Dell PowerStore

REST API Developers Guide

Version 4.1

February 2025
Rev. A09

DELL Technologies

Notes, cautions, and warnings

NOTE: A NOTE indicates important information that helps you make better use of your product.

CAUTION: A CAUTION indicates either potential damage to hardware or loss of data and tells you how to avoid the problem.

WARNING: A WARNING indicates a potential for property damage, personal injury, or death.

Contents

Additional Resources

As part of an improvement effort, revisions of the software and hardware are periodically released. Some functions that are described in this document are not supported by all versions of the software or hardware currently in use. The product release notes provide the most up-to-date information about product features. Contact your service provider if a product does not function properly or does not function as described in this document.

NOTE: PowerStore X model customers: For the latest how-to technical manuals and guides for your model, download the *PowerStore 3.2.x Documentation Set* from the PowerStore Documentation page at dell.com/powerstoredocs.

Where to get help

Support, product, and licensing information can be obtained as follows:

- **Product information**—For product and feature documentation or release notes, go to the PowerStore Documentation page at dell.com/powerstoredocs.
- **Troubleshooting**—For information about products, software updates, licensing, and service go to Dell Support and locate the appropriate product support page.
- **Technical support**—For technical support and service requests, go to Dell Support and locate the **Service Requests** page. To open a service request, you must have a valid support agreement. Contact your Sales Representative for details about obtaining a valid support agreement or to answer any questions about your account.

1

Overview

Topics:

The PowerStore REST API

The PowerStore REST API is a set of resources (objects), operations, and attributes that provide interactive, scripted, and programmatic management control of the PowerStore cluster. Here are some examples of what you can do with the REST API:

- Provision new cluster components, such as appliances and volumes.
- Configure replication destinations and sessions, and rules.
- Fail over and fail back an appliance.
- Create snapshots for backup purposes.
- Gather metrics to use for historical analysis.
- Gather configuration information and logs to use for auditing and trending analysis.

Reference materials are also available on the appliance in several formats:

- Swagger UI—`https://<cluster management ip address>/swaggerui`
- JSON—`https://<cluster management ip address>/api/rest/openapi.json`

The REST API uses a Representational State Transfer (REST) architecture style to expose data. Using a REST API provides the following advantages:

- Presents a single, consistent interface for managing PowerStore clusters.
- Requires no additional tools, other than standard web browsers or command-line HTTP tools, such as wGET and cURL. For complex interactions, clients can use any procedural programming language, such as C++ or Java, or scripting language, such as Perl or Python, to make calls to the REST API.
- Uses well-known HTTP conventions in a standard manner to interact with the appliance.
- Is easy to transport in the network. REST API traffic looks and acts like standard HTTP network traffic, and requires no special ports open in the firewall or special settings in the switches.

The REST API connection is secured with SSL. The same authentication is used for the REST API and GUI.

Resource-oriented architecture and REST

REST is a client/server architectural style that uses the HTTP protocol in a simple, effective, way. REST is based on the following principles:

- Application state and functionality are organized into resources. Resources represent physical things, such as a specific appliance. Resources also represent logical things, such as a specific alert, or collections of entities, such as the volumes or virtual disks in the system.
- Each resource has a unique Universal Resource Identifier (URI), and each resource instance has a unique ID. For example, you can identify the alert collection with this path component: `/api/rest/alert`, and you can identify the alert instance that has an ID of 00c0d863-8a13-4e98-ba06-c4c3f6da615f with this URI: `/api/rest/alert/00c0d863-8a13-4e98-ba06-c4c3f6da615f`.
- Resources share a uniform interface between the client and server through standard HTTP protocol operations. The PowerStore REST API uses the HTTP `GET` operation to retrieve data about a resource collection or resource instance, `PATCH` to modify a resource instance, `POST` to create a resource instance, and `DELETE` to delete a resource instance. The REST API also uses `POST` for a limited set of other operations to implement resource-specific actions. Thus, an application can interact with a resource by knowing the URI pattern, resource identifier, and action required.

- Communication between the client and server occurs through HTTP requests and responses. In the PowerStore REST API, requests and responses represent resource data using JavaScript Object Notation (JSON).
- Each request is stateless, which means that the server does not store application state information. Instead, client requests contain all the necessary information to service the request.
- Resources in a REST API are self-documenting. A response from the server contains information about the requested resource in the form of attribute names and values.

JSON data exchange format

JavaScript Object Notation (JSON) is a text-based, platform-independent data-exchange format that is easy for humans and machines to read and write. It consists of two structures:

- A set of name:value pairs enclosed by curly brackets.
- A list of values enclosed by square brackets. This structure is used when the value in a name:value pair is an array.

The value in a name can be a simple value, such as a string or a number, or it can be either of the structures above (a list of name:value pairs in curly brackets, or a list of values in square brackets).

The following example shows part of a response body for a `GET dns` collection request in JSON format. In this content, the value for the addresses attribute is a list structure:

```
[
  {
    "id": "DNS1",
    "addresses":
    [
      "10.244.53.108",
      "10.244.53.109"
    ]
  }
]
```

For more information about JSON, see json.org.

2

REST request components

Topics:

HTTP request headers

The following table describes the HTTP request headers that the PowerStore REST API uses. The API uses these headers in standard ways.

Table 1. HTTP request headers

HTTP header	Value	Description
`Accept:`	`application/json`	Format of the body content wanted in the response. All requests use `Accept: application/json`, which is the default and only value accepted.
`Content-Type:`	`application/json`	The body content type of the request. This header is optional because it is the default and only supported value.
`Accept-Language:`	`en-US`	The requested locale is mapped to an available language. Only `en-US` is supported.
`DELL-EMC-TOKEN:`	Valid CSRF token value from authenticated GET request	Before issuing any REST call which changes the state of the object (such as `POST`, `PATCH` or `DELETE`) send a `GET` request to receive a CSRF token as response header named `DELL-EMC-TOKEN`. Use the value of a token from the response header that is obtained from the GET call as a Header value in the subsequent calls for this session.
`Range:`	`<first>-<last>`	For `GET` calls only. Part of paging support. This header requests rows first through last (optional) of the result set.
`X-DELL-MFA-TYPE:`	`SECURID`	When multifactor authentication (MFA) is supported, this is the input field for the second factor method that is used to authenticate. For PowerStoreOS 3.5, only `SECURID` is supported.
`X-DELL-MFA-FACTOR-TOKEN:`	`passCodeValue`	When MFA is supported, this is the input field for the second factor token that is used to authenticate. This header is useful for a client that can supply both a password and a second factor with no intermediate steps. `X-DELL-MFA-TYPE` is required when this header is passed.

Related references

URI patterns
Request parameters
REST requests

URI patterns

Basic URI patterns

The following table describes the basic URI patterns that the PowerStore REST API supports:

Table 2. Basic patterns in the REST API

URI pattern	HTTP Operations	Description
Collection type resource URI `/api/rest/<resource_type>`	`GET`	Retrieves a list of instances for the specified resource type.
	`POST`	Creates a new instance of the specified resource type, using data specified in the request body, if allowed.
Instance resource URI `/api/rest/<resource_type>/<id>`	`GET`	Retrieves the specified resource instance.
	`PATCH`	Modifies the specified resource instance, if allowed.
	`DELETE`	Deletes the specified resource instance, if allowed.
Instance action URI `/api/rest/<resourceType>/<id>/<action>`	`POST`	Performs the operation specified by `<action>` for the specified resource instance.
Class-level action URI `/api/rest/<resource_type>/<action>`	`POST`	Performs the operation specified by `<action>` for the specified non-singleton resource type.

NOTE: URI parameters cannot contain slashes (`/`) or other special characters.

Examples

Retrieving all instances of the storage_container resource type

```
GET /api/rest/storage_container
```

Retrieving storage_container instance 70b37e07-b7fb-4e69-83ea-928680c4dc31

```
GET api/rest/storage_container/70b37e07-b7fb-4e69-83ea-928680c4dc31
```

Creating a new storage container instance

```
POST /api/rest/storage_container
```

Deleting storage container instance 70b37e07-b7fb-4e69-83ea-928680c4dc31

```
DELETE /api/rest/storage_container/70b37e07-b7fb-4e69-83ea-928680c4dc31
```

Modifying storage container instance 70b37e07-b7fb-4e69-83ea-928680c4dc31

```
PATCH /api/rest/storage_container/70b37e07-b7fb-4e69-83ea-928680c4dc31
```

Mounting storage container instance 70b37e07-b7fb-4e69-83ea-928680c4dc31

```
POST /api/rest/storage_container/70b37e07-b7fb-4e69-83ea-928680c4dc31/mount
```

Exchanging SSL certificates with another PowerStore appliance

```
POST /api/rest/x509_certificate/exchange
```

Related references

HTTP request headers
Request parameters
REST requests

Request parameters

The PowerStore REST API supports the following request parameters:

Table 3. Request parameters

Request parameter	Applicable request types	Description
`select`	Collection and instance queries	Specifies a comma-separated list of attributes to return in a response. If you do not use this parameter, a query returns the `id` attribute only. For more information, see Specifying the attributes to return in a query response.
`<filter expression>`	Collection queries	Filters the response data against a set of criteria. Only matching resource instances are returned. Filtering is case insensitive. For more information, see Filtering response data.
`order`	Collection queries	Specifies how to sort response data. You can sort response data in ascending or descending order based on one of the following: • The attributes of the queried resource type. • The attributes of a resource type that is embedded in the primary resource type or related through a foreign key. • A computed attribute. For more information, see Sorting response data.
`async`	Most non-GET requests	Setting this parameter to `true (?async=true)` runs the request in the background. Most active management requests (ones that attempt to change the configuration) support this option. By default, requests are run synchronously. For more information, see Working with asynchronous requests.

To use request parameters, append the parameters to the request URI. The first request parameter appended to the URI begins with a ? character. Subsequent request parameters appended to the URI begin with a & character. You can combine request parameters and can use them in any order.

Examples

Using the `select` request parameter

The following request uses a `select` request parameter to return the values for the `id`, `severity`, and `description` attributes for all alert instances.

Table 4. Using the select request parameter

Request and response	Request and response example
Request	`GET https://1.2.3.4/api/rest/alert?select=id,severity,description_l10n`

Response	<pre>[
	  {
	  "id":"014f999e-96d8-4f85-853c-c6a0e2569088",
	  "severity":"Minor",
	  "description_l10n":"Cluster license installation failed due to Fail to obtain license file. The trial period will expire on 2019-11-27 18:47:26.985, after which no new storage provisioning will be allowed."
	  },
	  {
	  "id":"2d3cc48d-0296-4d15-afb9-252061cf7df9",
	  "severity":"Minor",
	  "description_l10n":"Bus 0 enclosure 0 LCCA is initializing."
	  },
	  {
	  "id":"37583ff1-b0fe-41f2-9b3d-e5e5d2e5e525",
	  "severity":"Info",
	  "description_l10n":"The remote system <undefined> volume discovery or refresh has failed."
	  },
	.
	.
	.</pre>

Using a filter expression

The following request uses a filter expression to return the values for `alert` instances that have the `severity` attribute set to **`Info`**:

Table 5. Using a filter expression

Request and response	Request and response example
Request	`GET https://1.2.3.4/api/rest/alert?select=id,severity&severity=eq.Info`

Response	<pre>[
	  {
	  "id":"37583ff1-b0fe-41f2-9b3d-e5e5d2e5e525",
	  "severity":"Info"
	  },
	  {
	  "id":"63d21fb9-897d-4d07-983f-0478436755b6",
	  "severity":"Info"
	  },
	  {
	  "id":"ab9175c4-1ded-473e-b22d-63ac1154937f",
	  "severity":"Info"
	  }
	.
	.
	.</pre>

Using the `order` request parameter

The following request uses an order expression to sort `alert` instances by `severity`:

Table 6. Using the order request parameter

Request and response	Request and response example
Request	`GET https://1.2.3.4/api/rest/alert?select=id,severity&order=severity`

Response	<pre>[
	  {
	  "id":"37583ff1-b0fe-41f2-9b3d-e5e5d2e5e525",
	  "severity":"Info"
	  },
	  {
	  "id":"63d21fb9-897d-4d07-983f-0478436755b6",
	  "severity":"Info"
	  },
	  {
	  "id":"ab9175c4-1ded-473e-b22d-63ac1154937f",
	  "severity":"Info"
	  },
	  {
	  "id":"014f999e-96d8-4f85-853c-c6a0e2569088",
	  "severity":"Minor"
	  },
	  {
	  "id":"2d3cc48d-0296-4d15-afb9-252061cf7df9",
	  "severity":"Minor"
	  },
	  {
	]</pre>

Using the `is_async` request parameter

The following asynchronous request deletes a `local_user` instance. The request returns the job ID:

Table 7. Using the is_async request parameter

Request and response	Request and response example
Request	`DELETE https://1.2.3.4/api/rest/local_user/4?is_async=true`

Response	<pre>{
	  "id": "1ed0d49e-101b-4b8a-85dc-886f907e8070"
	}</pre>

Related references

HTTP request headers
URI patterns
REST requests

REST requests

A JSON request body for the REST API consists of a collection of name:value pairs for a single resource type. The request body has the following syntax:

- For number or boolean values:

```
{
    <attributeName1>:<value1>,
```

```
    <attributeName2>:<value2>,
    .
    .
    .
}
```

- For IP, string, or datetime values:

```
{
    <attributeName1>:"<value1>",
    <attributeName2>:"<value2>",
    .
    .
    .
}
```

For example, the request body for a create request for the `local_user` resource type could contain the following values:

```
{
    "name": "user1",
    "password": "myPassword"
    "role_id": "operator"
 }
```

Related references

HTTP request headers
URI patterns
Request parameters

3

REST responses

Topics:

Response components

Each response to a REST API request consists of a response header, HTTP status code (in the response header), and JSON response body, if applicable:

- The response header contains metadata about the response being sent.
- The HTTP status code in the response header indicates whether a request is successful or unsuccessful.
  - An HTTP status code in the `2xx` family, such as `200 OK` or `201 Created`, indicates a correctly processed request.
  - An HTTP status code in the `4xx` family indicates an error in the request. For example, a 400 status code indicates a badly formed request, and a 401 status code indicates an authorization error.
  - An HTTP status code in the `5xx` family indicates a server failure. For example a 500 status code indicates an internal server error, and a 503 status code indicates that the REST service is temporarily unavailable.
- The JSON response body varies according to the request type and whether a request was successful. For example, a collection response body is returned in response to a successful `GET` collection request, and an instance response body is returned in response to an instance request.

HTTP and HTTPS response headers

A response from the REST API always includes HTTP or HTTPS response headers that contain metadata about the response being sent.

NOTE: All HTTP responses from PowerStore now include the Strict-Transport-Security header as part of the HTTP Strict Transport Security (HSTS) feature. This feature is enabled by default. No configuration is required.

An `HTTPS 301 (Permanently Moved)` message appears if you try to access management over`http://`. You are then redirected to an `https://` address.

The following HTTP headers appear in REST API responses:

Table 8. HTTP response headers in the REST API

HTTP header	Description
Content-Type	This header is set to `application/json`, although it can be `application/zip`, `document/text`, or `application/binary`.
Content-Length	This header refers to the byte length of the HTTP body.
Set-Cookie	The login session ticket (`auth_cookie`) is returned in the first request of the session and required for all subsequent requests, unless you pass the user ID and password with

Table 8. HTTP response headers in the REST API (continued)

HTTP header	Description
	each request. Because the API uses cookie-based authentication, the HTTP client must support cookies in order to use the API. For more information, see Connecting and authenticating.
DELL-EMC-TOKEN	Before issuing any REST call which changes the state of the object (such as `POST`, `PATCH` or `DELETE`) send a `GET` request to receive a CSRF token as response header named `DELL-EMC-TOKEN`. Use the value of the token from the response header that is obtained from the GET call as a Header value in the subsequent calls for this session.
Content-Range	Paginated responses contain this header, which indicates how many instances were successful and how many were returned.

HTTPS headers use HSTS to ensure browsers use the more-secure HTTPS when connecting to servers. HSTS and the Strict-Transport-Security help prevent man-in-the-middle attacks.

This feature only works on browsers. The Strict-Transport-Security header is cached on the browser.

The following HTTPS headers appear in REST API responses:

Table 9. HTTPS response headers in the REST API

HTTPS header	Description
Strict-Transport-Security	This header indicates that HSTS is being used for security.
Strict-Transport-Security-Options	This header allows the following options to be used with the Strict-Transport-Security header: • `max-age`: This directive ensures that the browser redirects any **`http`** requests to **`https`** automatically for one year. • `includeSubDomains`: This directive states that the HSTS policy applies to all subdomains as long as those subdomains support `https`.

HTTP status codes

Every response to a REST API request includes an HTTP status code, which indicates whether the request was successful. If requests are unsuccessful (that is, if they return `4xx` and `5xx` HTTP status codes) the system returns a message entity that describes the problem.

The following table describes the HTTP status codes that apply to the REST API:

Table 10. HTTP status codes in the REST API

Status code	Name	Applies to	Description
200	`OK`	All of the following: • `GET` requests • Action `PATCH` requests with output data • Action `POST` requests with output data	Successful request. For a `GET` request, the response body contains the requested resource. For an action `POST` and `PATCH` request, the response body contains the output arguments.
201	`Created`	`POST` requests for creating resources	Successful request. The response body contains the `id` attribute only.
202	`Accepted`	Asynchronous `POST`, `PATCH`, and `DELETE` requests	Request is in process. The response body is the ID of the job resource instance executing the request.

Table 10. HTTP status codes in the REST API (continued)

Status code	Name	Applies to	Description
204	`No Content`	Non-`GET` requests.	Successful request. There is no body content in the response.
206	`Partial Content`	`GET` requests	Successful request. The response body contains a partial response. Used for pagination of a long response.
400	`Bad Request`	All requests	Problem with request intercepted before execution. The system detected a problem with the request and did not proceed. The request may have a badly formed URI or badly formed parameters, headers, or body content.
401	`Unauthorized`	All requests	Authentication error. The username and password combination or the auth-cookie sent with the request are not valid.
403	`Forbidden`	All requests	Authorization error. The role of the authenticated user does not have the privilege required to perform the requested operation.
404	`Not Found`	All requests	Resource does not exist. This can be caused by an invalid ID in an instance URI.
405	`Method Not Allowed`	All requests	Specified resource does not support the request's operation. For example, requesting a `DELETE` on a hardware resource can cause this error.
416	`Range Not Satisfiable`	GET requests	The specified range of items is invalid. The value of the Range header or the URL parameters `limit` or `offset`, or both, are invalid.
422	`Unprocessable Entity`	Non-`GET` requests	Error during execution of the request. The response body contains an error message that describes the problem with the request. Example causes: • The system is out of space. • A range error occurred. • There are inconsistent properties on a `POST` or `PATCH`
500	`Internal Server Error`	All requests	Internal error.
503	`Service Unavailable`	All requests	The REST service is temporarily unavailable. Wait and try the request again.

JSON collection response body

A JSON collection response body occurs in response to a `GET` collection request that results in a `200 OK` or a `206 Partial Content` HTTP status code. For collections with fewer than 100 elements, the response body for a collection contains the identifier for all instances in the resource type collection. The response is a JSON array of zero or more instances. Each instance in the response has one or more attribute name-value pairs. The returned name-value pairs are determined by the `select` URI parameter, which defaults to the `id` attribute. You can specify which attributes to return for each instance by using the `select` query parameter. You can specify which instances are returned by filtering the data using an attribute expression as a query parameter.

Example

GET collection response for an event

The following example illustrates the components of a collection resource. It shows a collection resource returned in response to a `GET` collection request for the `event` resource type. In this example, the query returns the `id` and `severity` of each event in the appliance. Spaces outside the quoted strings are used for readability, and are not significant.

Table 11. GET collection response for an event

Request and response	Request and response example
Request	`GET https://1.2.3.4/api/rest/event?select=id,severity`
Response	List of all instances in the specified collection that meet the request criteria. The response includes attribute values as a set of name:value pairs for each returned instance: `[ { "id": "0a739865-a233-4ef0-ae20-6ed014ad06fe", "severity": "Minor" }, { "id": "51c0db9e-1d2b-4289-97b7-f361d67f7807", "severity": "Minor" }, . . . ]`

JSON instance response body

A JSON instance response occurs in response to a `GET` instance request that results in a `200 Success` HTTP status code. By default, this response body contains only the identifier of the requested resource instance. Use the `select` request parameter to specify the additional attribute values to return.

Example

GET instance response for an event

The following example illustrates an instance resource. It shows an instance resource returned in response to a `GET` instance request for the `event` resource type with an identifier of 299a2f56-fea7-4e85-af0d-992294526911. Spaces outside the quoted strings are used for readability, and are not significant.

Table 12. GET instance response for an event

Request and response	Request and response example
Request	`GET https://1.2.3.4/api/rest/event/299a2f56-fea7-4e85-af0d-992294526911?select=id,severity`
Response	`{ "id": "299a2f56-fea7-4e85-af0d-992294526911", "severity": "Info" }`

JSON create response body

A JSON create response body occurs in response to a `POST` operation for create that results in a `201 Created` HTTP return code. This response body contains the `id` attribute for the new resource.

Example

Create response for a local_user

The following examples illustrate the components of a minimal instance resource. It shows a minimal instance resource returned in response to a successful `POST` request for creating a new local_user resource. The request body contains the arguments used to populate the new resource.

Table 13. Create response for a local_user

Request and response	Request and response example
Request	`POST https://1.2.3.4/api/rest/local_user`
Request body	`{ "name": "User1", "password": "myPassword", "role_id": "1" }`
Response	`{ "id": "4" }`

Response with no body

A response with no body occurs in response to a `DELETE` request that results in a `204 No Content` status code. In this circumstance, response headers are returned with the empty response body. A response with no body also occurs in response to an action `POST` request that does not have output data. This response with no body also occurs in response to a successful synchronous `DELETE` instance request that does not have output data and that results in a `204 No Content` status code.

JSON job response body

A JSON response body occurs in response to an asynchronous request that results in a `202 Accepted` HTTP return code. This response body contains the job id. You can query the `job` resource instance to find out whether the job completed and to get the response to the asynchronous request. For a description of the `job` resource type, see the job topic in the Reference content.

Example

Deletion response for a local_user

The following example returns a job resource instance in response to a successful asynchronous `DELETE` request.

Table 14. Deletion response for a local_user

Request and response	Request and response example
Request	`DELETE https://1.2.3.4/api/rest/local_user/4?is_async=true`
Response	`{ "id": "476c903a-1bc2-4370-9a4b-426594ed9604" }`

Error response

An error response is returned in response to an unsuccessful request; that is, a request that returns a 4xx or 5xx HTTP status code. Unlike the response bodies returned by successful requests, an error response cannot be queried independently.

For a description of the error response attributes, see the Reference content.

Example

Error response for a `POST` request

The following example returns an error response in response to an unsuccessful `POST` request:

Table 15. Error response for a POST request

Request, request body, and response	Request, request body, and response example
Request	`POST https://1.2.3.4/api/rest/volume_group/257f2597-7c11-46c9-8941-3793e1cb2baa/refresh`
Request body	`{ "from_object_id": "22f3bf53-ad83-4466-8e5e-5b0fade650da" }`
Response	`{ "messages":[ { "code": "0xE0A070020007", "severity": "Error",`

Table 15. Error response for a POST request (continued)

Request, request body, and response	Request, request body, and response example
	"message_l10n": "Family mismatch for snapshot target 257f2597-7c11-46c9-8941-3793e1cb2baa and source 22f3bf53-ad83-4466-8e5e-5b0fade650da.", "arguments": ["257f2597-7c11-46c9-8941-3793e1cb2baa", "22f3bf53-ad83-4466-8e5e-5b0fade650da"] }] }

4

JSON encodings

Topics:

JSON base value encodings

The following table shows the JSON encodings for each base type:

NOTE: A property that has no value appears as the type `null`.

Table 16. JSON base value encodings

Type name	JSON definition	Format after "<name>":	Example	Notes
short	type: integer format: int16	<int value>	"drive_count":600	N/A
integer	type: integer format: int32	<int value>	"answer": 42	N/A
long	type: integer format: int64	<int value>	"size": 123456789	N/A
float	type: number format: float	<float value>	"progress": 99.9	N/A
double	type: number format: double	<float value>	"throughput": 123456.78	N/A
string	type: string	<string value>	"description":"some text"	Use \ to escape the quote (") and control characters.
byte	type: string format: byte	<base64 encoded character string>	"bitmask":"c3VyZS4="	Base64 encoded byte sequence.
binary	type: string format: binary	<octet sequence string>	"checksum": "1a2b3c4d1234dcba"	Hex encoded byte sequence.
boolean	type: boolean	true|false	"force":true	Case insensitive.
date	type: string format: date	"yyyy-mm-dd"	"expiration_date": "2020-02-02"	As defined by full-date (RFC3339).
timestamp	type: string format: timestamp	"hh:mm:ss[.sss]"	"daily_start_time": "03:30:00"	As defined by partial-time (RFC3339).

Table 16. JSON base value encodings (continued)

Type name	JSON definition	Format after "<name>":	Example	Notes
`date-time`	type: `string` format: `date-time`	`yyyy-mm-ddThh:mm:ss[.sss]Z`	`"updated": "2015-07-14T18:21:32.621Z"`	As defined by `full-date` (RFC3339). All times are expressed in UTC time. The optional `[.sss]` contains fractional milliseconds.
`password`	type: `string` format: `password`	`<string value>`	`"password":"wordpass"`	This is a string in the API and may be presented differently by a client (to hide typed input). This is handled differently on the server (for example, the values are not logged).
`ip-address`	type: `string` format: `ip-address`	String containing an IPv4 address, IPv6 address, or hostname.	`"mgmtAddr":"128.222.1.2"`	In this API, some attributes support IPv4 only, while others support both IPv4 and IPv6. Some attributes also support DNS names. The help topics for individual resource types in the Reference content indicate which IP address options are supported by that resource type.
`URI`	type: `string` format: `uri`	`"<url in a string>"`	`"href": "https://foo.com/bar.jsp"`	N/A
`embedded`	type: `string` format:	`{ "<propName>":<value1>, … }`	`" health": { "value":0, "description": "OK", "resolution":"" }`	In this example, `health` is an embedded type with three attributes: `value`, `description`, and `resolution`.
`enum`	type: `string` format:	`<string value>`	`"severity":"Error"`	Enumeration values are single token strings. Each enumeration is defined in the *Dell PowerStore REST API Reference Guide*.
`id`	type: `string` format:	`<string value>`	`"id":"123"`	<id> value of the referenced resource instance.

JSON list encoding

A JSON list is a list of values with the following format:

```
[ <value1>, <value2>, <value3>,...]
```

where:

- Square brackets enclose the list.
- Commas separate each value.
- `<value>` can be another list or any of the base value encoding formats.

JSON lists can be empty.

Examples

An empty list

```
"addresses":[ ]
```

A list with one value

```
"addresses":["1.2.3.4"]
```

A list with three values

```
"addresses":["1.2.3.4","5.6.7.8","4.3.2.1"]
```

5

Managing a REST API session

Topics:

Connecting and authenticating

All requests to the REST API must be authenticated. The REST API uses the standard HTTP Basic access authentication mechanism to authenticate REST requests. The same users are valid for REST or UI access.

To log in to the REST API server, use the following request components:

Table 17. Request components for connecting and authenticating

Header, operation, and URI pattern	Header, operation, and URI pattern example

Header	`Accept: application/json`
	`Content-Type: application/json (if the request has a non-empty request body)`
	`Authorization: <base 64 encoding of username and password>`

Operation	`GET`, `PATCH`, `POST`, or `DELETE`
URI pattern	URI pattern for one of the requests listed in the Operation section

NOTE: It is a best practice to maintain a single session when issuing multiple commands. Working within a single session is more secure. In addition, authenticating with a username and password for every command results in a potentially excessive number of login audit events in the PowerStore audit log.

To avoid exposure to a cross-site request forgery (CSRF), requests other than `GET` require the `DELL-EMC-TOKEN` header. Before issuing any REST call which changes the state of the object (such as `POST`, `PATCH` or `DELETE`), send a `GET` request to receive a CSRF token. Use the value of token from the response header that is obtained from the `GET` call as a Header value in the subsequent calls for this session.

The server returns the following in response to a successful login:

- A `200 OK` HTTP status code.
- `auth_cookie` header, which is required to authenticate all subsequent requests, unless you resubmit the user ID and password with each request. It is also required for logging out of the session.

  NOTE: Once the `auth_cookie` value is set, a browser automatically sets the cookie value for each request.

If the authentication fails, the server returns a `401 Unauthorized` HTTP status code.

Obtaining login session information

Query the `login_session` resource type to find out basic information about the current session. The following table describes the information that is returned in response to a successful query of the `login_session` resource type:

Table 18. Response to a query of the login_session resource type

Attribute	Description
`id`	The unique identifier of the `login_session` resource instance.
`user`	Information about the user logged into this session defined by the `local_user` resource type.
`role_ids`	Roles for the user who is logged into this session defined by the `role` resource type.
`idle_timeout`	Number of seconds after last use until this session expires.
`is_password_change_required`	Indicates whether the password must be changed in order to use this session created for the built-in admin account. The values are: • `true` - Password must be changed. • `false` - Password does not need to be changed. For information about changing the password for a local user, see the Help topic for the `local_user` resource type in the Reference content.
`is_built_in_user`	Indicates whether the user is a system-defined user, such as **`admin`**.

Example login session information

```
[
  {
    "id":"85d0dbfc-d364-47b9-ad90-0fe03efdf860",
    "user"1",
    "role_ids":["1"],
    "idle_timeout":36000,
    "is_password_change_required":false,
    "is_built_in_user":true
  }
]
```

Logging out of the REST API session

Use the following request components to log out of an existing REST API session:

Table 19. Request components to log out of a REST API session

Header, operation, and URI pattern	Header, operation, and URI pattern example

Header	`Accept: application/json`
	`Content-Type: application/json`
	`Authorization: <auth_cookie>`

Operation	`POST`
URI pattern	`https://1.2.3.4/api/rest/logout`

The server returns a `204 No Content` HTTP status code and an empty response body in response to a successful logout.

6

Querying a resource

Topics:

Pagination

When you query a resource, the server limits the size of the returned collection by default. This collection is limited to the number of instances up to a specified pagination limit, which defaults to 100. If the requested collection has fewer instances than this limit, the response contains the status code `200 OK` and all requested instances in its body.

When the requested collection is larger than the limit, the response status code changes to `206 (Partial Content)`. The response body contains only the partial collection with a size equal to the pagination limit.

Also, a special header `Content-Range` is added. This header contains information about how many items were returned, their position in the whole collection, and the total size of the collection. For example, `0-99/1000` indicates that first 100 items were returned out of 1000.

Pagination parameters and ordering

You can control the number and position of the returned partial collection by using pagination parameters and ordering. Pagination parameters can be passed as a URL parameters `limit` and `offset`, or in a request header `Range`, which specifies indexes of the first and last items in the total collection to return. <add a link to the section about ordering>

Table 20. Pagination parameters

Parameter	Type	Description
`offset`	`integer`	Use this parameter to specify what the first item in the returned collection. The default is 0.
`limit`	`integer`	The maximum number of entries that the query returns.

The default pagination limit is 100; you can enter values from 1 to 2000. If 0 or less is entered as the value of the `limit`, the response is the status code `416 Range Not Satisfiable`. A response to a request with a valid `limit` parameter contains results up to the specified maximum number of items. If you specify a larger value, the server changes the limit to 2000.

When you limit the number of returned instances, by default, the first instances from the collection are returned based on the ordering. To get a partial collection starting from an index other than 0, use the URL parameter `offset`. Its default value is 0; other valid values are positive integers up to the total size of the collection. Invalid offset values cause the response status code `416 Range Not Satisfiable`.

Another way that you can request a specific range of items from a collection is to specify the first and last index of the instances of the partial collection using the request header `Range`. For example, a request with the header `Range: "100-199"` retrieves the second batch of 100 items in the collection.

Retrieving data for multiple occurrences in a collection

To retrieve data for multiple occurrences of a resource type, use the following request components:

Table 21. Retrieving data for multiple occurrences in a collection

Headers, operation, and URI pattern	Headers, operation, and URI pattern example
Headers	`Accept: application/json` `Authorization: <auth_cookie>`
Operation	`GET`
URI pattern	`/api/rest/<resource_type>` where *<resource_type>* is the resource type for the collection you want to return. For additional functionality, such as filtering instances, you can append one or more request parameters to the URI.
Body	Empty.

If the request succeeds, the server returns a `200 OK` HTTP status code and a collection resource in the response body. If the request does not succeed, the server returns a 4xx or 5xx HTTP status code and an error response. If there are no instances in a collection, the server returns a `200 OK` HTTP status code and an empty body containing only `[]`.

By default, the response to a `GET` collection request includes only the unique identifier for each instance of the specified resource type. You can use the following request parameters to customize the returned data:

Table 22. Request parameters to customize the returned data

Request parameters	Description
`select`	Requests data for a specified set of attributes.
`order`	Specifies how to sort the response data.
`<attribute expression>`	Filters the response data against a set of criteria. Only matching resource instances are returned. Filtering is case insensitive.
`limit`	Limits the maximum number of items in the returned collection (1-2000).
`offset`	Provides an index of the first returned instance.

Retrieving data for all alerts in the system

The following request returns information about all alerts in the system. The `select` parameter specifies that the values for the `id`, `severity`, and `description` attributes should be returned. This example shows three returned instances:

Table 23. Retrieving data for all alerts in the system

Header, request, and body	Header, request, and body example
Header	`Accept: application/json`
Request	`GET https://1.2.3.4/api/rest/alert?` `select=id,event_code,severity,description_l10n`
Request body	Empty.

Table 23. Retrieving data for all alerts in the system (continued)

Header, request, and body	Header, request, and body example

Response body	<pre>[
	  {
	  "id":"014f999e-96d8-4f85-853c-c6a0e2569088",
	  "severity":"Minor",
	  "description_l10n":"Cluster license installation failed due to Fail to obtain license file. The trial period will expire on 2019-11-27 18:47:26.985, after which no new storage provisioning will be allowed."
	  },
	  {
	  "id":"2d3cc48d-0296-4d15-afb9-252061cf7df9",
	  "severity":"Minor",
	  "description_l10n":"Bus 0 enclosure 0 LCCA is initializing."
	  },
	  {
	  "id":"37583ff1-b0fe-41f2-9b3d-e5e5d2e5e525",
	  "severity":"Info",
	  "description_l10n":"The remote system <undefined> volume discovery or refresh has failed."
	  }
	]</pre>

Retrieving data for a specified resource instance

To retrieve data for a specified resource instance, use the following request components:

Table 24. Retrieving data for a specified resource instance

Headers, operation, and URI pattern	Headers, operation, and URI pattern example

Headers	<pre>Accept: application/json
	Authorization: <auth_cookie></pre>

Operation	`GET`
URI pattern	<pre>/api/rest/<resource_type>/<id></pre> where: ● *<resource_type>* is the resource type of the desired instance. ● *<id>* is the unique identifier of the desired instance. For additional functionality, such as returning specific attributes, you can append one or more request parameters to the URI.
Body	Empty.

If the request succeeds, the server returns a `200 OK` HTTP status code and an instance resource in the response body. If the request does not succeed, the server returns a 4xx or 5xx HTTP status code and a message entity in the response body.

By default, the response to a `GET` instance request includes only the unique identifier (id attribute) of the specified resource instance. You can use the following request parameters to customize what data is returned:

Table 25. Request parameters to customize the returned data

Request parameter	Description
`select`	Requests data for a specified set of attributes.

Example

The following request returns the values for the `id`, `name`, and `serial_number` attributes for the `appliance` resource instance that has an id of `J8WRPD2`.

Table 26. Example of retrieving data for a specified resource instance

Header, request, and response body	Header, request, and response body example
Header	`Accept: application/json`
Request	`GET https://1.2.3.4/api/rest/appliance/J8WRPD2?select=id,name,service_tag`
Response body	`{ "id": "J8WRPD2", "name": "H2025-appliance-1", "service_tag": "J8WRPD2" }`

Specifying the attributes to return in a query response

Use the `select` request parameter in a collection query to specify the set of attributes to return in a response. If you do not use this parameter, a query will return the `id` attribute only.

When you use the `select` request parameter, you can refer to attributes in a related resource type, as described in the Syntax section below.

Syntax

As the first parameter on the request URI: `?select=<attr1>,<attr2>,<attr3>...`

As a subsequent parameter on the request URI: `&select=<attr1>,<attr2>,<attr3>...`

where the attributes whose values you want to retrieve are listed in a comma-separated list.

You can use nested notation syntax (`<reference_attribute_name>(<attr1?,<attr2>...)`) in a `select` expression to return the values of attributes from related resource types. A related resource type is a resource type that is either referred to explicitly in the definition of the target resource type or embedded in the target resource type.

Considerations

The following considerations apply to using the `select` parameter:

- If a `select` request is made for an attribute that is not defined on the resource type, the server returns a `400 Bad Request` error.
- If an attribute has a valid, empty string value, the server returns the value as `<attribute>:""`.
- Although a response normally contains only the requested attributes, this is not guaranteed. You should therefore be prepared to ignore unrequested properties.

Retrieving the name and create_time for a local_user

The following request retrieves values for the `name` and `create_time` attributes in the `local_user` resource collection:

Table 27. Retrieving the name and create_time for a local_user

Headers, request, and response body	Headers, request, and response body example
Headers	`Accept: application/json` `Authorization: <auth_cookie>`
Request	`GET https://1.2.3.4/api/rest/local_user?select=name,role_id`

Response body	`[`
	`  {`
	`  "name": "admin",`
	`  "role_id": "1"`
	`  },`
	`  {`
	`  "name": "stadmin",`
	`  "role_id": "2"`
	`  },`
	`  {`
	`  "name": "vmadmin",`
	`  "role_id": "4"`
	`  }`
	`]`

Filtering response data

Use one or more filter expressions in a request parameter to specify matching criteria for a list of resources returned by a collection query. A filter expression works like an SQL WHERE clause. You specify a filter expression composed of boolean predicates, and the expression is applied against the attribute values of the requested resource type. Only those instances that cause the filter expression to evaluate to true are returned in the query response. The system ANDs together multiple filter expressions.

Filter expressions use operators such as `gt`, `gte`, `lt`, and `lte`. The interpretation of these operators is type-dependent. For example, `gt` used with `datetime` attributes means the date value to the right of `gt` must be more recent than the date value to the left of `gt`.

Filtering is case insensitive for any property called `name`.

Using a filter expression can save bandwidth, because the server removes extra data before returning data to the client. However a filter expression does not reduce the amount of work the server performs to answer the request.

NOTE: Complex requests can be slow or can fail.

Syntax

As the first parameter on the request URI: `?<attribute_name>=<filter_expr>`

As a subsequent parameter on the request URI: `&<attribute_name>=<filter_expr>`

where `<filter_expr>` is defined by the following syntax :

`[not.]<operator>.<filter value>` You can use nested notation syntax (`<reference_attribute_name>(<attr1>,<attr2>...)`) in a `filter` expression to filter by attributes from a related resource type.

NOTE: Some resource types and attributes do not support filtering, which is indicated in their descriptions.

You can also filter by attributes that are not requested in the select URL parameter. A related resource type is a resource type that is either referred to explicitly in the definition of the target resource type or embedded in the target resource type.

Supported operators

Table 28. Supported operators

Comparator	Description	Applicable data types	Example
`eq`	Equal. Returns true when the expression on the left of `eq` equals the expression on the right.	All	`?role_id=eq.3` True if the value of `role_id` is `Operator`.
`gt`	Greater than. Returns true when the expression on the left of `=eq` is greater than the expression on the right.	All	`?low_value=gt.5` True if the value of `low_value` is greater than 5.
`gte`	Greater than or equal. Returns true when the expression on the left of `=eq` is greater than or equal to the expression on the right.	All	`?software_version=gte.5.3` True if the value of `software_version` is greater than or equal to 5.3.
`lt`	Less than Returns true when the expression on the left of `=eq` is less than the expression on the right.	All	`?modify_time=lt.2018-05-07T17:56:28.859+00:00` True if the value of `modify_time` is earlier than 2018-05-07T17:56:28.859+00:00.
`lte`	Less than or equal. Returns true when the expression on the left of `=eq` is less than or equal to the expression on the right.	All	`?modify_time=lte.2018-05-07T17:56:28.859+00:00` True if the value of `modify_time` is 2018-05-07T17:56:28.859+00:00 or earlier.
`neq`	Not equal. Returns true when the expression on the left of `=eq` is not equal to the expression on the right.	All	`?role_id=neq.3` True if the value of `role_id` is anything other than `Operator`.
`ilike`	Case-insensitive substring match. Returns true when the expression on the left of `=eq`, including its case, matches the expression on the right. Use * for a wildcard.	String	`?name=ilike.User*` True if the value of `name` starts with `User`.
`in`	One of a list of values. Returns true when the expression on the left of `=eq` matches any value in the comma-separated list on the right.	All	`?role_id=in.(operator,administrator,service)` True if the value of `role_id` is `Operator`, `Administrator`, or `VM Administrator`.
`is`	Checks for exact equality to null, true, or false values.	null (All), true, false (Boolean)	`?description=is.null`
`cs`	Contains.	List of any type	`?supported_speeds=cs.{1_Gbps,10_Gbps}` True if the `speeds` array contains 1 Gbps and 10 Gbps.

Table 28. Supported operators (continued)

Comparator	Description	Applicable data types	Example
cd	Contained in.	List of any type	?server_addresses=cd.{"1.2.3.4","5.6.7.8"} True if the values in the server addresses list are all either 1.2.3.4 or 5.6.7.8.
not	Negates the operator that follows.	All	?resource_type=not.in.(Operator,Administrator,VM Administrator) True if the value of role_id is not Operator, Administrator, or VM Administrator.
and	All the conditions in the parentheses that follow must be true in order for the request to evaluate as true. **NOTE:** The default for multiple filters is to AND them, so the use of and is optional unless you are using both and and or in the filter expression.	All	?and=(size.gt.50,performance_policy_id.eq.default_high) True if the value of size is greater than 50 MB and the value of performance_policy is High.
or	At least one of the conditions in the parentheses that follow must be true in order for the request to evaluate as true.	All	?or=(size.gt.50,performance_policy_id.eq.default_high) True if either the value of size is greater than 50 MB or the value of performance_policy is High or both.

Sorting response data

Use the order request parameter to specify sort criteria for one attribute in a list of resources returned by a collection query. The order parameter works like an SQL Order By clause. You can specify one of these sort orders for the attribute:

- asc: (Default) Sorts the response data in ascending order.
- desc: Sorts the response data in descending order.

Sorting is case insensitive for any property called name.

Syntax

As the first parameter on the request URI: ?order=<order_expr>

As a subsequent parameter on the request URI: &order=<order_expr>

where <order_expr> is defined by the following syntax:

prop_expr1[.asc|.desc],[prop_expr2[.asc|.desc]...

where:

- prop_expr1 and prop_expr2 are attribute names that are defined for the resource being queried.
- asc and desc are case insensitive.

Retrieving values for all alerts, sorted by state

The following request retrieves values for the id, description, and alert_state attributes for all alert instances and sorts this information by alert_state in ascending order.

Table 29. Retrieving values for all alerts and sorting by state

Headers, requests, and response body	Headers, request, and response body example
Headers	`Accept: application/json` `Authorization: <auth_cookie>`
Request	`GET https://1.2.3.4/api/rest/alert?select=id,description_l10n,state&order=state`

Response body	<pre>[
	  {
	  "id": "1bf5e114-b476-492b-a37e-7d9c1b38e7b9",
	  "description_l10n": "Firmware version of the rights DAE PSU is no_errors. Current firmware version is 09.16s.",
	  "state": "Active"
	  },
	  {
	  "id": "27f6caef-219a-406c-ad74-2ab801e56b19",
	  "description_l10n": "Firmware version of the tops DAE Controller is no_errors. Current firmware version is 2.30.0s.",
	  "state": "Active"
	  },
	  {
	  "id": "0ed1b64a-91aa-444e-a832-cec924e41850",
	  "description_l10n": "Port state was changed from downs to ups.",
	  "state": "Cleared"
	  },
	  {
	  "id": "d406b80d-db51-4853-a835-23ed5512ae0d",
	  "description_l10n": "Port state was changed from downs to ups.",
	  "state": "Cleared"
	  }
	]</pre>

Including data from a related resource type in a query

You can extend the scope of a collection query to retrieve data from a related resource type. The REST model describes two kinds of relations:

- A referenced resource type
- An embedded resource type

Referenced resource types

A referenced resource type describes an instance of a linked object type. This resource type is declared in the REST model as a reference to an object (for a many-to-one relation) or an array of objects (for a one-to-many or many-to-many relation). To return information about instances and their related instances, use the following syntax in the `select` parameter to specify the wanted attributes from the related resource type:

```
...<attribute1>,<attribute2>,<related_resource> (id,related_attribute2,
related_attribute3...)
```

If you omit the list of parameters, only the `select` attribute and its value are returned for the referenced object:

```
...<attribute1>,<attribute2>,<related_resource>
```

Examples

The following queries use the `select` request parameter to return information about nodes and their related appliances.

Table 30. Example 1: Returning specified parameters for the appliance related to each node

Component	Details
Headers	`Accept: application/json` `Authorization: <auth_cookie>`
Request	`https://1.2.3.4/api/rest/node?select=id,name,appliance(id,name)`
Response body	`[ { "id":"N1", "name":"H0112-appliance-1-node-A", "appliance": { "id":"A1", "name":"H0112-appliance-1" } }, { "id":"N2", "name":"H0112-appliance-1-node-B", "appliance": { "id":"A1", "name":"H0112-appliance-1" } } ]`

Table 31. Example 2: Returning only the id parameter for the appliance related to each node

Component	Details
Headers	`Accept: application/json` `Authorization: <auth_cookie>`
Request	`https://1.2.3.4/api/rest/node?select=id,name,appliance`
Response body	`[ { "id":"N1", "name":"H0112-appliance-1-node-A", "appliance": { "id":"A1" } }, { "id":"N2", "name":"H0112-appliance-1-node-B", "appliance": { "id":"A1" } } ]`

Table 32. Example 3: Returning specified parameters for all nodes related to an appliance

Component	Details
Headers	`Accept: application/json` `Authorization: <auth_cookie>`
Request	`https://1.2.3.4/api/rest/appliance?select=id,name,nodes(id,name)`

Response body	<pre>[
	  {
	  "id":"A1",
	  "name":"H0112-appliance-1",
	  "nodes":
	  [
	  {
	  "id":"N1",
	  "name":"H0112-appliance-1-node-A"
	  },
	  {
	  "id":"N2",
	  "name":"H0112-appliance-1-node-B"
	  }
	  ]
	  }
	]</pre>

Table 33. Example 4: Returning only the id parameter for all nodes related to an appliance

Component	Details
Headers	`Accept: application/json` `Authorization: <auth_cookie>`
Request	`https://1.2.3.4/api/rest/appliance?select=id,name,nodes`

Response body	<pre>[
	  {
	  "id":"A1",
	  "name":"H0112-appliance-1",
	  "nodes":
	  [
	  {
	  "id":"N1"
	  },
	  {
	  "id":"N2"
	  }
	  ]
	  }
	]</pre>

Embedded resource types

An embedded resource type describes an instance of nested objects. This resource type is declared in the REST model as a reference to an object or an array of objects with a special note in its description field.

- Embedded resources cannot be queried without a parent object.
- Embedded resources always return all fields. These objects do not allow the user to select which fields to return.

Use the following syntax in the `select` parameter to query an embedded resource type:

```
...<attribute1>,<attribute2>,<embedded_resource>
```

NOTE: Extending the scope of a collection query is not supported for all resources. Resources that do not support this functionality include a note in their description stating: `Filtering on the fields of this embedded resource is not supported.`

In addition, you can display selected attributes of an embedded resource:

```
...<attribute1>,<attribute2>,<embedded_resource>-
>>embedded_attribute1,<embedded_resource>->>embedded_attribute2...
```

Table 34. Example 5: Querying the protection data information for a volum

Component	Details
Headers	`Accept: application/json` `Authorization: <auth_cookie>`
Request	`https://1.2.3.4/api/rest/volume?select=id,protection_data`

Response body	[
	{
	"id":"3f6208d1-b325-4901-9fc1-9e82d943b857",
	"protection_data":
	{
	"family_id": "3f6208d1-b325-4901-9fc1-9e82d943b857",
	"parent_id": null,
	"source_id": null,
	"creator_type": "User",
	"copy_signature": null,
	"source_timestamp": null,
	"creator_type_l10n": "User",
	"is_app_consistent": null,
	"created_by_rule_id": null,
	"created_by_rule_name": null,
	"expiration_timestamp": null
	}
	},
	{
	"id":"9d102db7-048d-4d24-85c2-28c7a5fc91ee",
	"protection_data":
	{
	"family_id": "9d102db7-048d-4d24-85c2-28c7a5fc91ee",
	"parent_id": null,
	"source_id": null,
	"creator_type": "User",
	"copy_signature": null,
	"source_timestamp": null,
	"creator_type_l10n": "User",
	"is_app_consistent": null,
	"created_by_rule_id": null,
	"created_by_rule_name": null,
	"expiration_timestamp": null
	}
	},
	{
	"id":"f0480b0f-cc6f-431b-9e18-e4b39b2b13e1",
	"protection_data":
	{
	"family_id": "f0480b0f-cc6f-431b-9e18-e4b39b2b13e1",
	"parent_id": null,
	"source_id": null,
	"creator_type": "User",

Table 34. Example 5: Querying the protection data information for a volum (continued)

Component	Details
	"copy_signature": null, "source_timestamp": null, "creator_type_l10n": "User", "is_app_consistent": null, "created_by_rule_id": null, "created_by_rule_name": null, "expiration_timestamp": null } }]

7

Creating other types of requests

Topics:

Creating a resource instance

To create a resource instance, use the following request components:

Table 35. Components of a POST request

Component	Details

Headers	`Accept: application/json`
	`Content-Type: application/json`
	`Authorization: <auth_cookie>`

Operation	`POST`
URI pattern	`/api/rest/<resource_type>` where *<resource_type>* is the resource type of the instance that you want to create.

Body	`{`
	`"argument1":<value>,`
	`"argument2":<value>,`
	`"argument3":<value>`
	`.`
	`.`
	`.`
	`}`
	where the comma-separated list contains all required arguments and any optional arguments. Use double quotes around a `string`, `date-time`, or `ip-address` value.

NOTE: The unique identifier of the new instance is generated automatically by the server.

If the request succeeds, it returns a `201 Created` HTTP status code and a create response body. This resource contains the `id` attribute for the new resource. If the request does not succeed, the server returns a `4xx` or `5xx` HTTP status code and a message entity in the response body.

Creating a local_user

The following request creates an instance of the `local_user` resource type.

Table 36. Example of a POST request

Component	Details

Headers	`Accept: application/json`
	`Content-Type: application/json`
	`Authorization: <auth_cookie>`

Request	`POST https://1.2.3.4/api/rest/local_user`

Request body	`{`
	`  "name": "User1",`
	`  "password": "myPassword",`
	`  "role_id": "1"`
	`}`

Response body	`{`
	`  "id": "4"`
	`}`

Modifying a resource instance

To modify a resource instance, use the following request components:

Table 37. Components of a PATCH request

Component	Details

Headers	`Accept: application/json`
	`Content-Type: application/json`
	`Authorization: <auth_cookie>`

Operation	`PATCH`
URI pattern	`/api/rest/<resource_type>/<id>` where: • `<resource_type>` is the resource type of the instance that you want to modify. • `<id>` is the unique identifier of the instance that you want to modify. There are three varieties of `PATCH` operations: • `"<attribute>": "new_value"` replaces the existing value of the specified attribute with the new value. • `"add_<attribute>": ["<new_value1>", "<new_value2>", ...]` This only applies to list-type attributes, and adds the specified values to an existing list of values. • `"remove_<attribute>": ["<existing value1>", "<existing_value2>"...]` This only applies to list-type attributes, and removes the specified values from an existing list of values. The Reference content documents which varieties of `PATCH` are supported for each resource type. For additional functionality, such as making the request an asynchronous request, you can append one or more request parameters to the URI.

Body	`{`
	`  "argument1":<value>,`
	`  "argument2":<value>,`
	`  .`
	`  .`

Table 37. Components of a PATCH request (continued)

Component	Details
	. } where the comma-separated list contains all required arguments and any optional arguments. Use double quotes around a `string`, `date-time`, or `ip-address` value.

If the request succeeds, it returns a `204 No Content` HTTP status code and an empty response body. If the request does not succeed, the server returns a 4xx or 5xx HTTP status code in the response header and a message entity in the response body.

Examples

Replacing existing values in a volume group

The following request modifies the `name` and `description` values for the `volume_group` resource instance that has an `id` of `ad09bfa8-f8d8-41b5-96a9-c15c9ebdf214`:

Table 38. Example 1 of a PATCH request

Component	Details

Headers	`Accept: application/json`
	`Content-Type: application/json`
	`Authorization: <auth_cookie>`

Request	`PATCH https://1.2.3.4/api/rest/volume_group/ad09bfa8-f8d8-41b5-96a9-c15c9ebdf214`

Request body	`{`
	`"name":"Storage resources for Marketing",`
	`"description":"Volumes for storing competitive analaysis information"`
	`}`

Response body	Empty.

Adding additional volumes to a volume group

The following request adds two volumes to the `volume_group` resource instance that has an `id` of `ad09bfa8-f8d8-41b5-96a9-c15c9ebdf214`:

Table 39. Example 2 of a PATCH request

Component	Details

Headers	`Accept: application/json`
	`Content-Type: application/json`
	`Authorization: <auth_cookie>`

Request	`PATCH https://1.2.3.4/api/rest/volume_group/ad09bfa8-f8d8-41b5-96a9-c15c9ebdf214`
Request body	`{` `"add_volume_ids":["2b2e4948-5f23-495e-8e72-e26c5734c83f",`

Table 39. Example 2 of a PATCH request (continued)

Component	Details
	<pre> "6eaaa9bd-70e7-4b5f-9aca-fd2731d3c176"] }</pre>
Response body	Empty.

Removing volumes from a volume group

The following request removes two volumes from the `volume_group` resource instance that has an `id` of `ad09bfa8-f8d8-41b5-96a9-c15c9ebdf214`:

Table 40. Example 3 of a PATCH request

Component	Details

Headers	<pre>Accept: application/json
	Content-Type: application/json
	Authorization: <auth_cookie></pre>

Request	<pre>PATCH https://1.2.3.4/api/rest/volume_group/ad09bfa8-f8d8-41b5-96a9-c15c9ebdf214</pre>

Request body	<pre>{
	  "remove_volume_ids":["2b2e4948-5f23-495e-8e72-e26c5734c83f",
	  "6eaaa9bd-70e7-4b5f-9aca-fd2731d3c176"]
	}</pre>

Response body	Empty.

Deleting a resource instance

To delete a resource instance, use the following request components:

Table 41. Example of a resource instance delete operation

Component	Details

Headers	<pre>Accept: application/json
	Authorization: <auth_cookie></pre> If a resource type has request arguments for the `DELETE` operation, you must also use the following header: <pre>Content-Type: application/json</pre>

Operation	`DELETE`
URI pattern	<pre>/api/rest/<resource_type>/<id></pre> where: • `<resource_type>` is the resource type of the instance that you want to delete. • `<id>` is the unique identifier of the instance that you want to delete.

Table 41. Example of a resource instance delete operation (continued)

Component	Details
	For additional functionality, such as making the request an asynchronous request, you can append one or more request parameters to the URI.
Body	For most resource types, the body of a `DELETE` request is empty. However, if a resource type has request arguments for the `DELETE` operation, they are passed as a comma-separated list of standard JSON name:value pairs.

If the request succeeds, it returns a `204 No Content` HTTP status code and an empty response body. If the request does not succeed, the server returns a 4xx or 5xx HTTP status code in the response header and a message entity in the response body.

Deleting a volume_group instance

The following request deletes the `volume_group` instance that has an `id` of `23772434-6373-4748-aa1a-f4197475812a`:

Table 42. Example of a volume group delete operation

Component	Details

Headers	`Accept: application/json`
	`Content-Type: application/json`
	`Authorization: <auth_cookie>`

Request	`DELETE https://1.2.3.4/api/rest/volume_group/23772434-6373-4748-aa1a-f4197475812a`
Request body	Empty.
Response body	Empty.

Performing an instance-level resource-specific action

Some resource types have operations that let you perform resource-specific actions on resource instances beyond the standard delete and modify actions. For example, you can use the `volume` resource type's `refresh` operation to refresh the contents of a volume from another volume in the same family.

To perform a resource-specific action on a resource instance, use the following request components:

Table 43. Example of an instance-level resource action

Component	Details

Headers	For operations without request arguments:
	`Accept: application/json`
	`Authorization: <auth_cookie>`
	For operations with request arguments:
	`Accept: application/json`
	`Content-Type: application/json`
	`Authorization: <auth_cookie>`

Operation	`POST`

Table 43. Example of an instance-level resource action (continued)

Component	Details
URI pattern	`/api/rest/<resource_type>/<id>/<action_name>` where: • `<resource_type>` is the resource type of the instance for which you want to perform an action. • `<id>` is the unique identifier of the instance for which you want to perform an action. • `<action_name>` is the action that you want to perform. For additional functionality, such as making the request an asynchronous request, you can append one or more request parameters to the URI.
Body	For operations without request arguments: Empty. For operations with input data: `{ "argument1":<value>, "argument2":<value>, . . . }` where the comma-separated list contains all required arguments and any optional arguments. Use double quotes around a `string`, `date-time`, or `ip-address` value.

The success response for an instance-level resource-specific action differs depending on whether the action performed has output data:

- For actions that do not have output data, a successful request returns a `204 No Content` HTTP status code and an empty response body.
- For actions that have output data, a successful request returns a `200 OK` HTTP status code, and the body has the specified out attributes in an instance resource response body.

If the request does not succeed, the server returns a 4xx or 5xx HTTP status code in the response header and a message entity in the response body.

Creating a snapshot of a specified volume

The following request creates a snapshot of the `volume` instance that has an `id` of `a47a6bc8-ad32-4e07-bd51-15adc831dfee`:

Table 44. Example of a snapshot creation request

Component	Details

Headers	`Accept: application/json`
	`Content-Type: application/json`
	`Authorization: <auth_cookie>`

Request	`POST /api/rest/volume/a47a6bc8-ad32-4e07-bd51-15adc831dfee/snapshot`
Request body	(Optional) `{ "description": "DB copy before application upgrade" }`

Table 44. Example of a snapshot creation request (continued)

Component	Details

Response body	<pre>{
	  "id": "57cdb822-490e-4755-b8b3-99bb779b1472"
	}</pre>

Performing a class-level action

Some resource types have operations that let you perform class-level actions. For example, exchanging SSL certificates with another PowerStore appliance.

To perform a class-level action, use the following request components:

Table 45. Components of a class-level action

Component	Details

Headers	For operations without request arguments:
	<pre>Accept: application/json
	Authorization: <auth_cookie></pre>
	For operations with request arguments:
	<pre>Accept: application/json
	Content-Type: application/json
	Authorization: <auth_cookie></pre>

Operation	`POST`
URI pattern	<pre>/api/rest/<resource_type>/<action_name></pre> where: ● `<resource_type>` is the resource type of the class for which you want to perform an action. ● `<action_name>` is the action that you want to perform. For additional functionality, such as making the request an asynchronous request, you can append one or more request parameters to the URI.

Body	For operations without request arguments:
	Empty.
	For operations with input data:
	<pre>{
	  "argument1":<value>,
	  "argument2":<value>,
	  .
	  .
	  .
	}</pre>
	where the comma-separated list contains all required arguments and any optional arguments. Use double quotes around a `string`, `date-time`, or `ip-address` value.

The success response for a class-level action differs depending on whether the action performed has output data:

- For actions that do not have output data, a successful request returns a `204 No Content` HTTP status code and an empty response body.
- For actions that have output data, a successful request returns a `200 OK` HTTP status code, and the body has the specified out attributes in a class response body.

If the request does not succeed, the server returns a 4xx or 5xx HTTP status code in the response header and a message entity in the response body.

Exchanging SSL certificates

The following request exchanges SSL certificates with another PowerStore appliance:

Table 46. Example of a SSL certificate exchange request

Component	Details

Headers	`Accept: application/json`
	`Content-Type: application/json`
	`Authorization: <auth_cookie>`

Request	`POST /api/rest/x509_certificate/exchange`

Request body	`{`
	`  "service": "Management_HTTP",`
	`  "address": "10.244.53.108",`
	`  "port": 8080,`
	`  "username": "user1",`
	`  "password": "myPassword"`
	`}`

Response body	Empty.

Working with asynchronous requests

By default, all REST API requests are synchronous, which means that the client/server connection stays open until the request completes and the response is returned.

Alternatively, you can make any active management request (one that changes the system rather than just querying it) into an asynchronous request by adding URL parameter is`_async=true` to the request URL. Asynchronous requests can be more reliable than synchronous requests. With an asynchronous request, you start a job, and the server returns an associated job resource instance almost immediately. You can query the `job` resource instance when convenient to get the HTTP response code and response body for the request. If you create a synchronous request and the network connection is lost, or the REST client or server goes down while the request is processing, there is no way to obtain the request status when it does eventually complete.

Syntax

As the first parameter on the request URI:

```
?is_async=true
```

As a subsequent parameter on the request URI:

```
&is_async=true
```

Usage

The following considerations apply to asynchronous requests:

- A valid asynchronous request returns a `202 Accepted` HTTP status code and a minimal `job` resource instance in the response body.
- An invalid asynchronous request returns immediately with the appropriate error code in the response header and a message entity in the response body.

To view the status of an asynchronous request, retrieve data for the appropriate `job` resource instance. For example, if an asynchronous `PATCH local_user` user request returns a `job` resource instance with an `id` of `1b6df699-7083-4440-912c-22e2ed89c530`, you can use an instance query to retrieve the asynchronous request data from this `job` resource. Query at least the `state` and `response_body` attributes. If the `state` is `Completed` or `Failed`, the `response_body` contains the error response.

Creating an asynchronous request

The following example uses the `is_async=true` request parameter on a request to modify an `volume_group` instance.

Table 47. Example of an asynchronous request

Request component	Details

Headers	`Accept: application/json`
	`Content-Type: application/json`
	`Authorization: <auth_cookie>`

Request	`PATCH https://1.2.3.4/api/rest/volume_group/e436abe3-d78e-4d1d-a349-349368803c59?is_async=true`

Request body	`{`
	`  "description": "Storage resources for the Finance department"`
	`}`

Response body	`{`
	`  "id": "1b6df699-7083-4440-912c-22e2ed89c530"`
	`}`

Viewing an asynchronous request

The following example shows the `job` instance that is associated with the request that is shown above:

Table 48. Example of the job instance associated with the asynchronous request

Request component	Details

Headers	`Accept: application/json`
	`Content-Type: application/json`
	`Authorization: <auth_cookie>`

Request	`GET https://1.2.3.4/api/rest/job/1b6df699-7083-4440-912c-22e2ed89c530?select=id,description_l10n,state,response_body`
Request body	Empty.

Response body	`{`
	`  "id": "1b6df699-7083-4440-912c-22e2ed89c530",`
	`  "description_l10n": "Modify a volume group.",`
	`  "state": "COMPLETED",`
	`  "response_body": null`
	`}`

8

Reference content

This section describes the resource types, methods, and attributes in the PowerStore Management REST API, along with other API information such as datatypes and enumerations.

In addition to the PowerStore REST API Reference Guide on Dell Support, reference materials are also available on the appliance in several formats:

- Swagger UI—`https://<cluster management ip address>/swaggerui`
- JSON—`https://<cluster management ip address>/api/rest/openapi.json`