NetApp® SANtricity® Rest API and Client Libraries 1.0

Developers Guide

October 2016 | 215-11381_A0
doccomments@netapp.com

NetApp

Table of Contents

About this Guide

This guide describes general concepts, basic REST API workflows, and client libraries setup.

Overview of the Rest API

The SANtricity REST API is an application programming interface designed for experienced developers. Actions performed through the REST API are applied upon execution and without user prompts or confirmation dialogs. The REST API is URL-based, and the accompanying API documentation is completely interactive. Each URL contains a description of the corresponding operation and the ability to perform the action directly through the API Documentation. The API Documentation is accessible by navigating to http://localhost:8080/docs/rest/index.html through a browser.

Each URL endpoint presented within the API documentation has a corresponding POST, DELETE, or GET option. These URL endpoint options, or more properly known as HTTP verbs, are the actions available to the developer through the API documentation. For more information on HTTP verbs, refer to HTTP Verbs.

Data within the REST API is encoded through JavaScript Object Notation (JSON). The structured JSON data from the REST API can be easily parsed by programming languages (C, C++, cURL, Java, Python, Perl, etc.) JSON is a simple key-value pair based encoding with support for list and subject objects. Objects start and end with curly braces (i.e., { }) while lists start and end with brackets (i.e., [ ]). JSON understands values that are strings, numbers, and booleans. Numbers are floating point values. The API documentation provides a JSON template for each applicable URL operation, allowing the developer to simply enter parameters under a properly formatted JSON command.

Overview of the Client Libraries

The SANtricity REST API provides client libraries for Python and Java software development kits (SDK). Used in conjunction with the Web Services Proxy, the Python and Java SDKs facilitate access to the storage system for automation and integration into third-party web or script-based management tools. Each SDK library fully externalizes the Rest API, allowing developers to make calls on specific objects as needed. A primary benefit of the Python and Java SDK is the reduced time spent on development due to the benefits of the REST-based APIs. The setup needed to utilize the SDKs varies based on the library.

For information on how to install the Web Services Proxy, refer to the *NetApp SANtricity Web Services Proxy Installation Guide*.

For detailed information on how to install and get started using the SDKs, refer to the Python SDK and Java SDK sections.

General API Concepts

API Version

The REST API utilizes two top level URLs. The utils URL is an un-versioned URL used to view information about the version of the API. The second top-level URL are the version URLs. Currently, the REST API supports v2. v1 was a pre-release of the REST API, which never received a general release and is not included in this REST API.

v2 is the first version of the REST API to receive a general release. Because v2 has undergone revisions since its initial release, you should review the specific build of the REST API to verify you have the needed build level. However, the overall structure of the URLs is governed by the version.

When v3 is released, support will be continued for v2, v3, and other versions going forward until v2 is eventually deprecated. Currently, there are no plans to deprecate v2.

Logging in to the API

Web Services Proxy has two default user logins and permission levels:

- Read-write access
  - User ID: `rw`
  - Password: `rw`
- Read-only access
  - User ID: `ro`
  - Password: `ro`

To log in, type the following URL in a web browser:

- http://<host:port>/utils/login

In addition, you can use “Basic Authentication” to log in to the service. If a login session has not been established, a Basic Authentication challenge is sent to the client. User profile and password management for the API is performed through the `users.properties` file (located under the directory folder for the installation by default). For more information on how to change passwords and manage user profiles through the `users.properties` file, refer to the *NetApp SANtricity Web Services Proxy User Guide*.

Embedded versus Proxy

Embedded Web Services

The RESTful API server is embedded on the E2800 hardware platform. The APIs can be accessed over the HTTPs protocol on port 8443 (default). In reference to the storage array on which the REST service is running, the storage-system ID parameter is set to `1` for the embedded web service by default. For information on setting up SSL certificates, refer to SSL Configuration and Using Java SDK.

Proxy Mode

The RESTful API server is installed separately on a host machine. The APIs can be accessed through either HTTP (8080) or HTTPS (8443) protocols. Storage arrays must be added to the proxy server. For information on how to add a storage array to the proxy server, refer to Managing Storage Systems.

HTTP Verbs

URLs within the REST API utilize the HTTP verbs POST, GET, and DELETE. The HTTP verbs within the REST API documentation allow each URL to provide interactive functionality. DELETE removes items or resets a counter. GET is utilized for retrieval and POST causes an action to occur. Read-only access is required to perform GET. To perform POST or DELETE, you must have read-write access.

Some REST APIs used different types of verbs than what we utilize. For examples, some APIs use PUT instead of POST. Even when updating an object, v2 utilizes POST throughout the REST API in favor of PUT.

HTTP Status Codes

HTTP status codes return information about a performed operation. Whenever a HTTP status code is returned, a corresponding code number and text description of the operation is displayed. The following are the most common HTTPS status codes returned with the REST API:

Code	Error Description
200	OK – The operation succeeded.
201	Create – The operation succeeded and the object was created.
404	Object not found – Used in many instances to indicate the requested object is not available.
422	Bad User Input – Used to indicate something incorrect with the user input. The response body is populated with a JSON payload that has details about what went wrong.

General Storage Concepts

Storage Systems

A storage system is a 1:1 representation of an E-Series array. Any management commands made against an array must be directed to a storage-system's unique ID or WWN.

In this guide, array and storage-system are often used interchangeably.

A storage-system is uniquely identified globally by its world-wide-name (WWN) and locally to the name by its ID. The ID may be generated by the system, or provided at creation time. The ID or WWN may be used to identify the storage-system when making API calls through the name.

Disk, Pools, and Volumes

Disk pools and volume groups are modeled identically within the REST API. Disk pools are a collection of drives while volumes are created on disk pools. You have the ability to designate the specific storage type through the `raidLevel` value. For example, a `raidLevel` value of raid1 creates a mirror.

E-Series features such as Dynamic Disk Pools are compatible when creating storage pools with the REST API. For information on how to create a storage pool, refer to Provisioning Storage. For more information on general E-Series concepts such as Dynamic Disk Pools, refer to the *SANtricity Storage Manager Storage Concepts Guide*.

Host and Mappings

On an E-Series storage system, a host object represents a host consuming storage. There are three parts to a host:

- The description of the host
- The host type
- The host port

The host type helps the storage system communicate with the host. The supported host can retrieve from the host-type endpoint. The E-Series storage system can discover available host ports. The unassociated-hosts-ports endpoint shows all available ports. The host type and host port is required when creating a host.

To use a volume, you must first map it to a host. Mapping a volume to a host is performed through the volume mappings endpoint.

SSL Configuration

The following section details SSL configuration through the Web Services Proxy.

Getting started

NOTE: The following SSL configuration workflow process can alternatively be performed directly through the interactive API Documentation.

Secure Sockets Layer (SSL) is utilized to pass traffic through the network securely through a Hyper Text Transfer Protocol Secure (HTTPS) connection. The generation and importing of a signed SSL certificate is required for HTTPS connections. To configure an HTTPS connection, the port must be changed from 8080 to 8443 (if you are using the default port configurations).

When navigating to the 8443 port, a warning message is displayed. This is a result of the server generating out a self-signed SSL certificate during the initial startup. The self-signed SSL certificate is suitable to get an encryption on the channel, but you may still be subject to man-in-the-middle attacks. Using CORS from a browser does not work because it must have a properly signed SSL certificate. In addition, a number of scripting languages have a tendency to encounter issues when attempting to utilize an SSL without a properly generated and signed certificate. To ensure a proper SSL configuration, you must manually generate, sign, and import an SSL certificate.

NOTE: For the embedded API, the default SSL configuration is set to use only the public-facing address for the controller in SANtricity System Manager. SSL is required when accessing through embedded. Non-encrypted connections to embedded are disallowed.

Generating an SSL certificate

Perform the following to generate an SSL certificate:

1. Stop the service using the Services Command.

   NOTE: The Services Command must be run as root.

   The service is stopped.

2. From the default install location, access the working directory.

   NOTE: The default install location is /home/opt/netapp/santricity _web_services_proxy.

3. Under the working directory, remove the previous keystore generated during the server's initial startup.

4. Utilizing the `genkeypair` keytool command already installed on your server, create your own key.

   ```
   keytool -genkeypair -keyalg RSA -keysize 2048 -alias jetty -dname CN=<THE SERVER
   DNS NAME> -keypass changeit -storepass changeit -keystore keystore -ext
   san=ip:<THEIR IP ADDRESS>,dns:<THE SERVER DNS NAME> -validity 999
   ```

   or

   ```
   keytool -genkeypair -keyalg RSA -keysize 2048 -alias jetty -dname CN=servername -
   keypass changeit -storepass changeit -keystore keystore -ext
   san=ip:192.168.1.1,dns:servername -validity 999
   ```

   NOTE: It is recommended you use a RSA key with 2048 or better.

5. Specify the alias as jetty.
6. For the distinguished/common name, enter the name of your host.

   NOTE: By default, the password is `changeit`. If needed, the password can be modified under the wsconfig file.

7. Generate the certificate.

   The following message appears in the terminal window.

   ```
   When prompted for a password, use "changeit", unless you specify a specific one in
   the wsconfig.xml file
   When prompted for your first and last name, use the IP address or DNS name of the
   host, whichever one you plan on using in URLs
   ```

   A new self-generated SSL certificate is created. A certificate authority known to your web browser and client must now sign the self- generated certificate.

Signing the generated SSL certificate

Perform the following to sign a generated SSL certificate:

1. Using the certificate request command, generate a certificate request.

   ```
   keytool -certreq -alias jetty -file mycertreq.cet -keystore keystore -dname
   CN=servername -ext san=ip:192.168.1.1,dns:servername
   ```

   NOTE: The file name entered under the certificate request command is left to your discretion.

   A certificate request is generated. The certificate request file now must now be signed by the certificate authority.

2. Send the generated certificate request file to the certificate authority of your choosing.

   NOTE: NetApp does not endorse nor promote a specific certificate authority.

   The certificate authority signs the certificate request and returns a signed certificate. In addition, you receive a certificate from the certificate authority itself. This certificate must be imported into your keystore.

Importing the certificate into the keystore

Perform the following to import the certificate into the keystore:

1. Import the certificate authority certificate into the certificate authority keystore through the keytool.

   ```
   keytool -import -trustcacerts -alias root -file <CA CERT FILE> -keystore keystore
   keytool -import -trustcacerts -alias jetty -file <signed cert from ca> -keystore
   keystore
   ```

   NOTE: When importing the certificate authority certificate, you must specify the alias it was generated from, in this case jetty.

   The certificate authority certificate is imported into the keystore.

2. Through the Service Command, restart the service.

The service is started.

3. Save the certificate in your keystore.
4. Within a web browser, perform an HTTPS command.

   A certified SSL connection is established.

Generating an SSL Certificate on an Application client

If you do not already have the certificate, import it from the certifying authority. Follow the prescribed import process for your specific operating system and web browser.

Managing Storage Systems

The following section details managing storage systems through the Web Services Proxy.

A storage-system object can be thought of as representing the management connection to a particular array. Deleting a storage-system from the name does not destroy any objects that are currently defined on the array. Likewise, adding a storage-system to the name merely initiates connectivity to the array so that the name may begin monitoring the status of the array and make it available for requests through the REST API. The first step in managing a storage array is adding the storage array.

Auto-discovering a storage array

NOTE: You can bypass the manual discovery of storage arrays process if you already know the IP address for the physical array.

Perform the following to manually discover a storage array:

1. From the API Documentation, go to POST /discovery.

   The POST object for the discovery endpoint is displayed.

2. Under the system-id field within the GET object, enter the system ID for the storage system.

3. Click the **Model Schema**.

   The JSON command for the POST object is displayed under the Body field.

4. Under the `startIP` value, enter the starting IP address for the network scan.

5. Under the `endIP` value, enter the ending IP address for the network scan.

   NOTE: Through the specified `startIP` and `endIP` values, you can conduct narrow or broad, enterprise-wide network scans.

6. To utilize in-band agents for the network scan, enter `true` under the `useAgents` value.

7. If needed, enter the number of seconds for the connection timeout setting under the `connectionTimeout` value.

   NOTE: The `connectionTimeout` value is configured to 30 seconds by default.

8. To specify the maximum number of ports used for the network scan, enter the desired number under the `maxPortsToUse` value.

9. Review the updated JSON command for the POST object.

   Example

```
{
 "startIP": "xxx.x.x.x1",
 "endIP": "xxx.x.x2",
 "useAgents": "true",
 "connectionTimeout": 30,
 "maxPortsToUse": 0
}
```

10. Click **Try it out!**

    Details for the network scan action is displayed in the Response Body section of the POST operation.

11. Verify a Response Code of 200 is displayed under the Response Body section.

    NOTE: If a Response Code of 200 is not displayed under the Response Body section, the POST operation for the discovery end-point has been unsuccessful.

12. Copy the Requested URL under the POST operation.

13. Go to GET /discovery.

14. Paste the Requested URL from the POST operation under the `requestID` field within the GET operation.

    NOTE: If an entry is not entered under the `requestID` field, the most current URL is utilized by default.

15. Click **Try it out!**

    Results for the discovery action are displayed under the Response Body section of the GET operation.

16. Under the Response Body, locate and copy the `wwn` value.

17. Paste the `wwn` value under your copy buffer.

Manually Adding a Storage Array

Perform the following to manually add a storage array:

1. From the API Documentation, go to POST /storage-systems.

   The POST object for the storage systems endpoint is displayed.

2. Click the **Model Schema**.

   The JSON command for the POST object is displayed under the Body field.

3. To assign an ID to the storage system, enter the desired identifier under the id value.

   NOTE: If you do not specify an identifier under the id value, an identifier is generated for the storage system automatically.

4. Under the `controllerAddresses` value, enter the IP address for the storage system.

   NOTE: You need to specify a single IP address under the `controllerAddresses` value, because the POST operation locates the other IP addresses within the network.

5. If applicable, enter the password for the storage system under the `password` value.

   NOTE: If the storage system does not require a password, leave the `password` value blank.

6. Go to your copy buffer, and retrieve the copied `wwn` value from the GET /discovery operation.

7. Under `wwn` within the POST /storage-systems operation, paste the copied `wwn` value from the GET /discovery operation.

8. To enable trace logging for the storage system, enter `true` under the `enableTrace` value.
9. To associate meta tags to the storage system, enter the information under the `metaTags` values.

   NOTE: You can delete the `enableTrace` and `metaTags` values under the Body field if you choose not to configure these parameters.
10. Review the updated JSON command for the POST object.

    Example

```
{
 "id": "1",
 "controllerAddresses": [
 "xxx.x.x.x"
 ],
 "password": "",
 "wwn": "X0X8000050X0X0X0X0X0X0X0X0X0X0X0"
}
```

11. Click **Try it out!**

    Results for the added storage system action is displayed in the Response Body section of the POST operation.
12. Verify a Response Code of 201 displays under the Response Body section.

    NOTE: If a Response Code of 201 does not display under the Response Body section, the POST operation for the host end-point has been unsuccessful.
13. Go to GET /storage-systems.

    The GET object for the storage systems endpoint is displayed.
14. Click **Try it out!**

    Details for the added storage system is displayed under the Response Body section of the GET operation.
15. Copy the Requested URL under the GET operation.
16. To view the results for the global events through the command line, open a command line tool.

    NOTE: You can also view results for the global events through a browser by entering the copied Requested URL under the address bar.
17. Enter the copied URL under the command line tool.

    Example

    http://localhost:8080/devmgr/v2/storage-systems/

    Results for the global events is displayed under the command line tool.

Removing a Storage Array

Perform the following to manually remove a storage array:

1. From the API Documentation, go to DELETE /storage-systems/[system-id]/.

   The DELETE object for the storage system endpoint is displayed.

2. Under the system-id field within the DELETE object, enter the system ID for the storage system.

3. Click **Try it Out!**

   Results for the removed storage system is displayed under the Response Body section of the DELETE operation.

   NOTE: In the context of a storage-system, deletion is actually just a removal from management. Various cached data (i.e., statistics, events, etc.) may be lost upon removal from the name, but the array and its data are not directly affected.

Provisioning Storage

The following section details provisioning storage through the Web Services Proxy.

Creating Storage Pools

Perform the following to create a storage pool:

1. From the API Documentation, go to GET /storage-systems/(system-id)/drives/.

   The GET object for the Drives endpoint is displayed.

2. Under the system-id field within the GET object, enter the system ID for the storage system.
3. Click **Try it Out!**

   A list of all hard drives in the storage system displays under the Response body of the GET object.

4. Under the Response body of the GET object, locate the first `driveRef` value.
5. Copy the value for the first `driveRef` value.
6. Paste the first `driveRef` value into your copy buffer.
7. Under the Response body of the GET object, locate the second `driveRef` value.
8. Repeat Step 5 through 7 to copy the second `driveRef` value into your copy buffer.
9. Navigate to POST /storage-systems/(system-id)/storage-pools/.

   The POST object for the Storage Pools endpoint is displayed.

10. Under the system-id field within the POST object, enter the system ID for the storage system.
11. Click the **Model Schema**.

    The JSON command for the POST object is displayed under the Body field.

12. Retrieve the copied `driveRef` values from your copy buffer.
13. Locate `diskDriveIds` under the Body field.
14. Paste the copied `driveRef` values into the `diskDriveIds` value under the Body field.
15. Under the `Name` value, enter the desired name for the storage pool.
16. Under the `raidLevel` value, enter the desired raid level for the storage pool.

    NOTE: For reference, click **Model** under the POST operation to view all possible values for `raidLevel`. For example, entering a value of `raid0` under `raidLevel` creates a stripe storage pool.

17. Review the updated JSON command for the POST object.

Example

```
{
 "raidLevel": "raid0",
 "diskDriveIds": [
 "3231343336353837303932383200000000000000",
"3213433363538373039323838000000000000000"
 ],
 "eraseSecuredDrives": "false",
 "name": "Example Storage Pool"
}
```

18. Click **Try it out!**

    Details for the created storage pool action is displayed in the Response Body section of the POST operation.

19. Verify a Response Code of 200 displays under the Response Body section.

    NOTE: If a Response Code of 200 does not display under the Response Body section, the POST operation for the Storage Pool end-point has been unsuccessful.

20. Copy the Requested URL under the POST operation.

21. To view the results for the created storage pools through the command line, open a command line tool.

    NOTE: You can also view results for the created storage pools through the GET operation for the Storage Pools endpoint within the API documentation.

22. Enter the copied URL under the command line tool.

    Example

    http://localhost: 8080/devmgr/v2/storage-systems/1/storage-pools/

    Results for the created storage pool is displayed under the command line.

23. For future volume creation, locate and copy the `volumeGroupRef` value under the storage pool results within the command line tool.

24. Paste the `volumeGroupRef` value under your copy buffer.

Creating Volumes

You must first create a storage pool before creating a volume.

Perform the following to create a volume:

1. From the API documentation, go to POST /storage-systems/(storage-systems)/[system-id]/volumes/.
2. Enter the system ID for the storage system under the system-id field.
3. Under the POST object, click the **Model Schema**.

   The JSON command for the POST object is displayed under the Body field.

4. Retrieve the `volumeGroupRef` value from your copy buffer.

NOTE: The `volumeGroupRef` value is obtained through the Creating Storage Pools procedure.

5. Under the `poolId` value within the POST object, insert the copied `volumeGroupRef` value.
6. Under the `name` value, enter the desired name for the volume.
7. Under the `sizeUnit` value, enter the size unit for the volume (e.g., enter gb to create a gigabyte unit).

   NOTE: For reference, click **Model** under the POST operation to view all possible size unit values for `sizeUnit`.

8. Under the `size` value, enter the size of the unit for the volume.
9. Under the `segSize` value, enter the segment size for the volume.

   NOTE: A default value is utilized whenever the `segSize` value is left blank.

10. Review the updated JSON command for the POST object.

    Example

```
{
 "poolId": "3233343536373839303132333100000000000000",
 "name": "testvol1",
 "sizeUnit": "gb",
 "size": 2,
 "segSize": 32
}
```

11. Click **Try it out!**

    Details for the created volume action is displayed in the Response Body section of the POST operation.

12. Verify a Response Code of 200 displays under the Response Body section.

    NOTE: If a Response Code of 200 does not display under the Response Body section, the POST operation for the Volume end-point has been unsuccessful.

13. Copy the Requested URL under the POST operation.
14. To view the results for the created volume through the command line, open a command line tool.

    NOTE: You can also view results for the created volume through the GET operation for the volumes endpoint within the API documentation.

15. Enter the copied URL under the command line tool.

    Example

    http://localhost: 8080/devmgr/v2/storage-systems/1/volumes/

    Results for all volumes on the storage system is displayed under the command line tool.

16. Under the displayed volume results, locate and copy the `volumeRef` value for the created volume.
17. Paste the `volumeRef` value into your copy buffer.

NOTE: To display results for a specific volume, enter the volume ID (i.e., `volumeRef` value) at the end of the copied URL.

Setting up the Host

Retrieving the Host Type

Perform the following to retrieve the host type:

1. From the API documentation, go to GET /storage-systems/[system-id]/host-types/.
2. Enter the system ID under the system-id field within the GET object.
3. Click **Try it out!**

   Results are displayed under the Response Body section of the GET operation.
4. Under the Response Body section, copy the host type structure.

```
[
  {
    "name": "Windows Server 2003",
    "index": 0,
    "code": "Windows Server 2003",
    "used": true,
    "default": true
  }
]
```

5. Paste the host type structure into your copy buffer.

Retrieving the Host Port

Perform the following to retrieve the host port:

1. From the API documentation, go to GET /storage-systems/[system-id]/host-ports/.
2. Enter the system ID under the system-id field within the GET object.
3. Click **Try it out!**

   Results are displayed under the Response Body section of the Get operation.
4. Under the Response Body section, copy the `address` value.
5. Paste the `address` value into your copy buffer.

Creating the Host

Perform the following to create the host:

1. From the API documentation, go to POST /storage-systems/[system-id]/hosts/.
2. Enter the system ID for the storage system under the system-id field.
3. Under the POST object, click the **Model Schema**.

The JSON command for the POST object is displayed under the Body field.

4. Under the `Name` value, enter the desired name for the host.
5. Retrieve the host type structure from your copy buffer.
6. Under the `hostType` value within the POST object, insert the copied host type structure.

   NOTE: For reference, click **Model** under the POST operation to view details for all possible values under the Body field.

7. Retrieve the `address` value from your copy buffer.
8. Under the `port` value within the POST object, insert the copied `address` value.
9. Under the `label` value, enter the desired name for the host port.
10. Under the `type` value, enter **fc** for Fiber Channel.
11. Review the updated JSON command for the POST object.

```
{
 "name": "win-host-2",
  "hostType": {
    "name": "Windows Server 2003",
    "index": 0,
    "code": "Windows Server 2003"
    "default": true,
    "used": true
 },
 "ports": {
    "type": "fc",
    "ports": "3738393031323438"
    "label": "port1"
 }
}
```

12. Click **Try it out!**

    Details for the created host action is displayed in the Response Body section of the POST operation.

13. Verify a Response Code of 200 displays under the Response Body section.

    NOTE: If a Response Code of 200 does not display under the Response Body section, the POST operation for the host end-point has been unsuccessful.

14. Copy the Requested URL under the POST operation.
15. To view the results for the created host through the command line, open a command line tool.

    NOTE: You can also view results for the created host through the GET operation for the host endpoint within the API documentation.

16. Enter the copied URL under the command line tool.

    Example

http://localhost: 8080/devmgr/v2/storage-systems/1/hosts/

Results for the created host is displayed under the command line tool.

17. For future host mapping, locate and copy the `hostRef` value under the host results within the command line tool.
18. Paste the `hostRef` value under your copy buffer.

Mapping a Volume to a Host

Perform the following map a volume to a host:

1. From the API documentation, go to POST /storage-systems/[system-id]/volume-mappings/.
2. Enter the system ID for the storage system under the system-id field.
3. Under the POST object, click the **Model Schema**.

   The JSON command for the POST object is displayed under the Body field.
4. Retrieve the `volumeRef` value from your copy buffer.
5. Under the `mappableObjectId` value within the POST object, insert the copied `volumeRef` value.
6. Retrieve the `hostRef` value from your copy buffer.
7. Under the `targetId` value within the POST object, insert the copied `hostRef` value.
8. Under the `lun` value within the POST object, enter `1`.

   NOTE: For reference, click **Model** under the POST operation to view details for all possible values under the Body field.
9. Review the updated JSON command for the POST object.

   Example

```
{
 "mappableObjectId": "32333435363738393031323335000000000000",
 "targetId": "32333435363738393031333237000000000000",
 "lun": 1
}
```

10. Click **Try it out!**

    Details for the created volume mapping are displayed in the Response Body section of the POST operation.
11. Verify a Response Code of 200 displays under the Response Body section.

    NOTE: If a Response Code of 200 does not display under the Response Body section, the POST operation for the volume mapping end-point has been unsuccessful.
12. Copy the Requested URL under the POST operation.
13. To view the results for the volume mapping through the command line, open a command line tool.

NOTE: You can also view results for the volume mapping through the GET operation for the volume mapping endpoint within the API documentation.

14. Enter the copied URL under the command line tool.

    Example

    http://localhost: 8080/devmgr/v2/storage-systems/1/volume-mappings/

    Results for the volume mapping is displayed under the command line tool.

Monitor Storage

The following section details monitoring storage systems through the Web Services Proxy.

Monitoring Global Events

Perform the following to monitor global status events:

1. From the API Documentation, go to GET /events.

   The GET object for the global events endpoint is displayed.

2. If needed, enter the sequence number of the last received event under the `lastKnown` field.

3. If needed, enter the number of seconds to wait for a new event under the `wait` field.

4. Click **Try it out!**

   A list of global events is displayed under the Response Body of the GET object.

5. Copy the Requested URL under the GET operation.

   NOTE: The parameters display at the end of the Request URL in a `KEY=VALUE` format separated by an ampersand (&).

6. To view the results for the global events through the command line, open a command line tool.

   NOTE: You can also view results for the global events through a browser by entering the copied Requested URL under the address bar.

7. Enter the copied URL under the command line tool.

   Example

   http://localhost:8080/devmgr/v2/storage-systems/1/events?lastKnown=1&wait=30

   Results for the global events is displayed under the command line tool.

Monitoring the Status of Events for the Storage Device

Perform the following to monitor the status of events for a storage device:

1. From the API Documentation, go to GET /storage-systems/(system-id)/events/.

   The GET object for the Storage Device Events endpoint is displayed.

2. Under the system-id field within the GET object, enter the system ID for the storage system.

3. If needed, enter the sequence number for the last received event under the `lastKnown` field.

4. If needed, enter the number of seconds to wait for a new event under the `wait` field.

5. Click **Try it out!**

   The status of events for the specified storage system is displayed under the Response Body of the GET object.

6. Copy the Requested URL under the GET operation.

   NOTE: The parameters display at the end of the Request URL in a `KEY=VALUE` format separated by an ampersand (&).

7. To view the status of events for the storage device through the command line, open a command line tool.

   NOTE: You can also view the status of events for the storage device through a browser by entering the copied Requested URL under the address bar.

8. Enter the copied URL under the command line tool.

   Example

   http://localhost:8080/devmgr/v2/storage-systems/1/events?lastKnown=-1&wait=30

   Results for the status of events for the storage device is displayed under the command line tool.

Monitoring MEL Events

Perform the following to monitor the MEL Events:

1. From the API Documentation, go to GET /storage-systems/(system-id)/mel-events/.

   The GET object for the MEL Events endpoint is displayed.

2. Under the system-id field within the GET object, enter the system ID for the storage system.

3. If needed, enter the starting sequence number for the query under the `startSequenceNumber` field.

4. If needed, enter the maximum number of retrieve results for the query under the `count` field.

5. If needed, specify whether the query only retrieves results currently in the cache under the `cacheOnly` field.

6. If needed, specify whether the query only retrieves results classified as critical under the `critical` field.

7. If needed, specify whether the query retrieves debug entries under the `includeDebug` field.

8. Click **Try it out!**

   A list of MEL events matching the specified query is displayed under the Response Body of the GET object.

9. Copy the Requested URL under the GET operation.

   NOTE: The parameters display at the end of the Request URL in a `KEY=VALUE` format separated by an ampersand (&).

10. To view the results for the MEL events through the command line, open a command line tool.

    NOTE: You can also view results for the MEL events through a browser by entering the copied Requested URL under the address bar.

11. Enter the copied URL under the command line tool

    Example

    http://localhost:8080/devmgr/v2/storage-systems/1/mel-events?startSequenceNumber=-1&count=100&cacheOnly=false&critical=false&includeDebug=false

    Results for the MEL events is displayed under the command line tool.

Firmware Management

The following section details managing firmware through the Web Services Proxy.

Obtaining a list of available Firmware Files

Perform the following to obtain a list of available Firmware Files:

1. From the API Documentation, go to GET /firmware/cfw-files.

   The GET object for the controller firmware files endpoint is displayed.

2. Click **Try it out!**

   Results for the controller firmware list action are displayed in the Response Body section of the GET operation.

3. Verify a Response Code of 200 displays under the Response Body section.

   NOTE: If a Response Code of 200 does not display under the Response Body section, the GET operation for the controller firmware list end-point has been unsuccessful.

4. Copy the Requested URL under the GET operation.

5. To view the details for the controller firmware list through the command line, open a command line tool.

   NOTE: You can also view results for the controller firmware list through a browser by entering the copied Requested URL under the address bar.

6. Enter the copied URL under the command line tool.

   Example

   http://localhost:8080/devmgr/v2/firmware/cfwfiles

   Detailed information for the controller firmware list is displayed under the command line tool.

   NOTE: Firmware files can be uploaded through the /firmware/upload endpoint.

Verifying Firmware File Compatibility

Perform the following to verify Firmware File compatibility for a storage system:

1. From the API Documentation, go to POST /firmware/compatibility-check.

   The POST object for the controller firmware compatibility check is displayed.

2. Click the **Model Schema**.

   The JSON command for the POST object is displayed under the Body field.

3. Under the `storageDeviceIds` value, specify the desired storage systems to verify firmware compatibility.

   NOTE: If needed, you can specify multiple storage devices under `storageDeviceIds` as comma separated entries.

4. Review the updated JSON command for the POST object.

   Example

```
{
 "storageDeviceIds": [
 "xxxxxxxx-xxxx-xxxx-xxxx-xxxx", "xxxxxxxx-xxxx-xxxx-xxxx-xxxx"
 ],
 "releasedBuildsOnly": "true"
}
```

5. Click **Try it out!**

   Results for the controller firmware compatibility action are displayed in the Response Body section of the POST operation.

6. Verify a Response Code of 200 displays under the Response Body section.

   NOTE: If a Response Code of 200 does not display under the Response Body section, the POST operation for the controller firmware compatibility end-point has been unsuccessful.

7. Copy the Requested URL under the POST operation.

8. Go to GET /firmware/compatibility-check.

   The GET object for the firmware compatibility check is displayed.

9. Paste the copied Requested URL from the POST operation under the `requestID` field of GET /firmware/compatibility-check.

10. Click **Try it out!**

    Results for the controller firmware compatibility status is displayed under the Response Body section of the GET operation.

11. Verify a Response Code of 200 displays under the Response Body section.

    NOTE: If a Response Code of 200 does not display under the Response Body section, the GET operation for the controller firmware list end-point has been unsuccessful.

12. Copy the Requested URL under the GET operation.

13. To view the status of the controller firmware compatibility check through the command line, open a command line tool.

    NOTE: You can also view status for the controller firmware compatibility check through a browser by entering the copied Requested URL under the address bar.

14. Enter the copied URL under the command line tool.

    Example

    http://localhost:8080/devmgr/v2/firmware/compatibility-check

    Results for the controller firmware compatibility check are displayed under the command line tool.

Upgrading Controller Firmware

Perform the following to initiate a controller firmware upgrade on a storage system:

1. From the API Documentation, go to POST /storage-systems/[system-id]/cfw-upgrade.

   The POST object for the controller firmware upgrade endpoint is displayed.

2. Under the system-id field within the POST object, enter the system ID for the storage system.
3. Click the **Model Schema**.

   The JSON body for the POST object is displayed under the Body field.

4. To identify a specific controller firmware file for the upgrade, enter the filename under the `cfwFile` value.
5. To identify a specific nvsram file for the upgrade, enter the item under the `nvsramFile` value.
6. To stage the firmware, enter true under the `stageFirmware` value.
7. To skip the check of MEL events for the storage system, enter true under the `skipMelCheck` value.
8. Review the updated JSON command for the POST object.

   Example

```
{
 "cfwFile": ""
 "nvsramFile": ""
 "stageFirmware": "false",
 "skipMelCheck": "false"
}
```

9. Click **Try it out!**

   Results for the controller firmware upgrade action are displayed in the Response Body section of the POST operation.

10. Verify a Response Code of 202 displays under the Response Body section.
11. Copy the Requested URL under the POST operation.
12. To view details for the controller firmware upgrade through the command line, open a command line tool.

    NOTE: You can also view details for the controller firmware upgrade check through a browser by entering the copied Requested URL under the address bar.

13. Enter the copied URL under the command line tool.

    Example

    http://localhost:8080/devmgr/v2/storage-systems/1/cfw-upgrade

    Details for the controller firmware upgrade are displayed under the command line tool.

Using Python SDK

Downloading the SANtricity WebAPI – Python SDK Client Library

GitHub Repository

The SANtricity WebAPI - Python SDK client library is available at the following location on the NetApp GitHub repository:

- https://github.com/NetApp/santricity-webapi-pythonsdk.

PyPI

You can also download the SANtricity WebAPI – Python SDK Client Library from PyPi using `pip install netapp.santricity`.

Requirements

The SANtricity WebAPI - Python SDK client library requires Python 2.7 or later.

Installation

setuptools Installation

You can install the Python bindings through the Python setuptools.

After the download is completed, enter the following command:

```
python setup.py install
```

Manual Installation

If you chose not install the Python bindings through setuptools, you can perform the installation manually by first downloading the latest release of the package. After the download is completed, enter the following command to import the package:

```
import netapp.santricity
```

Getting Started

To get started using the Python SDK, access the `api_client.py` file, and specify the host URL for your REST service.

Example – Python Sample Script

```
#/usr/bin/python

from netapp.santricity.configuration import Configuration
from netapp.santricity.api_client import ApiClient
from netapp.santricity.api.v2.storage_systems_api import StorageSystemsApi
from netapp.santricity.models.v2.storage_system_response import StorageSystemResponse
from pprint import pprint

config = Configuration()
config.host = "http://localhost:8080" #
config.username = "rw"
```

```
config.password = "rw"

api_client = ApiClient()
config.api_client = api_client

storage_system = StorageSystemsApi(api_client=api_client)

ssr = storage_system.get_all_storage_systems()
```

Using HTTPS with the Python SDK

The Python SDK uses a Configuration object to manage the primary settings controlling the connection between the application that uses the SDK and the proxy. The following are some of the more relevant settings for using an HTTPS connection to the proxy:

- `host` - The URL of the proxy server to use, including protocol and port
- `ssl_ca_cert` - Path to a certificate bundle to use for verifying the connection
- `verify_ssl` - Parameter to indicate if SSL verification should be performed or not

Additionally, the following two SSL-related parameters are also available, which are used to identify a client certificate to be presented to the server:

- `cert_file`
- `key_file`

Depending on the level of security required, some or all of the settings may be adjusted.

Using HTTPS with the default self-signed certificates

This method is not generally recommended, but is helpful to know how to quickly get started using HTTPS with the proxy.

Configure the Host Attribute

Make sure that you have the URL to the proxy that utilizes SSL. If the default ports were used, this will be port 8443. Enter the following command to specify the host URL for the configuration object:

```
from netapp.santricity.configuration import Configuration
config = Configuration
confg.host = https://testproxy.foo.com:8443
```

Configuring the SSL Verification

If you are using the default self-signed certificate, the client cannot verify the authenticity of the proxy server without additional configuration. In this case, attempts to make an API call results in `certificate verify failed` errors.

To bypass these errors, either disable SSL verification, which is less secure or export and then import the certificate to be used by the client SDK.

Disabling SSL Verification

To bypass `certificate verify failed` errors, set the `verify_ssl` attribute of the Configuration object to `False`.

```
from netapp.santricity.configuration import Configuration
config = Configuration
confg.host = https://testproxy.foo.com:8443
config.verify_ssl = False
```

After configuration, errors do not occur when performing requests. However, a warning appears indicating that an unverified connection was made.

```
/venvs/python-sdk/lib/python2.7/site-packages/urllib3/connectionpool.py:838:
InsecureRequestWarning: Unverified HTTPS request is being made. Adding certificate
verification is strongly advised. See:
https://urllib3.readthedocs.io/en/latest/security.html
InsecureRequestWarning
```

The aforementioned warning can be disabled by adding a couple of lines to the program.

```
import urllib3
urllib3.disable_warnings()
```

Using an exported certificate with the client

Using this method exports the self-signed certificate from the proxy web server and makes it available to the client SDK. This method allows you to keep the SSL setting to `true`.

Export the certificate

Use the Java keytool to export the certificate to the `proxy-cert-export.cer` file.

```
CWD = /opt/netapp/santricity_web_services_proxy
./jre/bin/keytool -export -rfc -alias jetty -storepass changeit -file proxy-cert-
export.cer -keystore working/keystore
```

Convert the certificate to PEM format

The Python SDK expects the file containing the certificate bundle to use to verify the connection to be in PEM format. The export will be in x.509binary format, so it will need to be converted.

```
openssl x509 -inform DER -in proxy-cert-export.cer -out prox-cert-export.pem
```

Configure application to use certificate

Copy the PEM formatted certificate to a location accessible to the application using the client SDK. Update the Configuration object's ssl_ca_certattribute to the location of the file and enable SSL verification.

```
from netapp.santricity.configuration import Configuration
config = Configuration
confg.host = https://testproxy.foo.com:8443
config.verify_ssl = True
config.ssl_ca_cert = "path/to/proxy-cert-export.pem"
```

The client can now make verified SSL connections to the proxy server based on the default self-signed certificate.

Using HTTPS with a signed certificate

This section details how to use an authentic or “real” signed certificate and the SDK. Content within this section is not a complete guide for configuring the Web Services proxy with a signed certificate. For complete information on how to configure the Web Services proxy with a signed certificate, refer to SSL Configuration.

The following sections assume that the proxy server has been configured to use a signed certificate.

Application Configuration

Similar to the self-signed certificate method, you must configure the host to use the HTTPS based URL for the proxy and enable SSL verification. You also must configure the application to use a CA certificate file that contains a set of trusted CA certificates.

```
from netapp.santricity.configuration import Configuration
config = Configuration
confg.host = https://testproxy.foo.com:8443
config.verify_ssl = True
config.ssl_ca_cert = "path/to/ca-file.pem"
```

If the certificate was signed from a trusted authority, there should be no additional configuration needed on the client host (as long as you have access to a PEM file containing the set of trusted root CA certificates).

CA certificate files

The SDK must have access to a file containing a set of trusted CAs. This requirement is applicable regardless if you are using a globally trusted certificate authority, enterprise, or local certificate authority. This file should be in PEM format by default.

Using a custom CA certificate file

If you do not have a certificate file of generally trusted CAs or you choose not to use such a file, you can obtain the root certificate from the issuing CA and that item available to the application in its own file.

Using system-wide trusted CA files

Most operating systems have a system wide repository of trusted CAs. If this is available in PEM format, you can use this as the file specified for the `config.ssl_ca_cert` attribute in the application.

```
Default system trusted CAs = /etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem
```

You can use the system-wide trusted CA files even if you use your own CA to sign the certificate for the web service proxy. In this case, simply add the root certificate for your own CA to the system’s CA configuration.

Using Java SDK

Downloading the SANtricity WebAPI – Java SDK Client Library

GitHub Repository

The SANtricity WebAPI - Java SDK client library is available at the following location on the NetApp GitHub repository:

- https://github.com/NetApp/santricity-webapi-javasdk

Maven Central

You can also download the SANtricity WebAPI – Java SDK Client Library through Maven Central by adding the item to your dependency file.

Requirements

The SANtricity WebAPI - Java SDK client library requires an installation of Apache Maven.

Installation

Our build scripts require Apache Maven for pulling down the dependencies from Maven Central and compiling the source. Instead, if you would like to use a different build system, please check the Dependency List to download the appropriate ones.

The Java SDK client library must be installed to your local Maven repository.

To install the Java SDK, perform the following command:

```
mvn install
```

Optionally, you can deploy the Java SDK installation to a remote Maven repository.

To deploy the Java SDK installation remotely, configure the settings of the repository and perform the following command:

```
mvn deploy
```

Dependencies

Dependency List

The following is a list of used and declared dependencies.

Group ID	Artifact ID	Version	Scope	Type	Optional
io.swagger	swagger-annotations	1.5.8	compile	jar	false
org.glassfish.jersey.core	jersey-client	2.23	compile	jar	false
org.glassfish.jersey.media	jersey-media-multipart	2.23	compile	jar	false

Group ID	Artifact ID	Version	Scope	Type	Optional
org.glassfish.jersey.media	jersey-media-json-jackson	2.23	compile	jar	false
org.glassfish.jersey.core	jersey-common	2.23	compile	jar	false
com.fasterxml.jackson.core	jackson-annotations	2.7.0	compile	jar	false
com.fasterxml.jackson.core	jackson-databind	2.7.0	compile	jar	false
com.fasterxml.jackson.datatype	jackson-datatype-joda	2.1.5	compile	jar	false
javax.ws.rs	javax.ws.rs-api	2.0.1	compile	jar	false
com.brsanthu	migbase64	2.2	compile	jar	false

Maven Users

The following dependency must be added to your project's POM:

```
<dependency>
   <groupId>com.netapp.santricity</groupId>
   <artifactId>santricity-java-client</artifactId>
   <version>1.0</version>
   <scope>compile</scope>
</dependency>
```

Gradle Users

The following dependency must be added to your project's build file:

```
compile "com.netapp.santricity:santricity-java-client:1.0"
```

All Other Users

First, enter the following command to generate the JAR file:

```
mvn package
```

Next, manually install the following JAR files:

- `target/santricity-java-client-1.0.0.jar`
- `target/lib/*.jar`

Getting Started

After installation is complete, start the Java SDK through command prompt.

Example

```
import com.netapp.santricity.ApiClient;
```

```
import com.netapp.santricity.ApiException;
import com.netapp.santricity.api.v2.StorageSystemsApi;
import com.netapp.santricity.models.v2.*;
import com.netapp.santricity.api.v2.DiagnosticsApi;

import java.util.ArrayList;
import java.util.List;

import static java.lang.Thread.currentThread;
import static java.lang.Thread.sleep;

/**
 * Sample code illustrates SDK usage for gathering failure list
 */
public class FailureList {
    public static void main(String[] args) {
        ApiClient apiClient = new ApiClient();
        /**
         * Configure the ApiClient with the Santricity proxy URL & set proper
credentials
         */
        apiClient.setBasePath("http://localhost:8080/devmgr/v2");
        apiClient.setUsername("rw");
        apiClient.setPassword("rw");

        /**
         * Add a storage system to the proxy
         */
        String sysId = "config1";
        StorageSystemsApi ssApi = new StorageSystemsApi(apiClient);
        StorageSystemCreateRequest ssCreatReq = new StorageSystemCreateRequest();
        List<String> ctrlAddrs = new ArrayList<String>();
        ctrlAddrs.add("<Ctrl A IP Address>");
        ctrlAddrs.add("<Ctrl B IP Address>");
        ssCreatReq.setControllerAddresses(ctrlAddrs);
        ssCreatReq.setId(sysId);

        try {
            ssApi.newStorageSystem(ssCreatReq);
            sleep(1000);                // one second
            /**
             * Get List of failure events
             */
            DiagnosticsApi diag = new DiagnosticsApi(apiClient);
            sleep(1000);                // one second
            List<FailureData> failureList = diag.getFailures(sysId, false);
            for (FailureData failData : failureList) {
                System.out.println("Failure name: " +
failData.getFailureType().name());
            }
        } catch (ApiException apiExp) {
            System.out.println("An exception occurred. Please check error: \n " +
apiExp.getMessage());
        } catch (InterruptedException ex) {
            currentThread().interrupt();
        }
    }
}
```

Configuring Java to trust REST API Self Signed Certificates

If you need to use the default self-signed certificate, perform the following steps to permit the Java application to trust the certificate for the REST API.

1. From a browser, navigate to the API documentation using HTTPS (e.g., https://hostname.foo.com:8443/devmgr/docs)
2. Export the certificate to a file.

   NOTE: Export to the default DER encoding is acceptable.
3. Use the keytool command to import the certificate to the cacerts file for your jvm.

   Example

```
cd C:\Program Files\Java\jdk1.8.0_65 jre\bin\keytool.exe -import -trustcacerts -
keystore jre\lib\security\cacerts -alias rest-host-cert -file rest-host.cer
```

   NOTE: You must enter the password for the keystore. By default, the password is `changeit`.

Using HTTPS with the Java SDK

The Java SDK is compatible with SSL. The SDK utilizes the default Java keystore for trusted certificates. As a result, you can install the appropriate CA certificate in the default Java keystore. The default Java CA certificate store is located at `$JAVA_HOME/jre/lib/security/cacerts`.

If needed, SSL host verification can be disabled with the ApiClient object's `disableSSLVerification()` method.

Enabling SSL Connections

To use an SSL connection with the Java SDK, simply use the appropriate HTTPS URL for the target REST API server. Ensure to specify the HTTPS protocol and appropriate port for the encrypted connection.

```
ApiClient apiClient = new ApiClient();
apiClient.setBasePath("https://127.0.0.1:8443/devmgr/v2")
```

Certificate Management

The trusted root certificates for the Java SDK are managed using the Java keystore integrated with the Java JVM. This same keystore is used to run the application utilizing the SDK. If you need to import a certificate to utilize SSL with HTTPs connection to the REST server, you must import the certificate to the Java keystore.

Importing a certificate

If any of the following conditions apply, import a certificate to the Java keystore:

- The REST API service (proxy or embedded) is using a self-signed certificate. This is the default state “out-of-box”.
  - In this case, the self-signed certificate is imported.
- The REST API service (proxy or embedded) is using a signed certificate by an untrusted Certificate Authority (CA).

- This might be the case if you signed the certificate with your own CA or an enterprise CA was used to sign the certificate.
- In this case, the root CA certificate is imported.

Example of importing a certificate using the Java keytool

```
cd C:\Program Files\Java\jdk1.8.0_65
jre\bin\keytool.exe -import -trustcacerts -keystore
jre\lib\security\cacerts -alias rest-host-cert -file rest-host.cer
```

Disabling SSL Host Verification

NOTE: Disabling SSL host verification can be a security risk and is not recommended. However, this information is being provided for certain use cases where the security implications are understood and acceptable.

Disabling SSL host verification can be used as a workaround to importing an untrusted certificate from the REST API server to the client system. This permits the use of the HTTPS URL and encryption without requiring the full level of trust.

Example of disabling SSL Host Verification

```
apiClient apiClient = new ApiClient();
apiClient.setBasePath("https://127.0.0.1:8443/devmgr/v2");
apiClient.disableSSLVerification()
```

Reference Exception

- Connections to a proxy with the default self-signed certificate with correct hostname but no certificate installed

```
API Exception for self-signed cert without cert installed
```

- Connections to proxy with self-signed certificate with mis-matched hostname

```
API Exception for self-signed cert when name doesn't match
```

Sample Scripts

REST API

Sample scripts that perform the following tasks are available pre-bundled with the Web Proxy application:

- View SSL configuration, get a Certificate Signing Request (CSR), and import a new certificate.
- View, configure and test syslog receivers.
- View historical data and rolled up statistics for I/O latency, IOPS, and throughput.
- Create volume sets based on application types (application tagging).
- Collect and view support bundle artifacts specific to the E2800.

Client Libraries

Java SDK

SupportBundle.java

The purpose of this sample program is to demonstrate how to retrieve a support bundle using the REST API.

Usage

To use the sample script, you must edit the file to setup the appropriate URL and credentials for your environment. The settings to update are dependent on whether you are connecting through the Web Services Proxy or directly to the REST API on an embedded controller.

To configure for use with the Web Services Proxy

- You must have the storage array added to the Web Services Proxy and know the system-id of the storage array.
- Configure the ``proxyUrl`` variable with the URL of the proxy. Include the port number and the ``/devmgr/v2`` path at the end of the URL.
- The default username and password should work. However, if you have changed the username and password update the program to use with your credentials.

To configure for use with the embedded REST API (E2800 controllers)

- Configure the ``proxyUrl`` variable with the URL of the proxy.
    - The E2800 automatically redirects HTTP requests to the HTTPS port. The program does not work with the redirection. As a result, you must specify the HTTPS protocol and port in the URL.
    - Include the ``/devmgr/v2`` path at the end of the URL.
    - The self-signed certificates use the IP address as the certificate name, so host verification only works when accessing the URL with the IP address.

- The default username (``rw``) works.
- Update the password to match the password set on the array for logging into the Storage System Manager.
- Configure SSL
    - **Option 1 (**Recommended**)**
        - Configure the java JVM to trust the self-signed certificate for the REST API. For detailed information on how to configure the java JVM to trust self-signed certificates, refer to Configuring Java to trust REST API Self Signed Certificates.
    - **Option 2 (**Not Recommended**)**
        - Disable SSL host verification with ``ApiClient.disableSSLVerification()``. For detailed information on how to disable SSL host verification, refer to Disabling SSL Host Verification.

Python Sample Scripts

Sample scripts that perform the following tasks are available pre-bundled with the Python SDK [<REST API PATH>/samples/python]:

- Asynchronous Mirroring
- Configuration
- Consistency groups
- Copies
- Device alerts (email)
- Device alerts
- Auto Support log
- Auto Support configuration
- Events monitoring
- Hardware inventory
- Pools
- Requirements
- Restlibs
- Snapshots
- Symbol
- Volumes

Copyright information

Trademark information

How to send your comments

You can help us to improve the quality of our documentation by sending us your feedback. You can receive automatic notification when production-level (GA/FCS) documentation is initially released or important changes are made to existing production-level documents.

If you have suggestions for improving this document, send us your comments by email to *doccomments@netapp.com*. To help us direct your comments to the correct division, include in the subject line the product name, version, and operating system.

If you want to be notified automatically when production-level documentation is released or important changes are made to existing production-level documents, follow Twitter account @NetAppDoc.

You can also contact us in the following ways:

- NetApp, Inc., 495 East Java Drive, Sunnyvale, CA 94089 U.S.
- Telephone: +1 (408) 822-6000
- Fax: +1 (408) 822-4501
- Support telephone: +1 (888) 463-8277


NetApp