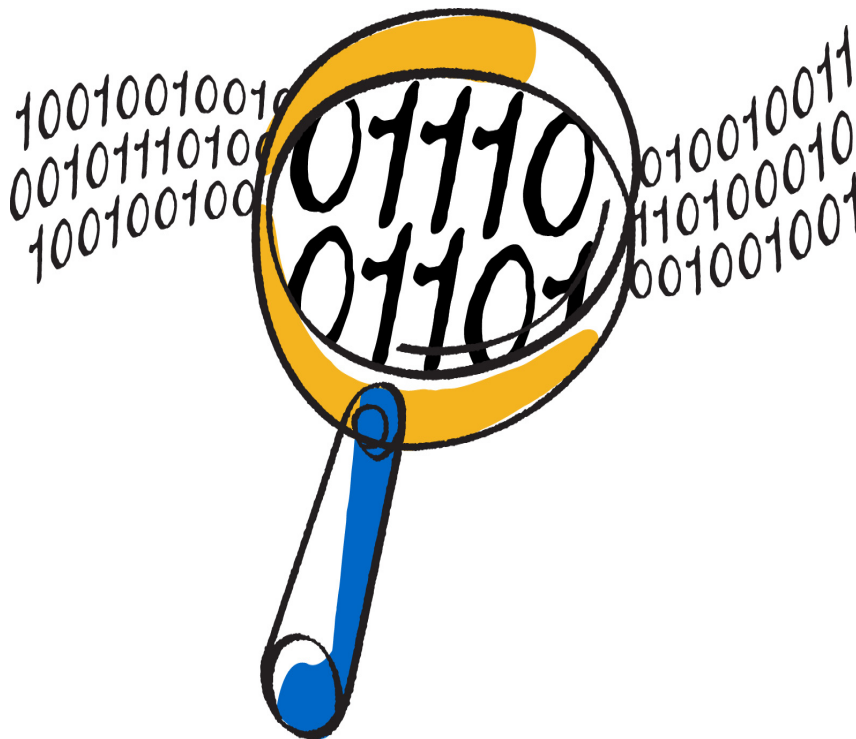




Mars OS REST API Developer's Guide for FlashRay™ Systems

Release 1.0.2



NetApp, Inc.
495 East Java Drive
Sunnyvale, CA 94089
U.S.A.

Telephone: +1 (408) 822-6000
Fax: +1 (408) 822-4501
Support telephone: +1 (888) 463-8277
Web: www.netapp.com
Feedback: doccomments@netapp.com

Part number: 215-09742_A0
Wednesday, February 11, 2015

Contents

Mars OS REST API Overview	4
Mars OS REST APIs	5
Why REST API architecture?	5
Security	6
How the resources are accessed	8
URI template	9
Basic URI	9
Sub-resource URI	9
Instance URI	9
Action URI	9
Bulk URI	10
Query URI	10
URI length	10
Features and examples	11
What can you do with Mars OS REST APIs?	12
Login and session management	13
Login	13
Login message exchange	13
Session management	15
Interaction with HTTP	17
HTTP Request methods	17
HTTP Request headers	17
HTTP Request body	19
HTTP Response status	20

HTTP Response headers	21
Response body	22
Internationalization and localization	24
Sample request with header "Accept-Language:ko" for message in Korean	24
Sample response with error message text in Korean	24
Bulk operation support	25
Bulk creation	25
Bulk action	28
Bulk deletion	29
Versioning	30
Backward compatibility	30
Discovery API	33
Copyright information	34
Trademark information	35
How to send your comments	36

Mars OS REST API Overview

This chapter discusses the following topics:

Mars OS REST APIs	5
How the resources are accessed	8
URI template	9

Mars OS REST APIs

This guide is intended for software developers who want to build applications to monitor or manage a FlashRay system. Developers who understand REpresentational State Transfer (REST) programming conventions can access and manage instances in the Mars OS cluster through its RESTful web service.

Mars OS REST API is the name of the library used to manage FlashRay systems. It is platform- and language-independent, and as long as the API Uniform Resource Identifiers (URIs) and data model schemae are followed, it can be used by any client to manage FlashRay systems.

Why REST API architecture?

RESTful web services enable simple, stateless communications between client and server through HTTP and HTTPS protocols. Clients do not have to know how the server stores or produces the resources, and servers need not be concerned with a client's user interface or state.

The basic design of the RESTful web service includes the data model of the resources through Uniform Resource Identifier (URI) templates and the Uniform Resource Locator (URL) links that web clients use to access resources. The URIs or URLs identify the Internet path of a web resource, which allows interaction with representations of the resources through unique web-based names using specific protocols. Once identified, the web client can access and do Create/Read/Update/Delete (CRUD) operations on them.

The web service includes a data model of the resources, URI templates (the URL links that web clients use to access resources), HTTP Request and Response message formats, client-server handshakes, caches, security, and the RESTful web service framework.

Uniform interface

A uniform interface has the following advantages:

- Resources are manipulated only through representations (using CRUD operations through HTTP methods).
- Individual resources are identified in requests by using URIs to locate the web-manageable resources, which have hierarchy and sub-resources through the URI structure. See [URI template on page 9](#).

Examples of resources for Mars OS include the cluster, LUNs and igroups. URIs, body contents, query parameters, request headers, and response codes using Hypertext As the Engine of Application State (HATEAOS) are self-descriptive and predictable.

Statelessness

Statelessness has the following advantages:

- Except for API login sessions, RESTful APIs do not require servers to maintain client states across multiple HTTP requests (assuming a state spans multiple requests). All information needed to fulfill a request is simply available.
- When clients make requests over HTTP to the FlashRay system, they exchange representations of the state of FlashRay objects. These representations are passed back and forth with the explicit assumption that neither party need know anything about the objects, other than what is presented in a single request or response.

Safe and idempotent operations

Resources remain in the same state, regardless of how many times clients access the same resource instance.

Other advantages

Other advantages of the REST API architectural style include compatible and adaptive version control, secure services, internationalization support, client library and documentation support.

Security

Mars OS has an embedded web server that provides RESTful web services, and this release supports HTTPS as well as HTTP calls to Mars OS REST APIs. If HTTPS is used, web clients and services communicate without leaking confidential information to third parties.

RBAC support

Mars OS supports role-based access control for API requests, and returns 403 Forbidden for unauthorized requests.

Session management

The Mars OS Web Service requires a REST API client to log in to a FlashRay system with the same user name and password as the credentials used for the command line interface.

After login and authentication, a session token is created for the client to access the service for subsequent API calls. Without authentication or a valid session token, the client will be denied access to the service. If a username and password or session is invalid, the client request will be rejected by the web service with a `401 Unauthorized` error.

Sessions have maximum idle time. A session will expire and be terminated if there is no access to it beyond the maximum idle time – but there is an API for users to get session information and explicitly keep the session alive.

How the resources are accessed

Data access and retrieval is done using HTTP Request and Response message formats, client-server handshakes, cache and security services, and the RESTful web service framework. Web clients connect to the server through port 80 (for HTTP) or 443 (for HTTPS), using a URL like `https://<cluster-ip>/webmapi`.

Self-signed certificates are used for SSL sessions, so an API client must accept the SSL certificate during the SSL handshake to use the REST APIs.

How data is exchanged

The REST API issues calls through HTTP Request and Response. The HTTP message bodies carry data objects in readable strings in the default format, JSON (JavaScript Object Notation). For example, clients might use `HTTP Request Accept` headers to specify the accepting data format — `Accept: application/json`.

RESTful clients can also use the third party libraries to do JSON parsing and formatting, such as the Java-based Jackson JSON processor. The JSON web site lists many others.

Methods

Clients execute CRUD (Create/Read/Update/Delete) operations through HTTP methods (GET/POST/PUT/DELETE) and URIs. The four standard HTTP methods are provided by the web server, and the client uses them to manipulate the resource representations. Responses are wrapped with HTTP Response status, context, data, or message properties.

- GET is used for all show commands. This method gets same result, regardless of the number of times clients request the same resource instance. Once the URI of an object is known, users can issue HTTP GET requests to retrieve the current representation of the object.
- POST is used to used to create, modify, or update new objects and event notifications.
- PUT is used to create, modify, or update existing objects. This method keeps the resource in same state, regardless of the number of times clients create the same resource instance.
- DELETE is used to delete objects, but makes no resource state change after resource instances have been deleted.

URI template

URIs are the only way to identify and reference the web resources that are the objects in the Mars OS system. The URI template was designed following REST API URI conventions:

- Top level resources map to the categories of Mars OS system objects.
- Sub-resources map to the objects inside of each category.
- Actions are the special operations on the resources, which cannot be simply applied with CRUD.
- Bulk means to apply the same operation on a number of instances of the same resource.
- Query string identifies either extra conditions of the instances (such as sorting, filtering, listing) or mentions extra conditions of operations. Hyphens are used as delimiters to improve readability.

This pragmatic solution makes the published API easy to maintain and use.

Basic URI

```
/webmapi/{resource-path}
```

Example: /webmapi/luns

Sub-resource URI

```
/webmapi/{resource-path}/{sub-resource path}
```

Example: /webmapi/luns/maps

Instance URI

```
/webmapi/{resource-path}/{id}
```

Example: /webmapi/luns/test-lun-1

Action URI

Action without instance ID

```
/webmapi/{resource-path}/action/{action-operation}
```

Example: /webmapi/luns/action/map

Action with instance ID

```
/webmapi/{resource-path}/{id}/action/{action-operation}
```

Example: /webmapi/luns/test-lun-1/action/online

NOTE: The segment of "action/{action-operation}" must be the last one in the URI.

Bulk URI

Bulk operation without IDs

`/webmapi/{resource-path}/bulk`

Example: `/webmapi/luns/bulk`

Bulk operation with IDs

`/webmapi/{resource-path}/bulk?ids={id1},{id2},..&force=true`

Example: `/webmapi/luns/bulk?ids=test-lun-1,test-lun-2&force=true`

Query URI

`/webmapi/{resource-path}?{query-string}`

Example: `/webmapi/luns/{id}? force=true`

URI length

Most HTTP status URIs have a maximum length (4Kb). Both web client and server consider a URI length equal to or greater than 4,096 as an invalid URI. All URI strings are URL-encoded.

Features and examples

This chapter discusses the following topics:

What can you do with Mars OS REST APIs?	12
Login and session management	13
Interaction with HTTP	17
Internationalization and localization	24
Bulk operation support	25
Versioning	30
Discovery API	33

What can you do with Mars OS REST APIs?

The feature descriptions and examples in this chapter are provided to help you to understand how you can use the Mars OS REST APIs to do tasks like the following:

- Discover the capabilities and configuration of a FlashRay system
- Monitor the performance and health of a FlashRay system
- Configure the settings of the FlashRay system to fit into your data center
- Provision new storage, or reconfigure existing storage within the cluster, making it available to clients

Login and session management

Login

Mars OS REST API clients must log in to the REST API service on Mars OS and get a session token from it for continuous access. Login requests with valid usernames and base64-encoded passwords are sent through `POST /webmapi/sessions`:

```
Request:
POST https://webmapi.example.com/webmapi/sessions
Accept: application/json
x-webmapi-version: 1.0.1
. . .
{"username": "myusername, "password": "bWFYc19zZXNzaW9uXzE0OTAyMzQ="}

Response:
200 OK
x-webmapi-version: 1.0.1
x-webmapi-authorization: YWRtaW46dGVzdHBhc3N3b3Jk=
Content-Type: application/json
. . .
{
  . . .
  "results": "YWRtaW46dGVzdHBhc3N3b3Jk="
}
```

To log out and terminate a REST API session, the client calls URI `DELETE /webmapi/sessions`:

```
Request:
DELETE https://webmapi.example.com/webmapi/sessions
x-webmapi-authorization: YWRtaW46dGVzdHBhc3N3b3Jk=
Accept: application/json
x-webmapi-version: 1.0.1

Response:
200 OK
. . .
```

Login message exchange

Authentication/Authorization HTTP Messages Exchange

```
Client->Mars: Discover API version request
GET https://mars.example.com/webmapi/versions
Header
  User-Agent: any client application
  Accept: application/json
```

14 | Features and examples

```
Content-type: application/json
x-webmapi-version: *
.....
Mars->Client: Respond with version and login url
Header
  HTTP/1.1 200 OK
  Content-Type: application/json
  x-webmapi-version: 1.0.1
Body
  {"webmapi-version": "1.0.1",
   "webmapi-compatible-versions": ["1.0.1", "1.0"]}
.....
Client->Mars: Any request without authentication
GET https://mars.example.com/webmapi/luns
Header
  User-Agent: any client application
  Accept: application/json
  Content-type: application/json
  x-webmapi-version: 1.0.1
.....
Mars->Client: Respond with challenge response
Header
  HTTP/1.1 401 Unauthorized
  Content-Length: 0
  x-webmapi-version: 1.0.1
  . . .
.....
Client->Mars: Send login POST request
POST https://mars.example.com/webmapi/sessions
Header
  User-Agent: any client application
  Accept: application/json
  Content-type: application/json
  x-webmapi-version: 1.0.1
Body
  '{"username":"admin", "password":"bWFYc19zZXNzaW9uXzE0OTAyMzQ="}'
.....
Mars->Client: Respond with OK status and have session token in the header "x-
webmapi-authorization"
Header
  HTTP/1.1 201 Created
  Authorization: NjVhNjUyNTctZTVmNS00NGY5LWJkZDMtMmViMWQ2MzYwZWU0
  x-webmapi-version: 1.0.1
Body
  {"status":
```

```

{"responseTime":1398131428,"statusCode":"SUCCESS"},"result":
"NjVhNjUyNTctZTVmNS00NGY5LWJkZDMtMmViMWQ2MzYwZWU0"}
.....
Client->Mars: Send subsequent request with session token in the header "x-
webmapi-authorization"
GET https://mars.example.com/webmapi/luns
Header
    User-Agent: any client application
    Accept: application/json
    Content-type: application/json
    x-webmapi-authorization: NjVhNjUyNTctZTVmNS00NGY5LWJkZDMtMmViMWQ2MzYwZWU0
    x-webmapi-version: 1.0.1
.....
Mars->Client: Respond OK status and results in the message body
Header
    HTTP/1.1 200 OK
    Content-Type: application/json
    x-webmapi-version: 1.0.1
Body
    {
        ...
        "results: { ...}
    }

```

Session management

Get session information

```

Request:
curl -v -H "Content-type: application/json" \
-H "x-webmapi-authorization: OGUyNDVjNTMtYTJkMS00OWVjLWFmN2UtZTMwYjY5YmRkZGQ1" \
-H "x-webmapi-version: 1.0.1" -H "x-webmapi-clientid: any client id" \
-H "x-webmapi-requestid: request123" \
-H "x-webmapi-useragent: webmapi" \
-X GET "http://192.168.13.27/webmapi/sessions"

Response:
HTTP/1.1 200 OK
Date: Tue, 01 Apr 2014 01:06:01 GMT
...
{"status":{"responseTime":1396314361,"statusCode":"SUCCESS"},
"result":{"username":"admin","last-update-time":1396313842,"time-to-live":81}}

```

16 | Features and examples

Keep Alive (Renew) a session

```
Request:
curl -v -H "Content-type: application/json" \
-H "x-webmapi-authorization: OGUyNDVjNTMtYTJkMS00OWVjLWFmN2UtZTMwYjY5YmRkZGQ1" \
-H "x-webmapi-version: 1.0.1" -H "x-webmapi-clientid: any client id" \
-H "x-webmapi-requestid: request123" \
-H "x-webmapi-useragent: webmapi" \
-X PUT "http://192.168.13.27/webmapi/sessions"

Response: the last update time gets updated and ttl is reset
HTTP/1.1 200 OK
Date: Tue, 01 Apr 2014 01:08:27 GMT
. . .
{"status":{"responseTime":1396314507,"statusCode":"SUCCESS"}}
```

Send no-update of session

Send no-update of session last update time for a specified request through header `x-webmapi-session-noupdate`.

```
Request:
curl -v -H "Content-type: application/json" \
-H "x-webmapi-authorization: OGUyNDVjNTMtYTJkMS00OWVjLWFmN2UtZTMwYjY5YmRkZGQ1" \
-H "x-webmapi-session-noupdate: true" \
-H "x-webmapi-version: 1.0.1" -H "x-webmapi-clientid: any client id" \
-H "x-webmapi-requestid: request123" \
-H "x-webmapi-useragent: webmapi" \
-X GET "http://192.168.13.27/webmapi/luns"

Response: the last update time gets updated and ttl is reset
HTTP/1.1 200 OK
Date: Tue, 01 Apr 2014 01:08:27 GMT
. . .
```

Interaction with HTTP

HTTP Request methods

The HTTP methods to use in FlashRay operations are listed in the following table:

Request Method	CRUD Operation	URI Example
GET <code>/ {resource}</code> <code>[?query-string]</code>	Read instances from the resource or get subset of the instances from query	GET <code>/luns</code> , GET <code>/luns?filter=name(eq)testlun</code>
GET <code>/ {resource}/{id}</code>	Read instance by resource instance id	GET <code>/luns/109923-2342</code>
POST <code>/ {resource}</code>	Create instance for the resource	POST <code>/luns</code>
PUT <code>/ {resource}</code>	Update instance with specified attributes in the body	PUT <code>/luns</code>
PUT <code>/ {resource}/{id}</code> <code>/action/ {action-name}</code>	Update instance by executing action	PUT <code>/luns/109923-2342/action/online</code> ; PUT <code>/disk/109-00-001/action/zerodisk</code>
DELETE <code>/ {resource}/{id}</code> <code>[?query-string]</code>	Delete instance by id with extra options in the query string	DELETE <code>/luns/109923-2342</code> ; DELETE <code>/luns/109923-2342?force=true</code>

HTTP Request headers

The HTTP headers to use when sending requests to a Mars OS system are listed in the following table:

Header	Required?	Description
Accept: application/json	Optional	Indicates the kinds of responses your client is able to receive. Valid values are application/json. If you don't specify a value, the Mars APIs use application/json.

Header	Required?	Description
Accept-Language: en-us	Optional	Indicates the client's preferred language for text messages. The Mars APIs use the first language from the list that is a supported Mars OS language. If there's no matching language, or you don't supply a value, text messages are returned in US English.
Content-Type: application/json	Optional	Indicates how the data in the body of a request is formatted. The format must be <code>application/json</code> . (The default is <code>application/json</code> .)

REST API headers

Header	Required?	Description
x-webmapi-authorization: bWFyc19zZXNzaW9uXzE0OTAyMzQ=	Required	The authorization key returned by a previous call to the session creation API. Requests other than discovery or session creation requests return <code>401 Unauthorized</code> when this header is absent.
x-webmapi-useragent: any user agent	Required	An arbitrary identifier for your client. Mars OS APIs return <code>400 Bad Request</code> when this header is absent
x-webmapi-version: 1.0.1	Required	The API version of the client requests and anticipated responses. Mars OS APIs return <code>400 Bad Request</code> when this header is absent.

Header	Required?	Description
x-webmapi-clientid: any client id	Optional	An arbitrary identifier used by your client. This identifier is not used by the Mars APIs, but is included in replies.
x-webmapi-requestid: 10010	Optional	An arbitrary identifier used by your client for this particular request. This identifier is not used by the Mars APIs, but is included in the reply for this API request.
x-webmapi-session: nouupdate	Optional	No update for the session last update time for this request.

HTTP Request body

POST/PUT requests can have the API input object in the body as plain text with JSON format according to the request header `Content-Type` definition. GET and DELETE requests do not have body.

Example of POST request in cURL command

```
curl -v \
  -H "Content-type: application/json" \
  -H "x-webmapi-authorization: bWFyc19zZXNzaW9uXzE0OTAyMzQ=" \
  -H "x-webmapi-version: 1.0.1" \
  -H "x-webmapi-useragent: webmapi" \
  -H "x-webmapi-clientid: any client id" \
  -H "x-webmapi-requestid: 123" \
  -X POST -d '{"name":"testlun_1","size":"505m"}' \
  "http://192.168.13.27/webmapi/luns"
```

Example of GET request in cURL command

```
curl -v \
  -H "Content-type: application/json" \
  -H "x-webmapi-authorization: bWFyc19zZXNzaW9uXzE0OTAyMzQ=" \
  -H "x-webmapi-version: 1.0.1" \
  -H "x-webmapi-useragent: webmapi" \
  -H "x-webmapi-clientid: any client id" \
```

```
-H "x-webmapi-requestid: 123" \
-X GET \
"http://192.168.13.27/webmapi/luns"
```

HTTP Response status

The Mars OS Web Service uses only HTTP status code to report any issue at the HTTP layer. For any REST API call failure, the operating system may respond with HTTP/1.1 200 OK and describe the error in the HTTP message body.

HTTP status code

Status code	Status name	Trigger condition	Recommended action
100	Continue	Server return multiple responses	Client keeps accept response
200	OK	Request is valid and executed. The response of execution result is in the response body.	Process response message
201	Created	Request is valid and a new instance is created to handle it. The new instance is specified by the URL in the Location header.	Access the URL to retrieve the new instance
202	Accepted	Request is accepted and a task is created to handle it. The task information is in the response body.	Client sends for asynchronous request
400	Bad Request	Request is rejected for malformed request body, incomplete, or otherwise invalid.	Reconstruct proper HTTP request
401	Unauthorized	Request does not authenticate or session time out when accessing resources requiring authorization.	Authenticate from client
403	Forbidden	Request does not have authority to access the resource. It applies to RBAC when authenticated users try to access a resource beyond their permissions.	Re-authenticate with proper user privilege

Status code	Status name	Trigger condition	Recommended action
404	Not Found	Request does not have proper URL, or it accesses a non-existent resource. This error applies to the endpoint resource service not found, but does not apply to the individual instances of the resource not found.	Reconstruct proper HTTP URL
405	Method Not Allowed	HTTP method specified in the request is not supported for this object.	Reconstruct proper HTTP URL or use proper HTTP method
500	Internal Server Error	Request is received but server cannot proceed because of an internal error.	Restart Mars Web Server

HTTP Response headers

Header	Description
Content-Type: application/json	Header indicates the content format of the response; should be application/json.
Authorization: {\$session-token}	Header indicates the login request is successful and responds to the authorization command with the session token.
Content-Language: en-us	Header indicates language used for the response's content text message.
x-webmapi-version: 1.0.1	Header indicates the server used REST API version.
x-webmapi-clientid: any client id	Header echos back the client ID from the request header.
x-webmapi-requestid: 123	Header echoes back the request id from the request header.

Response body

The REST API returns data in the body of the HTTP response. Each REST API return response has following fields:

- Status — contains API execution status (SUCCESS, ERROR, PARTIALSUCCESS) and response time.
- Result — contains API execution output when status is SUCCESS or PARTIALSUCCESS. If an API does not have output, the response does not have a result field.
- Error — contains an API execution error when status is SUCCESS or PARTIALSUCCESS. Each error includes an error text message in the text field, a key of the error (error code) in the key field, and the language field tells which locale is used. If a known problem caused the error, the causeMessage will display it.

```
Success with data:{
  "status":{
    "responseTime":1386895667,
    "statusCode":"SUCCESS"
  },
  "result":{
    "instancesTotal":10,
    "instancesCount":10,
    "instancesLimit":100,
    "instances":[
      {
        "uuid":"uuidc350",
        "name":"testigroup_0",
        "ostype":"windows",
        "initiators":[
          "20:00:00:00:00:00:00:00",
          "20:00:00:00:00:00:00:01"
        ],
        "lun-maps":[
          {
            "igroup-name":"testlun0",
            "lu-name":"testigroup_0",
            "lun":0
          },
          {
            "igroup-name":"testlun01",
            "lu-name":"testigroup_0",
            "lun":1
          }
        ]
      }
    ]
  }
}
```

```

    }
  ]
}
}

```

Success without data:

```

{
  "status":{
    "responseTime":1386895881,
    "statusCode":"SUCCESS"
  }
}

```

Error with error message:

```

{"status":{"responseTime":1386896018,"statusCode":"ERROR"},
 "error":{"errorMessage":{"text":"UID service communication layer
  encountered error","key":
"WEBUI_UID_ERROR_COMMUNICATION","language":"en_US"},
  "causeMessage":"Fail to open socket","key":"0123","args":[],
  "language":"en-US"}}}

```

NOTE: Digits in the key field refer to Mars OS error codes.

Internationalization and localization

The Mars OS Web Service supports internationalized messages. Any user-visible message is wrapped by an internationalization message, which includes `messageText`, `messageKey` and `MessageArgs`. Based on API client request `Accept-Language` HTTP header, the Web Service can return the localized message in the `messageText` field. If the client-specified locale is not supported by Mars OS, the `messageText` would be in the default locale, which is English (en-US).

Sample request with header "Accept-Language:ko" for message in Korean

```
curl -v -H "x-webmapi-authorization: XYZ" -H "Content-type: application/json" -H "x-webmapi-version: 1.0.1" -H "x-webmapi-clientid: anyclientid" -H "x-webmapi-requestid: request123" -H "x-webmapi-useragent: webmapi" -H "Accept-language:ko" -X POST -d '{"name":"testlun_1","size":"505m"}' "http://localhost:8080/webmapi/luns"
```

Sample response with error message text in Korean

```
{
  "status":{
    "responseTime":1414427142,
    "statusCode":"ERROR"
  },
  "error":{
    "errorMessage":{
      "text":"무단 액세스",
      "key":"WEBUI_WEBAPI_ERROR_UNAUTHORIZED_ACCESS",
      "args":[

    ],
    "language":"ko"
  }
}
```

Bulk operation support

A bulk request calls the same REST API multiple times with different input parameters, but puts them into a single request. This helps reduce the overhead of opening and closing HTTP connections and network latency, but it requires handling more complex requests and responses.

After receiving each bulk request, the web service iterates through each call's input, then returns a single response with results from each call. To avoid overloading the web server with single requests and forcing the client to wait too long, the bulk operation usually has the following size limitations:

- A bulk request can have a maximum of 100 API calls.
- A bulk request can have a maximum of 100,000 characters.

Bulk operations should not be used for long-running REST API calls.

NOTE: Bulk requests apply only to CRUD operations.

Bulk creation

A bulk request uses `bulk` in the URL path to identify the bulk operation, and has `bulk` type in the body to contain the multiple input arguments.

```
curl -v -H "Content-type: application/json" -H "x-webmapi-authorization:
bWFYc19zZXNzaW9uXzE0OTAyMzQ="
-H "x-webmapi-version: 1.0.1" \
-H "x-webmapi-clientid: any client id" -H "x-webmapi-requestid: request123" -H
"x-webmapi-useragent: webmapi" \
-X POST -d '{"itemsCount":2, "items":[{"name":"test-lun-1","size":"505m"},
{"name":"test-lun-2","size":"505m"}]}' \
"http://192.168.13.27/webmapi/luns/bulk"
```

A bulk response follows the same REST API response format, but the result uses `BulkResultType` with lists of successful and failed items.

- When all calls are successful, the `statusCode` is `SUCCESS`, and the result shows a list of created items with a name and new instance identifier:

```
{
  "status":{
    "responseTime":1377139230602,
    "statusCode":"SUCCESS"
  },
  "result":{
    "itemResults":[
      {
```

```

        "id": "test-lun-1",
        "data": "12934242"
    },
    {
        "name": "test-lun-2",
        "data": "129323442"
    }
],
"itemResultsCount": 2,
"itemErrorCount": 0
}
}

```

- When calls are partially successful, the `statusCode` is `PARTIALSUCCESS`, and the result shows lists of created and failed items with names and individual error information:

```

{"status": {"responseTime": 1377139230602, "statusCode": "PARTIALSUCCESS"},
"result": {
  "itemResults": [{"id": "test-lun-1", "data": "12934242"}],
  "itemErrors": [{"id": "test-lun-2", "error": {"errorMessage":
{"text": "Lun already exists", "key": "10007", "args": [], "language": "en_US"}}}],
  "itemResultCount": 1,
  "itemErrorCount": 1
}
}

```

- When all calls have failed, the `statusCode` is `ERROR` and the result shows a list of failed items with names and individual error information:

```

{
  "status": {
    "responseTime": 1377139230602,
    "statusCode": "ERROR"
  },
  "result": {
    "itemsErrors": [
      {
        "id": "test-lun-1",
        "error": {
          "errorMessage": {
            "text": "Lun already exists",
            "key": "10007",
            "args": [
              ],
            "language": "en_US"
          }
        }
      }
    ]
  }
}

```

```

    }
  },
  {
    "id":"test-lun-2",
    "error":{
      "errorMessage":{
        "text":"Lun already exists",
        "key":"10007",
        "args":[

        ],
        "language":"en_US"
      }
    }
  }
],
"itemResultCount":0,
"itemErrorCount":2
}
}

```

When bulk requests cannot be processed, the `statusCode` is `ERROR`, and the error information is displayed:

```

{
  "status":{
    "responseTime":1377139230602,
    "statusCode":"ERROR"
  },
  "error":{
    "errorMessage":{
      "text":"Request has invalid field: nam",
      "key":"WEBUI_WEBAPI_ERROR_INVALID_FIELD",
      "args":[
        "nam"
      ],
      "language":"en_US"
    }
  }
}

```

Bulk action

A bulk action request applies the action to the instances identified by ID but without input arguments. Use `bulk` in the URL path to identify bulk operation, and put the bulk of IDs as query strings in the URL with a comma as the ID separator.

```
PUT /luns/bulk/action/online?ids=test-lun-1,test-lun-2
```

For an action with input arguments, use `bulk` in the URL path to identify bulk operations, and have `bulk` type in the body to contain the multiple input arguments.

```
curl -v -H "Content-type: application/json" -H
"x-webmapi-authorization: bWFYcl9zZXNzaW9uXzE0OTAyMzQ=" -H
"x-webmapi-version: 1.0.1" \ -H
"x-webmapi-clientid: any client id" -H
"x-webmapi-requestid: request123" -H
"x-webmapi-useragent: webmapi" \
-X PUT -d '{"itemsCount":2,
"item":[{"name":"test-lun-1","new-name":"newtest-lun-1"},
{"name":"test-lun-2","new-name":"newtest-lun-2"}]}' \
"http://192.168.13.27/webmapi/luns/bulk/action/rename"
```

A bulk action response follows the same REST API response format, but the result uses `BulkResultType` with a list of successful items and list of failed items.

Like partially successful examples, the `statusCode` is `PARTIALSUCCESS`, and the result shows lists of created and failed items with names and individual error information:

```
{
  "status":{
    "responseTime":1377139230602,
    "statusCode":"PARTIALSUCCESS"
  },
  "result":{
    "itemResults":[
      {
        "id":"test-lun-1",
        "data":true
      }
    ],
    "itemErrors":[
      {
        "id":"test-lun-2",
        "error":{
          "errorMessage":{
            "text":"Lun cannot rename for the new name already exists",
            "key":"10007",
```

```

        "args": [
            ],
            "language": "en_US"
        }
    }
    ],
    "itemResultCount": 1,
    "itemErrorCount": 1
}
}

```

Bulk deletion

For a bulk deletion request, use `bulk` in the URL path to identify the bulk operation, and put the bulk of the IDs as query strings in the URL with a comma as the ID separator.

```
DELETE /resources/bulk?ids=id1,id2,...&other-arguments
```

The list of ids can be up to 100 and the ids after 100 will be trimmed and also the list is also limited by the maximum length of URL, which is 2048 characters.

For example:

```
DELETE /luns/bulk?ids=test-lun-1,test-lun-2&force=true
```

For a bulk deletion response, follow the same REST API response format, but the result uses `BulkResultType` with lists of successful and failed items.

Like partially successful examples, the `statusCode` is `PARTIALSUCCESS`, and the result shows lists of created and failed items with names and individual error information:

```

{"status":{"responseTime":1377139230602,"statusCode":"PARTIALSUCCESS"},
 "result":{
  {
    "itemsResults": [{"id":"test-lun-1", "data":true}],
    "itemsErrors": [{"id":"test-lun-2", "error":{"errorMessage":
      {"text":"Lun does not exist","key":"10009","args":[],"language":"en_
US"}},
    "itemResultCount": 1,
    "itemErrorCount":1
  }
}
}

```

Versioning

The REST API version applies to the whole API library, not just to individual APIs within the library. The MAJOR, MINOR, and PATCH strings are separated by "." — the PATCH version is optional.

```
Format: MAJOR.MINOR[.PATCH]
```

```
Example
```

The REST API library release is usually aligned with the Mars OS releases and has the same major/minor release numbers. In addition, Mars OS supports a set of compatible REST API versions. See [Discovery API on page 33](#).

A RESTful client can get the Mars OS compatible REST API version through the immutable API `GET /webmap/api/versions`.

Backward compatibility

The REST API library has a long lifespan that builds constantly over an evolving system, making it hard to predict all resources and aspects that might be consumed through the APIs.

Over time, new resources or APIs, or existing APIs with additional features and attributes, will be introduced into the library. Clients might not be updated to the newest library, so REST API supports a set of compatible versions in addition to its library release version.

REST API provides backward compatibility by preserving published APIs in the following ways:

- When new APIs or new attributes are added to existing APIs, clients accessing earlier versions must ignore the updates.
- When an API in the library is deprecated, the API document includes a warning that it has been deprecated, but the release continues to support the API.
- When a breaking change is made to a library, version negotiation is used to serve the correct API representation version to clients running compatible versions.

Version negotiation

The REST API endpoint (Mars OS) supports its release version and a set of compatible versions. Clients can get the version information of the endpoint through the immutable API `GET /webmap/api/versions`.

RESTful clients communicate and negotiate compatible API representation with the endpoint through the API request HTTP header `x-webmap-api-version: {client-webmap-api-version}`.

The REST API endpoint checks each API request version with its supported versions and responds:

- OK 200/Created 201 with released API representation when a client version is same as the endpoint running version.
- OK 200/Created 201 with compatible API representation when a client version is different from the endpoint running version, but in compatible versions.
- Bad Request 400 when a client version is incompatible with the endpoint running version.

Negotiation scenarios

Application/json and LUNs are used as examples.

Client runs same version as REST API endpoint

Client->Endpoint: create a lun with latest API

POST https://mars.example.com/webmapi/luns

x-webmapi-version: 1.2

```
{
  "name": "test-lun",
  "size": 8388608,
  "comments": "testing lun"
}
```

Endpoint->Client: respond successful lun creation

HTTP/1.1 201 Created

x-webmapi-version: 1.2

```
{
  "status": { "responseTime": 1386895667, "statusCode": "SUCCESS" },
  "result": { "uuid": "xyb12345" }
}
```

Client runs compatible version 1.1 while endpoint runs 1.2, but supports 1.1

Client->Endpoint: create a lun with old API without new attribute "comments" and with string for size instead of integer of bytes

POST https://mars.example.com/webmapi/luns

x-webmapi-version: 1.1

```
{
  "name": "test-volumn",
  "size": "8g"
}
```

Endpoint->Client: respond successful lun creation

HTTP/1.1 201 Created

x-webmapi-version: 1.1

```
{
  "status": { "responseTime": 1386895667, "statusCode": "SUCCESS" },
  "result": { "uuid": "xyb12345" }
}
```

32 | Features and examples

```
}
.....

Client runs compatible version 1.1 while endpoint runs 1.2, but supports 1.1,
using deprecating API

Client->Endpoint: change lun name with deprecating API
PUT https://mars.example.com/webmapi/luns/rename
  x-webmapi-version: 1.1

  {"name":"test-lun",
   "new-name":"test-pun-101"}
.....
Endpoint->Client: respond with deprecating warning
HTTP/1.1 200 OK
x-webmapi-version: 1.1

{"status":{"responseTime":1386895667,"statusCode":"DEPRACATE"},
 "error":{"errorMessage":{"text":"API is deprecating","key":
   "WEBUI_API_ERROR_DEPRECATED","language":"en_US",
   "args":["PUT", "/luns/rename"]}}}
}
.....

Client runs incompatible version 1.0 while endpoint runs version 1.2

Client->Endpoint: get lif
GET https://mars.example.com/webmapi/lif
  x-webmapi-version: 1.0
.....
Endpoint->Client: respond bad request for missing header
HTTP/1.1 400 Bad Request
x-webmapi-version: 1.2

{"status":{"responseTime":1391219482,"statusCode":"ERROR"},
 "error":{"errorMessage":{"text":"Incompatible client version:
  x-webmapi-version 1.0","key":"WEBUI_WEBAPI_ERROR_INCOMPATIBLE","args":
  ["x-webmapi-version","1.0"],"language":"en_US"]}}}
.....
```

Discovery API

WebMAPI clients must know where and how to log in. Mars OS uses version API `GET /webmapi/versions` to provide its own supported and compatible API version and login URI. This immutable API, which is supported through all API versions, can be called without a login requirement.

Request:

```
GET https://webmapi.example.com/webmapi/versions
x-webmapi-version: *
```

Response:

```
200 OK
x-webmapi-version: 1.0.1
Content-Type: application/json
. . .
{"webmapi-version":"1.0.1", "
"webmapi-compatible-versions":["1.0.1", "1.0.2"] }
```

Copyright information

Copyright © 1994–2015 NetApp, Inc. All rights reserved. Printed in the U.S.

No part of this document covered by copyright may be reproduced in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an electronic retrieval system—without prior written permission of the copyright owner.

Software derived from copyrighted NetApp material is subject to the following license and disclaimer:

THIS SOFTWARE IS PROVIDED BY NETAPP "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WHICH ARE HEREBY DISCLAIMED. IN NO EVENT SHALL NETAPP BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

NetApp reserves the right to change any products described herein at any time, and without notice. NetApp assumes no responsibility or liability arising from the use of products described herein, except as expressly agreed to in writing by NetApp. The use or purchase of this product does not convey a license under any patent rights, trademark rights, or any other intellectual property rights of NetApp.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.277-7103 (October 1988) and FAR 52-227-19 (June 1987).

Trademark information

NetApp, the NetApp logo, Network Appliance, the Network Appliance logo, Akorri, ApplianceWatch, ASUP, AutoSupport, BalancePoint, BalancePoint Predictor, Bycast, Campaign Express, ComplianceClock, Cryptainer, CryptoShred, CyberSnap, Data Center Fitness, Data ONTAP, DataFabric, DataFort, Decru, Decru DataFort, DenseStak, Engenio, Engenio logo, E-Stack, ExpressPod, FAServer, FastStak, FilerView, Flash Accel, Flash Cache, Flash Pool, FlashRay, FlexCache, FlexClone, FlexPod, FlexScale, FlexShare, FlexSuite, FlexVol, FPolicy, GetSuccessful, gFiler, Go further, faster, Imagine Virtually Anything, Lifetime Key Management, LockVault, Mars, Manage ONTAP, MetroCluster, MultiStore, NearStore, NetCache, NOW (NetApp on the Web), Onaro, OnCommand, ONTAPI, OpenKey, PerformanceStak, RAID-DP, ReplicatorX, SANscreen, SANshare, SANtricity, SecureAdmin, SecureShare, Select, Service Builder, Shadow Tape, Simplicity, Simulate ONTAP, SnapCopy, Snap Creator, SnapDirector, SnapDrive, SnapFilter, SnapIntegrator, SnapLock, SnapManager, SnapMigrator, SnapMirror, SnapMover, SnapProtect, SnapRestore, Snapshot, SnapSuite, SnapValidator, SnapVault, StorageGRID, StoreVault, the StoreVault logo, SyncMirror, Tech OnTap, The evolution of storage, Topio, VelocityStak, vFiler, VFM, Virtual File Manager, VPolicy, WAFL, Web Filer, and XBB are trademarks or registered trademarks of NetApp, Inc. in the United States, other countries, or both.

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. A complete and current list of other IBM trademarks is available on the web at www.ibm.com/legal/copytrade.shtml.

Apple is a registered trademark and QuickTime is a trademark of Apple, Inc. in the United States and/or other countries. Microsoft is a registered trademark and Windows Media is a trademark of Microsoft Corporation in the United States and/or other countries. RealAudio, RealNetworks, RealPlayer, RealSystem, RealText, and RealVideo are registered trademarks and RealMedia, RealProxy, and SureStream are trademarks of RealNetworks, Inc. in the United States and/or other countries.

All other brands or products are trademarks or registered trademarks of their respective holders and should be treated as such.

NetApp, Inc. is a licensee of the CompactFlash and CF Logo trademarks.

NetApp, Inc. NetCache is certified RealSystem compatible.

How to send your comments

You can help us to improve the quality of our documentation by sending us your feedback. Your feedback is important in helping us to provide the most accurate and high-quality information.

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

You can also contact us in the following ways:

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