Dell Storage Manager REST API Cookbook

Abstract

This paper discusses code samples with the Dell™ Storage Manager (DSM) REST API which can provide and improve on orchestration and delivery of repetitive tasks while minimizing the potential for error.

January 2019

Revisions

Date	Description
May 2016	Initial release
October 2016	New code samples
January 2019	Multiple edits and update to new format.

Acknowledgements

This paper was produced by the following members of the Dell EMC storage engineering team:

Authors: Daniel Tan, Linux/UNIX Product Specialist
Mark Tomczik, Oracle Product Specialist

Table of contents

Executive summary

Dell™ Storage Manager (DSM), formerly known as Enterprise Manager, is a comprehensive, user-friendly, single-pane-of-glass framework for management and administration of various Dell EMC™ storage platforms in a unified environment.

There are circumstances within business environments in which tasks may be repetitive, tedious, and error-prone when attempted with a GUI-based framework. The DSM REST API can provide and improve on orchestration and delivery of these often-repetitive tasks while minimizing the potential for error.

This paper introduces and discusses code samples and the use of the REST API using Python™. All code samples within this paper are shared as-is without any explicit statement of warranty or ongoing support.

1 Getting started

The Python interpreter is used in this paper to interface with the DSM RESTful API to deliver commands and retrieve data proxied through DSM to a Dell EMC SC Series array. This section outlines installing and configuring Python as well as the required configuration for the DSM RESTful API interface.

The difference between the legacy Storage Center CompCU and the DSM RESTful API with Python, is that the former interacts directly with the SC Series array while the latter proxies all requests through a DSM Data Collector installation to an SC Series array.

1.1 REST overview

REST stands for Representational State Transfer. It relies on a stateless, client-server-based, cacheable communications protocol; in most cases the HTTP protocol is used.

REST is an architecture style for designing scalable networked applications relying on a simple yet effective HTTP transport protocol to make calls between machines, devices, or platforms.

REST relies on the HTTP CRUD (Create/Read/Update/Delete) operations to interact with the destination machine or device. These operations are more commonly known as the GET, POST, PUT, and DELETE HTTP methods.

Each DSM REST API call is referenced by a specific delimited string. The application of this string with one of the HTTP methods mentioned previously will direct DSM to either return data, create an object, modify an object, or remove an object accordingly.

The DSM REST API calls are shown in a table at the start of each section as well as referenced by the **REST** variable name within each section of code.

Use the following procedure to download the complete list of REST API calls, help files, and enumerated field values supported by the DSM Data Collector.

1. Go to http://www.dell.com/support/home/us/en/19/product-support/product/storage-sc2000/drivers/.
2. Refine the **Drivers and downloads** search criteria:
   - **Keyword**: Enter **REST API**.
   - **Category**: Set to **Systems Management**.
3. Browse the query results and download the applicable version of the **Public REST API for Dell Storage API**.

1.2 Install and configure Python

This paper uses the Python v2.7.10 interpreter on a Fedora® 23 server installation to create and manage the HTTP session connection, submit REST API calls to the DSM installation, and retrieve data for processing or validation. The following commands install Python, pip, and the required supporting modules. The installation methods presented are applicable to Red Hat® Enterprise Linux®, CentOS® and other compatible Linux distributions.

```
# yum -y update
# yum -y install python
# yum -y install python-pip
# pip --install upgrade pip
# pip install requests http simplejson httplib urllib urllib2
# python --version
Python 2.7.10
```

1.3 Deploy modules and logging

Python is an extensible scripting platform that allows additional features and functions to be added by using importable modules. Some examples of available modules are referenced by the following import statements in python script **main.py**.

```
# head -10 ./main.py
# import modules into Python script
import requests, json, http, httplib, urllib, urllib2
import os, sys, subprocess, math
import math, time
import logging
from simplejson import scanner

# setup logging to scapi.log

logging.basicConfig(level=logging.DEBUG
                    ,filename='scapi.log'
                    ,format='[%(asctime)s] %(levelname)s %(message)s')
```

Note: Logging defaults to capturing all information, warnings, and errors to the **scapi.log** file within the current working directory.

1.4 DSM REST API installation and help

The DSM REST API is supported with DSM 2015 R3, build 15.3.1.300 or newer.

The REST API is automatically installed with the default installation of the DSM Data Collector on the specified Microsoft® Windows® host.

The latest DSM Data Collector is available on Dell.com/support.

The information in this paper has been tested and validated with a modified version of the Python-based HttpClient class and methods created and developed to support Cinder and SC Series arrays with OpenStack. The non-modified HttpClient class code is open source and available through GitHub project openstack/cinder in master branch cinder/volume/drivers/dell_emc/sc and is located in file storagecenter_api.py.

1.5 Establish the DSM parameters

The **main.py** used for this paper contains the following initialization attributes to establish the connection to the DSM Data Collector.

```
# define env incl. DSM IP addr, port & login credentials

DSM_ip = '<nnn.nnn.nnn.nnn>'  # IP address of DSM instance
DSM_port = '3033'             # Default port of DSM instance
DSM_id = '<Username>'         # Login credentials for DSM
DSM_pass = '<Password>'       # Password
verify_cert = False           # Default = False
apiversion = '2.0'            # Default = 2.0
```

1.6 Define the base URL and header information

The base URL and header information are in the **./main.py** file and should remain as provided.

```
# define base URL for DSM REST API interface

baseURL = 'https://%s:%s/api/rest/' % (DSM_ip, DSM_port)

# define HTTP content headers

header = {}
header['Content-Type'] = 'application/json; charset=utf-8'
header['Accept'] = 'application/json'
header['x-dell-api-version'] = apiversion
```

1.7 Create a requests.Session() object

These two lines in ./main.py create a **requests.Session()** object named **connection**. This object is used throughout the rest of this paper to reference the open connection to the DSM Data Collector.

```
# define the connection session

connection = requests.Session()
connection.auth = (DSM_id, DSM_pass)
```

2 Connecting to a DSM Data Collector

This section discusses using the configured Python environment to interact with the DSM Data Collector. Python is a natively object-oriented language; methods such as opening a connection to the DSM Data Collector involve using and interacting with the **connection** object previously created.

All subsequent REST API calls are defined between the open and close connection code segments shown in the following subsections.

2.1 Open a connection

This code defines the REST API request and opens a connection to the DSM Data Collector by using the HTTP POST call.

```
# login to DSM instance
# declare and define the payload variable

payload = {}

# define the REST API call

REST = '/ApiConnection/Login'

# build the complete REST API URL

completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])

# execute REST API call via the HTTP POST method

print connection.post(completeURL
                      ,data=json.dumps(payload
                                       ,ensure_ascii=False).encode('utf-8')
                      ,headers=header
                      ,verify=verify_cert)
```

STDOUT

```
<Response [200]>
```

2.2 Close a connection

In a similar fashion, it is a good practice to close any open object connections prior to exiting a script as shown in the following.

```
# logout from DSM instance
# declare and define the payload variable

payload = {}
```

```
# define the REST API call

REST = '/ApiConnection/Logout'

# build the complete REST API URL

completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])

# execute REST API call via the HTTP POST method

print connection.post(completeURL
                      ,data=json.dumps(payload
                                       ,ensure_ascii=False).encode('utf-8')
                      ,headers=header
                      ,verify=verify_cert)
```

STDOUT

```
<Response [204]>
```

2.3 Understanding instanceId

Every object that the REST API calls can interact with has a key attribute named **instanceId**. The value of the **instanceId** key may be associated with a REST API open connection, SC Series array, server object, volume object, storage profile, and more.

These **instanceId** values are often injected into subsequent REST API calls to interact with, create, modify, or remove further object types. The values are denoted in **RED** where required in the following code samples.

Understanding this key attribute and how to extract this value from the raw JSON-based output sent by the DSM Data Collector is required.

2.3.1 <instanceId> API connection

```
# capture API connection instanceId
# declare and define the payload variable

payload = {}

# define the REST API call

REST = '/ApiConnection/ApiConnection'

# build the complete REST API URL

completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
```

```
# execute REST API call via the HTTP GET method

json_data = connection.get(completeURL
                           ,headers=header
                           ,verify=verify_cert)
stdout = json.loads(json_data.text)
conn_instanceId = stdout['instanceId']
print conn_instanceId
```

STDOUT

```
0
```

2.3.2 <instanceId> All SC Series arrays managed by a DSM Data Collector

```
# capture all SC series arrays managed by this DSM instance
# declare and define the payload variable

payload = {}

# define the REST API call

REST = '/ApiConnection/ApiConnection/%s/StorageCenterList' % conn_instanceId

# build the complete REST API URL

completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])

# execute REST API call via the HTTP GET method

json_data = connection.get(completeURL
                           ,headers=header
                           ,verify=verify_cert)
stdout = json.loads(json_data.text)
scList = {}
print "Name\t\tSerial Number\t\tinstanceId\t\tIP"
for i in range(len(stdout)):
    print "%s\t\t%s\t\t\t%s\t\t%s" % (stdout[i]['name']
                                     ,stdout[i]['scSerialNumber']
                                     ,stdout[i]['instanceId']
                                     ,stdout[i]['hostOrIpAddress'])
    scList[stdout[i]['name']] = {}
    scList[stdout[i]['name']]['instanceId'] = stdout[i]['instanceId']
    scList[stdout[i]['name']]['hostOrIP'] = stdout[i]['hostOrIpAddress']
```

STDOUT

```
Name          scSerialNumber          instanceId        IP
SC 4          862                     862               172.16.2.104
SC 18         716                     716               172.16.2.118
SC 9          101                     101               172.16.2.109
```

2.3.3 <instanceId> All volume folder objects

```
# declare and define the payload variable

payload = {}

# define the REST API call

REST = '/StorageCenter/StorageCenter/%s/VolumeFolderList' %
       (scList['SC 9']['instanceId'])

# build the complete REST API URL

completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])

# execute REST API call via the HTTP GET method

json_data = connection.get(completeURL
                           ,headers=header
                           ,verify=verify_cert)
stdout = json.loads(json_data.text)
volFolderList = {}
print "Name\t\t\t\tinstanceId\t\tParent"
for i in range(len(stdout)):
    if stdout[i]['name'] == "Volumes":
        continue
    print "%s\t\t\t\t%s\t\t%s" % (stdout[i]['name']
                                 ,stdout[i]['instanceId']
                                 ,stdout[i]['parent']['instanceName'])
    volFolderList[stdout[i]['name']] = {}
    volFolderList[stdout[i]['name']]['instanceId'] = stdout[i]['instanceId']
    volFolderList[stdout[i]['name']]['parent'] =
        stdout[i]['parent']['instanceName']
```

STDOUT

```
Name                        instanceId        Parent
IBMSVC                      101.1             Volumes
11gR2                       101.21            Oracle
Oracle                      101.20            Volumes
Stretch-Cluster             101.23            RAC
RAC                         101.22            11gR2
[snip]
```

2.3.4 <instanceId> All volume objects

```
# declare and define the payload variable

payload = {}

# define the REST API call

REST = '/StorageCenter/StorageCenter/%s/VolumeList' %
       (scList['SC 9']['instanceId'])

# build the complete REST API URL

completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])

# execute REST API call via the HTTP GET method

json_data = connection.get(completeURL
                           ,headers=header
                           ,verify=verify_cert)
stdout = json.loads(json_data.text)
volList = {}
print "Name\t\t\t\tinstanceId\t\tPath"
for i in range(len(stdout)):
    print "%s\t\t%s\t\t%s" % (stdout[i]['name']
                             ,stdout[i]['instanceId']
                             ,stdout[i]['volumeFolderPath'])
    volList[stdout[i]['name']] = {}
    volList[stdout[i]['name']]['instanceId'] = stdout[i]['instanceId']
    volList[stdout[i]['name']]['path'] = stdout[i]['volumeFolderPath']
```

STDOUT

```
Name                        instanceId          Path
clapton-boot-vol            101.5500            Unix/Linux/SLES/clapton/
dean Volume 2               101.6568            Unix/Linux/Red_Hat/7.0/dean/
Beck-boot-vol               101.5501            Unix/Linux/SLES/beck/
dean Volume 3               101.6569            Unix/Linux/Red_Hat/7.0/dean/
Server_Pool-vol             101.5502            Oracle VM 3.1.3/
[snip]
```

2.3.5 <instanceId> All server folder objects

```
# declare and define the payload variable

payload = {}
```

```
# define the REST API call

REST = '/StorageCenter/StorageCenter/%s/ServerFolderList' %
       (scList['SC 9']['instanceId'])

# build the complete REST API URL

completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])

# execute REST API call via the HTTP GET method

json_data = connection.get(completeURL
                           ,headers=header
                           ,verify=verify_cert)
stdout = json.loads(json_data.text)
srvFolderList = {}
print "Name\t\t\t\tinstanceId\t\tParent"
for i in range(len(stdout)):
    if stdout[i]['name'] == "Servers":
        continue
    print "%s\t\t\t\t%s\t\t%s" % (stdout[i]['name']
                                 ,stdout[i]['instanceId']
                                 ,stdout[i]['parent']['instanceName']
                                 )
    srvFolderList[stdout[i]['name']] = {}
    srvFolderList[stdout[i]['name']]['instanceId'] = stdout[i]['instanceId']
    srvFolderList[stdout[i]['name']]['parent'] =
        stdout[i]['parent']['instanceName']
```

STDOUT

```
Name                        instanceId        Parent
Oracle                      101.5             Servers
HP-UX                       101.6             Unix
AIX                         101.11            Unix
Solaris                     101.12            Unix
OVM                         101.16            Oracle
[snip]
```

2.3.6 <instanceId> All server objects

```
# declare and define the payload variable

payload = {}

# define the REST API call

REST = '/StorageCenter/StorageCenter/%s/ServerList' %
       (scList['SC 9']['instanceId'])
```

```
# build the complete REST API URL

completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])

# execute REST API call via the HTTP GET method

json_data = connection.get(completeURL
                           ,headers=header
                           ,verify=verify_cert)
stdout = json.loads(json_data.text)
srvList = {}
print "Name\t\t\t\t\tinstanceId\t\tPath"
for i in range(len(stdout)):
    print "%s\t\t\t\t%s\t\t%s" % (stdout[i]['name']
                                 ,stdout[i]['instanceId']
                                 ,stdout[i]['serverFolderPath'])
    srvList[stdout[i]['instanceName']] = {}
    srvList[stdout[i]['instanceName']]['instanceId'] = stdout[i]['instanceId']
    srvList[stdout[i]['instanceName']]['path'] = stdout[i]['serverFolderPath']
```

STDOUT

```
Name                        instanceId        Path
Clapton                     101.139           Unix/Linux/
Theodore                    101.128           Unix/Linux/Red_Hat/
Cluster_LabFC_01            101.131           Unix/Linux/
Cluster_LabiSCSI_01         101.132           Unix/Linux/
rx2660                      101.74            Unix/HP-UX/
[snip]
```

2.3.7 <instanceId> All operating system types

```
# declare and define the payload variable

payload = {}

# define the REST API call

REST = '/StorageCenter/StorageCenter/%s/ServerOperatingSystemList' %
       (scList['SC 9']['instanceId'])

# build the complete REST API URL

completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
```

```
# execute REST API call via the HTTP GET method

json_data = connection.get(completeURL
                           ,headers=header
                           ,verify=verify_cert)
stdout = json.loads(json_data.text)
osList = {}
print "Name\t\t\t\tinstanceId\t\tscName"
for i in range(len(stdout)):
    print "%s\t\t\t\t%s\t\t%s" % (stdout[i]['instanceName']
                                 ,stdout[i]['instanceId']
                                 ,stdout[i]['scName'])
    osList[stdout[i]['instanceName']] = {}
    osList[stdout[i]['instanceName']]['instanceId'] = stdout[i]['instanceId']
    osList[stdout[i]['instanceName']]['scName'] = stdout[i]['scName']
```

STDOUT

```
Name                        instanceId        scName
Windows 2012                862.12            SC 4
Windows 2012 MPIO           862.13            SC 4
HP UX 11i v3                862.22            SC 4
AIX 7.1 MPIO                716.46            SC 18
Red Hat Linux 7.x           101.66            SC 9
[snip]
```

2.4 Putting it together

The remainder of section 2 presents several components of DSM REST API shown in earlier subsections of section 2.

A sample script is provided in section 5 that shows how these components can be assembled and used. For further information on the sample script, see section 5.

2.5 Console output

The control flow between the Python script and the DSM Data Collector happens with HTTP/REST API calls while the data flow exchange is contained within JSON-based data structures.

REST API calls to and from the DSM Data Collector can return either HTTP response codes or data.

A complete list of HTTP response codes are available at http://www.restapitutorial.com/httpstatuscodes.html.

REST API calls are implemented by the various methods discussed in this section, and each generates **STDOUT** to the console.

To execute the command and discard any return codes or data, run the following:

```
connection.get(completeURL, headers=header, verify=verify_cert)
```

To execute the command and print return codes or data to **STDOUT**, run the following:

```
print connection.post
          (completeURL
          ,data=json.dumps(payload
                          ,ensure_ascii=False).encode('utf-8')
          ,headers=header
          ,verify=verify_cert)
```

To execute the command and capture any return codes or data in the variable **json_data** for further parsing and analysis, run the following:

```
json_data = connection.post
                (completeURL
                ,data=json.dumps(payload
                                ,ensure_ascii=False).encode('utf-8')
                ,headers=header
                ,verify=verify_cert)

# stdout is returned as a JSON data structure and needs to be converted back
# to a Python dictionary data structure to extract its key value pairs

stdout = json.loads(json_data.text)
print stdout['instanceId']
```

2.5.1 HTTP response codes

Response codes are in a single line of data entry; the output here represents successful completion of the previously executed REST API call.

```
<Response [200]>
```

2.5.2 Python dictionary data structure

The following example data structure results from the prior command:

```
{u'instanceId': u'101.8683', u'scSerialNumber': 101, u'replicationSource':
False, u'liveVolume': False, u'vpdId': 8248, u'objectType': u'ScVolume',
u'volumeFolderPath': u'Unix/Linux/RestTest/', u'hostCacheEnabled': False,
u'inRecycleBin': False, u'instanceName': u'DSM_Rest_00', u'statusMessage  ':
u'', u'status': u'Down', u'storageType': {u'instanceId': u'101.1',
u'instanceName': u'Assigned - Redundant - 2 MB', u'objectType':
u'ScStorageType'}, u'cmmDestination': False, u'replicationDestination': False,
u'volumeFolder': {u'instanceId': u'101.221', u'instanceName': u'RestTest',
u'objectType': u'ScVolumeFolder'}, u'deviceId':
u'6000d31000065000000000000002038', u'active': False,
u'portableVolumeDestination': False, u'deleteAllowed': True, u'name':
u'DSM_Rest_00', u'scName': u'SC 9', u'secureDataUsed': False, u'serialNumber':
u'00000065-00002038', u'replayAllowed': False, u'flashOptimized': False,
u'configuredSize': u'10737418240 Bytes', u'mapped': False, u'cmmSource': False}
```

Note: The Python **pprint** module can also be used to reformat this data into more console-friendly output with the **import pprint** and **pprint.pprint(stdout)** statements.

3 Working with the SC Series array

The primary functions and syntax of script is contained between the open and close code segments discussed in sections 2.1 and 2.2. This paper provides simple guidance for the Python syntax to promote readability and understanding of the use of REST API calls with the DSM Data Collector.

The following code assumptions are made:

- Assumes a true or successful condition (return code) from each requested REST API call
- Contains minimal error trapping and management to promote readability and understanding
- Creates the data structures required to facilitate passing the data between REST API calls

3.1 Server folders

Server folders enable the ability to logically group server objects by name, purpose, or other defined criteria.

Table 1 Folder REST API calls used in this section

REST API	Method
/StorageCenter/ScServerFolder	POST
/StorageCenter/ScServerFolder/<instanceId>	PUT DELETE

3.1.1 Create a server folder

```
# create Storage Center server folder object managed by DSM / SC 9

payload = {}

# user-defined string / folder name

payload['Name'] = 'RestTest'

payload['StorageCenter'] = scList['SC 9']['instanceId']

# Storage Center instanceId + ".0" represents (/) the root level folder

payload['Parent'] = scList['SC 9']['instanceId'] + ".0"

# user-defined string / notes

payload['Notes'] = 'Created via REST API'

REST = '/StorageCenter/ScServerFolder'
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.post(completeURL
                            ,data=json.dumps(payload
                                             ,ensure_ascii=False).encode('utf-8')
                            ,headers=header
                            ,verify=verify_cert)
```

```
stdout = json.loads(json_data.text)
print stdout

#srvFolderList = {} created earlier in Section 2.3.4

srvFolderList[stdout['name']] = {}
srvFolderList[stdout['name']]['instanceId'] = stdout['instanceId']
srvFolderList[stdout['name']]['parent'] = 'Servers'
```

STDOUT

```
{u'status': u'Up', u'deleteAllowed': True, u'name': u'RestTest', u'parent':
{u'instanceId': u'101.0', u'instanceName': u'Servers', u'objectType':
u'ScServerFolder'}, u'instanceId': u'101.33', u'scName': u'SC 9', u'notes':
u'Created via REST API', u'scSerialNumber': 101, u'instanceName': u'RestTest',
u'okToDelete': True, u'folderPath': u'', u'root': False, u'statusMessage': u'',
u'objectType': u'ScServerFolder'}
```

3.1.2 Modify a server folder

The following code can be used to either modify the name or the parent folder of the server folder object or to modify both concurrently. The **payload['Name']** and **payload['Parent']** attributes both contain string-based values where the former represents the desired name of the folder object and the latter represents the instanceId value of the parent folder.

```
# rename server folder object

payload = {}

# user-defined string / new folder name

payload['Name'] = 'RestTest_001'

# Storage Center instanceId + ".0" represents (/) the root level folder

payload['Parent'] = scList['SC 9']['instanceId'] + ".0"

REST = '/StorageCenter/ScServerFolder/%s' %
       srvFolderList['RestTest']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.put(completeURL
                           ,data=json.dumps(payload
                                            ,ensure_ascii=False).encode('utf-8')
                           ,headers=header
                           ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout
srvFolderList[payload['Name']] = {}
```

```
# renamed folder retains the same instanceID value as the original folder

srvFolderList[payload['Name']]['instanceId'] = stdout['instanceId']

# remove duplicate list entry

del srvFolderList['RestTest']
```

STDOUT

```
{u'status': u'Up', u'deleteAllowed': True, u'name': u'RestTest_001', u'parent':
{u'instanceId': u'101.0', u'instanceName': u'Servers', u'objectType':
u'ScServerFolder'}, u'instanceId': u'101.33', u'scName': u'SC 9', u'notes':
u'Created via REST API', u'scSerialNumber': 101, u'instanceName':
u'RestTest_001', u'okToDelete': True, u'folderPath': u'', u'root': False,
u'statusMessage': u'', u'objectType': u'ScServerFolder'}
```

3.1.3 Delete a server folder

```
# remove server folder object

payload = {}
REST = '/StorageCenter/ScServerFolder/%s' %
       srvFolderList['RestTest_001']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.delete(completeURL
                              ,headers=header
                              ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout

del srvFolderList['RestTest_001']    # remove list entry
```

STDOUT

```
{u'result': True}
```

3.2 Servers

This section presents the REST API calls used to create, manage, and remove SC Series server objects.

Table 2 Server REST API calls used in this section

REST API	Method
/StorageCenter/ScPhysicalServer	POST
/StorageCenter/ScPhysicalServer /<instanceId>	PUT
/StorageCenter/ScServer/<instanceId>	DELETE
/StorageCenter/ScPhysicalServer/<instanceId>/AddHba	POST
/StorageCenter/ScPhysicalServer/<instanceId>/AddToCluster	POST
/StorageCenter/ScPhysicalServer/<instanceId>/RemoveFromCluster	POST

3.2.1 Create an FC or iSCSI server

```
# create a new server object

payload = {}
payload['Name'] = 'theodore'
payload['OperatingSystem'] = osList['Red Hat Linux 6.x']['instanceId']
payload['StorageCenter'] = scList['SC 9']['instanceId']
payload['ServerFolder'] = srvFolderList['RestTest']['instanceId']
payload['Notes'] = 'Created via REST API'
REST = '/StorageCenter/ScPhysicalServer'
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.post(completeURL
                            ,data=json.dumps(payload
                                             ,ensure_ascii=False).encode('utf-8')
                            ,headers=header
                            ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout
srvList[payload['Name']] = {}
srvList[payload['Name']]['instanceId'] = stdout['instanceId']
srvList[payload['Name']]['path'] = stdout['serverFolderPath']
srvList[payload['Name']]['scName'] = stdout['scName']
```

STDOUT

```
{u'volumeCount': 0, u'removeHbasAllowed': False, u'instanceName': u'theodore',
u'alertOnConnectivity': True, u'objectType': u'ScPhysicalServer', u'type':
u'Physical', u'instanceId': u'101.204', u'childStatus': u'Up',
u'serverFolderPath': u'RestTest/', u'portType': [], u'hbaCount': 0,
u'statusMessage': u'No paths could be found', u'status': u'Down',
u'scSerialNumber': 101, u'serverFolder': {u'instanceId': u'101.49',
u'instanceName': u'RestTest', u'objectType': u'ScServerFolder'},
```

```
u'connectivity': u'Down', u'alertOnPartialConnectivity': True, u'deleteAllowed':
True, u'pathCount': 0, u'name': u'theodore', u'hbaPresent': False,
u'connectedToAllControllers': True, u'scName': u'SC 9', u'notes': u'Created via
REST API', u'mapped': False, u'operatingSystem': {u'instanceId': u'101.30',
u'instanceName': u'Red Hat Linux 6.x', u'objectType':
u'ScServerOperatingSystem'}}
```

3.2.2 Add an HBA port to a server

The **payload['HbaPortType']** attribute contains EnumValueName values as detailed in Figure 1.

Name	Visibility	UI Name	Description
FibreChannel	Public	Fibre Channel	Fibre Channel
Iscsi	Public	iSCSI	iSCSI
Sas	Public	SAS	SAS as a front end transport
Unknown	Public	Unknown	The transport type is unknown

Figure 1 EnumValueName values in the **payload['HbaPortType']** attribute

```
# add single FC WWPN to server object

payload = {}
payload['HbaPortType'] = 'FibreChannel'
payload['WwnOrIscsiName'] = '21000024FF27DBCC'
REST = '/StorageCenter/ScPhysicalServer/%s/AddHba' %
       srvList['theodore']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.post(completeURL
                            ,data=json.dumps(payload
                                             ,ensure_ascii=False).encode('utf-8')
                            ,headers=header
                            ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout
```

STDOUT

```
{u'portWwnList': [u'21000024FF27DBCC'], u'iscsiIpAddress': u'0.0.0.0',
u'pathCount': 4, u'name': u'21000024FF27DBCC', u'remoteStorageCenter': False,
u'instanceId': u'101.3303892443', u'scName': u'SC 9', u'notes': u'',
u'scSerialNumber': 101, u'server': {u'instanceId': u'101.204', u'instanceName':
u'theodore', u'objectType': u'ScPhysicalServer'}, u'connectivity': u'Up',
u'iscsiName': u'', u'portType': u'FibreChannel', u'instanceName':
u'21000024FF27DBCC', u'objectType': u'ScServerHba'}
```

PortType Enum Summary

For the latest enumerated field values, refer to the REST API help files located in section 1.1.

3.2.3 Create a server cluster

```
# create a server cluster which contains server objects

payload = {}
payload['Name'] = 'RestTest_SrvCluster'
payload['OperatingSystem'] = osList['Red Hat Linux 6.x']['instanceId']
payload['StorageCenter'] = scList['SC 9']['instanceId']
payload['ServerFolder'] = srvFolderList['RestTest']['instanceId']
payload['Notes'] = 'Created via REST API'
REST = '/StorageCenter/ScServerCluster'
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.post(completeURL
                            ,data=json.dumps(payload
                                             ,ensure_ascii=False).encode('utf-8')
                            ,headers=header
                            ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout
srvClusterList = {}
srvClusterList[payload['Name']] = {}
srvClusterList[payload['Name']]['instanceId'] = stdout['instanceId']
```

STDOUT

```
{u'volumeCount': 0, u'removeHbasAllowed': False, u'instanceName':
u'RestTest_SrvCluster', u'alertOnConnectivity': True, u'objectType':
u'ScServerCluster', u'type': u'Cluster', u'instanceId': u'101.207',
u'childStatus': u'Up', u'serverFolderPath': u'RestTest/', u'portType': [],
u'hbaCount': 0, u'statusMessage': u'', u'status': u'Up', u'scSerialNumber': 101,
u'serverFolder': {u'instanceId': u'101.50', u'instanceName': u'RestTest',
u'objectType': u'ScServerFolder'}, u'connectivity': u'Up',
u'alertOnPartialConnectivity': True, u'deleteAllowed': True, u'pathCount': 0,
u'name': u'RestTest_SrvCluster', u'hbaPresent': False,
u'connectedToAllControllers': True, u'scName': u'SC 9', u'notes': u'Created via
REST API', u'mapped': False, u'operatingSystem': {u'instanceId': u'101.30',
u'instanceName': u'Red Hat Linux 6.x', u'objectType':
u'ScServerOperatingSystem'}}
```

3.2.4 Add a server to a server cluster

```
# add new server object to a server cluster folder

payload = {}
REST = '/StorageCenter/ScPhysicalServer/%s/AddToCluster' %
       srvClusterList['RestTest_SrvCluster']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
print connection.post(completeURL
                      ,data=json.dumps(payload
                                       ,ensure_ascii=False).encode('utf-8')
                      ,headers=header
                      ,verify=verify_cert)
```

STDOUT

```
<Response [204]>
```

3.2.5 Remove a server from a server cluster

```
# remove server object from a server cluster folder

payload = {}
REST = '/StorageCenter/ScPhysicalServer/%s/RemoveFromCluster' %
       srvClusterList['RestTest_SrvCluster']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
print connection.post(completeURL
                      ,data=json.dumps(payload
                                       ,ensure_ascii=False).encode('utf-8')
                      ,headers=header
                      ,verify=verify_cert)
```

STDOUT

```
<Response [204]>
```

3.2.6 Remove a server

```
# remove a server object

payload = {}
REST = '/StorageCenter/ScServer/%s' % srvList['theodore']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
print connection.delete(completeURL, headers=header, verify=verify_cert)
```

STDOUT

```
<Response [200]>
```

3.3 Volume folders

Volume folders enable the ability to logically group volume objects by name, purpose, size, or other defined criteria.

Table 3 Volumes folder REST API calls used in this section

REST API	Method
/StorageCenter/ScVolumeFolder	POST
/StorageCenter/ScVolumeFolder/<instanceId>>	PUT DELETE

3.3.1 Create a volume folder

```
# create Storage Center volume folder object managed by DSM / SC 9

payload = {}

# user-defined string / folder name

payload['Name'] = 'RestTest'
payload['StorageCenter'] = scList['SC 9']['instanceId']

# Storage Center instanceId + ".0" represents (/) the root level folder

payload['Parent'] = scList['SC 9']['instanceId'] + ".0"

# user-defined string / notes

payload['Notes'] = 'Created via REST API'

REST = '/StorageCenter/ScVolumeFolder'
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.post(completeURL
                            ,data=json.dumps(payload
                                             ,ensure_ascii=False).encode('utf-8')
                            ,headers=header
                            ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout

#volFolderList = {} created earlier in Section 2.3.3

volFolderList[stdout['name']] = {}
volFolderList[stdout['name']]['instanceId'] = stdout['instanceId']
volFolderList[stdout['name']]['instanceId'] = 'Volumes'
```

STDOUT

```
{u'status': u'Up', u'deleteAllowed': True, u'instanceName': u'RestTest',
u'name': u'RestTest', u'parent': {u'instanceId': u'101.0', u'instanceName':
u'Volumes', u'objectType': u'ScVolumeFolder'}, u'instanceId': u'101.101',
u'scName': u'SC 9', u'notes': u'Created via REST API', u'scSerialNumber': 101,
u'parentIndex': 0, u'okToDelete': True, u'folderPath': u'', u'root': False,
u'statusMessage': u'', u'objectType': u'ScVolumeFolder'}
```

3.3.2 Modify a volume folder

The following code can be used to modify either the name or the parent folder of the volume folder object or to modify both concurrently. The **payload['Name']** and **payload['Parent']** attributes both contain string-based values where the former represents the desired name of the folder object while the latter represents the instanceId value of the parent folder.

```
# rename volume folder object

payload = {}

# User-defined string / new folder name

payload['Name'] = 'RestTest_001'


# Storage Center instanceId + ".0" represents (/), i.e. the root level folder

payload['Parent'] = scList['SC 9']['instanceId'] + ".0"

REST = '/StorageCenter/ScVolumeFolder/%s' %
       volFolderList['RestTest']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.put(completeURL
                           ,data=json.dumps(payload
                                            ,ensure_ascii=False).encode('utf-8')
                           ,headers=header
                           ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout

volFolderList[payload['Name']] = {}

# renamed folder retains the same instanceID value as the original folder

volFolderList[payload['Name']]['instanceId'] = stdout['instanceId']

del volFolderList['RestTest']    # remove duplicate list entry
```

STDOUT

```
{u'status': u'Up', u'deleteAllowed': True, u'instanceName': u'RestTest_001',
u'name': u'RestTest_001', u'parent': {u'instanceId': u'101.0', u'instanceName':
u'Volumes', u'objectType': u'ScVolumeFolder'}, u'instanceId': u'101.101',
u'scName': u'SC 9', u'notes': u'Created via REST API', u'scSerialNumber': 101,
u'parentIndex': 0, u'okToDelete': True, u'folderPath': u'', u'root': False,
u'statusMessage': u'', u'objectType': u'ScVolumeFolder'}
```

3.3.3 Delete a volume folder

```
# remove volume folder object

payload = {}
REST = '/StorageCenter/ScVolumeFolder/%s' %
       volFolderList['RestTest_001']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.delete(completeURL
                              ,headers=header
                              ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout
del volFolderList['RestTest_001']    # remove list entry
```

STDOUT

```
{u'result': True}
```

3.4 Volumes

This section discusses the use of the REST API calls to create, manage and remove SC Series volume objects.

Table 4 Volume REST API calls used in this section.

REST API	Method
/StorageCenter/ScVolume	POST
/StorageCenter/ScVolume/<instanceId>	PUT DELETE
/StorageCenter/ScVolume/<instanceId>/MapToServer	POST
/StorageCenter/ScVolume/<instanceId>/Expand	POST
/StorageCenter/ScVolume/<instanceId>/ExpandToSize	POST
/StorageCenter/ScVolume/<instanceId>/PreallocateStorage	POST
/StorageCenter/ScVolume/<instanceId>/Unmap	POST
/StorageCenter/ScVolume/<instanceId>/Recycle	POST

3.4.1 Create a volume

```
# create Storage Center volume object managed by DSM / SC 9

payload = {}
payload['Name'] = 'RestTest_Vol'
payload['Size'] = '10GB'
payload['StorageCenter'] = scList['SC 9']['instanceId']
payload['VolumeFolder'] = volFolderList['RestTest']['instanceId']
payload['Notes'] = 'Created via REST API'
REST = '/StorageCenter/ScVolume'
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.post(completeURL
                            ,data=json.dumps(payload
                                             ,ensure_ascii=False).encode('utf-8')
                            ,headers=header
                            ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout
volList[stdout['name']] = {}
volList[stdout['name']]['instanceId'] = stdout['instanceId']
volList[stdout['name']]['parent'] = stdout['volumeFolderPath']
volList[stdout['name']]['deviceId'] = stdout['deviceId']
volList[stdout['name']]['size'] = stdout['configuredSize']
```

STDOUT

```
{u'instanceId': u'101.8703', u'scSerialNumber': 101, u'replicationSource':
False, u'liveVolume': False, u'vpdId': 8263, u'objectType': u'ScVolume',
u'volumeFolderPath': u'RestTest/', u'hostCacheEnabled': False, u'inRecycleBin':
False, u'instanceName': u'RestTest_Vol', u'statusMessage': u'', u'status':
u'Down', u'storageType': {u'instanceId': u'101.1', u'instanceName': u'Assigned -
Redundant - 2 MB', u'objectType': u'ScStorageType'}, u'cmmDestination': False,
u'replicationDestination': False, u'volumeFolder': {u'instanceId': u'101.223',
u'instanceName': u'RestTest', u'objectType': u'ScVolumeFolder'}, u'deviceId':
u'6000d31000006500000000000002047', u'active': False,
u'portableVolumeDestination': False, u'deleteAllowed': True, u'name':
u'RestTest_Vol', u'scName': u'SC 9', u'secureDataUsed': False, u'serialNumber':
u'00000065-00002047', u'replayAllowed': False, u'flashOptimized': False,
u'configuredSize': u'10737418240 Bytes', u'mapped': False, u'cmmSource': False}
```

3.4.2 Map a volume to a server

```
# map a volume object to a server

payload = {}
payload['Server'] = srvList['clapton']['instanceId']
REST = '/StorageCenter/ScVolume/%s/MapToServer' %
       volList['RestTest_Vol']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.post(completeURL
                            ,data=json.dumps(payload
                                             ,ensure_ascii=False).encode('utf-8')
                            ,headers=header
                            ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout
```

STDOUT

```
{u'instanceId': u'101.11236', u'scName': u'SC 9', u'scSerialNumber': 101,
u'controller': {u'instanceId': u'101.102', u'instanceName': u'SN 102',
u'objectType': u'ScController'}, u'lunUsed': [1], u'mapCount': 4, u'volume':
{u'instanceId': u'101.8707', u'instanceName': u'RestTest_Vol', u'objectType':
u'ScVolume'}, u'connectivity': u'Up', u'readOnly': False, u'objectType':
u'ScMappingProfile', u'hostCache': False, u'mappedVia': u'Server', u'server':
{u'instanceId': u'101.139', u'instanceName': u'clapton', u'objectType':
u'ScPhysicalServer'}, u'instanceName': u'8707-139', u'lunRequested': u'N/A'}
```

3.4.3 Expand a volume

3.4.3.1 Increase capacity

```
# expand a volume object by capacity

payload = {}
payload['ExpandAmount'] = '10GB'
REST = '/StorageCenter/ScVolume/%s/Expand' %
       volList['RestTest_Vol']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.post(completeURL
                            ,data=json.dumps(payload
                                             ,ensure_ascii=False).encode('utf-8')
                            ,headers=header
                            ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout
volList[stdout['instanceName']]['size'] = stdout['configuredSize']
```

STDOUT

```
{u'instanceId': u'101.8704', u'scSerialNumber': 101, u'replicationSource':
False, u'liveVolume': False, u'vpdId': 8264, u'objectType': u'ScVolume',
u'volumeFolderPath': u'RestTest/', u'hostCacheEnabled': False, u'inRecycleBin':
False, u'instanceName': u'RestTest_Vol', u'statusMessage': u'', u'status':
u'Down', u'storageType': {u'instanceId': u'101.1', u'instanceName': u'Assigned -
Redundant - 2 MB', u'objectType': u'ScStorageType'}, u'cmmDestination': False,
u'replicationDestination': False, u'volumeFolder': {u'instanceId': u'101.224',
u'instanceName': u'RestTest', u'objectType': u'ScVolumeFolder'}, u'deviceId':
u'6000d31000006500000000000000002048', u'active': False,
u'portableVolumeDestination': False, u'deleteAllowed': True, u'name':
u'RestTest_Vol', u'scName': u'SC 9', u'secureDataUsed': False, u'serialNumber':
u'00000065-00002048', u'replayAllowed': False, u'flashOptimized': False,
u'configuredSize': u'21474836480 Bytes', u'mapped': False, u'cmmSource': False}
```

3.4.3.2 Expand to a specific size

```
# expand a volume object to size

payload = {}
payload['NewSize'] = '20GB'
REST = '/StorageCenter/ScVolume/%s/ExpandToSize' %
       volList['RestTest_Vol']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[
json_data = connection.post(completeURL
                            ,data=json.dumps(payload
                                             ,ensure_ascii=False).encode('utf-8')
                            ,headers=header
                            ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout
volList[stdout['instanceName']]['size'] = stdout['configuredSize']
```

STDOUT

```
{u'instanceId': u'101.8704', u'scSerialNumber': 101, u'replicationSource':
False, u'liveVolume': False, u'vpdId': 8264, u'objectType': u'ScVolume',
u'volumeFolderPath': u'RestTest/', u'hostCacheEnabled': False, u'inRecycleBin':
False, u'instanceName': u'RestTest_Vol', u'statusMessage': u'', u'status':
u'Down', u'storageType': {u'instanceId': u'101.1', u'instanceName': u'Assigned -
Redundant - 2 MB', u'objectType': u'ScStorageType'}, u'cmmDestination': False,
u'replicationDestination': False, u'volumeFolder': {u'instanceId': u'101.224',
u'instanceName': u'RestTest', u'objectType': u'ScVolumeFolder'}, u'deviceId':
u'6000d31000006500000000000000002048', u'active': False,
u'portableVolumeDestination': False, u'deleteAllowed': True, u'name':
u'RestTest_Vol', u'scName': u'SC 9', u'secureDataUsed': False, u'serialNumber':
u'00000065-00002048', u'replayAllowed': False, u'flashOptimized': False,
u'configuredSize': u'21474836480 Bytes', u'mapped': False, u'cmmSource': False}
```

3.4.4 Pre-allocate a volume

The volume pre-allocation is successful only if the volume has been and is currently mapped to a server object within the same SC Series array.

```
# preallocate a volume object

payload = {}
REST = '/StorageCenter/ScVolume/%s/PreallocateStorage' %
       volList['RestTest_Vol']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.post(completeURL
                            ,data=json.dumps(payload
                                             ,ensure_ascii=False).encode('utf-8')
                            ,headers=header
                            ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout
```

STDOUT

```
{u'instanceId': u'101.8706', u'scSerialNumber': 101, u'replicationSource':
False, u'liveVolume': False, u'vpdId': 8266, u'objectType': u'ScVolume',
u'volumeFolderPath': u'RestTest/', u'hostCacheEnabled': False, u'inRecycleBin':
False, u'instanceName': u'RestTest_Vol', u'statusMessage': u'', u'status':
u'Up', u'storageType': {u'instanceId': u'101.1', u'instanceName': u'Assigned -
Redundant - 2 MB', u'objectType': u'ScStorageType'}, u'cmmDestination': False,
u'replicationDestination': False, u'volumeFolder': {u'instanceId': u'101.226',
u'instanceName': u'RestTest', u'objectType': u'ScVolumeFolder'}, u'deviceId':
u'6000d31000065000000000000000204a', u'active': True,
u'portableVolumeDestination': False, u'deleteAllowed': True, u'name':
u'RestTest_Vol', u'scName': u'SC 9', u'secureDataUsed': False, u'serialNumber':
u'00000065-0000204a', u'replayAllowed': True, u'flashOptimized': False,
u'configuredSize': u'21474836480 Bytes', u'mapped': True, u'cmmSource': False}
```

3.4.5 Modify a volume

The following code can be used exclusively to modify the name or the parent folder of the volume object or both concurrently. The **payload['Name']** and **payload['VolumeFolder']** attributes both contain string-based values where the former represents the desired name of the volume object while the latter represents the instanceId value of the parent folder.

```
# modify a volume object

payload = {}
payload['Name'] = 'RestTest_Vol_renamed'
payload['VolumeFolder'] = volFolderList['RestTest']['instanceId']
REST = '/StorageCenter/ScVolume/%s' % volList['RestTest_Vol']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.put(completeURL
                           ,data=json.dumps(payload
                                            ,ensure_ascii=False).encode('utf-8')
                           ,headers=header
                           ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout
volFolderList[payload['Name']] = {}
volFolderList[payload['Name']]['instanceId'] = stdout['instanceId']
del volList['RestTest_Vol']    # remove list entry
```

STDOUT

```
{u'instanceId': u'101.8704', u'scSerialNumber': 101, u'replicationSource':
False, u'liveVolume': False, u'vpdId': 8264, u'objectType': u'ScVolume',
u'volumeFolderPath': u'RestTest/', u'hostCacheEnabled': False, u'inRecycleBin':
False, u'instanceName': u'RestTest_Vol_renamed', u'statusMessage': u'',
u'status': u'Down', u'storageType': {u'instanceId': u'101.1', u'instanceName':
u'Assigned - Redundant - 2 MB', u'objectType': u'ScStorageType'},
u'cmmDestination': False, u'replicationDestination': False, u'volumeFolder':
{u'instanceId': u'101.224', u'instanceName': u'RestTest', u'objectType':
u'ScVolumeFolder'}, u'deviceId': u'6000d31000006500000000000002048', u'active':
False, u'portableVolumeDestination': False, u'deleteAllowed': True, u'name':
u'RestTest_Vol_renamed', u'scName': u'SC 9', u'secureDataUsed': False,
u'serialNumber': u'00000065-00002048', u'replayAllowed': False,
u'flashOptimized': False, u'configuredSize': u'21474836480 Bytes', u'mapped':
False, u'cmmSource': False}
```

3.4.6 Unmap a volume

```
# unmap a volume object from a server

payload = {}
REST = '/StorageCenter/ScVolume/%s/Unmap' %
       volList['RestTest_Vol']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
print connection.post(completeURL
                      ,data=json.dumps(payload
                                       ,ensure_ascii=False).encode('utf-8')
                      ,headers=header
                      ,verify=verify_cert)
```

STDOUT

```
<Response [204]>
```

3.4.7 Delete a volume

A volume object can either be placed in the recycle bin (it can be restored at a later time) or the volume object can be permanently deleted using the following details.

3.4.7.1 Move to recycle bin

```
# recycle a volume object

payload = {}
REST = '/StorageCenter/ScVolume/%s/Recycle' %
       volList['RestTest_Vol_renamed']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.post(completeURL
                            ,data=json.dumps(payload
                                             ,ensure_ascii=False).encode('utf-8')
                            ,headers=header
                            ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout
```

STDOUT

```
{u'result': True}
```

3.4.7.2 Remove permanently

```
# remove a volume object

payload = {}
REST = '/StorageCenter/ScVolume/%s' %
       volList['RestTest_Vol_renamed']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.delete(completeURL
                              ,headers=header
                              ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout
del volList['RestTest_Vol_renamed']    # remove list entry
```

STDOUT

```
{u'result': True}
```

3.5 Storage profiles

This section discusses the use of REST API calls to create, manage, and remove storage profile objects on an SC Series array.

Table 5 Security profile REST API calls used in this section

REST API	Method
/StorageCenter/ScStorageProfile	POST
/StorageCenter/ScStorageProfile/<instanceId>	PUT DELETE
/StorageCenter/ScStorageProfile/GetList	POST

3.5.1 Create a storage profile

The **payload['RaidTypeUsed']** attribute contains the EnumValueName values in Figure 2.

Name	Visibility	UI Name	Description
Mixed	Public	Mixed	Indicates that a mix of RAID 10 and RAID 5/RAID 6 stor RAID 10 Storage will be used for writable data in each data in each selected storage tier, dependent upon the dependent upon the stripe width selection in the ScStor
Raid10Only	Public	RAID 10 only	Indicates that only RAID 10 storage should be used RAID 10 Storage will be used for writable and historica
Raid5Or6Only	Public	RAID 5/RAID 6 only	Indicates that only RAID 5/RAID 6 storage should be us Either RAID 5 or RAID 6 storage will be used for writabl redundancy level of the tier. The stripe width used for R ScStorageSettings object.

Figure 2 Storage profiles

The additional code segment included below captures all storage profile information from all DSM-managed SC Series arrays and populates the Python dictionary **spList**.

```
# create a new named storage profile object

payload = {}
payload['Name'] = 'RestTest_sp'
payload['RaidTypeUsed'] = 'Mixed'
payload['StorageCenter'] = scList['SC 9']['instanceId']
payload['UseTier1Storage'] = True
payload['UseTier2Storage'] = True
payload['UseTier3Storage'] = True
payload['Notes'] = 'Created via REST API'
REST = '/StorageCenter/ScStorageProfile'
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.post(completeURL
                            ,data=json.dumps(payload
```

```
                                                    ,ensure_ascii=False).encode('utf-8')
                                ,headers=header
                                ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout

# capture all storage profile info managed by DSM into spList

payload = {}
REST = '/StorageCenter/ScStorageProfile/GetList'
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.post(completeURL
                                ,data=json.dumps(payload
                                                    ,ensure_ascii=False).encode('utf-8')
                                ,headers=header
                                ,verify=verify_cert)
stdout = json.loads(json_data.text)
spList = {}
for i in range(len(stdout)):
    spList[stdout[i]['instanceName']] = {}
    spList[stdout[i]['instanceName']]['instanceId'] = stdout[i]['instanceId']
    spList[stdout[i]['instanceName']]['scName'] = stdout[i]['scName']
```

STDOUT

```
{u'instanceName': u'RestTest_sp', u'useTier3Storage': True, u'name':
u'RestTest_sp', u'volumeCount': 0, u'scName': u'SC 9', u'notes': u'Created via
REST API', u'scSerialNumber': 101, u'userCreated': True, u'useTier2Storage':
True, u'instanceId': u'101.14', u'raidTypeUsed': u'Mixed',
u'allowedForNonFlashOptimized': True, u'objectType': u'ScStorageProfile',
u'allowedForFlashOptimized': True, u'tiersUsedDescription': u'Tier 1, Tier 2,
Tier 3', u'useTier1Storage': True, u'raidTypeDescription': u'RAID 10 Active,
RAID 5 Replay'}
```

RaidTypeUsed Enum Summary

For the latest enumerated field values, refer to the REST API help files located in section 1.1.

3.5.2 Modify a storage profile

The following code can be used to modify either a single attribute or multiple attributes of a storage profile object concurrently. This code segment cannot be used to change the SC Series attribute.

```
# modify a storage profile object identified by instanceId

payload = {}
payload['Name'] = 'RestTest_sp_renamed'
payload['RaidTypeUsed'] = 'Mixed'
payload['UseTier1Storage'] = True
payload['UseTier2Storage'] = True
```

```
payload['UseTier3Storage'] = True
REST = '/StorageCenter/ScStorageProfile/%s' %
       spList['RestTest_sp']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.put(completeURL
                           ,data=json.dumps(payload
                                            ,ensure_ascii=False).encode('utf-8')
                           ,headers=header
                           ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout
```

STDOUT

```
{u'instanceName': u'RestTest_sp_renamed', u'useTier3Storage': True, u'name':
u'RestTest_sp_renamed', u'volumeCount': 0, u'scName': u'SC 9', u'notes':
u'Created via REST API', u'scSerialNumber': 101, u'userCreated': True,
u'useTier2Storage': True, u'instanceId': u'101.14', u'raidTypeUsed': u'Mixed',
u'allowedForNonFlashOptimized': True, u'objectType': u'ScStorageProfile',
u'allowedForFlashOptimized': True, u'tiersUsedDescription': u'Tier 1, Tier 2,
Tier 3', u'useTier1Storage': True, u'raidTypeDescription': u'RAID 10 Active,
RAID 5 Replay'}
```

3.5.3 Delete a storage profile

```
# delete a named storage profile object

payload = {}
REST = '/StorageCenter/ScStorageProfile/%s' %
       spList['RestTest_sp']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.delete(completeURL
                              ,headers=header
                              ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout
```

STDOUT

```
{u'result': True}
```

3.6 Replays

This section provides the REST API calls used to create, manage, and remove Replays (also known as snapshots) and view volume objects on an SC Series array.

Table 6 Replay REST API calls used in this section

REST API	Method
/StorageCenter/ScVolume/<instanceId>/CreateReplay	POST
/StorageCenter/ScReplay/GetList	POST
/StorageCenter/ScReplay/<instanceId>/CreateView	POST
/StorageCenter/ScReplay/<instanceId>	PUT
/StorageCenter/ScReplay/<instanceId>/Expire	POST

3.6.1 Create a Replay

The creation of a Replay is based on the ScVolume REST API call and the RestTest folder and volume created in sections 3.4.1 and 3.3.1, respectively, with the assumption that neither object has been deleted. An additional code segment is included here to capture all Replay information from all DSM-managed SC Series arrays and populate the Python dictionary **rplyList**.

The volume Replay creation is successful only if the volume is mapped to a server object within the same SC Series array.

```
# create a replay object from RestTest_Vol volume object

payload = {}

# Description has a 30 char field len

payload['Description'] = 'RestTest_Vol_replay'

# ExpireTime in minutes, 0 = Never Expire

payload['ExpireTime'] = '15'

REST = '/StorageCenter/ScVolume/%s/CreateReplay' %
       volList['RestTest_Vol']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.post(completeURL
                            ,data=json.dumps(payload
                                             ,ensure_ascii=False).encode('utf-8')
                            ,headers=header
                            ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout
rplyList = {}
rplyList[stdout['createVolume']['instanceName']] = {}
rplyList[stdout['createVolume']['instanceName']]['instanceId'] =
    stdout['instanceId']
```

```
rplyList[stdout['createVolume']['instanceName']]['description'] =
    payload['Description']
rplyList[stdout['createVolume']['instanceName']]['scName'] =
    stdout['scName']

# capture all storage profile info managed by DSM into rplyList

payload = {}
REST = '/StorageCenter/ScReplay/GetList'
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.post(completeURL
                            ,data=json.dumps(payload
                                             ,ensure_ascii=False).encode('utf-8')
                            ,headers=header
                            ,verify=verify_cert)
stdout = json.loads(json_data.text)

#rplyList = {} created earlier in Section 3.6.1

for i in range(len(stdout)):
    rplyList[stdout[i]['createVolume']['instanceName']] = {}
    rplyList[stdout[i]['createVolume']['instanceName']]['instanceId'] =
        stdout[i]['instanceId']
    rplyList[stdout[i]['createVolume']['instanceName']]['description'] =
        stdout[i]['description']
    rplyList[stdout[i]['createVolume']['instanceName']]['scName'] =
        stdout[i]['scName']
```

STDOUT

```
{u'scSerialNumber': 101, u'globalIndex': u'101-8739-1', u'description':
u'RestTest_Vol_replay', u'spaceRecovery': False, u'instanceId': u'101.8739.1',
u'scName': u'SC 9', u'consistent': False, u'expires': True, u'freezeTime':
u'2016-05-06T12:52:59-05:00', u'createVolume': {u'instanceId': u'101.8739',
u'instanceName': u'RestTest_Vol', u'objectType': u'ScVolume'}, u'expireTime':
u'2016-05-06T13:07:59-05:00', u'source': u'Manual', u'writesHeldDuration': 5937,
u'active': False, u'markedForExpiration': False, u'objectType': u'ScReplay',
u'instanceName': u'05/06/2016 12:52:59 PM', u'size': u'0 Bytes'}
```

3.6.2 Create a View Volume from a Replay

```
# create a View Volume from a Replay

payload = {}
payload['Name'] = 'RestTest_Vol_replay_View'
payload['VolumeFolder'] = volFolderList['RestTest']['instanceId']
REST = '/StorageCenter/ScReplay/%s/CreateView' %
       rplyList['RestTest_Vol']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
```

```
json_data = connection.post(completeURL
                            ,data=json.dumps(payload
                                             ,ensure_ascii=False).encode('utf-8')
                            ,headers=header
                            ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout
```

STDOUT

```
{u'instanceId': u'101.8740', u'scSerialNumber': 101, u'replicationSource':
False, u'liveVolume': False, u'vpdId': 8298, u'objectType': u'ScVolume',
u'volumeFolderPath': u'RestTest/', u'hostCacheEnabled': False, u'childStatus':
u'Up', u'inRecycleBin': False, u'instanceName': u'RestTest_Vol_replay_View',
u'statusMessage': u'', u'status': u'Up', u'storageType': {u'instanceId':
u'101.1', u'instanceName': u'Assigned - Redundant - 2 MB', u'objectType':
u'ScStorageType'}, u'cmmDestination': False, u'replicationDestination': False,
u'volumeFolder': {u'instanceId': u'101.255', u'instanceName': u'RestTest',
u'objectType': u'ScVolumeFolder'}, u'deviceId':
u'6000d31000006500000000000000206a', u'active': True,
u'portableVolumeDestination': False, u'deleteAllowed': True, u'name':
u'RestTest_Vol_replay_View', u'scName': u'SC 9', u'secureDataUsed': False,
u'serialNumber': u'00000065-0000206a', u'replayAllowed': True,
u'flashOptimized': False, u'configuredSize': u'10737418240 Bytes', u'mapped':
False, u'cmmSource': False}
```

3.6.3 Modify Replay expiration

```
# modify a Replay and set Replay to never expire

payload = {}
payload['Description'] = 'RestTest_Vol_replay'
payload['ExpireTime'] = '0'                    # set replay to never expire
REST = '/StorageCenter/ScReplay/%s' % rplyList['RestTest_Vol']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.put(completeURL
                           ,data=json.dumps(payload
                                            ,ensure_ascii=False).encode('utf-8')
                           ,headers=header
                           ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout
```

STDOUT

```
{u'scSerialNumber': 101, u'globalIndex': u'101-8739-1', u'description':
u'RestTest_Vol_replay', u'spaceRecovery': False, u'instanceId': u'101.8739.1',
u'scName': u'SC 9', u'consistent': False, u'expires': False, u'freezeTime':
u'2016-05-06T12:52:59-05:00', u'createVolume': {u'instanceId': u'101.8739',
u'instanceName': u'RestTest_Vol', u'objectType': u'ScVolume'}, u'expireTime':
u'1969-12-31T18:00:00-06:00', u'source': u'Manual', u'writesHeldDuration': 5937,
u'active': False, u'markedForExpiration': False, u'objectType': u'ScReplay',
u'instanceName': u'05/06/2016 12:52:59 PM', u'size': u'0 Bytes'}
```

3.6.4 Expire a Replay

```
# expire a Replay

payload = {}
REST = '/StorageCenter/ScReplay/%s/Expire' %
       rplyList['RestTest_Vol_replay']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
print connection.post(completeURL
                      ,data=json.dumps(payload
                                       ,ensure_ascii=False).encode('utf-8')
                      ,headers=header
                      ,verify=verify_cert)
```

STDOUT

```
<Response [204]>
```

3.7 Replay profiles

This section provides the REST API calls used to create, manage, and remove Replay profile objects on an SC Series array.

Table 7 Replay profile REST API calls used in this section

REST API	Method
/StorageCenter/ScReplayProfile	POST
/StorageCenter/ScReplayProfile/<instanceId>	PUT DELETE
/StorageCenter/ScReplayProfile/GetList	POST

3.7.1 Create a Replay profile

The **payload['Privilege']** and **payload['SessionTimeout']** attributes contain EnumValueName values as detailed in the corresponding tables that follow. An additional code segment is included here to capture all Replay profile information from all DSM-managed SC Series arrays and populate the Python dictionary **rpList**.

```
# create a new named replay profile object

payload = {}
payload['Name'] = 'RestTest_rp'
payload['StorageCenter'] = scList['SC 9']['instanceId']
payload['Notes'] = 'Created via REST API'
REST = '/StorageCenter/ScReplayProfile'
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.post(completeURL
                            ,data=json.dumps(payload
                                             ,ensure_ascii=False).encode('utf-8')
                            ,headers=header
                            ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout

# capture all storage profile info managed by DSM

payload = {}
REST = '/StorageCenter/ScReplayProfile/GetList'
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.post(completeURL
                            ,data=json.dumps(payload
                                             ,ensure_ascii=False).encode('utf-8')
                            ,headers=header
                            ,verify=verify_cert)
stdout = json.loads(json_data.text)
rpList = {}
for i in range(len(stdout)):
    rpList[stdout[i]['instanceName']] = {}
    rpList[stdout[i]['instanceName']]['instanceId'] = stdout[i]['instanceId']
    rpList[stdout[i]['instanceName']]['scName'] = stdout[i]['scName']
```

STDOUT

```
{u'ruleCount': 0, u'name': u'RestTest_rp', u'volumeCount': 0, u'scName': u'SC
9', u'notes': u'Created via REST API', u'scSerialNumber': 101, u'userCreated':
True, u'instanceName': u'RestTest_rp', u'instanceId': u'101.18',
u'enforceReplayCreationTimeout': False, u'replayCreationTimeout': 0,
u'moreVolumesAllowed': True, u'objectType': u'ScReplayProfile', u'type':
u'Standard', u'expireIncompleteReplaySets': False}
```

3.7.2 Modify a Replay profile

```
# modify a replay profile object identified by instanceId

payload = {}
payload['Name'] = 'RestTest_rp_renamed'
payload['Notes'] = 'Created via REST API'
REST = '/StorageCenter/ScReplayProfile/%s' % rpList['RestTest_rp']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.put(completeURL
                           ,data=json.dumps(payload
                                            ,ensure_ascii=False).encode('utf-8')
                           ,headers=header
                           ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout
```

STDOUT

```
{u'ruleCount': 0, u'name': u'RestTest_rp_renamed', u'volumeCount': 0, u'scName':
u'SC 9', u'notes': u'Created via REST API', u'scSerialNumber': 101,
u'userCreated': True, u'instanceName': u'RestTest_rp_renamed', u'instanceId':
u'101.18', u'enforceReplayCreationTimeout': False, u'replayCreationTimeout': 0,
u'moreVolumesAllowed': True, u'objectType': u'ScReplayProfile', u'type':
u'Standard', u'expireIncompleteReplaySets': False}
```

3.7.3 Delete a Replay profile

```
# delete a named replay profile object

payload = {}
REST = '/StorageCenter/ScReplayProfile/%s' % rpList['RestTest_rp']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.delete(completeURL
                              ,headers=header
                              ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout
```

STDOUT

```
{u'result': True}
```

3.8 Recycle bin

This section discusses restoring a volume object from the recycle bin to its original parent folder in the Volumes tree.

Table 8 Recycle bin REST API calls used in this section

REST API	Method
/StorageCenter/ScVolumeFolder/<instanceId>/Restore	POST

3.8.1 Restore a volume from the recycle bin

```
# restore a volume object

payload = {}
REST = '/StorageCenter/ScVolume/%s/Restore' %
       volList['RestTest_Vol_renamed']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.post(completeURL
                            ,data=json.dumps(payload
                                             ,ensure_ascii=False).encode('utf-8')
                            ,headers=header
                            ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout
```

STDOUT

```
{u'result': True}
```

3.9 Users

This section provides the REST API calls used to create, manage, and remove user accounts and objects on an SC Series array.

Table 9 User REST API calls used in this section

REST API	Method
/StorageCenter/ScUser	POST
/StorageCenter/ScUser/<instanceId>	PUT DELETE
/StorageCenter/ScUser/<instanceId>/ResetPassword	POST
/StorageCenter/ScUser/<instanceId>/Unlock	POST
/StorageCenter/ScUser/Restore	POST

3.9.1 Create an SC Series user account

The **payload['Privilege']** and **payload['SessionTimeout']** attributes contain the EnumValueName values detailed in the corresponding tables below. An additional code segment is included here to capture all user account information from all DSM-managed SC Series arrays and populate the Python dictionary **usrList**.

```
# create a new user account on SC 9

payload = {}
payload['Name'] = 'restusr'
payload['RealName'] = 'REST User'
payload['Password'] = 'rest123'
payload['EmailAddress'] = 'restusr@mycompany.com'
payload['Privilege'] = 'Admin'
payload['SessionTimeout'] = 'Hours12'
payload['StorageCenter'] = scList['SC 9']['instanceId']
payload['Notes'] = 'Created via REST API'
REST = '/StorageCenter/ScUser'
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else RES

print connection.post(completeURL
                      ,data=json.dumps(payload
                                       ,ensure_ascii=False).encode('utf-8')
                      ,headers=header
                      ,verify=verify_cert)
usrList = {}
usrList[payload['Name']] = {}
usrList[payload['Name']]['password'] = payload['Password']

# capture all user account info

payload = {}
REST = '/StorageCenter/ScUser/GetList'
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.post(completeURL
                            ,data=json.dumps(payload
                                             ,ensure_ascii=False).encode('utf-8')
                            ,headers=header
                            ,verify=verify_cert)
stdout = json.loads(json_data.text)
for i in range(len(stdout)):
    if stdout[i]['instanceName'] in usrList.keys():
        print "Key already exists!"
    else:
        usrList[stdout[i]['instanceName']] = {}
        usrList[stdout[i]['instanceName']]['instanceName'] =
            stdout[i]['instanceName']
        usrList[stdout[i]['instanceName']]['instanceId'] =
            stdout[i]['instanceId']
        usrList[stdout[i]['instanceName']]['realName'] =
            stdout[i]['realName']
```

```
usrList[stdout[i]['instanceName']]['scName'] =
    stdout[i]['scName']
usrList[stdout[i]['instanceName']]['locked'] =
    stdout[i]['locked']
```

STDOUT

```
<Response [201]>
```

3.9.1.1 Privilege Enum Summary

For the latest enumerated field values, refer to the REST API help files located in section 1.1.

Name	Visibility	UI Name	Description
Admin	Public	Administrator	The User is an Administrator
Reporter	Public	Reporter	A read-only User
VolumeManager	Public	Volume Manager	A Volume Manager User

Figure 3 Payload[Privilege] EnumValueName values

3.9.1.2 SessionTimeout Enum Summary

For the latest enumerated field values, refer to the REST API help files located in section 1.1.

Name	Visibility	UI Name	Description
Days1	Public	1 Day	1 Day
Days2	Public	2 Days	2 Days
Hours1	Public	1 Hour	1 hour
Hours12	Public	12 Hours	12 Hours
Hours2	Public	2 Hours	2 Hours
Hours4	Public	4 Hours	4 Hours
Hours8	Public	8 Hours	8 Hours
Minutes15	Public	15 Minutes	15 Minutes
Minutes30	Public	30 Minutes	30 Minutes
Minutes45	Public	45 Minutes	45 Minutes

Figure 4 Payload[timeout] EnumValueName values

3.9.2 Modify an SC Series user account

The following code can be used exclusively to modify a single attribute or multiple attributes of a user account object concurrently. This code segment cannot be used to change or reset a user account name, password, or account status (locked or unlocked) attribute.

```
# modify a user account

payload = {}
payload['RealName'] = 'REST User Renamed'
payload['EmailAddress'] = 'restusr@mycompany.com'
payload['Notes'] = 'Created via REST API'
REST = '/StorageCenter/ScUser/%s' % usrList['restusr']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.put(completeURL
                           ,data=json.dumps(payload
                                            ,ensure_ascii=False).encode('utf-8')
                           ,headers=header
                           ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout
```

STDOUT

```
{u'createdByGroupLogIn': False, u'modifiedOn': u'2016-04-25T14:23:59-05:00',
u'passwordDaysUntilExpiration': 0, u'instanceId': u'101.14', u'scSerialNumber':
101, u'objectType': u'ScUser', u'distinguishedName': u'', u'realName': u'REST
User Renamed', u'title': u'', u'preferredLanguage': u'en_US', u'directoryUser':
False, u'sessionTimeout': u'Hours12', u'location': u'',
u'passwordInExpirationWarningWindow': False, u'emailAddress2': u'',
u'department': u'', u'dateUpdated': u'2016-04-25T14:23:59-05:00',
u'instanceName': u'restusr', u'mobilePhone': u'', u'dateCreated': u'2016-04-
25T11:06:13-05:00', u'privilege': u'Admin', u'emailAddress':
u'restusr@mycompany.com', u'createdBy': u'Admin', u'updatedBy': u'Admin',
u'homePhone': u'', u'businessPhone': u'', u'locked': False, u'name': u'restusr',
u'scName': u'SC 9', u'createdOn': u'2016-04-25T11:06:13-05:00', u'enabled':
True, u'emailAddress3': u'', u'notes': u'Created via REST API'}
```

3.9.3 Reset an SC Series user account password

```
# reset / change password of a specified user account

payload = {}

# SC 'admin' account password

payload['AuthorizationPassword'] = '***'    # SC 'admin' account password
payload['NewPassword'] = 'NewPass!'
REST = '/StorageCenter/ScUser/%s/ResetPassword' %
       usrList['restusr']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
```

```
print connection.post(completeURL
                      ,data=json.dumps(payload
                                       ,ensure_ascii=False).encode('utf-8')
                      ,headers=header
                      ,verify=verify_cert)
```

STDOUT

```
<Response [204]>
```

3.9.4 Unlock an SC Series user account

```
# unlock a user account

payload = {}
REST = '/StorageCenter/ScUser/%s/Unlock' % usrList['restusr']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
print connection.post(completeURL
                      , data=json.dumps(payload
                                        ,ensure_ascii=False).encode('utf-8')
                      , headers=header
                      , verify=verify_cert)
```

STDOUT

```
<Response [200]>
```

3.9.5 Delete or restore an SC Series user account

```
# delete a user account

payload = {}
REST = '/StorageCenter/ScUser/%s' % usrList['restusr']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.delete(completeURL
                              ,headers=header
                              ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout

# restore a user account

payload = {}
payload['Name'] = usrList['restusr']['instanceName']
payload['StorageCenter'] = scList[usrList['restusr']['scName']]['instanceId']
REST = '/StorageCenter/ScUser/Restore'
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.post(completeURL
                            ,data=json.dumps(payload
```

```
                                                    ,ensure_ascii=False).encode('utf-8')
                                ,headers=header
                                ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout
```

STDOUT

```
{u'result': True}
```

```
{u'createdByGroupLogIn': False, u'modifiedOn': u'2016-04-25T14:33:02-05:00',
u'passwordDaysUntilExpiration': 0, u'instanceId': u'101.14', u'scSerialNumber':
101, u'objectType': u'ScUser', u'distinguishedName': u'', u'realName': u'REST
User Renamed', u'title': u'', u'preferredLanguage': u'en_US', u'directoryUser':
False, u'sessionTimeout': u'Hours12', u'location': u'',
u'passwordInExpirationWarningWindow': False, u'emailAddress2': u'',
u'department': u'', u'dateUpdated': u'2016-04-25T14:33:02-05:00',
u'instanceName': u'restusr', u'mobilePhone': u'', u'dateCreated': u'2016-04-
25T11:06:13-05:00', u'privilege': u'Admin', u'emailAddress':
u'restusr@mycompany.com', u'createdBy': u'Admin', u'updatedBy': u'Admin',
u'homePhone': u'', u'businessPhone': u'', u'locked': False, u'name': u'restusr',
u'scName': u'SC 9', u'createdOn': u'2016-04-25T11:06:13-05:00', u'enabled':
True, u'emailAddress3': u'', u'notes': u'Created via REST API'}
```

3.10 Disks

This section discusses the use of the REST API calls to manage the disks, disk folders, and RAID rebalances.

Table 10 Disk REST API calls used in this section.

REST API	Method
/StorageCenter/ScDiskFolder/GetList	POST
/StorageCenter/ScDiskFolder/<instanceId>/DiskList	GET
/StorageCenter/ScDiskFolder/<instanceId>/DiskFolderTierList	GET
/StorageCenter/ScDiskFolderTier/<instanceId>/DiskList	GET
/StorageCenter/ScDiskFolder/<instanceId>/AddDisks	POST
/StorageCenter/ScDiskFolder/RaidRebalance	POST

3.10.1 All disks by disk folder

```
# list all disks by Disk Folder category

payload = {}
REST = '/StorageCenter/ScDiskFolder/GetList'
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.post(completeURL
                            ,data=json.dumps(payload
                                             ,ensure_ascii=False).encode('utf-8')
                            ,headers=header
                            ,verify=verify_cert)
stdout = json.loads(json_data.text)
Fldr = {}
for i in range(len(stdout)):
    Fldr[stdout[i]['scName']] = {}
    Fldr[stdout[i]['scName']][stdout[i]['instanceName']] = {}
    Fldr[stdout[i]['scName']][stdout[i]['instanceName']]['instanceId'] =
        stdout[i]['instanceId']
    Fldr[stdout[i]['scName']][stdout[i]['instanceName']]['spareCount'] =
        stdout[i]['spareCount']
    Fldr[stdout[i]['scName']][stdout[i]['instanceName']]['scSerialNumber'] =
        stdout[i]['scSerialNumber']

# get all disks by Disk Folder category Assigned on SC 9

payload = {}
REST = '/StorageCenter/ScDiskFolder/%s/DiskList' %
       Fldr['SC 9']['Assigned']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.get(completeURL
                           ,headers=header
                           ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout
```

STDOUT

```
[{u'managed': True, u'userConsumableSize': u'600127266816 Bytes', u'healthMask':
[], u'scSerialNumber': 101, u'diskClassIndex': 2, u'unassignedReason':
u'AlreadyAssigned', u'estimatedWearWarningEndurance': 0, u'size': u'600127266816
Bytes', u'secured': False, u'instanceId': u'101.129', u'enclosureIndex': 5,
u'controlType': u'Managed', u'objectType': u'ScDisk', u'instanceName': u'05-15',
u'diskFolder': {u'instanceId': u'101.3', u'instanceName': u'Assigned',
u'objectType': u'ScDiskFolder'}, u'statusMessage': u'', u'status': u'Up',
u'diskFolderIndex': 3, u'estimatedWearDate': u'1969-12-31T18:00:00-06:00',
u'spare': False, u'markedForRemoval': False, u'markedForSpare': False,
u'locked': False, u'name': u'05-15', u'healthy': True, u'scName': u'SC 9',
u'secureDataAllowed': False, u'diskEraseCapability': u'None', u'diskClass':
{u'instanceId': u'101.2', u'instanceName': u'10K', u'objectType':
```

```
u'ScDiskClass'}, u'endurance': 0, u'reportsWear': False}, {u'managed': True,
u'userConsumableSize': u'600127266816 Bytes', u'healthMask': [],
u'scSerialNumber': 101, u'diskClassIndex': 2, u'unassignedReason':
u'AlreadyAssigned', u'estimatedWearWarningEndurance': 0, u'size': u'600127266816
Bytes', u'secured': False, u'instanceId': u'101.128', u'enclosureIndex': 5,
u'controlType': u'Managed', u'objectType': u'ScDisk', u'instanceName': u'05-10',
u'diskFolder': {u'instanceId': u'101.3', u'instanceName': u'Assigned',
u'objectType': u'ScDiskFolder'}, u'statusMessage': u'', u'status': u'Up',
u'diskFolderIndex': 3, u'estimatedWearDate': u'1969-12-31T18:00:00-06:00',
u'spare': False, u'markedForRemoval': False, u'markedForSpare': False,
u'locked': False, u'name': u'05-10', u'healthy': True, u'scName': u'SC 9',
u'secureDataAllowed': False, u'diskEraseCapability': u'None', u'diskClass':
{u'instanceId': u'101.2', u'instanceName': u'10K', u'objectType':
u'ScDiskClass'}, u'endurance': 0, u'reportsWear': False},
[snip]
```

3.10.2 All disks by tier

```
# list all disks by Tier category

payload = {}
REST = '/StorageCenter/ScDiskFolderTier/GetList'
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.post(completeURL
                            ,data=json.dumps(payload
                                             ,ensure_ascii=False).encode('utf-8')
                            ,headers=header
                            ,verify=verify_cert)
stdout = json.loads(json_data.text)
dskTier = {}
for i in range(len(stdout)):
    dskTier[stdout[i]['scName']] = {}
    dskTier[stdout[i]['scName']][stdout[i]['instanceName']] = {}
    dskTier[stdout[i]['scName']][stdout[i]['instanceName']]['instanceId'] =
        stdout[i]['instanceId']
    dskTier[stdout[i]['scName']][stdout[i]['instanceName']]['diskCount'] =
        stdout[i]['diskCount']
    dskTier[stdout[i]['scName']][stdout[i]['instanceName']]['totalSpace'] =
        stdout[i]['totalSpace']

# get all disks by Disk Tier Assigned - Tier3 on SC 18

payload = {}
REST = '/StorageCenter/ScDiskFolderTier/%s/DiskList' %
       dskTier['SC 18']['Assigned - Tier3']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.get(completeURL
                           ,headers=header
                           ,verify=verify_cert)
stdout = json.loads(json_data.text)
print stdout
```

STDOUT


```
[{u'managed': True, u'userConsumableSize': u'2000398934016 Bytes',
u'healthMask': [], u'scSerialNumber': 716, u'diskClassIndex': 4,
u'unassignedReason': u'AlreadyAssigned', u'estimatedWearWarningEndurance': 0,
u'size': u'2000398934016 Bytes', u'secured': False, u'instanceId': u'716.55',
u'enclosureIndex': 3, u'controlType': u'Managed', u'objectType': u'ScDisk',
u'instanceName': u'03-03', u'diskFolder': {u'instanceId': u'716.3',
u'instanceName': u'Assigned', u'objectType': u'ScDiskFolder'}, u'statusMessage':
u'', u'status': u'Up', u'diskFolderIndex': 0, u'estimatedWearDate': u'1969-12-
31T18:00:00-06:00', u'spare': False, u'markedForRemoval': False,
u'markedForSpare': False, u'locked': False, u'name': u'03-03', u'healthy': True,
u'scName': u'SC 18', u'secureDataAllowed': False, u'diskEraseCapability':
u'None', u'diskClass': {u'instanceId': u'716.4', u'instanceName': u'7K',
u'objectType': u'ScDiskClass'}, u'endurance': 0, u'reportsWear': False},
{u'managed': True, u'userConsumableSize': u'2000398934016 Bytes', u'healthMask':
[], u'scSerialNumber': 716, u'diskClassIndex': 4, u'unassignedReason':
u'AlreadyAssigned', u'estimatedWearWarningEndurance': 0, u'size':
u'2000398934016 Bytes', u'secured': False, u'instanceId': u'716.54',
u'enclosureIndex': 3, u'controlType': u'Managed', u'objectType': u'ScDisk',
u'instanceName': u'03-10', u'diskFolder': {u'instanceId': u'716.3',
u'instanceName': u'Assigned', u'objectType': u'ScDiskFolder'}, u'statusMessage':
u'', u'status': u'Up', u'diskFolderIndex': 0, u'estimatedWearDate': u'1969-12-
31T18:00:00-06:00', u'spare': False, u'markedForRemoval': False,
u'markedForSpare': False, u'locked': False, u'name': u'03-10', u'healthy': True,
u'scName': u'SC 18', u'secureDataAllowed': False, u'diskEraseCapability':
u'None', u'diskClass': {u'instanceId': u'716.4', u'instanceName': u'7K',
u'objectType': u'ScDiskClass'}, u'endurance': 0, u'reportsWear': False},
[snip]
```


3.10.3 All disks unmanaged

```
# list all unmanaged disk objects on SC 9

payload = {}
REST = '/StorageCenter/ScDiskFolder/%s/DiskList' %
       Fldr['SC 9']['Unmanaged']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.get(completeURL
                           ,headers=header
                           ,verify=verify_cert)
stdout = json.loads(json_data.text)
dskUnmanaged = []
for i in range(len(stdout)):
    dskUnmanaged.append(stdout[i]['instanceId'])
print stdout
```

STDOUT


```
[{u'managed': False, u'userConsumableSize': u'600127266816 Bytes',
u'healthMask': [], u'scSerialNumber': 101, u'diskClassIndex': 2,
u'unassignedReason': u'AlreadyAssigned', u'estimatedWearWarningEndurance': 0,
u'size': u'600127266816 Bytes', u'secured': False, u'instanceId': u'101.129',
u'enclosureIndex': 5, u'controlType': u'Managed', u'objectType': u'ScDisk',
u'instanceName': u'05-15', u'diskFolder': {u'instanceId': u'101.3',
u'instanceName': u'Assigned', u'objectType': u'ScDiskFolder'}, u'statusMessage':
u'', u'status': u'Up', u'diskFolderIndex': 3, u'estimatedWearDate': u'1969-12-
31T18:00:00-06:00', u'spare': False, u'markedForRemoval': False,
u'markedForSpare': False, u'locked': False, u'name': u'05-15', u'healthy': True,
u'scName': u'SC 9', u'secureDataAllowed': False, u'diskEraseCapability':
u'None', u'diskClass': {u'instanceId': u'101.2', u'instanceName': u'10K',
u'objectType': u'ScDiskClass'}, u'endurance': 0, u'reportsWear': False},
{u'managed': False, u'userConsumableSize': u'600127266816 Bytes', u'healthMask':
[], u'scSerialNumber': 101, u'diskClassIndex': 2, u'unassignedReason':
u'AlreadyAssigned', u'estimatedWearWarningEndurance': 0, u'size': u'600127266816
Bytes', u'secured': False, u'instanceId': u'101.128', u'enclosureIndex': 5,
u'controlType': u'Managed', u'objectType': u'ScDisk', u'instanceName': u'05-10',
u'diskFolder': {u'instanceId': u'101.3', u'instanceName': u'Assigned',
u'objectType': u'ScDiskFolder'}, u'statusMessage': u'', u'status': u'Up',
u'diskFolderIndex': 3, u'estimatedWearDate': u'1969-12-31T18:00:00-06:00',
u'spare': False, u'markedForRemoval': False, u'markedForSpare': False,
u'locked': False, u'name': u'05-10', u'healthy': True, u'scName': u'SC 9',
u'secureDataAllowed': False, u'diskEraseCapability': u'None', u'diskClass':
{u'instanceId': u'101.2', u'instanceName': u'10K', u'objectType':
u'ScDiskClass'}, u'endurance': 0, u'reportsWear': False},
[snip]
```


3.10.4 Add unmanaged disks to a disk folder

```
# add unmanaged disk objects to Assigned group

payload = {}
payload['Disks'] = []
payload['Disks'] = dskUnmanaged
REST = '/StorageCenter/ScDiskFolder/%s/AddDisks' %
       Fldr['SC 9']['Assigned']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
print connection.post(completeURL
                      ,data=json.dumps(payload
                                       ,ensure_ascii=False).encode('utf-8')
                      ,headers=header
                      ,verify=verify_cert)
```

STDOUT


```
{u'result': True}
```

3.10.5 Execute a RAID rebalance

```
# initiate a SC RAID rebalance

payload = {}
payload['StorageCenter'] = scList['SC 9']['instanceId']
REST = '/StorageCenter/ScDiskFolder/RaidRebalance'
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
print connection.post(completeURL
                      ,data=json.dumps(payload
                                       ,ensure_ascii=False).encode('utf-8')
                      ,headers=header
                      ,verify=verify_cert)
```

STDOUT

```
{u'result': True}
```

3.11 Alerts

This section discusses the use of the REST API calls to query, review, and acknowledge alerts on SC Series arrays.

Table 11 Alert REST API calls used in this section.

REST API	Method
/StorageCenter/StorageCenter/<instanceId>/ActiveAlertList	GET
/StorageCenter/StorageCenter/<instanceId>/HistoricalAlertList	GET
/StorageCenter/ScAlert/<instanceId>/Acknowledge	POST

3.11.1 Get active alerts

```
# get all active alerts off SC 9

payload = {}
REST = '/StorageCenter/StorageCenter/%s/ActiveAlertList' %
       (scList['SC 9']['instanceId'])
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.get(completeURL
                           ,headers=header
                           ,verify=verify_cert)
stdout = json.loads(json_data.text)
activeAlert = {}
for i in range(len(stdout)):
    activeAlert[stdout[i]['instanceName']] = {}
    activeAlert[stdout[i]['instanceName']]['instanceId'] =
        stdout[i]['instanceId']
    activeAlert[stdout[i]['instanceName']]['scSerialNumber'] =
        stdout[i]['scSerialNumber']
```

```
    activeAlert[stdout[i]['instanceName']]['alertStatus'] =
        stdout[i]['alertStatus']
    activeAlert[stdout[i]['instanceName']]['message'] =
        stdout[i]['message']
    activeAlert[stdout[i]['instanceName']]['createTime'] =
        stdout[i]['createTime']
```

STDOUT

3.11.2 Get alert history

```
# get all historical alerts off SC 9

payload = {}
REST = '/StorageCenter/StorageCenter/%s/HistoricalAlertList' %
       (scList['SC 9']['instanceId'])
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.get(completeURL
                           ,headers=header
                           ,verify=verify_cert)
stdout = json.loads(json_data.text)
```

STDOUT

3.11.3 Acknowledge active alerts

```
# acknowledge active alert on SC 9 referenced by instanceId

payload = {}
REST = '/StorageCenter/ScAlert/%s/Acknowledge' %
       activeAlert['[102-96] vios2-king']['instanceId']
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
print connection.post(completeURL
                      ,data=json.dumps(payload
                                       ,ensure_ascii=False).encode('utf-8')
                      ,headers=header
                      ,verify=verify_cert)
```

STDOUT

```
<Response [204]>
```

4 Working with Linux/UNIX

This section covers tools discussed in this paper that can facilitate the installation and use of Python on Linux/UNIX platforms.

4.1 wget

The **wget** command is a UNIX-based utility that enables the retrieval of content from a web URL address. An example of this command is:

```
# wget --no-check-certificate \
> https://www.python.org/ftp/python/2.7.11/Python-2.7.10.tgz
```

4.2 tar

The **tar** command is a UNIX-based utility that enables the extraction and uncompresses the previously downloaded Python-2.7.10.tgz file.

```
# tar xvzf ./Python-2.7.10.tgz
# cd ./Python-2.7.10
# ./configure
# ./make
# ./make install
```

4.3 vim

The **vim** editor is suggested for use, because it can display code/syntax highlighting and makes for easier writing, management, and troubleshooting of Python code. The vim Python color scheme can be retrieved and installed.

```
# cd /tmp
# wget http://ianbits.googlecode.com/svn/trunk/vim/python.vim
# mkdir -p ~/.vim/syntax
# mv ./python.vim ~/.vim/syntax
```

5 Sample script

This section presents the various components of the Storage Center REST API discussed in previous sections and assembles it into a cohesive, functional, and complete sample script. This sample script is annotated to explain statements or sections as needed.

Additional code can be injected into this complete sample script where denoted by the placeholder:

```
"=====   YOUR CODE GOES HERE   =====".
```

The following assumptions are made regarding the code:

- Assumes a true or successful condition (return code) from each requested REST API call
- Contains minimal error trapping and management to promote readability and understanding
- Creates the data structures required to facilitate passing the data between REST API calls

```
#
# main.py
#
# import modules into Python script

import os, sys, subprocess, math
sys.path.append("/usr/lib64/python2.6/site-packages")
sys.path.append("/usr/local/lib/python2.7/site-packages")

import requests, json, http, httplib, urllib, urllib2
import math, time
import logging
from simplejson import scanner

# setup logging to scapi.log

logging.basicConfig(level=logging.DEBUG
                    ,filename='scapi.log'
                    ,format='[%(asctime)s] %(levelname)s %(message)s')

def stdout_inspect(input, how_many):
    for i in range(how_many):
            for key in input[i]:
                print ("%-25s: %s") % (key, input[i][key])
            print("")

if __name__ == '__main__':

    # define env incl. DSM IP addr, port & login credentials

    DSM_ip = '<nnn.nnn.nnn.nnn>'     # IP address of DSM instance
    DSM_port = '3033'                # Default port of DSM instance
    DSM_id = '<Username>'            # Login credentials for DSM
    DSM_pass = '<Password>'          # Password
    verify_cert = False              # Default = False
```

```
apiversion = '2.0'              # Default = 2.0

# disable warnings from requests module

if not verify_cert:
    requests.packages.urllib3.disable_warnings()

# define base URL for DSM REST API interface

baseURL = 'https://%s:%s/api/rest/' % (DSM_ip, DSM_port)

# define HTTP content headers

header = {}
header['Content-Type'] = 'application/json; charset=utf-8'
header['Accept'] = 'application/json'
header['x-dell-api-version'] = apiversion

# define the connection session

connection = requests.Session()
connection.auth = (DSM_id, DSM_pass)

# login to DSM instance

payload = {}
REST = '/ApiConnection/Login'
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
print connection.post(completeURL
                      ,data=json.dumps(payload
                                       , ensure_ascii=False).encode('utf-8')
                      ,headers=header
                      ,verify=verify_cert)

print ("")

# capture API connection instanceId

payload = {}
REST = '/ApiConnection/ApiConnection'
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.get(completeURL
                           ,headers=header
                           ,verify=verify_cert)
stdout = json.loads(json_data.text)
conn_instanceId = stdout['instanceId']

# capture all SC series arrays managed by this DSM instance

payload = {}
```

```
REST = '/ApiConnection/ApiConnection/%s/StorageCenterList' % conn_instanceId
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.get(completeURL
                           ,headers=header
                           ,verify=verify_cert)
stdout = json.loads(json_data.text)
scList = {}
print ("%-15s %-20s %-15s %-15s") %
      ("Name","Serial Number","instanceId","IP")
for i in range(len(stdout)):
    print ("%-15s %-20s %-15s %-15s") %
              (stdout[i]['name']
              , stdout[i]['scSerialNumber']
              , stdout[i]['instanceId']
              , stdout[i]['hostOrIpAddress'])
    scList[stdout[i]['name']] = {}
    scList[stdout[i]['name']]['instanceId'] = stdout[i]['instanceId']
    scList[stdout[i]['name']]['hostOrIP'] = stdout[i]['hostOrIpAddress']

print ("")

# loop through all SC series arrays managed by this DSM instance and
# capture all volume, volume folder, server, server folder and user account
# objects into volList, volFolderList, srvList, srvFolderList and usrList

volFolderList = {}
volFolderList_Total = 0
volList = {}
volList_Total = 0
srvFolderList = {}
srvFolderList_Total = 0
srvList = {}
srvList_Total = 0
osList = {}
for key in scList:

    # objects Volume Folder

    payload = {}
    REST = '/StorageCenter/StorageCenter/%s/VolumeFolderList' %
           (scList[key]['instanceId'])
    completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
    json_data = connection.get(completeURL
                               ,headers=header
                               ,verify=verify_cert)
    stdout = json.loads(json_data.text)
    volFolderList_Total += len(stdout)
    for i in range(len(stdout)):
        if stdout[i]['name'] == "Volumes":
            continue
```

```
    if volFolderList.has_key(stdout[i]['name']):
        volFolderList[stdout[i]['name']][key] = {}
    else:
        volFolderList[stdout[i]['name']] = {}
        volFolderList[stdout[i]['name']][key] = {}
    volFolderList[stdout[i]['name']][key]['instanceId'] =
        stdout[i]['instanceId']
    volFolderList[stdout[i]['name']][key]['parent'] =
        stdout[i]['parent']['instanceName']

# objects Volumes

payload = {}
REST = '/StorageCenter/StorageCenter/%s/VolumeList' %
       (scList[key]['instanceId'])
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.get(completeURL
                           ,headers=header
                           ,verify=verify_cert)
stdout = json.loads(json_data.text)
volList_Total += len(stdout)
for i in range(len(stdout)):
    if volList.has_key(stdout[i]['name']):
        volList[stdout[i]['name']][key] = {}
    else:
        volList[stdout[i]['name']] = {}
        volList[stdout[i]['name']][key] = {}
    volList[stdout[i]['name']][key]['instanceId'] =
        stdout[i]['instanceId']
    volList[stdout[i]['name']][key]['path'] =
        stdout[i]['volumeFolderPath']

# objects Server Folder

payload = {}
REST = '/StorageCenter/StorageCenter/%s/ServerFolderList' %
       (scList[key]['instanceId'])
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.get(completeURL
                           ,headers=header
                           ,verify=verify_cert)
stdout = json.loads(json_data.text)
srvFolderList_Total += len(stdout)
for i in range(len(stdout)):
    if stdout[i]['name'] == "Servers":
        continue
    if srvFolderList.has_key(stdout[i]['name']):
        srvFolderList[stdout[i]['name']][key] = {}
    else:
        srvFolderList[stdout[i]['name']] = {}
```

```
                srvFolderList[stdout[i]['name']][key] = {}
            srvFolderList[stdout[i]['name']][key]['instanceId'] =
                stdout[i]['instanceId']
            srvFolderList[stdout[i]['name']][key]['parent'] =
                stdout[i]['parent']['instanceName']

        # objects Servers

        payload = {}
        REST = '/StorageCenter/StorageCenter/%s/ServerList' %
               (scList[key]['instanceId'])
        completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
        json_data = connection.get(completeURL
                                   ,headers=header
                                   ,verify=verify_cert)
        stdout = json.loads(json_data.text)
        srvList_Total += len(stdout)
        for i in range(len(stdout)):
            if srvList.has_key(stdout[i]['name']):
                srvList[stdout[i]['name']][key] = {}
            else:
                srvList[stdout[i]['name']] = {}
                srvList[stdout[i]['name']][key] = {}
            srvList[stdout[i]['instanceName']][key]['instanceId'] =
                stdout[i]['instanceId']
            srvList[stdout[i]['instanceName']][key]['path'] =
                stdout[i]['serverFolderPath']

        # objects Operating Systems

        payload = {}
        REST = '/StorageCenter/StorageCenter/%s/ServerOperatingSystemList' %
               (scList[key]['instanceId'])
        completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
        json_data = connection.get(completeURL
                                   ,headers=header
                                   ,verify=verify_cert)
        stdout = json.loads(json_data.text)
        for i in range(len(stdout)):
            osList[stdout[i]['instanceName']] = {}
            osList[stdout[i]['instanceName']]['instanceId'] =
                stdout[i]['instanceId']
            osList[stdout[i]['instanceName']]['scName'] = key

print "DSM Instance                 : %s" % DSM_ip
print "Total Volume Folders Managed : %d" % volFolderList_Total
print "Total Volumes Managed        : %d" % volList_Total
print "Total Server Folders Managed : %d" % srvFolderList_Total
print "Total Servers Managed        : %d" % srvList_Total
print ("")
```

```
# get disks info by both diskFolder and diskTier sorting

payload = {}
REST = '/StorageCenter/ScDiskFolder/GetList'
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.post(completeURL
                            ,data=json.dumps(payload
                                             ,ensure_ascii=False
                                             ).encode('utf-8')
                            ,headers=header
                            ,verify=verify_cert)
stdout = json.loads(json_data.text)

# Disk Folder - Fldr

Fldr = {}
for i in range(len(stdout)):
    Fldr[stdout[i]['scName']] = {}
    Fldr[stdout[i]['scName']][stdout[i]['instanceName']] = {}
    Fldr[stdout[i]['scName']][stdout[i]['instanceName']]['instanceId'] =
        stdout[i]['instanceId']
    Fldr[stdout[i]['scName']][stdout[i]['instanceName']]['spareCount'] =
        stdout[i]['spareCount']
    Fldr[stdout[i]['scName']][stdout[i]['instanceName']]['scSerialNumber'] =
        stdout[i]['scSerialNumber']

payload = {}
REST = '/StorageCenter/ScDiskFolderTier/GetList'
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
json_data = connection.post(completeURL
                            ,data=json.dumps(payload
                                             ,ensure_ascii=False
                                             ).encode('utf-8')
                            ,headers=header
                            ,verify=verify_cert)
stdout = json.loads(json_data.text)

#stdout_inspect(stdout, len(stdout))

dskTier = {}
for i in range(len(stdout)):
    dskTier[stdout[i]['scName']] = {}
    dskTier[stdout[i]['scName']][stdout[i]['instanceName']] = {}
    dskTier[stdout[i]['scName']][stdout[i]['instanceName']]['instanceId'] =
        stdout[i]['instanceId']
    dskTier[stdout[i]['scName']][stdout[i]['instanceName']]['diskCount'] =
        stdout[i]['diskCount']
    dskTier[stdout[i]['scName']][stdout[i]['instanceName']]['totalSpace'] =
        stdout[i]['totalSpace']
```

```
# print capacities and usage across all SC managed by DSM

for key in scList:
    print("=== %s" % key)
    for keyTier in dskTier[key].keys():
        print("Disks in %s : %s" %
             (keyTier, dskTier[key][keyTier]['diskCount']))
        payload = {}
        REST = '/StorageCenter/ScDiskFolderTier/%s/StorageUsage' %
               dskTier[key][keyTier]['instanceId']
        completeURL = '%s%s' %
                      (baseURL, REST if REST[0] != '/' else REST[1:])
        json_data = connection.get(completeURL
                                   ,headers=header
                                   ,verify=verify_cert)
        stdout = json.loads(json_data.text)
        capacity = stdout['totalSpace'].split()
        totalSpace = float(int(capacity[0])/1073741824)
        print ("Capacity Total            : %d GB" % totalSpace)
        capacity = stdout['allocatedSpace'].split()
        allocatedSpace = float(int(capacity[0])/1073741824)
        print ("Capacity Alloc            : %d GB" % allocatedSpace)
        print ("Percentage Used           : %d" %
               float((allocatedSpace/totalSpace)*100)) + '%'
        print ("")

=====   YOUR CODE GOES HERE   =====

# logout from DSM instance

payload = {}
REST = '/ApiConnection/Logout'
completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])

#print connection.post(completeURL
#                      ,data=json.dumps(payload
#                                       ,ensure_ascii=False).encode('utf-8')
#                      ,headers=header
#                      ,verify=verify_cert)

print "Dell EMC"
print "The power to do more\n"
```

5.1 Console output

```
Name            Serial Number        instanceId      IP
SC 25           60707                60707           172.16.2.125
SC 18           716                  716             172.16.2.118
SC 9 - 101      101                  101             172.16.2.109
SC 10           103                  103             172.16.2.110
SC 4            862                  862             172.16.2.104

DSM Instance                 : 172.16.21.165
Total Volume Folders Managed : 284
Total Volumes Managed        : 680
Total Server Folders Managed : 60
Total Servers Managed        : 141

=== SC 25
Disks in Assigned - Tier3 : 24
Capacity Total            : 21424 GB
Capacity Alloc            : 1339 GB
Percentage Used           : 6%

=== SC 9 - 101
Disks in Assigned - Tier1 : 24
Capacity Total            : 12854 GB
Capacity Alloc            : 7539 GB
Percentage Used           : 58%

=== SC 4
Disks in Assigned - Tier3 : 96
Capacity Total            : 85699 GB
Capacity Alloc            : 33135 GB
Percentage Used           : 38%

=== SC 18
Disks in Assigned - Tier3 : 12
Capacity Total            : 20493 GB
Capacity Alloc            : 10188 GB
Percentage Used           : 49%

=== SC 10
Disks in Assigned - Tier1 : 24
Capacity Total            : 12854 GB
Capacity Alloc            : 16 GB
Percentage Used           : 0%

Dell EMC
The power to do more
```

5.2 _FolderDestroy

This code can be injected into the working code sample shown in section 5 replacing the placeholder, **"===== YOUR CODE GOES HERE ====="**.

This code recursively parses the specified folder path and permanently destroys all subfolders and volumes within this folder path.

```
def _FolderDestroy(scName, instanceId):
    """
    This function will recusively parse the specified folder path
    and permanently destroy all subfolders and volumes within this
    folder path
    """
    payload = {}

    REST = '/StorageCenter/ScVolumeFolder/%s/VolumeFolderList' %
           (instanceId)
    completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
    json_data = connection.get(completeURL
                               ,headers=header
                               ,verify=verify_cert)
    stdout = json.loads(json_data.text)
    if len(stdout) > 0:
        for i in range(len(stdout)):
            _FolderDestroy(scName, stdout[i]['instanceId'])
            print("INFO: Removing folder %s, instanceId %s") %
                 (stdout[i]['name'], stdout[i]['instanceId'])
            payload = {}
            REST = '/StorageCenter/ScVolumeFolder/%s' %
                   (stdout[i]['instanceId'])
            completeURL = '%s%s' %
                          (baseURL, REST if REST[0] != '/' else REST[1:])
            json_data = connection.delete(completeURL
                                          ,headers=header
                                          ,verify=verify_cert)
            print("")

    payload = {}
    REST = '/StorageCenter/ScVolumeFolder/%s/VolumeList' % (instanceId)
    completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
    json_data = connection.get(completeURL
                               ,headers=header
                               ,verify=verify_cert)
    stdout = json.loads(json_data.text)
    if len(stdout) > 0:
        for i in range(len(stdout)):
            print("INFO: Removing volume %s, instanceId %s") %
                 (stdout[i]['name'], stdout[i]['instanceId'])
            payload = {}
            REST = '/StorageCenter/ScVolume/%s' % (stdout[i]['instanceId'])
```

```
                completeURL = '%s%s' %
                              (baseURL, REST if REST[0] != '/' else REST[1:])
                json_data = connection.delete(completeURL
                                              ,headers=header
                                              ,verify=verify_cert)

"""
Call the _FolderDestroy function and instructing it to destroy all
subfolders and volumes contained within the /Unix/Linux/RestTest path on SC
9
"""
_FolderDestroy("SC 9 - 101"
               ,volFolderList[os.path.basename("/Unix/Linux/RestTest")]
                             ["SC 9 - 101"]
                             ['instanceId']
               )
print("")
```

5.3 _VolFolderRepl

This code can be injected into the working code sample shown in section 5 replacing the placeholder, **"===== YOUR CODE GOES HERE ====="**.

This code parses the source folder on the source SC Series array and replicates all volumes contained within to a matching folder structure on the target SC Series array.

```
def _VolFolderRepl(scName, instanceId, scTarget):
    """
    This function will parse the specified source folder path on
    the designated SC array and replicate all volumes contained within to
    a matching target folder path on the target SC array. This function
    will not parse / navigate subfolders within the source folder
    assumptions:
    . destination volume name will be prefixed with 'Repl of'
    . sync mode is assumed to be Async unless explicitly defined with
       the SyncMode param
    . QosNode will assume / use the first QosNode definition from the list
    """

    # capture all QosNode options for scName

    payload = {}
    REST = '/StorageCenter/ScReplicationQosNode'
    completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
    json_data = connection.get(completeURL
                               ,headers=header
                               ,verify=verify_cert)
    qos_out = json.loads(json_data.text)
    print("Qos Nodes available on %s:") % (scName)
    qos_list = []
```

```
        for item in range(len(qos_out)):
            if qos_out[item]['scName'] == scName:
                print(qos_out[item]['name'])
                qos_list.append(qos_out[item]['instanceId'])
        print("")
        print(qos_list)

        # parse volume folder and replicate volumes in for loop

        payload = {}
        REST = '/StorageCenter/ScVolumeFolder/%s/VolumeList' % (instanceId)
        completeURL = '%s%s' % (baseURL, REST if REST[0] != '/' else REST[1:])
        json_data = connection.get(completeURL
                                   ,headers=header
                                   ,verify=verify_cert)
        stdout = json.loads(json_data.text)
        if len(stdout) > 0:
            for i in range(len(stdout)):
                print("INFO: Repl volume %s, instanceId %s to %s") %
                      (stdout[i]['name']
                      ,stdout[i]['instanceId']
                      ,scTarget)
                payload = {}
                payload['DestinationStorageCenter'] =
                    scList[scTarget]['instanceId']
                payload['QosNode'] = qos_list[0]
                payload['StorageCenter'] = scList[scName]['instanceId']
                payload['SourceVolume'] = stdout[i]['instanceId']
                payload['ReplicateActiveReplay'] = True
                REST = '/StorageCenter/ScReplication'
                completeURL = '%s%s' %
                    (baseURL, REST if REST[0] != '/' else REST[1:])
                json_data = connection.post
                               (completeURL
                               ,data=json.dumps(payload
                                                ,ensure_ascii=False
                                                ).encode('utf-8')
                               ,headers=header
                               ,verify=verify_cert)
                json_out = json.loads(json_data.text)
        else:
            print("ERRO: Folder contains no volumes to replicate.")

    """
    Call the _VolFolderRepl function and instructing it to replicated
    all volumes contained within folder /Unix/Solaris on SC 4 to a
    matching folder structure on SC 25
    """
    _VolFolderRepl(
        "SC 4",
```

```
            volFolderList[os.path.basename("/Unix/Solaris")]["SC 4"]['instanceId'],
           "SC 9 - 101")
    print("")
```

5.4 _DispAllVolsMapped

This code can be injected into the working code sample shown in section 5 replacing the placeholder, **"===== YOUR CODE GOES HERE =====**".

This code receives a server object name as input, queries all SC Series arrays managed by the DSM installation, and displays each storage array where this server object exists, how many volumes are mapped to this server object, and the individual names of each volume object along with their unique index ID value and path.

```
def _DispAllVolsMapped(srvName):
    """
    Receives the server object name as srvName, and applies this to
    all known objects in the scListdictionary querying for all
    volume objects mapped to this server object
    """
    print("INFO: Querying all SC arrays for server object %s") % (srvName)
    print("")
    for key in scList:
        print("=== %s") % (key)
        volsMapped = {}
        payload = {}
        if srvList[srvName].has_key(key):
            REST = 'StorageCenter/ScServer/%s/MappingList' %
                   srvList[srvName][key]['instanceId']
            completeURL = '%s%s' % (baseURL,
                                    REST if REST[0] != '/' else REST[1:])
            json_data = connection.get(completeURL
                                       ,headers=header
                                       ,verify=verify_cert)
            stdout = json.loads(json_data.text)
            if (len(stdout)) > 0:
                for i in range(len(stdout)):
                    if (stdout[i]['volume']['instanceId']) in volsMapped:
                        continue
                    else:
                        volsMapped[stdout[i]['volume']['instanceId']] = {}
                        volsMapped[stdout[i]['volume']['instanceId']]
                                  ['instanceName'] =
                            stdout[i]['volume']['instanceName']
            else:
                print("INFO: Srv obj %s on %s is not mapped to any volumes") %
                      (srvName, key)
            print("INFO: Volumes mapped: %s") % (len(volsMapped))
            print("%-30s %-15s %s") % ("Volume Name", "InstanceId", "Path")
            for vol_key in volsMapped:
```

```
                print("%-30s %-15s %s") %
                    (volsMapped[vol_key]['instanceName'],
                     vol_key,
                     volList[volsMapped[vol_key]['instanceName']][key]['path'])
        else:
            print("WARN: Srv obj %s does not exist on %s") % (srvName, key)
        print("")

"""
Initiate the function to query the DSM installation for all instances of
the server object named rhevm across all SC arrays
"""
_DispAllVolsMapped("rhevm")
print("")
```

A Configuration details

Table 12 Component table

Component	Description
OS	Fedora 23 Server and Python 2.7.10
Cabling	Fibre Channel
Server	Dell PowerEdge™ R620
Storage	Dell Storage Center OS (SCOS) 6.5.30 with virtual port mode
Enterprise Manager/Dell Storage Manager	Enterprise Manager 2015 R3 or newer, or Dell Storage Manager 2016 R1 or newer

B Additional resources

B.1 Technical support and resources

Dell.com/support is focused on meeting your needs with proven services and support.

Storage technical documents and videos provide expertise that helps to ensure customer success on Dell EMC storage platforms.

B.2 Related documentation

Table 13 lists the referenced or recommended resources related to this document.

Table 13 Referenced or recommended resources

Vendor	Resource
Dell	*Dell Storage Center System Manager Administrator's Guide* available on the Knowledge Center at the SC Series Portal (login required)
Dell	*Dell Storage Center Connectivity Guide* available on the Knowledge Center at the SC Series Portal (login required)
Dell	*Dell Storage Center 6.0 Command Utility (CompCU) Reference Guide* available on the Knowledge Center at the SC Series Portal (login required)
GitHub	https://github.com/openstack/cinder/tree/master/cinder/volume/drivers/dell_emc
Python	https://www.python.org/