please use link_snapshot_by_snap_id for microcode greater than 5978.669.669.

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

Parameters

- **sg_id** – storage group id – str
- **snap_name** – snapshot name – str
- **link_sg_name** – target storage group name – str
- **_async** – if call should be async – bool
- **gen_num** – snapshot generation number – int

Returns
snapshot details – dict

link_snapshot_by_snap_id(*sg_id*, *link_sg_name*, *snap_name*, *snap_id*, *_async=False*)

Link a snapshot to another storage group using it's snap id.

Snap ids are only available on microcode 5978.669.669 and greater.

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

Parameters

- **sg_id** – storage group id – str
- **link_sg_name** – target storage group name – str
- **snap_name** – snapshot name – str
- **snap_id** – snapshot snap id – int
- **_async** – if call should be async – bool

Returns
snapshot details – dict

modify_rdf_group(*action*, *srdf_group_number*, *array_id=None*, *port_list=None*, *label=None*, *dev_list=None*, *target_rdf_group=None*, *consistency_exempt=None*)

Function to Modify Ports, devices or change label of RDF group.

Function can be used to Add ports, move volumes between rdf groups, remove ports or rename RDF group.

Parameters

- **action** – add_ports, remove_ports, move – str
- **srdf_group_number** – srdf group number for action on local array: int
- **array_id** – 12 digit serial of array – str
- **port_list** – list of ports to be added or removed e.g. [RF-1E:10, RF-2E:10] –list
- **label** – label for group up to 10 characters – str
- **dev_list** – list of volumes to be moved between RDF groups – list
- **target_rdf_group** – rdfg group to move volumes to – int
- **consistency_exempt** – ignore device for consistency checks – bool

modify_storage_group_snapshot(*src_storage_grp_id*, *tgt_storage_grp_id*, *snap_name*, *link=False*, *unlink=False*, *restore=False*, *new_name=None*, *gen_num=0*, *_async=False*, *relink=False*)

Modify a storage group snapshot.

This is for arrays with microcode less than 5978.669.669. Please use modify_storage_group_snapshot_by_snap_id for microcode greater than 5978.669.669.

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

Parameters

- **src_storage_grp_id** – name of the storage group – str
- **tgt_storage_grp_id** – target sg id (Can be None) – str
- **snap_name** – snapshot name – str
- **link** – link action required – bool
- **unlink** – unlink action required – bool
- **restore** – restore action required – bool
- **new_name** – new name for the snapshot – str
- **gen_num** – generation number – int
- **_async** – if call should be async – bool
- **relink** – relink action required – bool

Returns

modified storage group snapshot details – dict

modify_storage_group_snapshot_by_snap_id(*src_storage_grp_id*, *tgt_storage_grp_id*, *snap_name*, *snap_id*, *link=False*, *unlink=False*, *restore=False*, *new_name=None*, *_async=False*, *relink=False*, *remote=False*, *force=False*)

Modify a storage group snapshot using it's snap id.

Snap ids are only available on microcode 5978.669.669 and greater.

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

Parameters

- **src_storage_grp_id** – name of the storage group – str
- **tgt_storage_grp_id** – target sg id (Can be None) – str
- **snap_name** – snapshot name – str
- **snap_id** – snap id – int
- **link** – link action required – bool
- **unlink** – unlink action required – bool
- **restore** – restore action required – bool
- **new_name** – new name for the snapshot – str
- **_async** – if call should be async – bool
- **relink** – relink action required – bool
- **remote** – used when linking/relinking to established R1 Storage group,using this feature acknowledges that the data from snapshot will be transmitted to R2 site,removes the need to suspend/establish SRDF as part of link operation.Caution should be applied – bool
- **force** – forces operation to succeed that would normally fail.Example terminate snapshots in SG where devices have been added,using symforce will bypass the device count check.Should be used with caution. – bool

Returns
modified storage group snapshot details – dict

modify_storage_group_srdf(*storage_group_id*, *action*, *srdf_group_number*, *options=None*, *_async=False*)

Modify the state of a rdf group.

This may be a long running task depending on the size of the SRDF group, can switch to async call if required. Available actions are 'Establish', 'EnableConsistency', 'DisableConsistency', 'Split', 'Suspend', 'Restore', 'Resume', 'Failover', 'Failback', 'Swap', 'SetBias', and 'SetMode'.

Parameters

- **storage_group_id** – storage group id – str
- **action** – the rdf action, note enable/disable consistency feature requires Unisphere 9.2.1.6 or higher – str
- **srdf_group_number** – srdf group number – int
- **options** – srdf options e.g. {setMode': {'mode': 'Asynchronous'}} – dict
- **_async** – if call should be async – bool

Returns
storage group rdf details – dict

rename_snapshot(*sg_id*, *snap_name*, *new_name*, *gen_num=0*)

Rename an existing storage group snapshot.

This is for arrays with microcode less than 5978.669.669. Please use rename_snapshot_by_snap_id for microcode greater than 5978.669.669.

Parameters

- **sg_id** – storage group id – str
- **snap_name** – snapshot name – str
- **new_name** – new snapshot name – str
- **gen_num** – snapshot generation number – int

Returns
snapshot details – dict

rename_snapshot_by_snap_id(*sg_id*, *snap_name*, *new_name*, *snap_id*)

Rename an existing storage group snapshot using it's snap id.

Snap ids are only available on microcode 5978.669.669 and greater.

Parameters

- **sg_id** – storage group id – str
- **snap_name** – snapshot name – str
- **new_name** – new snapshot name – str
- **snap_id** – snapshot snap id – int

Returns
snapshot details – dict

restore_snapshot(*sg_id*, *snap_name*, *gen_num=0*)

Restore a storage group to its snapshot.

This is for arrays with microcode less than 5978.669.669. Please use restore_snapshot_by_snap_id for microcode greater than 5978.669.669.

Parameters

- **sg_id** – storage group id – str
- **snap_name** – snapshot name – str
- **gen_num** – snapshot generation number – int

Returns
snapshot details – dict

restore_snapshot_by_snap_id(*sg_id*, *snap_name*, *snap_id*)

Restore a storage group to its snapshot using it's snap id.

Snap ids are only available on microcode 5978.669.669 and greater.

Parameters

- **sg_id** – storage group id – str
- **snap_name** – snapshot name – str
- **snap_id** – snapshot snap id – int

Returns
snapshot details – dict

suspend_storage_group_srdf(*storage_group_id*, *srdf_group_number*, *suspend_options=None*, *_async=False*)

Suspend IO on the links for the given storage group.

Optional boolean parameters to set are "bypass", "metroBias", "star", "immediate", "hop2", "consExempt", "force", "symForce".

Parameters

- **storage_group_id** – storage group id – str
- **srdf_group_number** – srdf group number – int
- **suspend_options** – suspend parameters – dict
- **_async** – if call should be async – bool

Returns
storage group rdf details – dict

unlink_gen_snapshot(*sg_id*, *snap_name*, *unlink_sg_name*, *_async=False*, *gen_num=0*)

Unlink a snapshot from another storage group.

This is for arrays with microcode less than 5978.669.669. Please use unlink_snapshot_by_snap_id for microcode greater than 5978.669.669.

Parameters

- **sg_id** – storage group id – str
- **snap_name** – snapshot name – str
- **unlink_sg_name** – target storage group name – str

- **_async** – if call should be async – bool
- **gen_num** – snapshot generation number – int

Returns
snapshot details – dict

unlink_snapshot_by_snap_id(*sg_id*, *unlink_sg_name*, *snap_name*, *snap_id*, *_async=False*)

Unlink a snapshot from another storage group using it's snap id.

Snap ids are only available on microcode 5978.669.669 and greater.

Parameters

- **sg_id** – storage group id – str
- **unlink_sg_name** – target storage group name – str
- **snap_name** – snapshot name – str
- **snap_id** – snapshot snap id – int
- **_async** – if call should be async – bool

Returns
snapshot details – dict

11.9 PyU4V.rest_requests

rest_requests.py.

class PyU4V.rest_requests.**RestRequests**(*username*, *password*, *verify*, *base_url*, *interval*, *retries*, *application_type=None*, *proxies=None*)

Bases: object

RestRequests.

close_session()

Close the current session.

establish_rest_session(*headers=None*)

Establish a REST session.

Returns
session – object

file_transfer_request(*method*, *uri*, *timeout=None*, *download=False*, *r_obj=None*, *upload=False*, *form_data=None*)

Send a file transfer request via REST to the target API.

Valid methods are 'POST' and 'PUT'.

Parameters

- **method** – request method – str
- **uri** – target uri – str
- **timeout** – optional timeout override – int
- **download** – if download request – bool
- **r_obj** – download request payload – dict

- **upload** – if upload request – bool
- **form_data** – upload multipart form data – dict

Returns
server response, status code – dict, int

Raises
InvalidInputException, VolumeBackendAPIException, Timeout, SSLError, ConnectionError, HTTPError

rest_request(*target_url*, *method*, *params=None*, *request_object=None*, *timeout=None*)

Send a request to the target api.

Valid methods are 'GET', 'POST', 'PUT', 'DELETE'.

Parameters

- **target_url** – target url –str
- **method** – method – str
- **params** – Additional URL parameters – dict
- **request_object** – request payload – dict
- **timeout** – optional timeout override – int

Returns
server response, status code – dict, int

11.10 PyU4V.snapshot_policy

snapshot_policy.py.

class PyU4V.snapshot_policy.**SnapshotPolicyFunctions**(*array_id*, *rest_client*)

Bases: object

associate_to_storage_groups(*snapshot_policy_name*, *storage_group_names*, *_async=False*)

Associate a snapshot policy to storage group(s).

Parameters

- **snapshot_policy_name** – the snapshot policy name – str
- **storage_group_names** – List of storage group names – list
- **_async** – is the operation asynchronous – bool

Returns
resource object – dict

create_snapshot_policy(*snapshot_policy_name*, *interval*, *cloud_retention_days=None*, *cloud_provider_name=None*, *local_snapshot_policy_secure=False*, *local_snapshot_policy_snapshot_count=None*, *offset_mins=None*, *compliance_count_warning=None*, *compliance_count_critical=None*, *_async=False*)

Create a new snapshot policy.

Parameters

- **snapshot_policy_name** – the snapshot policy name – str

- **interval** – The value of the interval counter for snapshot policy execution. Must be one of '10 Minutes', '12 Minutes', '15 Minutes', '20 Minutes', '30 Minutes', '1 Hour', '2 Hours', '3 Hours', '4 Hours', '6 Hours', '8 Hours', '12 Hours', '1 Day', '7 Days' – str
- **cloud_retention_days** – part of cloud_snapshot_policy_details number of days to retain the policy – int
- **cloud_provider_name** – part of cloud_snapshot_policy_details the cloud provider name – str
- **local_snapshot_policy_secure** – secure snapshots may only be terminated after they expire or by Dell EMC support – bool
- **local_snapshot_policy_snapshot_count** – the max snapshot count of the policy – int
- **offset_mins** – Defines when, within the interval the snapshots will be taken for a specified Snapshot Policy. The offset must be less than the interval of the Snapshot Policy. For daily snapshots the offset is the number of minutes after midnight UTC, for weekly the offset is from midnight UTC on the Sunday. The format must be in minutes – int
- **compliance_count_warning** – The Number of snapshots which are not failed or bad when compliance changes to warning. – int
- **compliance_count_critical** – The Number of snapshots which are not failed or bad when compliance changes to critical. – int
- **_async** – is the operation asynchronous – bool

Returns
resource object – dict

delete_snapshot_policy(*snapshot_policy_name*)

Delete a snapshot policy

Parameters
snapshot_policy_name – the snapshot policy name – str

disassociate_from_storage_groups(*snapshot_policy_name*, *storage_group_names*, *_async=False*)

Disassociate a snapshot policy from storage group(s).

Parameters

- **snapshot_policy_name** – the snapshot policy name – str
- **storage_group_names** – List of storage group names – list
- **_async** – is the operation asynchronous – bool

Returns
resource object – dict

get_snapshot_policy(*snapshot_policy_name*)

Given a snapshot policy name, return snapshot policy details.

Parameters
snapshot_policy_name – name of the snapshot policy – str

Returns
snapshot policy details – dict

get_snapshot_policy_compliance(*storage_group_name*, *last_week=False*, *last_four_weeks=False*, *from_epoch=None*, *to_epoch=None*, *from_time_string=None*, *to_time_string=None*)

Get compliance attributes on a storage group.

Parameters

- **storage_group_name** – storage group name
- **last_week** – compliance in last week – bool
- **last_four_weeks** – compliance in last four weeks – bool
- **from_epoch** – timestamp since epoch – str e.g 1606820929 (seconds)
- **to_epoch** – timestamp since epoch – str e.g 1606820929 (seconds)
- **from_time_string** – human readable date – str e.g 2020-12-01 15:00
- **to_time_string** – human readable date – str e.g 2020-12-01 15:00

Returns
resource – dict

get_snapshot_policy_compliance_epoch(*storage_group_name*, *from_epoch=None*, *to_epoch=None*)

Get compliance attributes for the last four weeks.

Get compliance attributes on a storage group for the last four weeks

Parameters

- **storage_group_name** – storage group name
- **from_epoch** – timestamp since epoch – str e.g 1606820929 (seconds)
- **to_epoch** – timestamp since epoch – str e.g 1606820929 (seconds)

Returns
resource – dict

get_snapshot_policy_compliance_human_readable_time(*storage_group_name*, *from_time_string=None*, *to_time_string=None*)

Get compliance attributes for the last four weeks.

Get compliance attributes on a storage group for the last four weeks

Parameters

- **storage_group_name** – storage group name
- **from_time_string** – human readable date – str e.g 2020-12-01 15:00
- **to_time_string** – human readable date – str e.g 2020-12-01 15:00

Returns
resource – dict

get_snapshot_policy_compliance_last_four_weeks(*storage_group_name*)

Get compliance attributes for the last four weeks.

Get compliance attributes on a storage group for the last four weeks

Parameters
storage_group_name – storage group name

Returns
resource – dict

`get_snapshot_policy_compliance_last_week`(*storage_group_name*)

Get compliance attributes on a storage group for the last week.

Parameters
storage_group_name – storage group name

Returns
resource – dict

`get_snapshot_policy_list`()

Given a snapshot policy name, return snapshot policy details.

Returns
snapshot policy names – list

`get_snapshot_policy_storage_group_list`(*snapshot_policy_name*)

Get list of storage groups associated to specified snapshot policy.

Parameters
snapshot_policy_name – name of the snapshot policy – str

Returns
snapshot policy details – list

`modify_snapshot_policy`(*snapshot_policy_name*, *action*, *interval=None*, *offset_mins=None*, *snapshot_count=None*, *compliance_count_warning=None*, *compliance_count_critical=None*, *storage_group_names=None*, *new_snapshot_policy_name=None*, *_async=False*)

Modify a snapshot policy

This can be action: [Modify, Suspend, Resume, AssociateToStorageGroups, DisassociateFromStorageGroups]. A modify of the snapshot policy or adding or removing storage groups associated with the policy.

Parameters

- **snapshot_policy_name** – the snapshot policy name – str
- **action** – the modification action, must be one of 'AssociateToStorageGroups', 'DisassociateFromStorageGroups' 'Modify', 'Suspend', 'Resume' – str
- **interval** – The value of the interval counter for snapshot policy execution. Must be one of '10 Minutes', '12 Minutes', '15 Minutes', '20 Minutes', '30 Minutes', '1 Hour', '2 Hours', '3 Hours', '4 Hours', '6 Hours', '8 Hours', '12 Hours', '1 Day', '7 Days' – str
- **offset_mins** – The number of minutes after 00:00 on Monday to first run the service policy. The offset must be less than the interval of the Snapshot Policy. The format must be in minutes – int
- **snapshot_count** – The maximum number of snapshots that should be maintained for a specified Snapshot Policy. The maximum count must be between 1 to 1024. – int
- **compliance_count_warning** – The Number of snapshots which are not failed or bad when compliance changes to warning. The warning compliance count cannot be set to 0 and must be less than or equal to the maximum count of the Snapshot Policy. – int
- **compliance_count_critical** – The Number of snapshots which are not failed or bad when compliance changes to critical. If the warning compliance count is also set, the critical compliance count must be less than or equal to that. – int

- **storage_group_names** – List of storage group names – list
- **new_snapshot_policy_name** – change the name if set – str
- **_async** – is the operation asynchronous – bool

Returns
resource object – dict

modify_snapshot_policy_properties(*snapshot_policy_name*, *interval=None*, *offset_mins=None*, *snapshot_count=None*, *compliance_count_warning=None*, *compliance_count_critical=None*, *new_snapshot_policy_name=None*, *_async=False*)

Suspend a snapshot policy

Parameters

- **snapshot_policy_name** – the snapshot policy name – str
- **interval** – The value of the interval counter for snapshot policy execution. Must be one of '10 Minutes', '12 Minutes', '15 Minutes', '20 Minutes', '30 Minutes', '1 Hour', '2 Hours', '3 Hours', '4 Hours', '6 Hours', '8 Hours', '12 Hours', '1 Day', '7 Days' – str
- **offset_mins** – The number of minutes after 00:00 on Monday to first run the service policy. The offset must be less than the interval of the Snapshot Policy. The format must be in minutes – int
- **snapshot_count** – The maximum number of snapshots that should be maintained for a specified Snapshot Policy. The maximum count must be between 1 to 1024. – int
- **compliance_count_warning** – The Number of snapshots which are not failed or bad when compliance changes to warning. The warning compliance count cannot be set to 0 and must be less than or equal to the maximum count of the Snapshot Policy. – int
- **compliance_count_critical** – The Number of snapshots which are not failed or bad when compliance changes to critical. If the warning compliance count is also set, the critical compliance count must be less than or equal to that. – int
- **new_snapshot_policy_name** – change the name if set – str
- **_async** – is the operation asynchronous – bool

Returns
resource object – dict

resume_snapshot_policy(*snapshot_policy_name*, *_async=False*)

Suspend a snapshot policy

Parameters

- **snapshot_policy_name** – the snapshot policy name – str
- **_async** – is the operation asynchronous – bool

Returns
resource object – dict

suspend_snapshot_policy(*snapshot_policy_name*, *_async=False*)

Suspend a snapshot policy.

Parameters

- **snapshot_policy_name** – the snapshot policy name – str
- **_async** – is the operation asynchronous – bool

Returns
resource object – dict

static verify_combination(*last_week*, *last_four_weeks*, *from_epoch*, *from_time_string*)

Verify the valid combinations for compliance.

Parameters

- **last_week** – compliance in last week – bool
- **last_four_weeks** – compliance in last four weeks – bool
- **from_epoch** – timestamp since epoch – str e.g 1606820929 (seconds)
- **from_time_string** – human readable date – str e.g 2020-12-01 15:00

Returns
msg or None – str

verify_from_epoch(*from_epoch*, *to_epoch*, *to_time_string*)

Verify the the from_epoch param for compliance.

Parameters

- **from_epoch** – timestamp since epoch – str e.g 1606820929 (seconds)
- **to_epoch** – timestamp since epoch – str e.g 1606820929 (seconds)
- **to_time_string** – human readable date – str e.g 2020-12-01 15:00

Returns
msg or None – str query_params – dict

verify_from_time_string(*to_epoch*, *to_time_string*, *from_time_string*)

Verify the the from_time_string param for compliance.

Parameters

- **to_epoch** – timestamp since epoch – str e.g 1606820929 (seconds)
- **from_time_string** – human readable date – str e.g 2020-12-01 15:00
- **to_time_string** – human readable date – str e.g 2020-12-01 15:00

Returns
msg or None – str query_params – dict

verify_input_params(*last_week*, *last_four_weeks*, *from_epoch*, *to_epoch*, *from_time_string*, *to_time_string*)

Verify the input parameters for compliance.

Parameters

- **last_week** – compliance in last week – bool
- **last_four_weeks** – compliance in last four weeks – bool
- **from_epoch** – timestamp since epoch – str e.g 1606820929 (seconds)
- **to_epoch** – timestamp since epoch – str e.g 1606820929 (seconds)
- **from_time_string** – human readable date – str e.g 2020-12-01 15:00
- **to_time_string** – human readable date – str e.g 2020-12-01 15:00

Returns
msg or None – str query_params – dict

11.11 PyU4V.system

system.py.

class PyU4V.system.**SystemFunctions**(*array_id*, *rest_client*)

Bases: object

SystemFunctions.

acknowledge_alert(*alert_id*)

Acknowledges an alert.

Parameters
alert_id – alert_id uniquely identifying alert on Unisphere system –str

Returns
alert details – dict

change_local_user_password(*username*, *current_password*, *new_password*)

Function to allow users to change password for local user accounts.

Requires minimum version of Unisphere 9.2.1.x and API account must have security admin role Change local user password. :param username: username for local account – str :param current_password: existing password for local account – str :param new_password: new password for local account – str

delete_alert(*alert_id*)

Deletes Specified Alert.

Parameters
alert_id – alert_id uniquely identifying alert on Unisphere system –str

delete_health_check(*health_check_id*, *array_id=None*)

Delete a health check record.

Parameters

- **health_check_id** – health check id – str
- **array_id** – array id – str

delete_snmp_trap_destination(*snmp_id*)

Deletes specified SNMP trap receiver from configuration.

:param snmp_id unique identifier for snmp trap destination - str

download_all_settings(*file_password*, *dir_path=None*, *file_name=None*, *array_id=None*, *return_binary=False*)

Download all settings.

Export settings feature allows the saving of a subset of system settings to a file. The exported settings have a generic format and do not contain any specific information regarding particular storage array or Unisphere instance, thus making it applicable in any environment.

The intention is to help users to port the system wide settings to another instance of Unisphere, and also to capture single array settings so that they can be applied to another storage array within single instance or another instance of Unisphere at any point of time.

- **All settings:**
  - **Unisphere settings:**
    * Alert notifications

        * Performance metrics
        * Performance preferences
        * Performance user templates
    - **System settings:**
        * Alert policies
        * Alert level notifications
        * Performance thresholds
        * System thresholds

    By default settings will be written to a zip file in the current working directory unless a supplied directory path and/or file name are provided. If any extension is provided in the file name it will be replaced with .zip before the data is written to file.

    If instead the file writing process should be handled outside of PyU4V or uploaded directly to Unisphere without any file handling set return_binary to True. The response dict will have the settings binary data included in key 'binary_data'.

    Parameters

    - **file_password** – password to sign file (required) – str
    - **dir_path** – file save location – str
    - **file_name** – zip file name – str
    - **array_id** – array id – str
    - **return_binary** – return settings binary data – bool

    Returns
    download details – dict

download_audit_log_record(*array_id=None*, *return_binary=False*, *dir_path=None*, *file_name=None*, *timeout=None*)

Download audit log record for the last week in PDF

Parameters

- **array_id** – array serial number – str
- **return_binary** – return binary data instead of writing audit log record pdf to file – bool
- **dir_path** – file write directory path – str
- **file_name** – file name – str
- **timeout** – timeout – int

Returns
download details – dict

download_system_settings(*file_password*, *dir_path=None*, *file_name=None*, *array_id=None*, *return_binary=False*, *exclude_alert_policy_settings=False*, *alert_level_notification_settings=False*, *exclude_system_threshold_settings=False*, *exclude_performance_threshold_settings=False*)

Export System settings.

- **System settings:**
  - Alert policies
  - Alert level notifications
  - Performance thresholds
  - System thresholds

By default settings will be written to a zip file in the current working directory unless a supplied directory path and/or file name are provided. If any extension is provided in the file name it will be replaced with .zip before the data is written to file.

If instead the file writing process should be handled outside of PyU4V or uploaded directly to Unisphere without any file handling set return_binary to True. The response dict will have the settings binary data included in key 'binary_data'.

Parameters

- **file_password** – password to sign file (required) – str
- **dir_path** – file save location – str
- **file_name** – zip file name – str
- **array_id** – array id – str
- **return_binary** – return settings binary data – bool
- **exclude_alert_policy_settings** – exclude alert policy settings – bool
- **alert_level_notification_settings** – exclude alert level notification settings – bool
- **exclude_system_threshold_settings** – exclude system threshold settings – bool
- **exclude_performance_threshold_settings** – exclude performance threshold settings – bool

Returns

export details – dict

download_unisphere_settings(*file_password*, *dir_path=None*, *file_name=None*, *return_binary=False*, *exclude_alert_notification_settings=False*, *exclude_performance_preference_settings=False*, *exclude_performance_user_templates=False*, *exclude_performance_metric_settings=False*)

Download Unisphere settings.

- **Unisphere settings:**
  - Alert notifications
  - Performance metrics
  - Performance preferences
  - Performance user templates

By default settings will be written to a zip file in the current working directory unless a supplied directory path and/or file name are provided. If any extension is provided in the file name it will be replaced with .zip before the data is written to file.

If instead the file writing process should be handled outside of PyU4V or uploaded directly to Unisphere without any file handling set return_binary to True. The response dict will have the settings binary data included in key 'binary_data'.

Parameters

- **file_password** – password to sign file (required) – str
- **dir_path** – file save location – str
- **file_name** – zip file name – str
- **return_binary** – return settings binary data – bool
- **exclude_alert_notification_settings** – exclude alert notification settings – bool
- **exclude_performance_preference_settings** – exclude performance preference settings – bool
- **exclude_performance_user_templates** – exclude performance user templates – bool
- **exclude_performance_metric_settings** – exclude performance metric settings

Returns

download details – dict

get_alert_details(*alert_id*)

Gets the details of an alert.

Parameters

alert_id – alert_id uniquely identifying alert on Unisphere system – str

Returns

alert details – dict

get_alert_ids(*array=None*, *_type=None*, *severity=None*, *state=None*, *created_date=None*, *_object=None*, *object_type=None*, *acknowledged=False*, *description=None*)

Get a list of Alert Ids.

Parameters for this function can be combined to create a search pattern based on multiple filters to target results.

Parameters

- **array** – filters returned list to display Alert Ids that are associated with the specified array e.g. "000213234443" or "<like>443" – str
- **_type** – filters returned list to display Alert Ids that has the following type: ARRAY, PERFORMANCE, SERVER
- **severity** – filters returned list to display only Alert Ids with specified severity: NORMAL, INFORMATION, MINOR, WARNING, CRITICAL, FATAL – str
- **state** – filters returned list to display Alert Ids that has the following state: NEW, ACKNOWLEDGED, CLEARED – str
- **created_date** – filters returned list to display Alert Ids that are greater than(">1"), Less than("<1") or equal to the specified created_date "MMM-dd-yyyy HH:mm:ss.SSS" – str
- **_object** – filters returned list to display Alert Ids that are associated with the specified array object e.g. equal to "object=SRP_3" – str

- **object_type** – filters returned list to display Alert Ids that are associated with the specified array object type e.g. equal to "object_type=Director" – str
- **acknowledged** – filters returned list to display Alert Ids that are acknowledged or not – bool
- **description** – filters returned list to contain text matching description in body – str

Returns
list of alert ids – list

`get_alert_summary()`

Gets Alert Summary information.

Returns
summary of alerts on system - dict

`get_any_director_port(director, filters=None)`

Get a non-GuestOS port from a director.

Parameters

- **director** – director to search for ports with – str
- **filters** – filters to apply when search for port – str

Returns
port – int

`get_audit_log_list(start_time, end_time, array_id=None, user_name=None, host_name=None, client_host=None, message=None, record_id=None, activity_id=None, application_id=None, application_version=None, task_id=None, process_id=None, vendor_id=None, os_type=None, os_revision=None, api_library=None, api_version=None, audit_class=None, action_code=None, function_class=None)`

Get a list of audit logs from Unisphere between start and end date.

Retrieve a list of audit logs from Unisphere, it is possible to filter this list through the input parameters. Due to the potential to return large amounts of results both start and end time are required.

Parameters

- **start_time** – timestamp in milliseconds since epoch – int
- **end_time** – timestamp in milliseconds since epoch – int
- **array_id** – array serial number – str
- **user_name** – Optional value that filters returned list to display Audit Log Entries that contain the specified username only – str
- **host_name** – Optional value that filters returned list to display Audit Log Entries that contain the specified host_name only – str
- **client_host** – Optional value that filters returned list to display Audit Log Entries that contain the specified client_host only – str
- **message** – Optional value that filters returned list to display Audit Log Entries that contain the specified message only – str
- **record_id** – Optional value that filters returned list to display Audit Log Entries that have a matching record_id – int
- **activity_id** – Optional value that filters returned list to display Audit Log Entries that contain the specified activity_id only – str

- **application_id** – Optional value that filters returned list to display Audit Log Entries that contain the specified application_id only – str
- **application_version** – Optional value that filters returned list to display Audit Log Entries that contain the specified application_version only – str
- **task_id** – Optional value that filters returned list to display Audit Log Entries that contain the specified task_id only – str
- **process_id** – Optional value that filters returned list to display Audit Log Entries that contain the specified process_id only – str
- **vendor_id** – Optional value that filters returned list to display Audit Log Entries that contain the specified vendor_id only – str
- **os_type** – Optional value that filters returned list to display Audit Log Entries that contain the specified os_type only – str
- **os_revision** – Optional value that filters returned list to display Audit Log Entries that contain the specified os_revision only – str
- **api_library** – Optional value that filters returned list to display Audit Log Entries that contain the specified api_library only – str
- **api_version** – Optional value that filters returned list to display Audit Log Entries that contain the specified api_version only – str
- **audit_class** – Optional value that filters returned list to display Audit Log Entries that contain the specified audit_class only – str
- **action_code** – Optional value that filters returned list to display Audit Log Entries that contain the specified action_code only – str
- **function_class** – Optional value that filters returned list to display Audit Log Entries that contain the specified function_class only – str

Returns
audit logs – list

get_audit_log_record(*record_id*, *array_id=None*)

Get audit log details for a specific record.

Parameters

- **record_id** – audit log record id – int
- **array_id** – array serial number – str

Returns
audit log record details – dict

get_director(*director*)

Query for details of a director for a symmetrix.

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

Returns
director details – dict

get_director_list(*array_id=None*, *iscsi_only=False*)

Get a list of directors for a given array.

Parameters

- **array_id** – array serial number – str
- **iscsi_only** – return only iSCSI directors – bool

Returns
iSCSI directors – list

get_director_port(*director*, *port_no*)

Get details of the symmetrix director port.

Parameters

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

Returns
director port details – dict

get_director_port_list(*director_id*, *array_id=None*, *filters=None*)

Get a list of director ports for a specified director.

Parameters

- **director_id** – director id – str
- **array_id** – array id – str
- **filters** – user inputted filters see documentation for details

Returns
director ports – list

get_director_port_list_by_protocol_v4(*directors*, *protocol='SCSI_FC'*)

Get directors and ports for V4 by protocol.

In V4 the enabled_protocol can be one of NVMe_TCP SCSI_FC NVMe_FC RDF_FC iSCSI RDF_GigE

Parameters

- **directors** – directors – list
- **protocol** – protocol – str

Returns
port list

get_directory_port_iscsi_endpoint_list(*director_id*, *iscsi_endpoint*, *array_id=None*)

Get a list of director ports for a specified director.

Parameters

- **director_id** – director id – str
- **iscsi_endpoint** – if the port is an iSCSI target, applicable to front-end SE or OR directors only, default to not set – bool
- **array_id** – array id – str

Returns
director ports – list

get_disk_details(*disk_id*, *array_id=None*)

Get details for specified disk id.

Parameters

- **disk_id** – disk id – str
- **array_id** – array id – str

Returns
disk details – dict

get_disk_id_list(*array_id=None*, *failed=False*)

Get a list of disks ids installed.

Parameters

- **array_id** – array id – str
- **failed** – if only failed disks should be returned – bool

Returns
disk ids – list

get_fa_directors()

Get all FA directors on the array.

Returns
fa director strings – list

get_fc_director_port_list_v4(*directors*)

Get FC directors and ports for V4.

In V4 the enabled_protocol = SCSI_FC

Parameters
directors – directors – list

Returns
port list

get_health_check_details(*health_check_id*, *array_id=None*)

Gets details of individual health check.

Parameters

- **health_check_id** – health check id – str
- **array_id** – array id – str

Returns
health check details – dict

get_ip_interface(*director_id*, *port_id*, *interface_id*, *array_id=None*)

Get IP interface details

Parameters

- **director_id** – director id – str
- **port_id** – port id – str
- **interface_id** – interface id – str
- **array_id** – array id – str

Returns
IP interface details – dict

get_ip_interface_list(*director_id*, *port_id*, *array_id=None*)

Get a list of IP interfaces for a given array.

Parameters

- **director_id** – director id – str
- **port_id** – port id – str
- **array_id** – array id – str

Returns
IP interfaces – list

get_iscsi_director_port_list_v4(*directors*)

Get iSCSI directors and ports for V4.

In V4 the enabled_protocol = iSCSI

Parameters
directors – directors – list

Returns
port list

get_iscsi_ip_address_and_iqn(*port_id*)

Get the ip addresses from the director port.

Parameters
port_id – director port identifier – str

Returns
ip addresses, iqn – list, str

get_management_server_resources()

Get Details on Server Resources and REST Utilization.

returns: dictionary with details on server utilization –dict

get_nvme_director_port_list_v4(*directors*)

Get NVMe directors and ports for V4.

In V4 the enabled_protocol = NVMe_FC :param directors: directors – list :returns: port list

get_or_directors()

Get all OR directors on the array.

Returns
or director strings – list

get_port_group(*port_group_id*)

Get port group details.

Parameters
port_group_id – name of the portgroup – str

Returns
port group details – dict

get_port_identifier(*director*, *port_no*)

Get the identifier (wwn) of the physical port.

Parameters

- **director** – the id of the director – str
- **port_no** – the number of the port – str

Returns
wwn (FC) or iqn (iscsi) – str or None

get_rdf_director_port_list_v4(*directors*)

Get RDF directors and ports for V4.

In V4 the enabled_protocol = RDF_FC

Parameters
directors – directors – list

Returns
port list

get_rdf_gige_director_port_list_v4(*directors*)

Get RDF GIGE directors and ports for V4.

In V4 the enabled_protocol = RDF_GigE

Parameters
directors – directors – list

Returns
port list

get_server_logging_level()

Get Server logging level for Unisphere Server. returns: dict

get_snmp_trap_configuration()

Returns the details of SNMP Trap Receivers.

:returns SNMP configuration information–dict

get_system_health(*array_id=None*)

Query for system health information.

Parameters
array_id – array id – str

Returns
system health – dict

get_tagged_objects(*tag_name*)

Get a list of objects with specified tag.

Parameters
tag_name – tag name – str

Returns
tags – list

get_tags(*array_id=None*, *tag_name=None*, *storage_group_id=None*, *num_of_storage_groups=None*, *num_of_arrays=None*)

Query for a list of tag names.

The input parameters represent optional filters for the tag query, including any filters will apply that filter to the list of returned tags.

Parameters

- **array_id** – filter by array id – str
- **tag_name** – filter by tag name – str
- **storage_group_id** – filter by storage group id – str
- **num_of_storage_groups** – filter by tags that are in x or greater amount of storage groups – int
- **num_of_arrays** – filter by tags that in y or greater amount of arrays – int

Returns
tags – list

get_target_wwns_from_port_group(*port_group_id*)

Get the director ports' WWNs.

Parameters
port_group_id – the name of the port group – str

Returns
target_wwns – target wwns for the port group – list

list_system_health_check(*array_id=None*)

List previously run system health checks.

Parameters
array_id – array id – str

Returns
system health checks – list

perform_health_check(*array_id=None*, *description=None*)

Initiate a environmental health check.

Parameters

- **array_id** – array id – str
- **description** – description for health check, if not set this will default to 'PyU4V-array_id-date-time' – str

Returns
health check property details – dict

refresh_array_details(*array_id=None*)

Refresh Unisphere object model for specified array with latest configuration information.

Note, the usage of this call is restricted to run once every 5 minutes to avoid excessive usage. Usage if changes have been made from another Unipshere instance this call can be run to ensure systems are in sync.

Parameters
array_id – The storage array ID – string

returns: None or Status Code 429 with message

set_director_port_online(*director*, *port_no*, *port_online*)

Set Director Port online or offline.

Parameters

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

- **port_online** – True will attempt to online port, false will offline port – bool

Returns
director port details – dict

set_port_protocol(*director*, *port_number*, *protocol*, *enable=True*, *array_id=None*, *_async=None*)

Enable or disable protocol on OR director ports.

Parameters

- **director** – Director Id e.g. OR-1C – str
- **port_number** – Director Port Number – int
- **protocol** –
- **enable** –
- **array_id** –

Returns

set_server_logging_level(*server_log_level='WARN'*, *restapi_logging_enabled=False*)

Get Server logging level for Unisphere Server. :param server_log_level - INFO, WARN, DEBUG – string :param restapi_logging_enabled - bool returns: dict

set_snmp_trap_destination(*name*, *port*, *username=None*, *password=None*, *passphrase=None*)

Add new SNMP trap receiver to configuration.

Parameters

- **name** – Ipdaddress or DNS Name – str
- **port** – port SNMP server recieves trap on –int
- **username** – Username for SNMP v3Username for SNMP v3 –str
- **password** – Password for SNMP v3 –str
- **passphrase** – Passphrase for SNMP v3 –str

Returns

update_snmp_trap_destination(*snmp_id*, *name=None*, *port=None*, *username=None*, *password=None*, *passphrase=None*)

Update existing SNMP configuration item.

Parameters

- **snmp_id** – an id generated for a specific SNMP trap, use get_snmp_trap_configuration to find values – str
- **name** – New IP address/hostname for the SNMP trap – str
- **port** – New port number for the SNMP trap –str
- **username** – Updated username for the SNMP trap –str
- **password** – Updated password for SNMP trap –str
- **passphrase** – Updated passphrase for SNMP trap –str

Returns
dict

upload_settings(*file_password*, *file_path=None*, *array_id=None*, *binary_data=None*)

Upload Unisphere and/or system settings to Unisphere.

Allows for importing a zip file or binary data that contains settings that were previously exported from Unisphere.

The settings that a file upload may include are:

- **All settings:**
  - **Unisphere settings:**
    - Alert notifications
    - Performance metrics
    - Performance preferences
    - Performance user templates
  - **System settings:**
    - Alert policies
    - Alert level notifications
    - Performance thresholds
    - System thresholds

A password that was specified during export needs to be provided during import operation. This is to assure that the imported file has not been tampered with.

It is possible to upload system settings for more than one array, to do so pass a list of array IDs in to array_id input parameter. For Unisphere settings an array ID is not required.

Parameters

- **file_password** – password that file has been signed with – str
- **file_path** – full file location – str
- **array_id** – array id – str
- **binary_data** – binary settings data – bytes

Returns

upload details – dict

11.12 PyU4V.serviceability

serviceability.py.

class PyU4V.serviceability.**ServiceabilityFunctions**(*array_id*, *rest_client*)

Bases: object

download_grab_files(*array_id=None*, *node_name='Unisphere'*, *return_binary=False*, *dir_path=None*, *file_name=None*, *timeout=None*)

Download serviceability logs

Parameters

- **array_id** – array serial number – str

- **node_name** – Node name. Allowable values are Unisphere, Vasa0, Vasa1, Vasadb, Semgmt0, Semgmt1 – str
- **return_binary** – return binary data instead of writing audit log record pdf to file – bool
- **dir_path** – file write directory path, eg. "." will write to script execution directory – str
- **file_name** – file name, file exension .tar.gz will be applied by the function – str
- **timeout** – timeout, recommend setting a long timeout value as grab file generation can take some time, e.g. 1000 – int

Returns
download details – dict

get_application(*array_id=None*)

Get a list of all embedded applications running on the array.

Parameters
array_id – array id – str

Returns
ApplicationInformation – dict

get_ip_configuration(*array_id=None*)

Get current IPv4 information of U4P, VASA and SE and IPv6 information of U4P and SE.

Parameters
array_id – array id – str

Returns
IPInformation – dict

get_local_system()

Get local powermax system serial number in the embedded environment with version v4 or higher. :returns: symmetrix id – dict

get_ntp_settings(*array_id=None*)

Get current NTP server configuration.

Parameters
array_id – array id – str

Returns
ntp server – dict

get_solutions_enabler_application(*array_id=None*)

Get a list of information on each SE node and get the access Id, this function also returns se_nethost configuration for client server access.

Parameters
array_id – array id – str

Returns
SEInformation – dict

get_solutions_enabler_configuration(*array_id=None*)

Get SE base configuration values that are available in SE settings.

Parameters
array_id – array id – str

Returns
SEConfigInformation – dict

`get_symavoid_settings`(*array_id=None*)

Get a list of available symmetrix and symavoid symmetrix.

Parameters
array_id – array id – str

Returns
UnisphereInformation – dict

`get_unisphere_application_details`(*array_id=None*)

Get a list of information about Unisphere node. Get the access Id along with Unisphere server access.

Parameters
array_id – array id – str

Returns
UnisphereInformation – dict

`get_unisphere_configuration`(*array_id=None*)

Get Unisphere configuration information.

Parameters
array_id – array id – str

Returns
ConfigArray – dict

`import_custom_certificate`(*array_id=None*, *node_name=None*, *keyfile=None*, *certfile=None*, *trustfile=None*)

Import custom certificate.

Parameters

- **array_id** – array id – str
- **node_name** – Specify the node name (Semgmt0/Semgmt1) for which this certificate is imported. – str
- **keyfile** – Path to alternate key file all_key.pem. – str
- **certfile** – Path to alternate certificate file all.pem. – str
- **trustfile** – Path to alternate trust certificate file trust.pem. – str

Returns
SECertificateInfo – dict

`modify_nethosts`(*action*, *host_name*, *user*, *array_id=None*)

Add or remove user and host to nethosts file for client server configuration.

Current configuration can be checked with get_solutions_enabler_application() function

Parameters

- **action** – add or remove - str
- **host_name** – name or ip of host to be added/rmoved to/from nethost file – str
- **user** – username to be granted access – str
- **array_id** – array_id – str

Returns
net host configuration – dict

modify_ntp_settings(*ntp_server*, *array_id=None*)

Set a new NTP server information.

This call will restart SMAS Service so Unisphere and REST will be unavailable while the configuration change takes effect please do not attempt any other calls until this has completed and changes have been verified.

Parameters

- **ntp_server** – ntp server URL –str
- **array_id** – array id –str

Returns
ntp server – dict

modify_solutions_enabler_configuration(*array_id=None*, *allow_symforce=None*, *use_access_id=None*)

set SE configuration values.

Parameters

- **array_id** – array id – str
- **allow_symforce** – Indicates whether users can specify -symforce when performing RDF control operations. The allowable values for this option are TRUE and FALSE. The default setting is FALSE. Changing this value requires unisphere to be restarted before it will take effect in the UI –bool
- **use_access_id** – This option applies to Symmetrix Access Control. It specifies whether to use the access ID generated on client or server. Used only on the server side during client/server operations. Possible values are: CLIENT: The client access ID is used for every command performed. If a client access ID is not available the command will fail. SERVER: The server access ID is used for every command performed. ANY: If the client access ID is available it is used for every command performed. If it is not available then the server access ID is used. The default setting is SERVER. –str

Returns
ConfigArray – dict

modify_unisphere_service_access(*action*, *array_id=None*)

Enables Unisphere server access for remote support assistance.

Parameters
action – Unisphere server access options such as AllowServerAccess and BlockServer-Access –str

Returns
ntp server – dict

replace_self_signed_certificate(*array_id=None*, *node_name=None*)

Replace SE Management Self signed certificate.

Parameters

- **array_id** – array id – str
- **node_name** – Specify the node name (Semgmt0/Semgmt1) for which this self-signed certificate is created. – str

restart_unisphere_application(*array_id=None*)

Restart the Unisphere server.

Please do not attempt any other calls until this has completed and changes have been verified.

Parameters
array_id – array id –str

Returns
will not return as Unisphere server restarts.

update_ip_configuration(*array_id=None*, *action=None*, *natone_ip_address=None*, *natone_netmask=None*, *natone_gateway=None*, *nattwo_ip_address=None*, *nattwo_netmask=None*, *nattwo_gateway=None*)

Set new IPv4 information of U4P, VASA and SE. and set new IPv6 information of U4P and SE.

During the NAT IP PUT operation, the existing Unisphere/REST server will restart and with New IP address.

Please do not attempt any other calls until this has completed and changes have been verified.

Parameters

- **array_id** – array id – str
- **action** – update Options to modify (UpdateIPV4 or UpdateIPV6) – str
- **natone_ip_address** – primary ip address for unisphere instance connecting to port on node 1 of array either IPV4 or IPV6 – str
- **natone_netmask** – netmask for natone ip address – str
- **natone_gateway** – gateway for natone ip address IPv4 or IPv6 – str
- **nattwo_ip_address** – secondary ip address for unisphere instance connecting to port on node 2 of array either IPV4 or IPV6 – str
- **nattwo_netmask** – netmask for secondary IP connection either IPV4 or IPV6 – str
- **nattwo_gateway** – gateway for secondary IP connection either IPV4 or IPV6 – str

Returns
IPInformation – dict

update_symavoid_settings(*array_id=None*, *action=None*, *symm_list=None*)

Add to symavoid list or remove a symmetrix from the symavoid list

Parameters

- **array_id** – array id – str
- **action** – AddToSymmavoid or RemoveFromSymmavoid, case sensitive – str
- **symm_list** – list of symms – list

Returns
IPInformation – dict

11.13 PyU4V.clone

clone.py.

class PyU4V.clone.**CloneFunctions**(*array_id*, *rest_client*)

Bases: object

Clone Functions.

create_clone(*storage_group_id*, *target_storage_group_id*, *consistent=True*, *establish_terminate=False*, *array_id=None*, *force=False*, *star=False*, *skip=False*)

Create Clone.

Parameters

- **storage_group_id** – The Storage Group ID – string
- **consistent** – creates the clone crash consistent using ECA technology – bool
- **establish_terminate** – creates the clone and immediately terminates, very useful if you want to make an independent copy available immediately but don't intend to use for restore purposes – bool
- **array_id** – The storage array ID – string
- **target_storage_group_id** – name of storage group to contain clone devices – string
- **force** – Attempts to force the operation even though one or more volumes may not be in the normal, expected state(s) for the specified operation – bool
- **star** – Acknowledge the volumes are in an SRDF/Star configuration – bool
- **skip** – Skips the source locks action – bool

Returns
dict

establish_clone(*storage_group_id*, *target_storage_group_id*, *array_id=None*, *consistent=True*, *not_ready=False*, *vse=False*, *force=False*, *star=False*, *skip=False*, *_async=False*)

Perform establish against a clone storage group.

Parameters

- **storage_group_id** – The Storage Group ID – string
- **target_storage_group_id** – name of storage group to contain clone devices – string
- **array_id** – The storage array ID – string
- **consistent** – creates the clone crash consistent using ECA technology – bool
- **not_ready** – sets target storage group to not ready following establish operation – bool
- **vse** – uses VSE close – bool
- **force** – Attempts to force the operation even though one or more volumes may not be in the normal, expected state(s) for the specified operation – bool
- **star** – Acknowledge the volumes are in an SRDF/Star configuration – bool
- **skip** – Skips the source locks action – bool

- **_async** – if call should be async – bool

Returns
dict

get_clone_pairs_list(*storage_group_id*, *array_id=None*)

Get Clone Pairs List. :param: array_id The storage array ID – string :param: storage_group_id The Storage Group ID – string :returns: count and list of clone pairs – dict

get_clone_storage_group_pair_details(*storage_group_id*, *target_storage_group_id*, *array_id=None*)

Get Clone storage group pair details.

:param storage_group_id The Storage Group ID – string :param target_storage_group_id The Target Storage Group ID – string :param array_id The storage array ID – string

get_clone_target_storage_group_list(*storage_group_id*, *array_id=None*, *target_storage_group=None*, *target_storage_group_volume_count=None*, *volume_pair_count=None*, *state=None*, *modified_tracks=None*, *src_protected_tracks=None*, *src_modified_tracks=None*, *background_copy=None*, *differential=None*, *precopy=None*, *vse=None*)

Get Clone target storage group list.

Parameters

- **storage_group_id** – The Storage Group ID – string
- **array_id** – The storage array ID – string
- **target_storage_group** – Value that filters returned list to include target storage groups equal to or like the provided name – string
- **target_storage_group_volume_count** – Value that filters returned list to include target storage groups with the specified number of volumes – string
- **volume_pair_count** – Value that filters returned list to include target storage groups with the specified number of volume pairs – string
- **state** – Value that filters returned list to include target storage groups with pairs in the specified states – array
- **modified_tracks** – Value that filters returned list to include target storage groups with the specified modified tracks – string
- **src_protected_tracks** – Value that filters returned list to include target storage groups with the specified src protected tracks – str
- **src_modified_tracks** – Value that filters returned list to include target storage groups with the specified src modified tracks – string
- **background_copy** – Value that filters returned list to include target storage groups with background copy flag – boolean
- **differential** – Value that filters returned list to include target storage groups with differential flag – boolean
- **precopy** – Value that filters returned list to include target storage groups with the precopy flag – boolean
- **vse** – Value that filters returned list to include target storage groups with the vse – boolean

Returns
a list of target storage groups – list

restore_clone(*storage_group_id*, *target_storage_group_id*, *array_id=None*, *star=False*, *force=False*, *_async=False*)

Perform split actions against a clone storage group that is in the restored state.

Parameters

- **storage_group_id** – The Storage Group ID – string
- **target_storage_group_id** – The Storage Group ID of Target storage group – string
- **array_id** – The storage array ID – string
- **force** – Attempts to force the operation even though one or more volumes may not be in the normal, expected state(s) for the specified operation – bool
- **star** – Acknowledge the volumes are in an SRDF/Star configuration – bool
- **_async** – if call should be async – bool

Returns
dict

split_clone(*storage_group_id*, *target_storage_group_id*, *array_id=None*, *star=False*, *skip=False*, *force=False*, *_async=False*)

Perform split actions against a clone storage group that is in the restored state.

Parameters

- **storage_group_id** – The Storage Group ID – string
- **target_storage_group_id** – The Storage Group ID of Target storage group – string
- **star** – Acknowledge the volumes are in an SRDF/Star configuration – bool
- **skip** – Skips the source locks action – bool
- **force** – Attempts to force the operation even though one or more volumes may not be in the normal, expected state(s) for the specified operation – bool
- **array_id** – The storage array ID – string
- **_async** – if call should be async – bool

Returns
dict

terminate_clone(*storage_group_id*, *target_storage_group_id=None*, *array_id=None*, *force=False*, *symforce=False*, *star=False*, *skip=False*, *not_ready=False*, *restored=None*)

Terminate Clone Session.

Parameters

- **array_id** – The storage array ID – string
- **storage_group_id** – The Storage Group ID – string
- **target_storage_group_id** – name of storage group to contain clone devices – string
- **force** – Attempts to force the operation even though one or more volumes may not be in the normal, expected state(s) for the specified operation – bool
- **star** – Acknowledge the volumes are in an SRDF/Star configuration – bool

- **skip** – Skips the source locks action – bool
- **not_ready** – sets clone devices to not ready after operation – bool
- **restored** – removes the restore flag from clone session, leaves clone session intact for incremental clone operations. Used following restore session – bool

Returns
dict

11.14 PyU4V.volumes

volumes.py.

class PyU4V.volumes.**VolumesFunctions**(*array_id*, *rest_client*)

Bases: object

Enhanced Functions for retrieving Array Configuration Data.

get_volumes_details(*array_id=None*, *filters=None*, *select=None*, *exclude=None*)

Get list of volumes from array with selected parameters.

Parameters

- **array_id** – The storage array ID – string
- **filters** – filter parameters, used to narrow down the result list, filters can be any of the volume attributes, and operators vary depending on type, 'eq','ne','gt','ge','lt','le', equal, not equal, greater than, greater than or equal, less than, less than or equal, additional operators are 'like','ilike' for string match, or case in-sensitive string match, example filters=['num_of_storage_groups eq 1', 'identifier ilike findme'] – list
- **select** – selection of attributes to be in return, attributes can be listed using get_volumes_meta_data function If none selected base set of top level attributes are returned, by default API will only return id, to achieve this pass a list with empty string [''], to extend the list of attributes returned to include second level attributes you can pass along with list from get_volumes_meta_data() e.g. select=api.volumes.get_volumes_meta_data() + ['snapshots.name'] – list
- **exclude** – list of attributes to exclude, by default rdf_infos and snapshot details have been excluded as these can extend the running of the API call, to override simply pass an empty list or specify a list of attribute names that you want to exclude from the return, if values are passed in by select exclude is ignored – list

Returns
dict

get_volumes_meta_data()

Get details on available meta data.

Returns
dictionary with list of attributes and descriptions of attribute types – dict

11.15 PyU4V.storage_groups

storage_groups.py.

class PyU4V.storage_groups.**StorageGroupsFunctions**(*array_id*, *rest_client*)

Bases: object

Enhanced Functions for retrieving Array Configuration Data.

get_storage_groups_details(*array_id=None*, *filters=None*, *select=None*, *exclude=None*)

Get list of storage_groups from array with selected parameters.

Parameters

- **array_id** – The storage array ID – string
- **filters** – filter parameters, used to narrow down the result list, filters can be any of the storage_groups attributes and operators vary depending on type, 'eq','ne','gt','ge','lt','le', equal, not equal, greater than, greater than or equal, less than, less than or equal, additional operators are 'like','ilike' for string match, or case insensitive string match, example filters=['id ilike gk', 'num_of_volumes eq 32'] – list
- **select** – selection of attributes to be in return, attributes can be listed using get_storage_groups_meta_data function e.g. ['volumes.wwn','volumes.effective_wwn', 'type', 'effective_used_capacity_gb'] If none selected base set of top level attributes are returned, by default API will only return id, to achieve this pass a list with empty string [''] – list
- **exclude** – list of attributes to exclude, by default rdf_infos and snapshot details have been excluded as these can extend the running of the API call, to override simply pass an empty list or specify a list of attribute names that you want to exclude from the return, , if values are passed in by select exclude is ignored – list

get_storage_groups_meta_data()

Get details on available meta data for storage group objects.

returns: dictionary with list of attributes and descriptions of
attribute types – dict

11.16 PyU4V.performance_enhanced

performance_enhanced.py.

class PyU4V.performance_enhanced.**EnhancedPerformanceFunctions**(*array_id*, *rest_client*)

Bases: object

Enhanced Functions for retrieving latest diagnostic level performance Metrics.

get_all_performance_metrics_for_system(*array_id=None*)

Get latest data for all KPI metrics.

Parameters
array_id – 12 Digit Serial Number of Array – int

Returns
data for all available categories for the specified PowerMax Array diagnostic level metrics only, 5 min interval – dict

get_category_metrics(*category*, *array_id=None*)

Get latest data for KPI metrics for specificied performance category.

Parameters
array_id – 12 Digit Serial Number of Array – int

Returns
full list of all KPI metrics for latest diagnostic timestamp for the specified array and performance category – dict

get_performance_categories_list(*array_id=None*)

Get a list of performance categories and metrics that will be returned for each category.

Parameters
array_id – 12 Digit Serial Number of Array – int

Returns
a list of categories available for querying, returned list has dictionary of category represented by 'id' key and 'metrics' key detailing what will be returned for the given array represented by 'system' key – list

11.17 PyU4V.workload_planner

workload_planner.py.

class PyU4V.workload_planner.**WLPFunctions**(*array_id*, *rest_client*)

Bases: object

get_capabilities(***kwargs*)

get_headroom(***kwargs*)

get_wlp_information(***kwargs*)

11.18 PyU4V.utils

11.18.1 PyU4V.utils.config_handler

config_handler.py.

PyU4V.utils.config_handler.**set_logger_and_config**(*file_path=None*)

Set logger and config file.

Parameters
file_path – path to PyU4V configuration file – str

Returns
config parser – obj

11.18.2 PyU4V.utils.console

console.py

11.18.3 PyU4V.utils.constants

constants.py.

11.18.4 PyU4V.utils.decorators

decorators.py

PyU4V.utils.decorators.**deprecation_notice**(*func_class*, *start_version*, *end_version*)

Notify the user of function deprecation in a future version.

Parameters

- **func_class** – the class name for deprecated function – str
- **start_version** – the start Unisphere version – float
- **end_version** – the end Unisphere version – float

Returns

decorated function – decorator

PyU4V.utils.decorators.**refactoring_notice**(*func_class*, *path*, *start_version*, *end_version*)

Notify the user of function refactoring elsewhere in PyU4V.

Parameters

- **func_class** – the class name for deprecated function – str
- **path** – path to the new function – str
- **start_version** – the start Unisphere version – float
- **end_version** – the end Unisphere version – float

Returns

decorated function – decorator

PyU4V.utils.decorators.**retry**(*exceptions*, *total_attempts=3*, *delay=3*, *backoff=2*)

Decorator for retrying function calls.

Parameters

- **exceptions** – expected exceptions to retry on – class or tuple
- **total_attempts** – times to run function before failure. – int
- **delay** – initial delay between retries in seconds. – int
- **backoff** – delay multiplier between each attempt – int

11.18.5 PyU4V.utils.exception

exception.py

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

Bases: *PyU4VException*

InvalidInputException.

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

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

Bases: *PyU4VException*

MissingConfigurationFileException

message = 'PyU4V settings not be loaded, please check file location or univmax_conn input parameters.'

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

Bases: Exception

PyU4VException.

code = 500

headers = {}

message = 'An unknown exception occurred.'

safe = False

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

Bases: *PyU4VException*

ResourceNotFoundException.

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

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

Bases: *PyU4VException*

UnauthorizedRequestException.

message = 'Unauthorized request - please check credentials'

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

Bases: *PyU4VException*

VolumeBackendAPIException.

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

11.18.6 PyU4V.utils.file_handler

file_handler.py

PyU4V.utils.file_handler.**create_list_from_file**(*file_name*)

Given a file, create a list from its contents.

Parameters
file_name – the path to the file – str

Returns
file contents – list

PyU4V.utils.file_handler.**read_csv_values**(*file_name*, *convert=False*, *delimiter=','*, *quotechar='|'*)

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 = fh.read_csv_values(mycsv.csv) sg_name_list = data['sgname'] policy_list = data['policy']

Parameters

- **file_name** – path to CSV file – str
- **convert** – convert strings to equivalent data type – bool
- **delimiter** – delimiter kwarg for csv DictReader object – str
- **quotechar** – quotechar kwarg for csv DictReader object – str

Returns
CSV parsed data – dict

PyU4V.utils.file_handler.**write_binary_data_to_file**(*data*, *file_extension*, *file_name*, *dir_path=None*)

Write Unisphere binary data to file.

Parameters

- **data** – Unisphere REST response with data for writing – json response
- **file_extension** – file extension used for writing to file – str
- **file_name** – file name – str
- **dir_path** – file write directory path – str

Returns
file name and write directory – str

PyU4V.utils.file_handler.**write_dict_to_csv_file**(*file_path*, *dictionary*, *delimiter=','*, *quotechar='|'*)

Write dictionary data to CSV spreadsheet.

Parameters

- **file_path** – path including name of the file to be written to – str
- **dictionary** – data to be written to file – dict
- **delimiter** – delimiter kwarg for csv writer object – str
- **quotechar** – quotechar kwarg for csv writer object – str

PyU4V.utils.file_handler.**write_to_csv_file**(*file_name*, *data*, *delimiter=','*, *quotechar='|'*)

Write list data to CSV spreadsheet.

Parameters

- **file_name** – name of the file to be written to – str
- **data** – data to be written to file – list
- **delimiter** – delimiter kwarg for csv writer object – str
- **quotechar** – quotechar kwarg for csv writer object – str

11.18.7 PyU4V.utils.time_handler

time_handler.py

PyU4V.utils.time_handler.**format_time_input**(*time_in*, *return_seconds=False*, *return_milliseconds=False*)

Format timestamp as seconds/milliseconds for use in REST requests.

Parameters

- **time_in** – timestamp – int/float
- **return_seconds** – return time in seconds – bool
- **return_milliseconds** – return time in milliseconds – bool

Returns

timestamp – int

__init__.py.

CHAPTER

TWELVE

API INDEX

CHAPTER

THIRTEEN

WELCOME TO PYU4V'S DOCUMENTATION!

13.1 Overview

PyU4V is a Python module that simplifies interaction with the Unisphere for PowerMax REST API. It wraps REST calls with simple APIs that abstract the HTTP request and response handling.

Note: You can get the Unisphere for PowerMax REST documentation by navigating to https://developer.dell.com/apis/4458/ An OpenApi.json file can be download from this site for use with tools like PostMan or for interfacing with PyU4V custom API call mechanism. Please refer to this documentation when looking for information about query path parameters for functions that accept them.

13.2 Supported PyU4V Versions

PyU4V Version	10.1.0.1
Minimum Unisphere Version	10.1
Array Model	VMAX-3, VMAX AFA, PowerMax
Array uCode	HyperMax OS, PowerMax OS
Platforms	Linux, Windows
Python	3.6, 3.7, 3.8, 3.9, 3.10
Requirements	Requests, Six, urllib3, prettytable
Test Requirements	TestTools, Tox

Note: PyU4V officially supports Python 3.6, 3.7, 3.8, 3.9 & 3.10 Python 2.x support has been dropped since January 1st 2020.

Note: PyU4V version 10.1.x is compatible with scripts written for PyU4V versions >= 9.2. Please ensure to check change log to ensure that you are not using functions that have been marked as depreciated.

13.3 Getting Started

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

Configuration
Configuring PyU4V for your environment.

Quick Start Guide
Making your first calls with PyU4V.

Contribute to PyU4V
Contribute to the PyU4V project.

Issues & Support
How to get support with or open issues for PyU4V.

Programmers Guide
A range of examples demonstrating various PyU4V module usage.

Tools Guide
The tools guide for PyU4V

API Glossary
A glossary of all available functions.

13.4 Build your own PyU4V Docs

PyU4V docs have been built using Sphinx and included with the source PyU4V package, however if you would like to build the docs from scratch use the following commands:

```
$ pip install sphinx
$ pip install sphinx-rtd-theme
$ cd PyU4V/docs
$ make clean && make html
```

All of the necessary make files and sphinx configuration files are included with PyU4V so you can build the docs after the required dependencies have been installed.

Once the above commands have been run you will find newly generated html files within the `/PyU4V/docs/build` folder. Open `index.html` within a browser of your choosing to view the docs offline. Generating the docs is not required, we have bundled the most up-to-date docs with PyU4V so you can still navigate to `/PyU4V/docs/build/index.html` within your browser to view PyU4V docs offline.

13.5 Disclaimer

PyU4V 10 is distributed under the Apache 2.0 License. Unless required by applicable law or agreed to in writing, software distributed under the Apache 2.0 License is distributed on an **"as is" basis, without warranties or conditions of any kind**, either express or implied. See the License for the specific language governing permissions and limitations under the License.

PYTHON MODULE INDEX

p

INDEX

D

E

F

G

H

I

L

R

S

T

U

V

W