Dell EMC Data Domain REST API Programmer Guide

302-003-980 01

Copyright

Contents

Chapter 1 API overview

Introduction

This Programmer's Guide is intended for use by developers who want to integrate with DataDomain systems by using DataDomain REST APIs. It provides answers to common questions for REST API development, reference to examples, documentation, and supported features and conventions.

REST API Landing Page

You can access a landing page for the Data Domain REST API at:

https://<DDSystem IP>/api

This page contains links to the following information:

- Automatically generated documentation
- A Swagger definition file
- Web services WADL description
- Schema files
- Python reference sample files

In addition, you can find an in-house testing tool (which *is not* a production version) available to customers so that they can become familiar with the Data Domain REST API.

Data Domain RESTful Web Services

Data Domain RESTful Web Services are a set of APIs that allow you to manage and configure Data Domain Systems. The APIs allow you to integrate Data Domain systems with other partner and non-partner customers by providing a predefined set of operations executable through HTTP client/server communication over HTTPS. RESTful requests are stateless, so the server does not maintain a session for the client. Instead, REST relies on a self-contained request protocol that provides all of the information that the server needs to perform requested actions. The Data Domain REST server supports up to 200 clients.

You can invoke RESTful Web Services by using any HTTP client, such as a web browser or predefined command line utility like cURL. You can also choose to write a customized programmatic solution.

The client receives a success or a failure response from the server for every operation that it invokes. If the operation is successful in addition to the requested information, the response typically provides auxiliary information in the form of related URIs. These links help the client application navigate to other parts of available web services.

You can generate DD REST client code from DD REST swagger file (rest_api.json) using the Swagger utilities from the following link:

http://swagger.io/

Chapter 2 Data Domain REST API components and methods

Content Types

DataDomain web services support JSON and XML data formats (that is, application/json and application/xml). The client application can choose the data formats for the request and response.

The client applications should use the HTTP header `Content-Type` to specify the request format, and the HTTP header `Accept` to specify the expected response format. The server uses the following rules to determine the format of requests and responses:

- If `Content-Type` is set, the server expects the input data to be in the specified format. If the `Accept` header is specified, the server response to be in the specified data type format.
- If the `Content-Type` header is specified but the `Accept` is not, the format of response data is the same as the request data.
- If neither the `Content-Type` nor the `Accept` headers are specified, the default format for both request and response types are assumed to be application/json.

Resource URLs

Data Domain web services are represented in the form of URIs. The specified resource URIs allow requests to fetch or alter the state of a Data Domain system. Defined resources are aligned with Data Domain system object definitions and models. CIFS, NFS Exports, MTrees, Data Domain Boost Storage Units, Users, Tenants, and Tenant Units are examples of the resources that you can use to configure and manage Data Domain systems through the Data Domain REST API. The REST API also supports operations on system-wide objects such as SNMP, NTP, and file systems.

The Data Domain REST API server allows communication over HTTP protocol port 3009, and URIs are defined in the following format:

https://<DD System IP>:3009/rest/<Version Info:major.minor>/<RESOURCE>

The following URI naming conventions apply:

- Use the hyphen character to separate words in URI such as `dd-systems` or `tenant-units`.
- Use the underscore character to separate words in the query parts such as `pool_name`.

The REST service version information updates when a change in a related Data Domain resource causes a change in the request/response structures or the query parameters:

- A major version is increased when there is a change in the request input, a change in the response output, or when a query keyword is removed.
- A minor version is increased when a new query keyword is added for a URI.

Different versions of Data Domain systems support different sets of resources. For a list of supported resources and methods on a Data Domain system, refer to documentations provided at: https://<DD System IP>/api/doc/

Template, query parameters and object ID's

Resource URI's can contain template or query parameters.

A template parameter indicates the resource on which the operation is to be performed. Template parameters should be replaced by object ID's, which are unique identifiers that are associated with DD objects. Object ID's are URL-encoded and are returned as part of a RESTful response. API users must use object IDs in GET details, and in PUT and DELETE URIs to refer to a specific object.

For example, the SYSTEM-ID template parameter in the following request should be replaced by the URL-encoded ID of the Data Domain system for which we want to fetch a list of users."

```
GET https://<DD SYSTEM IP>:3009/rest/v1.0)/dd-systems/{SYSTEM-
ID}/users
```

On a Data Domain Operating system, a SYSTEM-ID of 0 can be used to indicate the local Data Domain system's ID in place of the actual URL-encoded SYSTEM-ID.

For example, the following request gets the list of users on the local Data Domain system:

```
GET https://<DD System IP>:3009/rest/v1.0/dd-systems/0/users.
```

The ID in the `GET https://<DD SYSTEM IP>:3009/rest/v1.0/dd-systems/{SYSTEM-ID}/users/{`***`ID`***`}` request is also a template parameter and represents the URL-encoded unique ID of the user for whom to fetch detailed information

The following list provides examples of URL-encoded ID's:

- Data Domain system ID, URL encoded system UUID:
  `d863debd00f9406a%3A967f077761e85d3d`
- MTree ID, URL encoded MTree name:
  `%2Fdata%2Fcol1%2Fnas-archive`
- Export ID, URL encoded export path:
  `%2Fdata%2Fcol1%2Fengineering%2Fsantaclara`
- CIFS share, URL-encoded share name:
  `nas-archive`

Query parameters provide additional criteria to which the response data needs to conform. Query parameters are part of the URL query string and must contain URL-encoded values.

For example, the following query fetches a list of users on a specific Data Domain System starting at page one, and assuming each page has five users:

```
GET https://<DD SYSTEM IP>:3009/rest/v1.0/dd-systems/{SYSTEM-
ID}/users?page=1&size=5
```

If a request is sent with an incomplete URI, the REST API framework returns a list of links that may be related so the client can correct itself. For example, if client sends a `GET /rest/v1.0/dd-systems/<system-id>/protocols` request, the REST API framework returns the following response:

```
{
    "details": "success",
    "code": 0,
    "link": [
        {
            "rel": "parent",
            "href": "/rest/v1.0/dd-systems/57a3b36ca0b49814%3Afec6675e7c1864bc"
        },
        {
            "rel": "ddboost",
            "href": "/rest/v1.0/dd-
systems/57a3b36ca0b49814%3Afec6675e7c1864bc/protocols/ddboost"
        },
        {
            "rel": "vdisk",
            "href": "/rest/v1.0/dd-
systems/57a3b36ca0b49814%3Afec6675e7c1864bc/protocols/vdisk"
        },
        {
            "rel": "cifs",
            "href": "/rest/v1.0/dd-
systems/57a3b36ca0b49814%3Afec6675e7c1864bc/protocols/cifs"
        },
        {
            "rel": "nfs",
            "href": "/rest/v1.0/dd-
systems/57a3b36ca0b49814%3Afec6675e7c1864bc/protocols/nfs"
        }
    ]
}
```

Supported schema

Each resource URI refers to schema elements that represent request or response data that is communicated between the client and server. Schema documentation for each resource is available at:
https://<DD System IP>/api/doc/

Resource schema documentation includes the following:

- Model data and associated examples for the request and response objects in XML or JSON format
- The response data and status codes expected based on successful and failed operations
- Request or response headers
- Supported template and query parameters

Supported Products

Both Data Domain Management Center (MC) and Data Domain Operating System (OS) products support REST requests. In addition, the Data Domain MC REST APIs support request-forwarding to managed Data Domain systems. To perform operations on a managed Data Domain system from Data Domain MC, you must specify the Data Domain system URL-encoded `SYSTEM-ID` as part of the request URL.

Chapter 3 DD REST API operations

REST Operations

GET

GET operations retrieve information about a resource without altering the state of the Data Domain system. Any data passed to a GET request is part of the URL and is passed as a query parameter. GET operations read data in one of the following manners:

Example

- Retrieve a list of objects associated with the resource. The operation usually allows paging the data and filtering or sorting based on particular object attributes. The returned response includes links to navigate to the first, last, next, and previous pages. Object data includes links to related resources for each object, if any are available.

  The following example retrieves CIFS shares available on a Data Domain system, starting with the `path /data/col1/share`. Data returned as part of the response is sorted in ascending fashion and paged (5 items per page):

  ```
  GET /rest/v1.0/dd-
  systems/57a3b36ca0b49814%3Afec6675e7c1864bc/protocols/
  cifs/shares?
  page=0&size=5&sort=name&filter=path%3D%2Fdata%2Fcol1%2Fshare
  ```

- Retrieve details (and any related links) associated with a specific object based on its URI-encoded object ID.

  The following example uses a GET operation to obtain detailed information about CIFS share ***share1***:

  ```
  GET /rest/v1.0/dd-
  systems/57a3b36ca0b49814%3Afec6675e7c1864bc/protocols/cifs/
  shares/share1
  ```

POST

POST operations enable you to create objects for Data Domain resources. For example, you can use POST to create a CIFS share that is configured according to embedded data within the request body. The response includes a link to the details associated with the newly-created object.

Example

The following shows a POST operation for creating a CIFS share named ***share1***:

```
POST /rest/v1.0/dd-
systems/57a3b36ca0b49814%3Afec6675e7c1864bc/protocols/cifs/shares
JSON Data sent as part of the request:
    "share_create":{
        "name":"share1",
        "path":"/data/col1/share1",
    }
```

PUT

PUT operations enable you to modify object data for a Data Domain resource. For example, you can use PUT to alter CIFS share attributes such as clients, users, and groups. In PUT operations, you identify target resources according to their URL-encoded object IDs.

Example
The following example shows a PUT operation for modifying ***share1*** information (a CIFS share):

```
PUT /rest/v1.0/dd-
systems/57a3b36ca0b49814%3Afec6675e7c1864bc/protocols/cifs/shares/s
hare1
```

In the example, JSON Data sent as part of the request modifies share's state, adds a client, and sets the `max_connections` attribute for the share:

```
"share_modify":{
    "disable":false,
    "max_connections":10,
    "clients":[
        {
            "name":"10.25.182.11",
            "delete":false
        }
    ]
}
```

DELETE

Use the DELETE operation to delete one or all of the objects for a specified resource.

Example
The following shows a DELETE request for a CIFS share:

```
DELETE /rest/v1.0/dd-
systems/57a3b36ca0b49814%3Afec6675e7c1864bc/protocols/cifs/shares/share1
```

Paging and Sorting

You can specify the page size (size) and the page number (page) as query parameters within GET operation REST API request.

Paging: Specify page size (`size`) and page number (`page`) as query parameters within GET operations. The default page size is 20, the maximum page size is 200, and page numbering starts at `0`.

Sorting: Specify the sorting (`sort`) criteria as query parameters within GET operations. For example, you can specify descending order with the hyphen (-) character, as in the following example (where `name` is an object attribute): `-name`

The sorting feature also supports multiple keys as shown in the following:

```
sort=<key1>,-<key2>,<key3>
```

Filtering

You can use filters (`filter`) with specific object attributes. Filters support regular expressions and multiple criteria.

For example: `filter=key1=val1`, `key2=val2`, and `key3=val3`.

String filters

Some examples of filters targeted at object `name` values include:

- `name = abc`: Find objects whose names contain the string `abc`.
- `name = ^abc$` Find objects whose exact name is `abc`.

Number range filters

The REST API supports the following number range formats:

Regular expression	REST number range filter syntax
a <= x<= b	(a, b)
x >= a	(a,)
x <= b	(, b)
x == a	(a, a)

Example: Request query string:

```
page=0&size=5&sort=name&filter=name=/data/col1/mt* and tenant_unit=tu*
```

Example: Response:

The paging request response includes the following information:

- `current_page`: Current page number.
- `page_entries`: Number of entries in the page.
- `total_entries`: Total number of entries.
- `page_size`: Size of each page.
- `page_links`: Structure containing links to first, previous, next, and last pages.

```
"paging_info": {
        "current_page": 0,
        "page_entries": 5,
        "total_entries": 7,
        "page_size": 5,
        "page_links": [
            {
                "rel": "first",
                "href": "/rest/v1.0/dd-
systems/57a3b36ca0b49814%3Afec6675e7c1864bc/mtrees?page=0&size=5"
            },
            {
                "rel": "last",
                "href": "/rest/v1.0/dd-
systems/57a3b36ca0b49814%3Afec6675e7c1864bc/mtrees?page=1&size=5"
            },
            {
                "rel": "next",
                "href": "/rest/v1.0/dd-
systems/57a3b36ca0b49814%3Afec6675e7c1864bc/mtrees?page=1&size=5"
            }
        ]
    }
```

HTTP Response Status Codes

REST API services can return the following REST response status codes:

Successful Service 2xx

Code	Description
200 (OK)	Used for any successful service that is not mentioned in 201
201 (Created)	Successfully created a resource such as Mtree, NFS Export, or CIFS Share.

Redirection 3xx

Code	Description
301 (Moved permanently)	The client URI is removed (obsoleted). The response body contains a new URI for the client to use if it is applicable

Client Error 4xx

Code	Description
400 (Bad request)	Used for any client error that is not specified in the previous 4-series errors.
401 (Unauthorized)	The client is not authenticated or authorized to perform the request.
403 (Forbidden)	Authentication was successful, but client does not have permission to perform the specified operation. For example the, client does not have privileges to run this operation on the specified Data Domain system.
404 (Not found)	The URI is invalid or the object was not found
405 (Method not allowed):	Returned if the specified method is not supported for the URI. For example, /rest/v1.0/auth does not support the GET method.
411 (Length required)	A request body is expected, but the body is empty.
413 (Request entity too large)	Request size is greater than 64k (65536 bytes), which exceeds the buffer length for socket packets.
414 (URI too long)	URI is longer than 2000 bytes. The 2000 byte maximum URI can work for most client and server combinations
415 (Unsupported media type)	Media type is not application/json or application/xml.

Server Error 5xx

Code	Description
500 (Internal error)	Used for server errors not mentioned in 501.
501 (Not implemented)	The specified service is not implemented.

Chapter 4 Authentication and Authorization

This chapter presents the following topics:

Authentication

In order for a client to use Data Domain Web Services, the client must first authenticate the user through the RESTful server. The RESTful server also performs RBAC (role base access control) to authorize the user based on their role on the system. Client authentication can be performed in two ways:

- Username and password authentication:
  The Client must send an authentication request to the Data Domain System and provide required username and password information. Valid users are then authenticated and authorized by the server and an authentication token is provided to the client. Since REST server does not store the client context between requests, each request needs to contain the authentication token. The subsequent requests specify the `X-DD-AUTH-TOKEN` in the HTTP header. The default session timeout is 30 minutes.
- Certificate based authentication:
  This authentication and authorization process is the Data Domain in-house client certification verification process through Data Domain web (REST) services. This process is executed on top of the SSL layer. Therefore, the normal SSL handshake and verification through HTTPS between client and server must complete before the authentication and authorization process begins. The client certificate mechanism leverages a strong public key infrastructure and uses asymmetric cryptography. As a result, the mechanism provides a very secure way to access Data Domain systems. The client certificate must contain the user information (user-name), and this user must be a valid local, Network Information Service (NIS), or Active Directory (AD) user on the server side (that is, on the Data Domain Management Center or standard Data Domain system). The following sections provide procedures for preparing client and server for Certificate Based Authentication.

Prepare client for certificate-based authentication

You can prepare the client or certificate-based authentication (CA)in the following way:

1. Obtain the CA certificate and the client certificate
   The client must obtain the CA certificate and the client certificate with a valid local or name service (NIS or AD) username on the Data Domain system.
2. Import the CA certificate on the Data Domain system. For example:

```
# ssh sysadmin@<DD hostname/IP> adminaccess \
  certificate import ca application \
  login-auth < cacert.pem
```

3. Copy the Data Domain system CA certificate to the client system.
   If needed, you can copy the Data Domain system CA certificate and save it on the local system so the system can verify the server communication. You can use Openssl to get the public CA of Data Domain system. For example:

```
# openssl s_client -showcerts -connect <DD hostname/IP>:3009 \
  < /dev/null
```

The following sample output shows a case in which the last server certificate is the CA one. You can copy the CA certificate from `-----BEGIN CERTIFICATE-----` to `-----END CERTIFICATE----` to use as needed.

Note: On a LINUX or UNIX system for system wide use, OpenSSL keeps the trusted CA certificates in /etc/ssl/certs.

```
$ openssl s_client -showcerts -connect 10.25.183.157:3009 < /dev/null
CONNECTED(00000003)
depth=1 /C=US/ST=CA/L=Santa Clara/O=Valued Datadomain Customer/OU=Root
CA/CN=ddve-25183157.brs.lab.emc.com
verify error:num=19:self signed certificate in certificate chain
verify return:0
---
Certificate chain
  0 s:/C=US/ST=CA/OU=Host Certificate/O=Valued DataDomain customer/CN=ddve-
25183157.brs.lab.emc.com
     i:/C=US/ST=CA/L=Santa Clara/O=Valued Datadomain Customer/OU=Root CA/CN=ddve-
25183157.brs.lab.emc.com
-----BEGIN CERTIFICATE-----
MIICgjCCAesCAQIwDQYJKoZIhvcNAQEFBQAwgY8xCzAJBgNVBAYTAlVTMQswCQYD
VQQIDAJDQTEUMBIGA1UEBwwLU2FudGEgQ2xhcmExIzAhBgNVBAoMGlZhbHVlZCBE
YXRhZG9tYWluIEN1c3RvbWVyMRAwDgYDVQQLDAdSb290IENBMSYwJAYDVQQDDB1k
ZHZlLTI1MTgzMTU3LmJycy5sYWIuZW1jLmNvbTAeFw0xNDEyMTcxNzAwMDhaFw00
NTEyMTAwMTAwMDhaMIGCMQswCQYDVQQGEwJVUzELMAkGA1UECAwCQ0ExGTAXBgNV
BAsMEEhvc3QgQ2VydGlmaWNhdGUxIzAhBgNVBAoMGlZhbHVlZCBEYXRhRG9tYWlu
IGN1c3RvbWVyMSYwJAYDVQQDDB1kZHZlLTI1MTgzMTU3LmJycy5sYWIuZW1jLmNv
bTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEApjSl3E0R2gGy0pxsvdHkQLEE
fKuup2bE3HT8t2QokJYAadw/jb4BaHZ3sKkR+nr5ZKj4QfHea15U6+uDaGF6iT/1
5g6pPj+Jh4Cnyn9iIjFJNPvQJXsk5A//JLh5Fr6lIQVizqJ9vOr88QvJsDPqVeJ4
ndE92XdxXOMDwMFPvlECAwEAATANBgkqhkiG9w0BAQUFAAOBgQAOY2DRbw9xBD/M
irnP4L8+T1dtk9i6ZMPP9rehlqzxo0TRhy/XfaX/wlhPNnymwAGT0m9U1Oib6AVk
/RuZM0WNnGzarRlk0KYStD84MOfGsPRbPNkPfk/tBNxxkCnMjdawdv9xyt6Flqin
/N0FUcNkL1qm5Y8JwSajx6AilXGt4w==
-----END CERTIFICATE-----
  1 s:/C=US/ST=CA/L=Santa Clara/O=Valued Datadomain Customer/OU=Root CA/CN=ddve-
25183157.brs.lab.emc.com
     i:/C=US/ST=CA/L=Santa Clara/O=Valued Datadomain Customer/OU=Root CA/CN=ddve-
25183157.brs.lab.emc.com
-----BEGIN CERTIFICATE-----
MIIClDCCAf2gAwIBAgIBADANBgkqhkiG9w0BAQUFADCBjzELMAkGA1UEBhMCVVMx
CzAJBgNVBAgMAkNBMRQwEgYDVQQHDAtTYW50YSBDbGFyYTEjMCEGA1UECgwaVmFs
dWVkIERhdGFkb21haW4gQ3VzdG9tZXIxEDAOBgNVBAsMB1Jvb3QgQ0ExJjAkBgNV
BAMMHWRkdmUtMjUxODMxNTcuYnJzLmxhYi5lbWMuY29tMB4XDTE0MTIxODAxMDAw
OFoXDTQ1MTIxMDAxMDAwOFowgY8xCzAJBgNVBAYTAlVTMQswCQYDVQQIDAJDQTEU
MBIGA1UEBwwLU2FudGEgQ2xhcmExIzAhBgNVBAoMGlZhbHVlZCBEYXRhZG9tYWlu
IEN1c3RvbWVyMRAwDgYDVQQLDAdSb290IENBMSYwJAYDVQQDDB1kZHZlLTI1MTgz
```

```
MTU3LmJycy5sYWIuZW1jLmNvbTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA
wKoti2sp12Yj3H3PfUui3Mxv3H64M8N4B8RPAgg3aeAK/EcdnK1SnqtIfoyAytc0
MG6uqVKY0A/GATp4TaxFATRnrlfdbbOk41eHvgkyaSO1IYMU0APpdGzQX+4sSbtH
aEJhseEEDa3F35qACk3TSzLYTJAbn+6D1RhgP6R5jn8CAwEAATANBgkqhkiG9w0B
AQUFAAOBgQCihdO8qfAThy1/0R3btmNYrNkhHobUA41bNr+Ca7aGVgNomO+sugcs
t8B5fN+UKFFTEjz4e4vOkA7RbIVG2+6MxJjmZEaWjgvBkpMWrikqxwAnJZPqX/WX
etec6lhTRjgoCURwK+OMyJypRrx7uOlxOiSgSrjjDLgVv2ZJ1+lO0Q==
-----END CERTIFICATE-----
---
Server certificate
subject=/C=US/ST=CA/OU=Host Certificate/O=Valued DataDomain
customer/CN=ddve-25183157.brs.lab.emc.com
issuer=/C=US/ST=CA/L=Santa Clara/O=Valued Datadomain Customer/OU=Root
CA/CN=ddve-25183157.brs.lab.emc.com
```

4. Import the client certificate to the browser.

   To use the client certificate from a browser, you must import the client certificate to the browser so the browser can display the certificate before sending the request to the DD system. The client must provide proof of possession to the browser by providing both private and public keys. The browser validates the client using private and public keys, but only sends the client public key to the DD system.

Importing the CA certificate to Chromium on Ubuntu 12.04

To import a client certificate on a Chromium browser (Ubuntu 12.04):

1. Open a Chromium browser instance.
2. Click **Edit**->**Preference**s
3. Navigate to **HTTPS/SSL** (**Manage Certificates**...)
4. Select the **Your Certificates** tab and click **Import**.
5. Go to your client certificate directory and choose the p12 certificate file. When prompted, specify a password to decrypt the PKCS12 file.

Importing a client certificate to Safari and Chromium on a MacBook Pro Computer

1. In the EMC_CA/certs directory, double-click the p12 certificate and follow the instructions for importing the client certificate on the MacBook.
2. Open Keychain Access and mark the client certificate as Trust
   a. Open Keychain Access
   b. Select **System** in the top left panel and select the category **My Certificates** in bottom left panel.
   c. Right click the certificate
   d. Select **Get Info**.
   e. Click **Trust**.
   f. From the **When using this certificate** area, select **Always Trust**.
3. Enable applications to access this certificate without requiring password:
   a. From /Applications/Utilities, open **Keychain Access**.
   b. From the **Keychains** pane, select the **System keychain**.
   c. Select **System** in the top left panel and select the category **My Certificates** in bottom left panel.
   d. Click the arrow next to the imported certificate.
   e. Double-click the private key.
   f. From the **Access Control** tab, select **Allow all applications to access this item**.
   g. Click **Save Changes** and authenticate as a local administrator when prompted.

Prepare the server for certificate-based authentication

To prepare the server for certificate-base authentication on a Data Domain system:

1. Verify that the client certificate installed properly on Data Domain system. For example

   ```
   adminaccess certificate show imported-ca application login-auth
   ```

2. Make sure the username is valid on Data Domain system.

   If the user is an NIS user, complete the followings steps on the target Data Domain system:

   a. Identify group information for NIS user by running the following Unix/Linux shell command to query name services for the user group information:

   ```
   #id anjala
   uid=200(anjala) gid=200(anjala) groups=200(anjala)
   ```

   b. Provide RBAC role for the user NIS group.

   ```
   # authentication nis groups add anjala role admin
   ```

   c. If there is no default value available from DHCP, ensure that the NIS server and domain information is set with the following CLI commands:

   ```
   authentication nis server add <server-list>
   authentication nis domain set <domain>
   ```

   d. Enable NIS in the following way:

   ```
   # authentication nis enable
   # authentication nis show
             NIS Summary:
             Domain: datadomain.com
             Servers: 10.25.209.6,10.25.232.17
             Admin Groups: anjala
             User Groups:
             Backup Operator Groups:
             Enabled: Yes
             Status: Good
   ```

   e. If the user is a local user, check the user's status in the following way:

   ```
   # user show list
   ```

   f. If the user doesn't exist, add the user with the desired role with the following command:

   ```
   # user add <username> role [admin|user…]
   ```

Client-side testing

Test with cURL

To test with cURL, at the command prompt, run the following command:

```
# curl --tlsv1 --cacert <path-to-dd-ca-cert>/<dd-cacert> \
 -v "https://<DD hostname>:3009/rest/v1.0/system" \
 --request GET --header "Content-Type: application/json" \
 --cert <path-to-client-cert>/<client-public-cert> \
 --key <path-to-client-cert>/<client-private-key>
```

If you don't want to specify the Data Domain system public CA certificate, use the `-k` option:

```
# curl --tlsv1 -k \
 -v "https://<DD hostname>:3009/rest/v1.0/system" \
 --request GET --header "Content-Type: application/json" \
 --cert <path-to-client-cert>/<client-public-cert>
```

Note:

- The Data Domain hostname is the same as common name in the host certificate of Data Domain system because (for security purposes) the peer name must match the subject name in the DD host certificate.
- The Data Domain hostname must be resolvable.

For example:

```
curl --tlsv1 --cacert /etc/ssl/certs/ddr_cacert.pem -v
"https://ddve-25183157.brs.lab.emc.com:3009/rest/v1.0/system" --request
GET --header "Content-Type: application/json" --cert
./EMC_CA/certs/anjala_client.cer --key ./EMC_CA/certs/anjala_client.key
* About to connect() to ddve-25183157.brs.lab.emc.com port 3009 (#0)
* Trying 10.25.183.157... connected
* successfully set certificate verify locations:
* CAfile: ./trustedCA/cacert.pem
 CApath: /etc/ssl/certs
* SSLv3, TLS handshake, Client hello (1):
* SSLv3, TLS handshake, Server hello (2):
* SSLv3, TLS handshake, CERT (11):
* SSLv3, TLS handshake, Request CERT (13):
* SSLv3, TLS handshake, Server finished (14):
* SSLv3, TLS handshake, CERT (11):
* SSLv3, TLS handshake, Client key exchange (16):
* SSLv3, TLS handshake, CERT verify (15):
* SSLv3, TLS change cipher, Client hello (1):
* SSLv3, TLS handshake, Finished (20):
* SSLv3, TLS change cipher, Client hello (1):
* SSLv3, TLS handshake, Finished (20):
* SSL connection using AES256-SHA
* Server certificate:
```

```
* subject: C=US; ST=CA; OU=Host Certificate; O=Valued DataDomain
customer; CN=ddve-25183157.brs.lab.emc.com
* start date: 2014-12-17 17:00:08 GMT
* expire date: 2045-12-10 01:00:08 GMT
* common name: ddve-25183157.brs.lab.emc.com (matched)
* issuer: C=US; ST=CA; L=Santa Clara; O=Valued Datadomain Customer;
OU=Root CA; CN=ddve-25183157.brs.lab.emc.com
* SSL certificate verify ok.
> GET /rest/v1.0/system HTTP/1.1
> User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0
OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23 librtmp/2.3
> Host: ddve-25183157.brs.lab.emc.com:3009
> Accept: */*
> Content-Type: application/json
>
< HTTP/1.1 200 OK
< Content-Type: application/json
< Content-Length: 401
< X-DD-AUTH-TOKEN: 3638545ebfb8770f1fe975e2a915accd2
< X-DD-UUID: 8bc604f1233c1073:d175e3d721fe50c3
< Cache-Control: no-cache
< Server: Data Domain OS 0.6000.0.0-521337
<
* Connection #0 to host ddve-25183157.brs.lab.emc.com left intact
* Closing connection #0
* SSLv3, TLS alert, Client hello (1):
{"name": "ddve-25183157.brs.lab.emc.com", "version": "Data Domain OS
0.6000.0.0-521337", "serialno": "AUDVBDSS1DAVVU", "uptime": "14:36:59 up
8 days, 22:40", "uptime_secs": 772849, "uuid":
"8bc604f1233c1073:d175e3d721fe50c3", "link": [{"rel": "self", "href":
"/rest/v1.0/system"}, {"rel": "related", "href":
"/rest/v1.0/ddsystems/8bc604f1233c1073%3Ad175e3d721fe50c3/stats/compression/measure
ment
s"}]}
```

Testing with browsers

To test in a browser environment:

1. Enter the following URL in the browser's navigation bar:

   https://<DD System IP>:3009/rest/v1.0/system

   The browser displays the client certificate.

2. Select your client certificate.

3. In the browser, validate the CA that issued the client certificate (to avoid future unknown CA certificate security warnings).

4. Send request to the Data Domain System.

   The Data Domain system information is displayed in the browser. For example:

```
<system_info xmlns="rest.datadomain.com">
 <name>ddve-25183157.brs.lab.emc.com</name>
 <version>Data Domain OS 0.6000.0.0-521337</version>
 <serialno>AUDVBDSS1DAVVU</serialno>
 <uptime>11:39:32 up 19:43</uptime>
 <uptime_secs>71003</uptime_secs>
 <uuid>8bc604f1233c1073:d175e3d721fe50c3</uuid>
 <link rel="self" href="/rest/v1.0/system"/>
 <link rel="related"
href="/rest/v1.0/ddsystems/8bc604f1233c1073%3Ad175e3d721fe50c3/stats/compre
ssion/measurements"/>
</system_info>.
```

Chapter 5 Workflow and Logs

Limits/operational requirements, sample workflow

A limit of 64 K is imposed on REST API requests. The framework prevents requests over this limit from going through by returning a response that indicates that the request is too long. The PCR POST request is a common case where the request can be too long. To avoid these limits, limit the size of POST requests to 64 K and add additional paths through the subsequent PCR PUT requests.

Example REST Request Workflow

- REST clients must first submit a `POST /rest/v1.0/auth` request, specifying the username and password information in the request `auth_info` structure. The returned response includes the `service_status`, the authentication token (`X-DD-AUTH-TOKEN`) embedded in the HTTP response header, and related links for the client to navigate.
- The client must set the session UUID as HTTP header X-DD-AUTH-TOKEN for subsequent requests. If the current session UUID expires, the client must resubmit the authentication request to obtain a new session UUID.
- When the session is over, the client must submit a DELETE authentication request (`DELETE /rest/v1.0/auth`) to log the user out, delete the session, and delete the authentication token. As with any request, the DELETE authentication request must provide a valid session UUID in the `HTTP X-DD-AUTH-TOKEN` header.

Sample curl requests

Sending an authentication request:

```
curl -X 'POST' --header 'Content-Type: application/json' --header
'Accept: application/json' -d '{ "auth_info":{
"username":"sysadmin", "password":"abc123" } }'
'https://10.25.182.115:3009/rest/v1.0/auth'
```

Sent a request to obtain a list of MTrees on the system:

```
curl -X 'GET' --header 'Content-Type: application/json' --header
'Accept: application/json' --header 'X-DD-AUTH-
TOKEN:f6178449353c39894387c7f9deff1900c'
'https://10.25.182.115:3009/rest/v1.0/dd-systems/0/mtrees'
```

Send a logout request:

```
curl -X 'DELETE' --header 'Content-Type: application/json' --header
'Accept: application/json' --header 'X-DD-AUTH-
TOKEN:c583120cd31dfc61a91087767990b7471'
'https://10.25.182.115:3009/rest/v1.0/auth'
```

More Python examples

Python examples are available for reference at the following URL:

https://<DD System IP>/api/ page

Logs

- Audit information is logged in the /ddr/var/log/audit.log file. By default, audit information is only logged for non-read only REST API operations like PUT, POST and DELETE.
- The length of time that it takes for REST requests operations to execute is logged in the /ddr/var/log/debug/sm/sms.info file. This log file includes the start and completion time of all executed RESTful web services.
- Information about RESTful clients that access REST services and their corresponding users is logged in the /ddr/var/support/autosupport file.

Chapter 6 Using Sample Test Tools

Getting started with Data Domain REST API Tools

You can use the Data Domain REST API in-house test tool or the Swagger tool to become familiar with the DD REST API. The landing page for REST APIs is located at the following URL and has links to both tools:

https://<DD System IP>/api

Note: The In-house Test tool is non-production version and is only available for becoming familiar with Data Domain REST APIs.

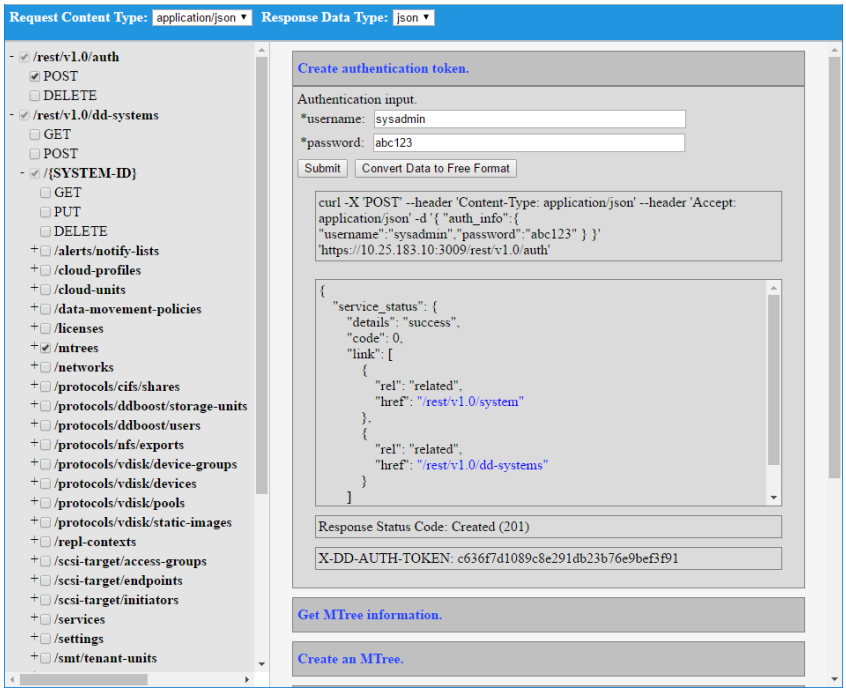
Using the In-house Test Tool

To use the In-house Test tool:

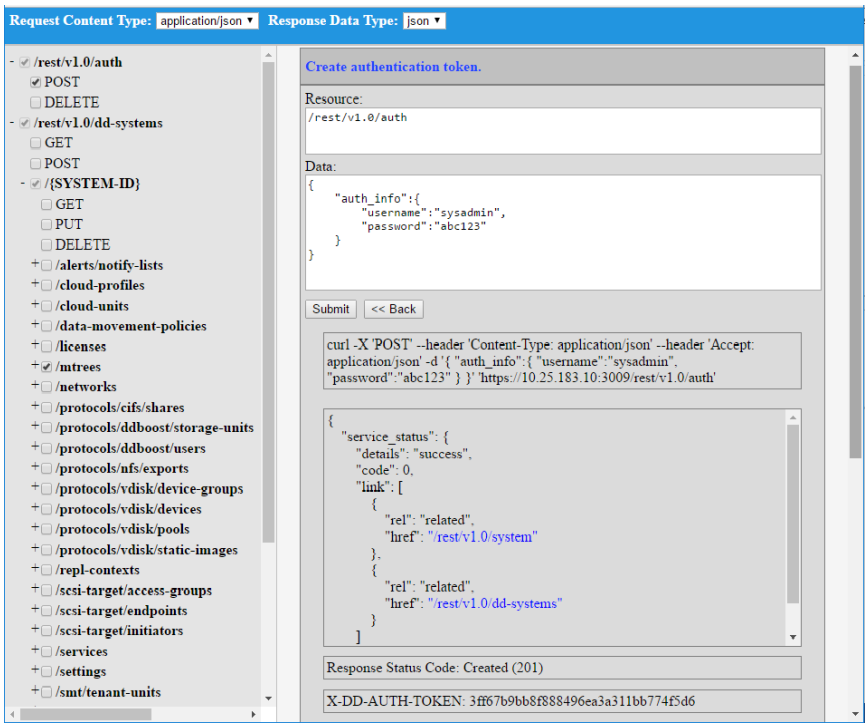
1. Go to https://<DD System IP>:3009 and accept the certificate.
2. Navigate to https://<DD System IP>/api/rest_test.html and accept the certificate.
3. From the top section, choose to communicate Request Content Types and Response Data Types in either JSON or XML format. The left navigation panel displays the list of supported URIs.
4. From the left navigation panel, select a URI for the instance of the Data Domain Management Center or Data Domain Operating System version. The content area displays the relevant URI's fields.
5. Provide the request information and submit the request. The returned response includes the response status code and related link information to navigate.

The In-house Test tool includes a free-format mode that provides the request data in the free JSON or XML format.

Following screen capture shows the In-house Test tool in *strict* format:



The following screen capture shows the In-house Test tool in free-format mode:



Using the Swagger API Documentation Tool

The Swagger tool is available at the following URL:

https://<DD System IP>/api/doc/

The main page lists all supported modules and their related URIs for either the Data Domain Management Center or Data Domain Operating System. Information includes:

- Response and request models .
- Example data in JSON or XML format.
- Response status code on success and failure.
- Request and response headers.
- Supported template and query parameters.

You can experiment the URIs by providing request data and submitting a request. The following screen capture shows an authentication request in the Swagger tool: