OnCommand® Workflow Automation 5.1

REST Web Services Primer

November 2019 | 215-14537_A0
doccomments@netapp.com

Contents

Introduction to the Document

This document provides information about OnCommand Workflow Automation (WFA) REST APIs and the document is specifically intended for developers who build RESTful clients.

In addition to this detailed and descriptive programmer's guide, the Reference Manual is available from within WFA. You can use this Reference Manual to look up the resource URLs, request and response formats, links and relations, XML or JSON schema definitions of input and output XMLs or JSONs, and sample HTTP request and response messages.

General

WFA allows external services to access various resources and perform CRUD operations on these resources using a RESTful API. The following functionalities are available through the RESTful APIs:

- Accessing workflow definitions and metadata
- Executing workflows and monitoring jobs (Jobs in this context are instances of workflows)
- Viewing users, roles, and changing passwords
- Executing and testing resource selection filters
- Executing and testing resource finders
- Managing credentials of storage or other data center objects
- Viewing data sources and data source types
- Taking backup and restoring databases
- Exporting and importing of entity using .dar files
- Searching entities using Free text search
- Updating AutoSupport configuration
- Updating LDAP configuration
- Validating WFA configurations
- WFA System version information

For more information about RESTful web services, see the following document:

REST In Practice: Hypermedia and Systems Architecture by Jim Webber et. al.

For testing and validating your clients written in Java, you must refer to the following frameworks:

REST ASSURED

REST CLIENT

NetApp WFA API supports developers who build clients of WFA services (workflow execution), using a RESTful application development style.

WFA REST APIs provide access to resource collections such as workflows, users, filters, and finders, through URI paths. To use a REST API, the client application must make an HTTP request and parse the response. The response includes a header and body, which is similar to all HTTP responses. The body includes an XML or JSON representation of the resource specified by the URI path.

HTTP verbs, such as GET, PUT, POST, and DELETE, can be used on these URIs to perform various CRUD operations on the resources. The links and relations that a particular object supports are provided as a part of the response.

Authentication and authorization

Every web service request must be accompanied by a valid username and password. WFA uses oneway SSL (HTTPS) with HTTP basic authentication (See *RFC 2617*). With HTTP basic authentication, the WFA server sends a “401 Unauthorized” challenge to the connecting client and the client must specify the username and password in response to the challenge. The client can also send the username and password in the original request to avoid this challenge or response cycle.

All users, except guest users, are allowed to execute workflows through the REST API. If a client authenticates with a valid username and password, but the authenticated user does not have the rights to access the specified resource (either because of the role performed by the user or the resource itself restricts the user from accessing the resource), then the WFA server might still send a “403 Forbidden” error response.

Versioning

WFA API currently does not support multiple versions.

Pattern for client API development

Developers who develop a client that uses WFA API must perform the following steps in developing the client. Clients must not use hardcoded URLs. Every time a client wants to use the API for achieving a particular task, the client must use the following procedure:

1. During client development, refer to the *API documentation* of the URL for the root object collection (for example, /rest/workflows) of the object set that the client wants to operate on. (See: WFA object collection for a complete list of root object collections available).

2. Create the following root URL:

```
https://{host:port}/rest/{<wfa-object-collection-
name>}
```

3. Add the required query parameters to the URL. Refer to the quick reference documentation to view the query parameters that are available for the particular object collection.

4. Issue an HTTP GET request on the URL. Ensure that the ‘Accept’ and ‘Authorization’ headers are properly filled. The ‘Accept’ header must be ‘application/xml’ for XML media type, and ‘application/json’ for JSON media type, and the ‘Authorization’ header must include the base64 encoded credentials. WFA uses basic authentication. It is best to use HTTPS instead of HTTP.

5. WFA supports both application/xml and application/json as the content type for all APIs.

6. If the HTTP code is 200 OK, extract the response body.

7. If there is more than one item in the collection, the client must extract from the response, the specific item on which the client wants to perform an operation.

8. During client development, refer to the documentation for the names of the ‘rel’ attribute that specify the operation that the client wants to perform on the object. Extract the link object from the response body matching the ‘rel’ attribute of interest.

9. Find the names of the ‘rel’ attribute that specify the operation that the client wants to perform on the object. Extract the link object from the response body matching the ‘rel’ attribute of interest.

10. If an input needs to be specified, refer to the XSD for the expected operation and build the request accordingly. Issue the supported HTTP request for that operation. The HTTP request matching the 'rel' attribute is provided in this document and in the quick reference document.

Note: Points 7, 8, and 9 are applicable only for XML.
A detailed 'hello world' style example is provided in the chapter, 'Hello World' workflow: An example illustrating API usage depicting this flow.

Note: JSON does not support atom links and relations.

Links and relations (applicable only for XML)

WFA REST API makes extensive use of atom links to provide references to objects and the actions that they support. These links are the mechanism that the client must use to access and operate on an object. The client must make use of the links to drill down further and perform operations on an object. If a client request body includes links, the server ignores them.

Links are context sensitive. Links appear in a response body based on which, the given user is allowed to perform in the given context. Due to this, links provide a way to gradually self-discover and explore the resources exposed by the API.

Typically, links are provided in the following form in response objects:

```
<ns2:link href="URL" rel="relationship"/>
```

The 'href' attribute provides the actual URL that should be used to access the resource. The clients must always use the URL specified in the above link to operate on individual target objects instead of hardcoding the URLs.

The 'rel' attribute value provides the relationship of the object; whose XML representation contains the link to a target object. The relationship also specifies the operations or actions that can be performed on a given resource. The relationship also indicates the HTTP request type to use when making a request with the 'href' attribute.

The following table contains the definitions of all the common links that might appear in responses. These relations are standard across all WFA object collections.

Note: If a particular relation does not appear in a response object, it might either mean that this particular operation is not supported for that object or mean that the operation or relation specified by the link is not relevant to the current context.

Relation (specified by rel)	Description of the relation	Relevant HTTP Request
Self	View the representation of an object.	GET
List	View the object collection.	GET
Add	Add a new object to the object collection.	POST
Update	Update an object.	PUT

Remove	Remove an object.	DELETE

Apart from a partial or complete set of the standard actions listed above, objects in a given object collection might support several non-standard actions and relations that are specific to that object. The following sections in this document that describe these object collections also contains a table that explains all the standard and non-standard links a given object supports.

WFA object collection

WFA API defines and exposes a collection of resources and their representations. These can be used by clients to access and operate on the collection or individual items within the collection.

The following WFA object collections are represented through the API:

- Users (/rest/users)
- Backup and Restore (/rest/backups)
- Dar export/import (rest/dars)
- Configurations(/rest/configurations)
- Free text search (/rest/search)
- AutoSupport configuration (/rest/system/asup)
- LDAP configuration (/rest/system/ldap)
- System information (/rest/system)
- Workflows and associated Jobs (/rest/workflows)
- Workflow Executions
- Filters (/rest/filters)
- Finders (/rest/finders)
- Credentials (/rest/credentials)
- Data Sources and Data Source types (/rest/data_sources and /rest/data_source_types) and associate acquisition jobs WFA has some internal APIs:

The following internal APIs should not be used by clients directly:

- Command executions (/rest/execution/command)
- Job execution (/rest/execution/jobcommand)
- Script execution(/rest/execution/script)

Users

The user collection allows authenticated and authorized clients to programmatically view user accounts and change passwords. WFA also provides REST API support for addition and deletion of users.

```
The root URI for the user collection is /rest/users. The client can start consuming
the API by invoking a GET on this URI. It returns a collection of users based on what
the logged in user is eligible to see and operate on.
```

```
The links that appear in each user resource in the collection can be used by the
client to further drill down and perform other operations on individual user objects.
The root URI for user management is /rest/users
```

Refer to the chapter: Users to learn more about the request and response content, available links and relations, and supported query parameters.

Backup and restore

It allows the authenticated and authorized clients to take the database backup and download and restore from the local file.

```
The root URI for backup and restore is /rest/backup.
```

Refer to the chapter: Backup and Restore to learn more about the request and response content, available links and relations, and supported query parameters.

Dar export and import

It allows the authenticated and authorized clients to export and import the entities to and from the local file.

```
The root URI for dars is /rest/dars.
```

It also allows export or import of a specific entity using their UUID.

Refer to the chapter: Dar export/import to learn more about the request and response content, available links and relations, and supported query parameters.

Configurations

It validates WFA configurations. In case of validation failure, it returns the list of Validation Representation for invalid objects; otherwise returns no content.

```
The root URI for validate configurations is /rest/configurations.
```

Refer to the chapter: Configuration to learn more about the request and response content, available links and relations, and supported query parameters.

Free text search

The search allows authenticated and authorized clients to search for different entities given a term and a search context. The term should be at least two characters and complete (It does not support regular expressions).

Search contexts available are: ALL, FINDERS, FILTERS, WORK_FLOWS, CATEGORIES, POLICIES, FUNCTIONS, COMMANDS, DICTIONARY_ENTRIES, EXECUTION_STATUS, CACHE_QUERIES, and STORE.

The root URI for search is /rest/search. Refer to the chapter: Free Text Search to learn more about the request and response content, available links and relations, and supported query parameters.

AutoSupport configurations

The AutoSupport configurations allow authenticated and authorized clients to change the AutoSupport configurations. It supports uploading the AutoSupport data to AutoSupport store at NetApp. Download allows the AutoSupport data to be downloaded to local machine.

```
The root URI for asup configurations is /rest/system/asup.
```

Refer to the chapter: AutoSupport Configuration to learn more about the request and response content, available links and relations, and supported query parameters.

LDAP configurations

The LDAP configurations allow authenticated and authorized clients to change the LDAP configurations.
It supports the validation of LDAP user credentials.

```
The root URI for LDAP configurations is /rest/system/ldap.
```

Refer to the chapter: LDAP Configuration to learn more about the request and response content, available links and relations and supported query parameters.

System information

It represents WFA system information mainly major and minor versions of different components.

```
The root URI for system information configuration is /rest/system.
```

Refer to the chapter: System Information to learn more about the request and response content, available links and relations, and supported query parameters.

Workflows

The workflow collection allows authenticated and authorized clients to retrieve a collection of workflow objects and to perform various actions on this workflow. It provides the ability to execute, monitor, and control workflows.

WFA delivers a set of sample workflows. Users can also create their own customized workflows as per their requirements. WFA API does not allow users to create, update, or delete workflows. Users must use the WFA designer GUI to create workflows. The API can only be used to access, execute, control, and monitor the sample or user-created workflows.

Each workflow object encapsulates the workflow providing metadata information, such as name and description about the workflow as well as the user input (input) and return parameter (output) information associated with the workflow.

Query parameters allow the client to filter the workflow collection based on a given category or based on a given name.

The workflow collection visible to a user will depend on which category of workflows the user is eligible to see and operate on.

```
The root URI for workflows is /rest/workflows.
```

Refer to the chapter: Workflows to view all supported query parameters, links, relations, and action.

Workflow Executions

Workflow Executions represents WFA workflow execution instance.

```
The root URI for workflow execution instance is /rest/workflows/executions.
```

Refer to the chapter: Workflow Executions to view all supported query parameters, links, relations, and action.

Filters

Filters represent a collection of filter objects. Each filter in the collection is an object that encapsulates individual resource selection criteria.

WFA has many sample filters. In addition to the sample filters, users can create their own customized filters. Currently, WFA API can only be used to view and test these filters. The creation of new filters, or modification or deletion of existing filters can only be done through the WFA designer GUI.

```
The root URI for filters is /rest/filters.
```

Refer to the chapter: Filters

Name	Description	Type	Default
workflow_execution_id	Workflow execution ID.	path	
child_command_index	Index of the CommandExecutionArguments which correspond to child workflow execution. When specified, this returns the list of CommandExecutionArguments of the child workflow execution corresponding to the index.	query	

Response body

```
element: (custom) media
types: application/xml or
application/json
```

LogMessage

It returns the list of LogMessage with a given workflow execution id and command/step index.

Parameters

Name	Description	Type	Default
workflow_execution_id	Workflow execution ID.	path	
command_index	Command/step index whose log messages are to be returned. The index starts with 0.	path	
child_command_index	Command/step index whose log messages are to be returned from within the child workflow. The index starts with 0.	query	

Response body

```
element: (custom) media
types: application/xml or
application/json
```

Filters to learn more about the request and response content, available links and relations, and supported query parameters.

Finders

Finders represent a collection of finder objects. Each finder represents a logical collection of resource selection filters that are executed together to find a resource that matches multiple criteria. RESTful clients can operate on these resources to view and to test a given finder and obtain results.

WFA has many sample finders. In addition to the sample finders, users can create their own custom finders. Currently, the WFA API can only be used to view and test these finders.

The creation of new finders, or modification or deletion of existing finders can only be done through the WFA designer GUI.

```
The root URI for finders is /rest/finders.
```

Refer to the chapter: Finders to learn more about the request and response content, available links and relations, and supported query parameters.

Credentials

Credentials represent a collection of credential objects. Each credential object represents the credential details of a given data center object. The data center object could be, for example, a Data ONTAP system, a VMware vCenter system or a NetApp management server.

```
REST APIs are available to retrieve, add, modify, and delete credentials of
a given system. The root URI for credentials is /rest/credentials.
```

Refer to the chapter: Credentials to learn more about the request and response content, available links and relations, and supported query parameters.

Data sources and data source types

Data sources represent a collection of data sources of a given type. Data source types represent the collection of different data source types that are created within WFA. The API allows clients to view these data sources and data source types and to programmatically run acquisition jobs on the data sources.

Each data source represents an instance of a given data source type that is created for the purpose of acquiring data to meet resource selection needs.

The root URI for data sources is `/rest/data_sources.` The root URI for data source types is `/rest/data_source_types.`

Refer to the chapter: Data Source and Data Source Types to learn more about the request and response content, available links and relations, and supported query parameters.

'Hello World' workflow: An example illustrating API usage

The information provided in this chapter is designed to serve developers who develop a client application that consumes various WFA services using the API.

The following topics are covered as example:

- Retrieving a workflow collection filtered by query parameter (for example, a name). This will also demonstrate how to fill the 'Accept' and 'Authorization' headers.
- Inspecting the response and extracting the links (applicable only for XML).
- Creating a request object and executing a workflow using the appropriate link (applicable only for XML).
- Extracting the 'self' link and executing an HTTP request to retrieve the status of the workflow (applicable only for XML).

Note: This is an example to illustrate the usage of the API. This example might not run as it is on an installed WFA server (as we do not ship the 'Hello World' workflow). The client should replace the 'Hello World' workflow with the exact name of the workflow the client wishes to execute. The rest of the services provided by the WFA server can be consumed by the client in a similar fashion.

Note: The example is shown with HTTP messages to illustrate the client/server interactions without the use of any languages. A client written in a particular language can use the standard HTTP client library and standard XML or JSON library supported by the language to build a client.

Getting a filtered workflow collection

Every WFA request requires authentication. WFA uses HTTP basic authentication. For this to work, the client must either fill the 'Authorization' header properly the first time or fill the credentials when challenged with a '401 Unauthorized' error code by the server. With HTTP basic authentication, the client must fill the 'Authorization' header with the proper user credentials (MIME base64 encoding of the form *username: password*).

The root URL for getting a workflow collection is /rest/workflows.

Example 3-1: Getting a filtered workflow collection by name Request:

XML request:

```
GET /rest/workflows?name=Hello%20World HTTP/1.1
Authorization: Basic <encoded_credentials>

Accept: application/xml
```

JSON request:

```
GET /rest/workflows?name=Hello%20World HTTP/1.1
Authorization: Basic <encoded_credentials>

Accept: application/xml
```

Response:

XML response

```
200 OK
Date: <date of
request> Content-
Type: application/xml

<collection xmlns:ns2="http://www.w3.org/2005/Atom">
    <workflow uuid="76b936ba-d52e-4304-b562-01676d35ad43">
    <name>Hello World</name>
<description>
         Hello World Example
    </description>
    <certification>NONE</certification>
    <categories>
        <category>API EXAMPLE</category>
    </categories>
    <userInputList>
        <userInput>
            <name>Name</name>
            <description>My Name</description>
            <type>String</type>
            <mandatory>true</mandatory>
        </userInput>
    </userInputList>
    <returnParameters>
        <returnParameter>
            <name>Name</name>
            <value>John Doe</value>
            <description>My Name</description>
        </returnParameter>
    </returnParameters>
    <ns2:link href="http://localhost/rest/workflows/76b936ba-
d52e-4304b562- 01676d35ad43" rel="self"/>
    <ns2:link href="http://localhost/rest/workflows" rel="list"/>
<ns2:link href="http://localhost/rest/workflows/76b936ba-
d52e-4304b562- 01676d35ad43/jobs" rel="execute"/>
    <ns2:link href="http://localhost/rest/workflows/76b936ba-
d52e-4304b562- 01676d35ad43/out" rel="out-parameter"/>
    <ns2:link href="http://localhost/rest/workflows/76b936ba-
d52e-4304b562- 01676d35ad43/preview" rel="preview"/>
    </workflow>
</collection>
```

JSON response

```
200 OK
Date: <date of request>
Content-Type: application/json

[
  {
    "uuid": "76b936ba-d52e-4304-b562-01676d35ad43",
    "name": "Hello  World",
    "description": "Hello World Example",
    "certification": "NONE",
    "categories": [
      "API EXAMPLE"
    ],
    "userInputList": [
      {
        "name": "Name",
        "description": "My Name",
        "type": "String",
        "mandatory": true
      }
    ],
    "returnParameters": [
         {
        "name": "Name",
             "value": "John Doe",
        "description": "My Name"
      }
       ]
  }
]
```

Inspecting the response and extracting the links (only for XML)

The client can now extract the link for ‘execute’ (highlighted in red) from the message, which is as follows:

http://localhost/rest/workflows/76b936ba-d52e-4304-b562-01676d35ad43/jobs

Using the link above, the client can execute the workflow. If the developers’ looks up this guide, they can see that the HTTP request associated with execute is ‘POST’ and the request type is ‘Workflow Input’. The client should now fill this request information as shown in the following section.

Creating a request object and executing a workflow

Using XML

Now the ‘execute’ URL has been extracted for a given workflow (from the ‘href’ attribute of the link with the ‘rel’ attribute matching ‘execute’), the client can execute the given workflow as shown in the following example:

Request:

```
POST /rest/workflows/76b936ba-d52e-4304-b562-01676d35ad43/jobs HTTP/1.1\
Accept: application/xml
Authorization: Basic encoded-credentials

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
     <workflowInput>
          <userInputValues>
               <userInputEntry key="Name" value="John Doe"/>
          </userInputValues>
          <comments>API Example Execution</comments>
          <executionDateAndTime>9/23/11 2:59 PM</executionDateAndTime>
</workflowInput>
```

Using JSON

The client can execute the given workflow using workflow uuid as shown in the following example:

Request:

```
POST /rest/workflows/76b936ba-d52e-4304-b562-01676d35ad43/jobs HTTP/1.1
Accept: application/json
Authorization: Basic encoded-credentials

{
  "executionDateAndTime": "9/23/11 2:59 PM",
  "comments": "API Example Execution",
  "userInputValues": [
    {
      "key": "Name",
      "value": "John Doe"
    }
  ]
}
```

Response:

Using XML

```
HTTP/1.1 201 Created
Date: request-date
Content-Type: application/xml
Location: http://localhost/rest/workflows/76b936ba-d52e-4304-b562-
01676d35ad43/jobs/83

<?xml version="1.0" encoding="UTF-8"?>
<workflow uuid="76b936ba-d52e-4304-b562-01676d35ad43">
    <name>Hello World</name>
          <description>
               Hello World Example
          </description>
          <certification>NONE</certification>
          <categories>
              <category>API EXAMPLE</category>
          </categories>
          <userInputList>
              <userInput>
                  <name>Name</name>
                  <description>My Name</description>
                  <type>String</type>
                  <mandatory>true</mandatory>
              </userInput>
    </userInputList>
    <returnParameters>
        <returnParameter>
            <name>Name</name>
            <value>John Doe</value>
            <description>My Name</description>
        </returnParameter>
    </returnParameters>
    <ns2:link href="http://localhost/rest/workflows/76b936ba-d52e-
4304b562- 01676d35ad43" rel="self"/>
    <ns2:link href="http://localhost/rest/workflows" rel="list"/>
<ns2:link href="http://localhost/rest/workflows/76b936ba-d52e-
4304b562- 01676d35ad43/jobs" rel="execute"/>
    <ns2:link href="http://localhost/rest/workflows/76b936ba-d52e-
4304b562- 01676d35ad43/out" rel="out-parameter"/>
    <ns2:link href="http://localhost/rest/workflows/76b936ba-d52e-
4304b562- 01676d35ad43/preview" rel="preview"/>
</workflow>
<jobStatus>
        <jobStatus>SCHEDULED</jobStatus>
        <jobType>Workflow Execution - Hello World</jobType>
        <scheduleType>Immediate</scheduleType>
        <plannedExecutionTime>Dec 14, 2012
12:47:23 PM</plannedExecutionTime>
        <comment>API Example Execution</comment>
 </jobStatus>
    <ns2:link
href="http://localhost/rest/workflows/76b936ba-d52e-
4304b562-01676d35ad43/jobs"     rel="add"/>
    <ns2:link
href="http://localhost/rest/workflows/76b936ba-d52e-
4304b562-01676d35ad43/jobs/83/resume"     rel="resume"/>
    <ns2:link
href="http://localhost/rest/workflows/76b936ba-d52e-
4304b562-01676d35ad43/jobs/83/cancel"     rel="cancel"/>
    <ns2:link
href="http://localhost/rest/workflows/76b936ba-d52e-
4304b562-01676d35ad43/jobs/83/plan/out"     rel="out"/>
    <ns2:link
href="http://localhost/rest/workflows/76b936ba-d52e-
4304b562-01676d35ad43/jobs/83"     rel="self"/>
</job>
```

As shown in the above example, the job is scheduled (highlighted in blue) and the client can extract the URL either from the location response header or from the self-link in the response body (highlighted in red). A GET on this URL provides the current status of the job.

Using JSON

```
{
  "jobId": 100,
  "workflow": {
    "uuid": "76b936ba-d52e-4304-b562-01676d35ad43",
    "name": "Hello  World",
    "description": "Hello World Example",
    "certification": "NONE",
    "categories": [
      "API EXAMPLE"
    ],
    "userInputList": [
      {
        "name": "Name",
        "description": "My Name",
        "type": "string",
        "mandatory": true
      }
    ],
    "returnParameters": [
      {
        "name": "Name",
        "value": "John Doe",
        "description": "My Name"
      }
    ]
  },
  "jobStatus":  {
    "jobStatus": "SCHEDULED",
    "jobType": "Workflow Execution – Hello World",
    "scheduleType": "Immediate",
    "plannedExecutionTime": "Dec 14, 2012 12:47:23 PM",
    "comment": "API Example Execution",
    "userInputValues": [
      {
        "key": "string",
        "value": "string"
      }
    ],
    "returnParameters": [
      {
        "key": "string",
        "value": "string"
      }
    ]
  }
}
```

Extracting the job's 'self' link to monitor and obtain the status of the workflow (only for XML)

After extracting the status link from the above response, the client can retrieve the status of the workflow as follows:

Request:

```
GET /rest/workflows/76b936ba-d52e-4304-b562-01676d35ad43/jobs/83 HTTP/1.1
Accept: application/xml
Authorization: Basic encoded-credentials
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/xml
<?xml version="1.0" encoding="UTF-8"?>
<workflow uuid="76b936ba-d52e-4304-b562-01676d35ad43">
    <name>Hello World</name>
<description>
      Hello World Example
    </description>
    <certification>NONE</certification>
    <categories>
        <category>API EXAMPLE</category>
    </categories>
    <userInputList>
        <userInput>
            <name>Name</name>
            <description>My Name</description>
            <type>String</type>
            <mandatory>true</mandatory>
        </userInput>
    </userInputList>
    <returnParameters>
        <returnParameter>
            <name>Name</name>
            <value>John Doe</value>
            <description>My Name</description>
        </returnParameter>
    </returnParameters>
    <ns2:link href="http://localhost/rest/workflows/76b936ba-d52e-
4304b562- 01676d35ad43" rel="self"/>
    <ns2:link href="http://localhost/rest/workflows" rel="list"/>
<ns2:link href="http://localhost/rest/workflows/76b936ba-d52e-
4304b562- 01676d35ad43/jobs" rel="execute"/>
    <ns2:link href="http://localhost/rest/workflows/76b936ba-d52e-
4304b562- 01676d35ad43/out" rel="out-parameter"/>
    <ns2:link href="http://localhost/rest/workflows/76b936ba-d52e-
4304b562- 01676d35ad43/preview" rel="preview"/>
</workflow>
    <jobStatus>
        <jobStatus>COMPLETED</jobStatus>
        <jobType>Workflow Execution - Hello World</jobType>
        <scheduleType>Immediate</scheduleType>
        <startTime>Dec 14, 2012 12:47:27 PM</startTime>
        <endTime>Dec 14, 2012 12:47:36 PM</endTime>
<plannedExecutionTime>Dec 14, 2012 12:47:23
PM</plannedExecutionTime>
        <comment>API Example Execution</comment>
        <returnParameters>
            <returnParameters value="John Doe" key="Name"/>
        </returnParameters>
    </jobStatus>

    <ns2:link href="http://localhost/rest/workflows/76b936ba-d52e-4304b562-
01676d35ad43/jobs"     rel="add"/>
    <ns2:link href="http://localhost/rest/workflows/76b936ba-d52e-4304b562-
01676d35ad43/jobs/83/resume"     rel="resume"/>
    <ns2:link href="http://localhost/rest/workflows/76b936ba-d52e-4304b562-
01676d35ad43/jobs/83/cancel"     rel="cancel"/>
    <ns2:link href="http://localhost/rest/workflows/76b936ba-d52e-4304b562-
01676d35ad43/jobs/83/plan/out"     rel="out"/>
    <ns2:link href="http://localhost/rest/workflows/76b936ba-d52e-4304b562-
01676d35ad43/jobs/83"     rel="self"/>
</job>
```

As shown above in green, the status of the workflow job shows that the job is completed. The client can now extract the return parameters from the job status in the response listed above.

Backup and Restore

The root URL is /rest/ backups.

An HTTP GET on this URL returns a URL of the backup file to be downloaded. The API accepts a query parameter named “full”. When this parameter is set to true, a full backup is taken that includes configuration parameters.

An HTTP POST or PUT on this URL restores the resources from external backup file. The KeyName of the backup file in the form data must be "backupFile”. The media type must be “multipart or formdata”. This API accepts a query parameter named “full”. When this parameter is set to true, even the configuration parameters are restored along with the resources from the backup file.
The response for the POST or PUT at this URL returns a response with no contents in case of a successful restore. In case of a failure, error message is returned as a response. In case of successful restore with minor issues, warning is returned in response. Typical examples of warnings include cache upgrade failures, mismatches in packs imported before and after the restore.

Security

Only users with backup, admin, or architect privileges can invoke this API to download the backup file.

A ‘403 Forbidden’ response will be returned other users.
Unauthenticated access (missing or invalid credentials) will be challenged with a ‘401 Unauthorized ‘response by the server.

HTTP error codes

Status Code	**Response**
Valid codes Success Codes: 200 OK Error Codes 400 Bad request 401 Unauthorized 403 Forbidden 404 Not found 405 Method Not Allowed 415 Unsupported Media Type 500 Server error	See *Appendix B* on HTTP status codes and detailed error messages for a detailed description of these error codes and when they will be returned.

Dar export/import

The root URL is /rest/dars. An HTTP GET on this URL returns a URL of the export file to be downloaded.

An HTTP POST or PUT on this URL imports the resources from the dar file. The KeyName for import file in the form data must be "dar". The media type should be "multipart/form-data".

Security

Only a user with backup, architect, or admin privileges can invoke this API to export/import dar file. A '403 Forbidden' response will be returned for other users.

Unauthenticated access (missing or invalid credentials) will be challenged with a '401 Unauthorized' by the server.

HTTP error codes

Status Code	**Response**
Valid codes Success Codes: 200 OK Error Codes 400 Bad request 401 Unauthorized 403 Forbidden 404 Not found 405 Method Not Allowed 415 Unsupported Media Type 500 Server error	See *Appendix B* on HTTP status codes and detailed error messages for a detailed description of these error codes and when they will be returned.

Configuration

The root URL is /rest/configurations. An HTTP Get on the URL/rest/configurations/validate the WFA configurations for every object.

In case of validation failure, it returns the list of ValidationRepresentation for invalid objects; otherwise returns no content.

Security

Only a user with admin privileges can invoke this API.

A ‘403 Forbidden’ response is returned for other users.

Unauthenticated access (missing or invalid credentials) will be challenged with a ‘401 Unauthorized’ by the server.

ValidationRepresentation

The following table shows the contents of ValidationRepresentation object.

Name	**Type**	**Mandatory**	**Description**
Type	String	True	Object type or entity type on which validation is performed.
Message	String	True	Validation error message
Identifiers	String	True	Identifiers of the object whose validation is failed.

HTTP error codes

Status Code	**Response**
Valid codes <u>Success Codes:</u> 200 OK <u>Error Codes</u> 400 Bad request 401 Unauthorized 403 Forbidden 404 Not found 405 Method Not Allowed 415 Unsupported Media Type 500 Server error	See *Appendix B* on HTTP status codes and detailed error messages for a detailed description of these error codes and when they will be returned.

- API Executions

The root URL is /rest/execution/api. It represents generic execution session.

Free Text Search

Search API represents WFA free text search features.

```
The root URL is /rest/search.
```

The following resources are part of this group:

```
/rest/search
```

Security

Only a user with admin or architect privileges can invoke this API.

A ‘403 Forbidden’ response is returned for other users.

Unauthenticated access (missing or invalid credentials) will be challenged with a ‘401 Unauthorized’ by the server

Search different entities

Search for different entities for a given a term and a search context.

URI:/rest/search

Parameters

Name	**Description**	**Type**	**Default**
Term	String term to search for in various entities in the system	query	
context	Defines the search context where the search has to be conducted. Valid values are: ALL, FINDERS, FILTERS, WORK_FLOWS, CATEGORIES, POLICIES, FUNCTIONS, COMMANDS, DICTIONARY_ENTRIES, EXECUTION_STATUS, CACHE_QUERIES, STORE, SCHEMES, REMOTE_SYSTEM_TYPES, PACKS	query	

Response body

```
element: (custom) media
types: application/xml or
application/json
```

Return list of SearchResult objects

Using XML

```
<?xml version="1.0" encoding="UTF-8"?>
 <searchResult id="...">
    <name>...</name>
    <type>...</type>
 <schemeNames>...</schemeNames>
 <version>
   <major>...</major>
   <minor>...</minor>
   <revision>...</revision>
```

```
</version>
</searchResult>
```

Using JSON

```
{
  "id": "...",
  "name": "...",
  "type": "...",
  "schemeNames": "...",
  "version": {
    "major": ...,
    "minor": ...,
    "revision": ...
  }
```

HTTP error codes

Status Code	**Response**
Valid codes Success Codes: 200 OK Error Codes 400 Bad request 401 Unauthorized 403 Forbidden 404 Not found 405 Method Not Allowed 415 Unsupported Media Type 500 Server error	See *Appendix B* on HTTP status codes and detailed error messages for a detailed description of these error codes and when they will be returned.

AutoSupport Configuration

AutoSupport API represents WFA auto support data and configurations.

`The root URL is /rest/system/asup.`

The following resources are part of this group:

```
/rest/system/asup
/rest/system/asup/download /rest/system/asup/send
```

Security

Only a user with admin privileges can invoke this API.

A ‘403 Forbidden’ response is returned for other users.

Unauthenticated access (missing or invalid credentials) will be challenged with a ‘401 Unauthorized’ by the server.

Return AutoSupport configuration

It returns AutoSupport configuration using HTTP post. Only admin can perform this task.\

```
URI: /rest/system/asup
```

Response body

```
element: asup-configuration media
types: application/xml or
application/json
```

asup-configuration

Using XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<asup-configuration>
   <enabled>...</enabled>
     <protocol>...</protocol>
     <destination>...</destination>
     <content>...</content>
     <sender-mail-address>...</sender-mail-address>
 <scheduler>
    <enabled>...</enabled>
    <week-day>...</week-day>
    <hour>...</hour>
 </scheduler>
<proxy>
   <host>...</host>
   <port>...</port>
 </proxy>
  <runtime>
     <site>...</site>
     <company>...</company>
     <host>...</host>
     <system-id>...</system-id>
     <os>...</os>
  </runtime>
 </asup-configuration>
```

Using JSON

```
{
  "protocol": "...",
  "destination": "...",
  "content": "...",
  "scheduler": {
    "hour": ...,
    "enabled": ...,
    "week-day": "..."
  },
  "proxy": {
    "host": "...",
    "port": "..."
  },
  "runtime": {
    "site": "...",
    "company": "...",
    "host": "...",
    "os": "...",
    "system-id": "..."
  },
  "enabled": ...,
  "sender-mail-address": "..."
}
```

Update AutoSupport configuration

It updates AutoSupport configuration. Only admin can perform this task.

Request body

```
element: asup-configuration media
types: application/xml or
application/json
```

asup-configuration
Using XML

```
<?xml version="1.0" encoding="UTF-8"?>
<asup-configuration>
   <enabled>...</enabled>
     <protocol>...</protocol>
     <destination>...</destination>
     <content>...</content>
     <sender-mail-address>...</sender-mail-address>
 <scheduler>
    <enabled>...</enabled>
    <week-day>...</week-day>
    <hour>...</hour>
 </scheduler>
<proxy>
   <host>...</host>
   <port>...</port>
 </proxy>
  <runtime>
     <site>...</site>
     <company>...</company>
     <host>...</host>
     <system-id>...</system-id>
     <os>...</os>
  </runtime>
 </asup-configuration>
```

Using JSON

```
{
  "protocol": "...",
  "destination": "...",
  "content": "...",
  "scheduler": {
    "hour": ...,
    "enabled": ...,
    "week-day": "..."
  },
  "proxy": {
    "host": "...",
    "port": "..."
  },
  "runtime": {
    "site": "...",
    "company": "...",
    "host": "...",
    "os": "...",
    "system-id": "..."
  },
  "enabled": ...,
  "sender-mail-address": "..."
}
```

Response body

```
element: asup-configuration media
types: application/xml or
application/json
```

Updated ASUP configuration

Download AutoSupport data

Download the AutoSupport data. Only admin user can perform this task.

```
URI: /rest/system/asup/download
```

Response body

```
element:(custom) media
types: application/xml or
application/json
```

Response with URL for AutoSupport data to be downloaded.

Upload AutoSupport data

It uploads the AutoSupport data to the AutoSupport store at NetApp.

```
URI: /rest/system/asup/send
```

Request body

```
element: asup-configuration media
types: application/xml or
application/json
```

asup-configuration

Response body

```
element: (custom) media
types: application/xml or
application/json
```

Response with no contents

HTTP error codes

Status Code	Response
Valid codes Success Codes: 200 OK Error Codes 400 Bad request 401 Unauthorized 403 Forbidden 404 Not found 405 Method Not Allowed 415 Unsupported Media Type 500 Server error	See *Appendix B* on HTTP status codes and detailed error messages for a detailed description of these error codes and when they will be returned.

LDAP Configuration

LDAP API represents LDAP configurations for WFA.

`The root URL is /rest/system/ldap.`

The following resources are part of this group:

```
/rest/system/ldap
/rest/system/ldap/test
```

Security

Only a user with admin privileges can invoke this API.

A ‘403 Forbidden’ response is returned for other users.

Unauthenticated access (missing or invalid credentials) will be challenged with a ‘401 Unauthorized’ by the server.

Return LDAP configuration

It returns LDAP configuration using HTTP GET. Only admin can perform this task.

`URI: /rest/system/ldap`

Response body

```
element: LdapConfiguration media
types: application/xml or
APPLICATION/JSON
```

LDAPConfiguration

Using XML

```
<?xml version="1.0" encoding="UTF-8"?>
<LdapConfiguration>
  <enabled>...</enabled>
  <ldapServerUrlList>...</ldapServerUrlList>
  <usernameAttribute>...</usernameAttribute>
  <searchTimeOutSeconds>...</searchTimeOutSeconds>
  <distinguishedNameAttribute>...</distinguishedNameAttribute>
  <mailAttribute>...</mailAttribute>
  <groupMembershipAttribute>...</groupMembershipAttribute>
  <adminGroups>...</adminGroups>
  <architectGroups>...</architectGroups>
  <operatorGroups>...</operatorGroups>
  <guestGroups>...</guestGroups>
  <approverGroups>...</approverGroups>
  <ldapServers>
    <ldapServer>
      <url>...</url>
      <bindUsername>...</bindUsername>
      <bindPassword>...</bindPassword>
      <baseDistinguishedName>...</baseDistinguishedName>
    </ldapServer>
  </ldapServers>
      </LdapConfiguration >
```

Using JSON

```
{
  "enabled": ...,
  "ldapServerUrlList": "...",
  "usernameAttribute": "...",
  "searchTimeOutSeconds": ...,
  "distinguishedNameAttribute": "",
  "mailAttribute": "..."
  "groupMembershipAttribute": "...",
  "adminGroups": "...",
  "architectGroups": "...",
  "operatorGroups": "...",
  "guestGroups": "...",
  "approverGroups": "...",
  "ldapServers": [
    {
      "url": "...",
      "bindUsername": "...",
      "bindPassword": "...", "baseDistinguishedName": "..."
    }
  ]
      }
```

Set LDAP configuration

Request body

```
element: LdapConfiguration media
types: application/xml or
application/json
```

The xml or json object for the new LdapConfiguration

Response body

```
element: (custom) media
types: application/xml or
application/json
```

REST Response with updated LdapConfiguration entity.

Test user credentials

It tests user credentials. Only admin can perform this task and is supported only for LDAP users.

```
URI:/rest/system/ldap/test
```

Request body

```
element: user media types:
application/xml or
application/json
```

LDAP User details
Using XML

```
<?xml version="1.0" encoding="UTF-8"?>
 <user name="...">
   <password>...</password>
   <domain>...</domain>
   <roleType>...</roleType>
 <categories>
     <category>...</category>
     <category>...</category>
     <!--...more "category" elements...-->
 </categories>
   <isLdap>...</isLdap>      <atom:link
xmlns:atom="http://www.w3.org/2005/Atom">...</atom:link><atom:link
xmlns:atom="http://www.w3.org/2005/Atom">...</atom:link><!--
  ...more "link" elements...-->
 </user>
```

Using JSON

```
{
  "name": "…",
  "password": "…",
  "domain": "…",
  "roleType": "…",
  "categories": [
    "..."
  ],
  "ldap": true
}
```

Response body

```
element: (custom) media
types: application/xml or
application/json
```

REST Response

HTTP error codes

Status Code	Response

Valid codes Success Codes: 200 OK Error Codes 400 Bad request 401 Unauthorized 403 Forbidden 404 Not found 405 Method Not Allowed 415 Unsupported Media Type 500 Server error	See *Appendix B* on HTTP status codes and detailed error messages for a detailed description of these error codes and when they will be returned.

System Information

System represents WFA system information. The root URL is /rest/system.

The following resources are part of this group:

```
/rest/system
```

Security

Unauthenticated access (missing or invalid credentials) is challenged with a ‘401 Unauthorized’ by the server.

Return software details

It returns software details using HTTP post.

```
URI: /rest/system
```

Response body

```
element: about media types:
application/xml or
application/json
```

About

The following table shows a sample XML containing the WFA system information:

```
<?xml version="1.0" encoding="UTF-8"?>
<about>
 <wfa-software-version>
   <majorVersion>4</majorVersion>
    <minorVersion>0...</minorVersion>
<maintenanceVersion>0...</maintenanceVersion>
<configurationVersion>0...</configurationVersion>
<contentVersion>0...</contentVersion>
</wfa-software-version>
 <wfa-vendor> NetApp </wfa-vendor>
  <atom:link xmlns:atom="http://www.w3.org/2005/Atom">...</atom:link>
<atom:link xmlns:atom="http://www.w3.org/2005/Atom">...</atom:link> <!--
...more "link" elements...-->
 </about>
```

The following table shows a sample JSON containing WFA system information:

```
{
  "wfa-software-version": {
    "majorVersion": "4",
    "minorVersion": "0",
    "maintenanceVersion": "0",
    "configurationVersion": "0",
    "contentVersion": "0"
  },
  "wfa-vendor": "NetApp"
}
```

HTTP error codes

Status Code	**Response**
Valid codes Success Codes: 200 OK Error Codes 400 Bad request 401 Unauthorized 403 Forbidden 404 Not found	See *Appendix B* on HTTP status codes and detailed error messages for a detailed description of these error codes and when they will be returned.
405 Method Not Allowed 415 Unsupported MediaType 500 Server error	

Users

The root URI is /rest/users. An HTTP GET on this URL returns a list of users.

The user API allows the client (admin users) to programmatically view, create, update, and delete user information and change the current user's password. A user can only change his or her password.

Currently the user collection does not support any query parameters for filtering the collection.

Security

Only users with admin privilege can add or delete users.

Only a user with admin or architect privileges can invoke the API to get the list of all users.

A user with the 'operator' role will not be able to view other users. In such cases, the response simply returns a collection containing only the user on behalf of whom the client invoked the API.

A user can only change his or her own password using this API.

Guest users cannot access this API. A '403 Forbidden' response is returned for guest users.

Unauthenticated access (missing or invalid credentials) will be challenged with a '401 Unauthorized' by the server.

User data structures

The collection of users returned on a HTTP GET on /rest/users is nothing but a collection of user objects, whose content is described in detail in this section.

Note: To obtain the corresponding XML schema definition and sample XML documents representing these data structures, see the Reference Manual, which is available from within the product.

User

The following table shows the contents of each individual USER object.

Name	**Type**	**Mandatory**	**Description**
User Name	String	True	Name of the user
User Role Type	UserRoleType	True	Role performed by the user. An enumeration with any one of the following:
			• Guest • Operator • Architect • Admin • System • Backup
User Categories	Array of Strings	False	Categories that the user has access to. Only valid for users with operator role.
Is LDAP	Boolean	True	Is this user in LDAP?

Links (only for XML)	Array of Atom Links	False	A collection of atom links that specifies available resources and their links. Allows clients to do hypermedia traversal.
Domain	String	False	Domain to which the user belongs in case the user is a LDAP user.

The following table shows a sample XML containing a collection of exactly one user:

```
<?xml version="1.0" encoding="UTF-8"?>
<collection xmlns:ns2="http://www.w3.org/2005/Atom">
    <user name="admin">
        <roleType>Admin</roleType>
        <categories/>
        <isLdap>false</isLdap>
        <ns2:link href="http://localhost/rest/users/admin" rel="self"/>
<ns2:link href="http://localhost/rest/users/admin/password"
rel="change_password"/>
    </user>
</collection>
```

The following table shows a sample JSON containing a collection of exactly one user:

```
[
 {
  "name": "admin",
  "roleType": "Admin",
  "ldap": false,
  "categories": []
 }
]
```

Password change

This request object is to encapsulate password change information for clients that want to change the password of the current user.

Name	**Type**	**Mandatory**	**Description**
Old Password	String	True	Current Password
New Password	String	True	New Password

User links, relations, and action (only for XML)

The following table illustrates the links and their relations available for each user in the user collection.

Relation	**Description of the relation**	**Relevant**	**Input Type**	**Output Type**
(specified by rel)		**HTTP Request**		
Self	View the representation of an individual user object.	GET	NA	*Users[]*
List	View the collection of users.	GET	NA	*Users[]*
change_password	Change the password of a given user.	PUT	*Password change*	*Users[]*

add_user	Add a new user	POST	*Users[]*	*Users[]*

Notes: User entries that are LDAP linked contains ldap_domain_name or user_name in the URI. Non LDAP users will just have user_name in the URI. So it is necessary that the client uses the link found in the link relations to access individual users. change_password link will not be available for LDAP users.

If current password is empty or null, the service returns the following error message:

```
"Unable to change password because old password is empty".
```

If new password is empty or null, the service returns the following error message:

```
"Unable to change password because new password is empty".
```

If current password is incorrect, the service returns the following error message:

```
"Unable to change password because old password does not match existing
password".
```

HTTP error codes

Status Code	Response
Valid codes Success Codes: 200 OK Error Codes 400 Bad request 401 Unauthorized 403 Forbidden 404 Not found 405 Method Not Allowed 415 Unsupported Media Type 500 Server error	See *Appendix B* on HTTP status codes and detailed error messages for a detailed description of these error codes and when they will be returned.

Workflows

The root URL is /rest/workflows. An HTTP GET on this URL returns a collection of workflows based on what the user can view. The client can also filter the collection using supported query parameters.

The workflow collection is a list of workflows, filtered based on the query parameters. If no query parameters are specified, this API returns a collection of workflows that are accessible to the user who invoked the API. See the table below for the description of the query parameters.

Query parameters for filtering the workflow collection

Parameter	**Value**	**Description**
Categories	String []	A set of one or more categories specified in the example below. If this query parameter is specified, only workflows that belong to at least one of the list of categories is returned.
Name	String	Name of the workflow that needs to be returned. If this query parameter is specified, the collection of workflows contains only one entry that represents the workflow specified by the name.

Query parameter examples

The following URL returns a workflow that is named as ‘Hello World’, if the workflow is in ready for production state.

```
https://localhost/rest/workflows?name=Hello%20World
```

The following URL returns a collection of workflows for which categories are either ‘Application Provisioning’ or ‘Setup’.

```
https://localhost/rest/workflows?categories=Application%20Provisioning&cate
gories=Setup
```

The following example URL returns all the workflows that are in production.

```
https://localhost/rest/workflows
```

Security

Users with admin or architect privileges can access all workflows using this API. An ‘operator’ can only access and execute workflows that are in categories that are accessible by the user.

A ’403’ forbidden is returned for guest users or for operators who try to execute workflows that they are not authorized to execute.

Unauthenticated access (missing or invalid credentials) is challenged with a ‘401 Unauthorized’ by the server.

Workflow data structures

This section provides a description of all the major data structures exposed through the workflows API.

Note: To obtain the corresponding XML schema definition and sample XML documents representing these data structures, see the Reference Manual, which is available from within the product.

KeyAndValuePair

The key and value pair is used to encapsulate a single key and its value.

Name	Type	Mandatory	Description
Key	String	True	Key
Value	String	True	Value

Workflow

The workflow object represents a single workflow. The following table describes the information encapsulated within this object.

Name	Type	Mandatory	Description
Workflow UUID	String	True	A 32 byte UUID string that uniquely identifies the workflow.
Workflow name	String	True	Name of the workflow
Workfl ow descripti on	String	False	Description
Workflow certification	Enum	False	Certified by. Can be any one of the following: • NONE, • PS, • COMMUNITY
			• USER_LOCKED • NETAPP
Workflow category/categorie s	Array of Strings	False	One or more categories to which the workflow belongs.
User Input	Array	False	A collection of allowed *User input* for this workflow.
Return Parameter	Array	False	A collection of possible *Return parameter* that will be returned after the workflow is executed or previewed.

Hyper media Links (only for XML)	Array of Atom links	False	A collection of atom links that specify available resources and their links. Allows clients to do hypermedia traversal. Each of these URLs and their associated methods are described in this document. This entry is read-only for the client.
minOntapVersion	String		The minimum Data ONTAP version of the storage system required for the workflow execution.
version	Version		Indicates the version of the workflow and has elements for major, minor, and revision.

The following is a sample XML output containing a collection of exactly one workflow.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?> <collection
xmlns:ns2="http://www.w3.org/2005/Atom">
 <workflow uuid="a1ea8848-15e3-4162-9282-bef1380df310">
   <name>Example workflow - Create a volume and NFS Export</name>
   <description>This workflow is a sample.</description>
   <categories>
      <category>Example</category>
   </categories>
   <userInputList>
      <userInput>
         <name>vol_name</name>
         <description>The name of the volume</description>
  <defaultValue>vol1</defaultValue>
         <type>String</type>
      </userInput>
      <userInput>
         <name>rwHosts</name>
         <description>Hosts with access</description>
         <type>String</type>
      </userInput>
    </userInputList>
    <returnParameters>
       <returnParameter>
          <name>exportpathname</name>
          <value>the_export.export_path</value>
          <description>The pathname </description>
        </returnParameter>
     </returnParameters>
     <ns2:link href="http://localhost/rest/workflows/" rel="list"/>
     <ns2:link href="http://localhost/rest/workflows/a1ea8848-15e3-41629282-
bef1380df310" rel="self"/>
     <ns2:link href="http://localhost/rest/workflows/a1ea8848-15e3-41629282-
bef1380df310/jobs" rel="execute"/>
     <ns2:link href="http://localhost/rest/workflows/a1ea8848-15e3-4162-
9282-bef1380df310/out" rel="out-parameter"/>
     <ns2:link href="http://localhost/rest/workflows/a1ea8848-15e3-4162-
9282-bef1380df310/preview" rel="preview"/>
    </workflow>
</collection>
```

The following is a sample JSON output containing a collection of exactly one workflow:

```
[
    {
      "uuid": "a1ea8848-15e3-4162-9282-bef1380df310",
      "name": "Example workflow - Create a volume and NFS Export",
      "description": "This workflow is a sample.",
      "categories": [
        "Example"
      ],
      "userInputList": [
        {
          "name": "vol_name",
          "description": "The name of the volume",
          "defaultValue": "vol1",
          "type": "string",
```

```
        },
        {
          "name": "rwHosts",
          "description": "Hosts with access",
          "type": "string"
        }
      ],
      "returnParameters": [
        {
          "name":  "exportpathname",
          "value": "the_export.export_path",
          "description": "The pathname "
        }
      ]
    }
]
```

User input

This object encapsulates a single user input of a workflow. A workflow might contain zero or more of such user inputs.

Name	Type	Mandatory	Description
Parameter Name	String	True	Name of the input parameter
Parameter Description	String	False	Description of the parameter
Parameter default value	String	False	Default value for the parameter, if any
Parameter Type	String	True	Type of parameter. Can be any one of String, Query or Enumeration

Allowed Values	Array of Strings	False	Range or list depicting a set of legally allowed values for this parameter. Used in case of Enumeration or Query types only.
ConditionalUserInput	User input	False	Indicates the user input on which this user input is conditionally dependent on.
conditionalUserInputValues/conditional UserInputValues	String	False	NA
Mandatory	Boolean	True	NA
rowSelectionType	Enum		Indicates if the row selection type is single or multiple for a query multiselect user input type.
Columns	Array of User Inputs		The set of user inputs in a Table user input type.

Workflow input

The workflow input object is used to encapsulate information required to execute a workflow. The contents of the workflow input object is explained in the following table.

Name	**Type**	**Mandatory**	**Description**
Input parameters	Array	False	Input parameters A collection of *KeyAndValuePair*
Execution Date and Time	String	False	Execution date and time. For example: 9/23/11 2:59 PM Relevant only for executing or resuming jobs. If left empty, the job will be executed immediately.
Comment	String	False	A free form string that can be used to describe the action by the client. This comment is stored against the job and can be retrieved later by the client.
ScheduleID	String	False	ID of the schedule to execute the workflow job.

The following is sample XML for a WorkflowInput that is sent by a client.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
     <workflowInput>
          <userInputValues>
               <userInputEntry key="ArrayIP" value="10.68.66.214"/>
               <userInputEntry key="VolName" value="TestWS"/>
          </userInputValues>
          <comments>Execution for creating Test volumes</comments>
          <executionDateAndTime>9/23/11 2:59 PM</executionDateAndTime>
</workflowInput>
```

The following is sample JSON for a WorkflowInput that is sent by a client:

```
{
  "executionDateAndTime": "9/23/11 2:59 PM",
  "comments": "Execution for creating Test volumes",
  "userInputValues": [
    {
      "key": "ArrayIP",
      "value": "10.68.66.214"
    },
    {
      "key": "VolName",
      "value": "TestWS"
    },
  ]
}
```

Return parameter

The return parameter object encapsulates the output of the workflow. Each entry represents a field that corresponds to a particular output of the workflow. The following table shows the contents of each individual RETURN PARAMETER object. This structure represents details of each return parameter that might be returned as a result of executing the workflow.

Name	**Type**	**Mandatory**	**Description**
Name	String	True	Name of the return parameter (for example, volume).
Value	String	True	Value of the parameter (for example, $VolumeName).
Description	String	False	Description (for example, This is a Volume).

Job status

This structure depicts status information about the job that was executed against the workflow.

Note: There are some discrepancies in the XSD schema where every entry is a string, which makes it difficult to do XSD validation for dates, integers and enumerations. This discrepancy will be fixed in future versions.

Name	**Type**	**Mandatory**	**Description**

Job Status	String	True	Status of the Job. It is one of the following: • SCHEDULED • PENDING • EXECUTING • COMPLETED • FAILED • PARTIALLY_SUCCESSFUL • ABORTING • CANCELED • OBSOLETE • PLANNING • PAUSED
Job Type	String	True	Job type – Cache / Workflow execution
Schedule Type	String	True	Schedule type – Delayed / Immediate / Recurring
Start Time	String	False	Actual start time of job execution
End Time	String	False	Actual time when the job execution
			completed
Planned Execution Time	String	False	Planned time for job execution
Error Message	String	False	Error message if any (if the job execution failed)
Execution Comment	String	False	Comment supplied by the client while executing the job

Workflow job

This structure depicts all aspects of a single instance of a workflow job. This is to represent a workflow job resource only and is typically returned when a workflow job status is queried or when a workflow job is created.

Name	**Type**	**Mandatory**	**Description**
Job ID	Integer	True	ID of the workflow job.
Workflow	Workflow	True	Workflow details of the job.
Job status	Job status	True	Current Job status of the job.
Links (only for XML)	Atom Links	False	Atom links depicting other sub resources and allowed operations for the job. These links are read only for the client.

The following is a sample XML output containing job.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<job xmlns:ns2="http://www.w3.org/2005/Atom" jobId="3">
```

```
<workflow uuid="a1ea8848-15e3-4162-9282-bef1380df310">
<name>Example workflow - Create a volume</name>
<description>This workflow is a sample.</description>
<categories><category>Example</category></categories>
<userInputList>
<userInput>
<name>vol_size</name>
<description>The size</description>
<defaultValue>1</defaultValue>

<type>Number</type>
<allowedValues><value>0.01-5000</value></allowedValues>
</userInput>
</userInputList>
<returnParameters>
<returnParameter>
<name>export-pathname</name>
<value>the_export.export_path</value>
<description>The pathname</description>
</returnParameter>
</returnParameters>
<ns2:link href="http://localhost/rest/workflows/" rel="list"/>
<ns2:link href="http://localhost/rest/workflows/a1ea8848-15e3- 41629282-bef1380df310"
rel="self"/>
<ns2:link href="http://localhost/rest/workflows/a1ea8848-15e3- 41629282-bef1380df310/jobs"
rel="execute"/>
<ns2:link href="http://localhost/rest/workflows/a1ea8848-15e3-4162- 9282-bef1380df310/out"
rel="out-parameter"/>  <ns2:link href="http://localhost/rest/workflows/a1ea8848-15e3-4162-
9282-bef1380df310/preview" rel="preview"/>
</workflow>
<jobStatus>
<jobStatus>SCHEDULED</jobStatus>
```

```
    <jobType>Workflow Execution - Example workflow - Create
a volume</jobType>
    <scheduleType>Delayed</scheduleType>
    <plannedExecutionTime>Sep 23, 2012 2:59:00 PM</plannedExecutionTime>
    <comment>Execution for creating Test volumes</comment>
 </jobStatus>
  <ns2:link href="http://localhost/rest/workflows/a1ea8848-15e3-4162-
9282bef1380df310/jobs" rel="add"/>
  <ns2:link href="http://localhost/rest/workflows/a1ea8848-15e3-4162-
9282bef1380df310/jobs/3/resume" rel="resume"/>
  <ns2:link href="http://localhost/rest/workflows/a1ea8848-15e3-4162-
9282bef1380df310/jobs/3/cancel" rel="cancel"/>
  <ns2:link href="http://localhost/rest/workflows/a1ea8848-15e3-4162-
9282bef1380df310/jobs/3/plan/out" rel="out"/>
  <ns2:link href="http://localhost/rest/workflows/a1ea8848-15e3-4162-
9282bef1380df310/jobs/3" rel="self"/>
</job>
```

The following is a sample JSON output containing job:

```
{
  "jobId": 3,
  "workflow": {
    "uuid": "a1ea8848-15e3-4162-9282-bef1380df310",
    "name": "Example workflow - Create a volume",
    "description": "This workflow is a sample.",
    "categories": [
      "Example"
    ],
    "userInputList": [
      {
        "name": "vol_size",
        "description": "The size",
        "defaultValue": "1",
        "type": "Number",
        "allowedValues": [
          "0.01-5000"
        ]
      }
    ],
    "returnParameters": [
      {
        "name": "export-pathname",
        "value": "the_export.export_path",
        "description": "The pathname"
      }
    ]
  },
  "jobStatus":  {
    "jobStatus": "SCHEDULED",
    "jobType": "Workflow Execution - Example workflow - Create a volume",
    "scheduleType": "Delayed",
    "plannedExecutionTime": "Sep 23, 2012 2:59:00 PM",
    "comment": "Execution for creating Test volumes"

  }
}
```

Workflow links, relations, and actions (only for XML)

The links available in the response of a request can be used by the client to drill down further and access and operate on an individual workflow object.

The following table provides a detailed list of links and their relations with respect to the workflow resources. These links are represented in each workflow object in the collection, when the server returns the collection. The link fields are read-only for the client and the client need not specify the links when submitting requests. If the client submits links in requests, the server ignores them.

Relation (specified by rel)	Description of the relation	HTTP Request	Request Type	Response Type
list	Provides a list of workflow objects based on certain criteria. If no criteria are specified through a query parameter, all workflows that the user is eligible to view are returned.	GET	NA	*Workflow[]*
self	Returns the workflow object	GET	NA	*Workflow[]*

	specified by the link. The URL in the 'href' attribute has the UUID of the Workflow. UUID uniquely identifies a specific workflow resource.			
execute	Executes a given workflow. This creates a new job object and returns the job object as a part of the response.	POST	*Workflow* input*[]*	*Workflow* job *[]*
out-parameter	Returns the possible output (return) parameter values of a workflow as a key value pair.	GET	NA	*Return* parameter*[]*
Preview	Runs a preview (dry run) of the workflow with the given user inputs and returns the output. This action will not execute an actual workflow on the storage system. After previewing, this method returns the result as return parameters.	POST	*Workflow* input*[]*	*Return* parameter*[]*
User_input_query_result	Returns the query result for a given user input type which is either "Query" type or "QueryMultiSelect" . Reservation flag is derived from the workflow and applied for query result. For dependent query parameter mappings are passed through query parameter.	GET	NA	UserInputQueryResult

Notes: If the format of user input pair string is incorrect, the service returns the following error message as part of the BAD_REQUEST response:

```
"Invalid user-input value 'some_input_with_value' ,should be name=value"
```

Default values are used if a user input value has not been provided (if applicable).

Array of parameter names or an empty array is mandatory. In case, all the parameters are requested, use '@ ()' (in case of PowerShell WS command line client).

When the client performs the execute operation (shown above), a job is created. Thus, the resulting response contains a job representation (WorkflowJob) that contains another set of links, which allows the user to operate on the job resource. The following table shows the link and relations that are relevant for the job object.

Relation (rel attribute)	**Description**	**HTTP Request**	**Request Type**	**Response Type**
Resume	Resumes this job. This method will resume or reschedule a job that was already scheduled for execution.	POST	*Workflow* input*[]*	*Workflow* job
Cancel	Cancels this job. This method rejects a workflow that was already scheduled for execution.	POST	*Workflow* input*[]*	*Workflow* job
Out	Returns the output parameters and their values for the job. **Query Parameters:** **parameters** Denotes the parameters for which the values need to be returned. Can specify zero or more such parameters. If no parameters are specified, then all the return parameters of the workflow are returned.	GET	NA	*Return* parameter*[]*
Self	Returns the representation of this job. This link can be used to inspect the current state of the job.	GET	NA	*Workflow* job

Notes: For resuming and executing operations, if no date and time is provided in the 'WorkflowInput', the workflow is resumed and executed immediately.

For canceling operation, the execution data and time need not be specified in the 'WorkflowInput'. If it is specified, it is ignored by the client.

If the user executing the service is an operator who is not allowed to resume or execute this workflow, due to workflow authorization restrictions (the workflow is assigned to a category, this user is not assigned to) the following error is returned:

```
"current user 'user name'      is not allowed to resume workflow ' workflowId'.
```

Only workflows with the status of 'Paused', 'Scheduled', 'Aborted' or 'Failed' can be resumed. When an operator tries to resume a workflow with any other status the following error is returned:

```
"Could not resume workflow execution with id ' the workflow job id'. Resume is
only allowed from status ' PAUSED'."
```

When an operator tries to resume a failed workflow, that has expired, the following error is returned:

```
"Resume of failed workflow execution with id ' the workflow job Id ' is not
allowed. Workflow execution is expired - more than ' number of expiration days'
days have passed since failure"
```

For the ‘status’ operation, the service returns output data (Return Parameters) only if the job status is in a final state. It must be in one of the following statuses: COMPLETED, FAILED, PARTIALLY_SUCCESSFUL or ABORTED.

Return Parameter Values are returned as per planning phase. Failure of the job will not result in any change to the returned values.

For the ‘out’ operation, if the status of the job in question is not one of COMPLETED, FAILED, PARTIALLY_SUCCESSFUL, ABORTED, then the service returns the following error message:

```
“The job’s status is <Current Job Status>, data can be retrieved only in
the following statuses: COMPLETED, FAILED, ABORTED”.
```

For the ‘out’ operation, if the requested parameter name is not defined for the workflow, the service returns the following error message:

```
  “Parameter ’parameter name’ not found. It's not defined as a return parameter”
```

For the ‘cancel’ operation, if the user executing the service is an operator who is not allowed to execute this workflow, due to workflow authorization restrictions (the workflow is assigned to a category, this user is not assigned to) the following error is returned:

```
"current user 'user name'  is not allowed to reject workflow ' workflowId'
```

Only workflows with the status of ‘Paused’, ‘Scheduled’, ‘Pending’ or ‘Running’ can be rejected. No error is returned if a user tries to reject a workflow with a wrong status.

When an operator is trying to reject a paused workflow, that an operator is not allowed to reject, the following error is returned:

```
    "Users with 'Operator' role are not allowed to reject workflow executions".
```

HTTP error codes

Status Code	Response
Valid codes Success Codes: 200 OK Error Codes 400 Bad request	See *Appendix B* on HTTP status codes and detailed error messages for a detailed description of these error codes and when they will be returned.

401 Unauthorized 403 Forbidden 404 Not found 405 Method Not Allowed 415 Unsupported Media Type 500 Server error	

Workflow Executions

WorkflowExecutionResource represents WFA workflow execution instance.

CommandExecutionArguments

It returns the list of CommandExecutionArguments with a given workflow execution id.

Parameters

Name	Description	Type	Default
workflow_execution_id	Workflow execution ID.	path	
child_command_index	Index of the CommandExecutionArguments which correspond to child workflow execution. When specified, this returns the list of CommandExecutionArguments of the child workflow execution corresponding to the index.	query	

Response body

```
element: (custom) media
types: application/xml or
application/json
```

LogMessage

It returns the list of LogMessage with a given workflow execution id and command/step index.

Parameters

Name	Description	Type	Default
workflow_execution_id	Workflow execution ID.	path	
command_index	Command/step index whose log messages are to be returned. The index starts with 0.	path	
child_command_index	Command/step index whose log messages are to be returned from within the child workflow. The index starts with 0.	query	

Response body

```
element: (custom) media
types: application/xml or
application/json
```

Filters

Filters are the resource selection primitive that WFA uses to automatically select resources while planning a workflow before workflow execution. In WFA, filters are SQL statements with associated meta-data and input parameters and the output dictionary item that describes the input and output to the filter.

WFA REST API can be used by clients to programmatically view the meta-data and other aspects about a filter. The API can also be used to execute a filter (for testing the filter) and obtain its results. Currently, WFA API cannot be used to create, update, or delete a filter.

Query parameters to filter collection of filters

Parameter	Value	Description
dictionary	String	Name of the dictionary item. For example, if cm_storage.aggregate is specified, only filters that return this dictionary item as output will be returned.

Query Parameter Example:

```
An HTTP GET on the following URL returns all filters.
https://localhost/rest/filters
An HTTP GET on the following URL returns all filters that filter the
dictionary item cm_storage.aggregate.
https://localhost/rest/filters?dictionary=cm_storage.aggregate
```

Security

Only users with the admin role or the architect role can use FILTERS API. If operators or guest users attempt to use this API, the server returns a ‘403 Forbidden’.

Any ‘unauthorized access’ (i.e. incorrect or missing credentials) will be challenged with a ‘401 Unauthorized’ by WFA server.

Filter data structures

This section provides a description of all the major data structures exposed through the filters API.

Note: To obtain the corresponding XML schema definition and sample XML documents representing these data structures, see the Reference Manual, which is available from within the product.

KeyAndValuePair

The key and value pair is used to encapsulate a single key and its value.

Name	Type	Mandatory	Description
Key	String	True	Key
Value	String	True	Value

Filter

The filter object represents a single filter. The following table describes the information encapsulated within this object.

Name	Type	Mandatory	Description
Filter UUID	String	True	A 32 byte UUID string that uniquely identifies the filter.
Filter name	String	True	Name of the filter.
Filter certification	Enum	False	Certified by. Can be any one of the following: • NONE • PS • COMMUNITY • USER_LOCKED • NETAPP
Input Parameters	Array	False	A set of string that specifies the input parameters that need to be passed as input to the filter.
Dictionary Name	String	False	The fully qualified name of the dictionary item that the filter is used to filter resources.
			(for example, cm_storage.aggregate)
Hyper media Links (only for XML)	Array of Atom links	False	A collection of atom links that specifies available resources and their links. Allows clients to do hypermedia traversal. Each of these URLs and their associated methods are described in this document. This entry is read-only for the client.

The following table shows a sample XML output of /rest/filters that returns a collection of exactly one filter.

```
<collection xmlns:ns2="http://www.w3.org/2005/Atom">
  <filter id="eb6fc609-bc4c-4b28-b7af-d80e7c620645">
```

```
     <name>CM aggregates based on ONTAP version</name>
     <certification>NETAPP</certification>
<parameters>
     <parameter>os_version</parameter>
</parameters>
<dictionaryName>cm_storage.Aggregate</dictionaryName>
    <ns2:link href="http://localhost/rest/filters/eb6fc609-bc4c-4b28-
b7afd80e7c620645" rel="self"/>
    <ns2:link href="http://localhost/rest/filters" rel="list"/>
<ns2:link href="http://localhost/rest/filters/eb6fc609-bc4c-4b28-
b7afd80e7c620645/test_no_reservations"rel="test_no_reservations"/>
<ns2:link href="http://localhost/rest/filters/eb6fc609-bc4c-4b28-
b7afd80e7c620645/test" rel="test"/>
</filter>
   <filter id="7d88e461-ec6e-401d-8da3-bea6f1606c32">
   <name>CM aggregate by used space %</name>
   <certification>NETAPP</certification>
<parameters>
   <parameter>used_size_threshold</parameter>
   <parameter>used_space</parameter>
</parameters>
<dictionaryName>cm_storage.Aggregate</dictionaryName>
   <ns2:link href="http://localhost/rest/filters/7d88e461-ec6e-401d-
8da3bea6f1606c32" rel="self"/>
  <ns2:link href="http://localhost/rest/filters" rel="list"/>   <ns2:link
href="http://localhost/rest/filters/7d88e461-ec6e-401d-
8da3bea6f1606c32/test_no_reservations"rel="test_no_reservations"/>
<ns2:link href="http://localhost/rest/filters/7d88e461-ec6e-401d-
8da3bea6f1606c32/test" rel="test"/>
 </filter>
</collection>
```

The following table shows a sample JSON output of /rest/filters that returns a collection of exactly one filter:

```
[
 {
   "id": "eb6fc609-bc4c-4b28-b7af-d80e7c620645",
   "name": "CM aggregates based on ONTAP version",
   "certification": "NETAPP",
   "dictionaryName": "cm_storage.Aggregate",
   "parameters": [
    "os_version"
   ]
 }
]
```

FilterTestResults

The FilterTestResults object encapsulates the output of a filter test execution.

Name	Type	Mandatory	Description
Filter name	String	True	Name of the filter.
Dictionary name	String	True	Fully qualified name of the dictionary item that this filter is associated with.

Parameters	*KeyAndValuePair*	false	A collection of KeyAndValuePair items. Each item specifies an input parameter and the corresponding value that was passed by the client to test the filter.
Columns	Array of column	False	Contains column description of the output columns of the query result.
Rows	Array of row	False	Array of row. Each row contains exactly one output dictionary item that matches the criteria.

The following is sample XML output of FilterTestResults run on a filter that filters Clustered Data ONTAP aggregates based on os_version. In this example, the os_version was specified as 8.1.

```
<filterTestResults>
  <filterName>CM aggregates based on ONTAP version</filterName>
  <dictionaryName>cm_storage.Aggregate</dictionaryName>
<parameters>
  <parameter value="8.1" key="os_version"/>
</parameters>
<columns>
   <column>#</column>
   <column>name</column>
   <column>node.cluster.primary_address</column>
   <column>node.name</column>
</columns>
<rows>
 <row>
   <cell value="1" key="#"/>
   <cell value="aggr0" key="name"/>
   <cell value="10.72.181.165" key="node.cluster.primary_address"/>
   <cell value="f3170-181-42" key="node.name"/>
 </row>
<row>
   <cell value="2" key="#"/>
   <cell value="f317018142_aggr1" key="name"/>
   <cell value="10.72.181.165" key="node.cluster.primary_address"/>
   <cell value="f3170-181-42" key="node.name"/>
 </row>
<row>
   <cell value="3" key="#"/>
   <cell value="aggr0_f3170_181_43_0" key="name"/>
   <cell value="10.72.181.165" key="node.cluster.primary_address"/>
   <cell value="f3170-181-43" key="node.name"/>
</row>
<row>
   <cell value="4" key="#"/>
   <cell value="f317018143_aggr1" key="name"/>
   <cell value="10.72.181.165" key="node.cluster.primary_address"/>
   <cell value="f3170-181-43" key="node.name"/>
 </row>
</rows>
</filterTestResults>
```

The following is sample JSON output of FilterTestResults run on a filter that filters Clustered Data ONTAP aggregates based on os_version. In this example, the os_version was specified as 8.1.

```
{
  "filterName": "CM aggregates based on ONTAP version",
  "dictionaryName": "cm_storage.Aggregate",
  "parameters": [
    {
      "key": "os_version",
      "value": "8.1"
    }
  ],
  "columns": [
    "#",
"name",
"node.cluster.primary_address",
"node.name"
  ],
  "rows": [
    {
      "cell": [
        {
          "key": "#",
          "value": "1"
        },
         {
          "key": "name",
          "value": "aggr0"
        },
         {
          "key": "node.cluster.primary_address",
          "value": "10.72.181.165"
        },
         {
          "key": "node.name",
          "value": "f3170-181-42"
        }
      ]
    },
    {
      "cell": [
        {
          "key": "#",
          "value": "2"
        },
         {
          "key": "name",
          "value": "f317018142_aggr1"
        },
         {
          "key": "node.cluster.primary_address",
          "value": "10.72.181.165"
        },
         {
          "key": "node.name",
          "value": "f3170-181-42"
        }
      ]
    },
    {
      "cell": [
        {
          "key": "#",
          "value": "3"
        },
         {
          "key": "name",
          "value": "aggr0_f3170_181_43_0"
        },
         {
          "key": "node.cluster.primary_address",
          "value": "10.72.181.165"
        },
         {
          "key": "node.name",
          "value": "f3170-181-43"
        }
      ]
    },
    {
      "cell": [
        {
          "key": "#",
          "value": "4"
        },
         {
          "key": "name",
          "value": "f317018143_aggr1"
        },
         {
          "key": "node.cluster.primary_address",
          "value": "10.72.181.165"
        },
         {
          "key": "node.name",
          "value": "f3170-181-43"
        }
      ]
    }
  ]
}
```

Filter links, relations, and actions (only for XML)

The following table shows the links and relations of a filter object.

Relation	**Description**	**HTTP Request**	**Request Type**	**Response Type**
list	Lists all the filters. Query parameters can be used to restrict the list based on specific criteria.	GET	NA	*Filter[]*
self	Returns this specific filter.	GET	NA	*Filter*

test	Tests the filter and returns the results. The results include reservation data. **Query Parameters** <parameter>=<value>: The test filter expects the client to specify the input parameters to the filter as query parameters. An example is as below: http://localhost/rest/filters/eb6fc609-bc4c-4b28-b7af-d80e7c620645/test?os_version=8.1&clustername=test	GET	NA	*FilterTestResults*
test_no_reservation	Tests the filter without applying RESERVATION data. This action also supports query parameters exactly as the 'test' action shown in the row above.	GET	NA	*FilterTestResults*

HTTP error codes

Status Code	**Response**
Valid codes Success Codes: 200 OK Error Codes 400 Bad request 401 Unauthorized 403 Forbidden	See *Appendix B* on HTTP status codes and detailed error messages for a detailed description of these error codes and when they will be returned.
404 Not found 405 Method Not Allowed 415 Unsupported MediaType 500 Server error	

Finders

Finders are a logical group of filters for the same dictionary item, where each filter specifies particular criteria. Finders return a resource collection that matches all the criteria of all the filters in the finder.

WFA REST API can be used by clients to programmatically view the meta-data and other aspects of a finder. The API can also be used to execute a finder (for testing the finder) and obtain its results. Currently, WFA API cannot be used to create, update, or delete a finder.

Query parameters to filter collection of filters

Parameter	Value	Description
dictionary	String	Name of the dictionary item. For example, if cm_storage.aggregate is specified, only finders that return this dictionary item as output are returned.

Query Parameter Example

```
An HTTP GET on the following URL returns all finders.
https://localhost/rest/finders
An HTTP GET on the following URL returns all finders that find the
dictionary item cm_storage.aggregate.
https://localhost/rest/finders?dictionary=cm_storage.aggregate SECURITY
```

Security

Only users with the admin role or the architect role can use the finders API. If operators or guests attempt to use this API, the server returns a ‘403 Forbidden’.

Any unauthorized access (either incorrect or missing credentials) will be challenged with a ‘401 Unauthorized’ by the server.

Finder data structures

This section provides a description of all the major data structures exposed through the finders API.

Note: To obtain the corresponding XML schema definition and sample XML documents representing these data structures, see the *Reference Manual*, that is available from in the product.

KeyAndValuePair

The key and value pair is used to encapsulate a single key and its value.

Name	Type	Mandatory	Description
Key	String	True	Key
Value	String	True	Value

Finder

The finder object represents a single finder. The following table describes the information encapsulated within this object.

Name	**Type**	**Mandatory**	**Description**
Finder UUID	String	True	A 32 byte UUID string that uniquely identifies the finder.
Finder name	String	True	Name of the finder.
Finder certification	Enum	false	Certified by. Can be any one of the following: • NONE • PS • COMMUNITY • USER_LOCKED • NETAPP
Input Parameters	Array	False	A set of String that specifies the input parameters that need to be passed as input to the finder.
Dictionary Name	String	False	The fully qualified name of the dictionary item that the filter is used to filter resources. (for example, cm_storage.aggregate).
Filters	Array of *Filter*	False	A logical grouping of filters corresponding to the same dictionary item that together forms this dictionary item finder.
Hyper media Links (only for XML)	Array of Atom links	False	A collection of atom links that specifies available resources and their links. Allows clients to do hypermedia traversal. Each of these URLs and the associated methods are described in this document. This entry is read-only for the client.

The following is a sample XML output of /rest/finders that returns a collection of exactly one finder.

```
<collection xmlns:ns2="http://www.w3.org/2005/Atom">
<finder id="915c8c3f-3863-4fef-a198-b35257d5de2d">
<name>Find CM aggregate in a given CM node</name>
<dictionaryName>cm_storage.Aggregate</dictionaryName>
<certification>NETAPP</certification>
<filters>
<filter id="e6ee693c-5707-4936-bf27-99d6c8b319db">
<name>CM aggregate by key</name>
<certification>WFA</certification>
<parameters>
<parameter>name</parameter>
<parameter>node.cluster.name</parameter>
<parameter>node.name</parameter>
</parameters>
<dictionaryName>cm_storage.Aggregate</dictionaryName>
<ns2:link href="http://localhost/rest/filters/e6ee693c-57074936- bf27-
99d6c8b319db" rel="self"/>
<ns2:link href="http://localhost/rest/filters" rel="list"/>
<ns2:link href="http://localhost/rest/filters/e6ee693c-5707- 4936-bf27-
99d6c8b319db/test_no_reservations"rel="test_no_reservations"/>
<ns2:link href="http://localhost/rest/filters/e6ee693c-5707- 4936-bf27-
99d6c8b319db/test" rel="test"/>
</filter>
```

```
</filters>
<parameters>
<parameter>name</parameter>
<parameter>node.cluster.name</parameter>
<parameter>node.name</parameter>
</parameters>
<ns2:link href="http://localhost/rest/finders/915c8c3f-3863-4fef-
a198b35257d5de2d/test_no_reservations"rel="test_no_reservations"/>
<ns2:link href="http://localhost/rest/finders/915c8c3f-3863-4fef-
a198b35257d5de2d/test" rel="test"/>
<ns2:link href="http://localhost/rest/finders/915c8c3f-3863-4fef-
a198b35257d5de2d" rel="self"/>
<ns2:link href="http://localhost/rest/finders" rel="list"/>
</finder>
</collection>
```

The following is a sample JSON output of /rest/finders that returns a collection of exactly one finder:

```
[
 {
  "id": "915c8c3f-3863-4fef-a198-b35257d5de2d",
  "name": "Find CM aggregate in a given CM node",
  "dictionaryName": "cm_storage.Aggregate",
  "certification": "NETAPP",
  "filters": [
    {
      "id": "e6ee693c-5707-4936-bf27-99d6c8b319db",
      "name": "CM aggregate by key",
      "certification": "WFA",
      "parameters": [
        "name",
         "node.cluster.name",
         "node.name"
      ]
    }
  ],
  "parameters": [
    "name",
  "node.cluster.name",
  "node.name"
  ]
 }
]
```

FinderTestResults

The FinderTestResults object encapsulates the output of a finder test execution.

Name	**Type**	**Mandatory**	**Description**
Finder Name	String	True	Name of the finder.
Dictionary	String	True	Fully qualified name of the dictionary item
Name			that this filter is associated with.
Parameters	*KeyAndValuePair*	false	A collection of KeyAndValuePair items. Each item specifies an input parameter and the corresponding value that was passed by the client to test the filter.
Columns	Array of column	False	Contains column description of the output columns of the finder result.

Rows	Array of row	False	Array of rows. Each row contains exactly one output dictionary item that matches all the criteria within the finder.

The following is a sample XML output of FinderTestResults run on a finder that finds cluster mode aggregates based on name, node name and cluster name.

```
<finderTestResults>
<finderName>Find CM aggregate in a given CM node</finderName>
<dictionaryName>cm_storage.Aggregate</dictionaryName>
<parameters>
  <parameter value="f2040-181-51" key="node.name"/>
  <parameter value="aggr0" key="name"/>
  <parameter value="clus51_52_test" key="node.cluster.name"/>
</parameters>
<columns>
  <column>#</column>
  <column>node.name</column>
  <column>node.cluster.primary_address</column>
  <column>name</column>
</columns>
<rows>
  <row selected="true">
  <cell value="1" key="#"/>
  <cell value="aggr0" key="name"/>
  <cell value="10.72.181.75" key="node.cluster.primary_address"/>
  <cell value="f2040-181-51" key="node.name"/>
</row>
</rows>
</finderTestResults>
      "node.cluster.primary_address",
      "name"
  ],
  "rows": [
    {
      "cell": [
        {
          "key": "#",
          "value": "1"
        },
             {
          "key": "name",
          "value": "aggr0"
        },
             {
          "key": "#",
          "value": "1"
        },
             {
          "key": "node.cluster.primary_address",
          "value": "10.72.181.75"
        },
             {
          "key": "node.name",
          "value": "f2040-181-51"
        }
      ]
    }
  ]
}
```

The following is a sample JSON output of FinderTestResults run on a finder that finds cluster mode aggregates based on name, node name, and cluster name.

The following table shows a sample XML output of FinderTestResults run on a finder that tries to find a cluster mode aggregate based on name, node name, and cluster name of a cluster name that does not exist.

```
<finderTestResults>
  <finderName>Find CM aggregate in a given CM node</finderName>
  <dictionaryName>cm_storage.Aggregate</dictionaryName>
<parameters>
  <parameter value="node1" key="node.name"/>
  <parameter value="aggr1" key="name"/>
  <parameter value="clus1" key="node.cluster.name"/>
</parameters>
  <columns/>
  <rows/>
  <reasonForNoResult>
No results were found. The following filters have returned empty results:
CM aggregate by key
 </reasonForNoResult>
</finderTestResults>
```

The following table shows a sample JSON output of FinderTestResults run on a finder that tries to find a cluster mode aggregate based on name, node name, and cluster name of a cluster name that does not exist.

```
{
    "finderName": "Find CM aggregate in a given CM node",
    "dictionaryName":"cm_storage.Aggregate",
    "parameters": [
     {
      "key": "node.name",
      "value": "node1"
     },
        {
      "key": "name",
      "value": "aggr1"
     },
        {
      "key": "node.cluster.name",
      "value": "clus1"
     }
    ],
    "columns": [
    ],
    "rows": [
    ],
    "reasonForNoResult": "No results were found. The following filters have returned empty results:
   CM aggregate by key"
}
```

Finder links, relations, and actions (only for XML)

The following table shows the links and relations of a finder object.

Relation	Description	HTTP Request	Request Type	Response Type
list	Lists all the finders. Query parameters can be used to restrict the list based on a particular dictionary item.	GET	NA	*Finder[]*
self	Returns this specific finder only.	GET	NA	*Finder*

test	Tests the finder and returns results. The results include reservation data. **Query Parameters** <parameter>=<value>: The test finder expects the client to specify the input parameters to the finder as query parameters. See the following example: http://localhost/rest/finders/915c8c 3f-3863-4fef-a198-b35257d5de2d/test?name=aggr0&node.cluster.name=clus51_52_test&node.name=f2040-181-51	GET	NA	*FinderTestRes ult s*
test_no_res ervation	Tests the finder without applying RESERVATION data. This action also supports query parameters exactly as the 'test' action shown in the row above.	GET	NA	*FinderTestRes ult s*

HTTP error codes

Status Code	**Response**
Valid codes Success Codes: 200 OK Error Codes 400 Bad request 401 Unauthorized 403 Forbidden 404 Not found 405 Method Not Allowed 415 Unsupported Media Type 500 Server error	See *Appendix B* on HTTP status codes and detailed error messages for a detailed description of these error codes and when they will be returned.

Remote System Types

The Remote System Types API allows WFA API clients:

- Get information about existing remote system type(s)

Example

```
An HTTP GET on the following URL returns all remote system types.
https://localhost/rest/remote_system_types

An HTTP GET on the following URL returns the specific remote system type
with the given uuid. https://localhost/rest/remote_system_types/3e4e2811-
1b60-42ad- 8eb8d8250d182166
```

Security

Only users with the admin role or the architect role can use the Remote System Types API.

If operators or guest users attempt to use this API, the server returns a ‘403 Forbidden’.

Any unauthorized access (i.e either incorrect or missing credentials) will be challenged with a ‘401 Unauthorized’ by the server.

Remote System Type data structures

This section provides a description of all the major data structures exposed through the remote system types API.

Note: To obtain the corresponding XML schema definition and sample XML documents representing these data structures, see the *Reference Manual* that Sis available from within the product.

Remote System Type

The Remote System Type object provides details of a single remote system type. The following table describes the information encapsulated within this object.

Name	**Type**	**Mandatory**	**Description**
Uuid	String	True	ID that uniquely identifies the resource instance.
Name	String	True	Name of the remote system type.
Description	String	True	Description of the remote system type.
Version	Version	True	Version of the remote system type and includes elements for major, minor, and revision.

Certification	Enum	True	Certification of the remove system type entry. Can be any one of the following: • NONE • PS • COMMUNITY • USER_LOCKED • NETAPP
Connection protocol	Enum	True	Connection protocols to be used for connecting to the remote system. Can be one of the following: • HttpsToHttp • HttpsOnly • HttpOnly • Others
Protocol details	Array of Protocol detail	True	List of Protocol detail entries (in order driven by Connection protocol). Each Protocol detail entry has the following fields: • Protocol • defaultPort
			defaultTimeout
Hyper media links (only for XML)	Array of atom links	False	A collection of atom links that specifies available resources and their links. Allows clients to do hypermedia traversal. Each of these URLs and the associated methods are described in this document. This entry is read-only for the client.

The following is sample XML containing the remote system type object

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<remoteSystemType xmlns:atom="http://www.w3.org/2005/Atom" uuid="3e4e2811

{
  "uuid": "3e4e2811-1b60-42ad-8eb8-d8250d182166",
  "name": "Data ONTAP Systems",
  "description": "System type for Data ONTAP Systems.",
  "version": {
    "major": 1,
    "minor": 0,
    "revision": 0
  },
  "certification": "NETAPP",
  "connectionProtocol": "HttpsToHttp",
  "protocol-details": [
    {
      "protocol": "HTTPS",
      "defaultPort": 443,
      "defaultTimeout": 60
    },
       {
      "protocol": "HTTP",
      "defaultPort": 80,
      "defaultTimeout": 60
    }
  ]
}

8eb8d8250d182166"/>
 <atom:link rel="list" href="https://localhost/rest/remote_system_types"/>

</remoteSystemType>
```

The following is sample JSON containing the remote system type object:

Links and relations (only for XML)

Relation	Description	HTTP Request	Request Type	Response Type
List	Lists all remote system type objects.	GET	NA	*Remote System Type[]*
self	Returns the specific remote system type only.	GET	NA	*Remote System Type*

HTTP error codes

Status Code	Response
Valid codes Success Codes: 200 OK Error Codes 400 Bad request 401 Unauthorized 403 Forbidden 404 Not found 405 Method Not Allowed 415 Unsupported Media Type 500 Server error	See *Appendix B* on HTTP status codes and detailed error messages for a detailed description of these error codes and when they will be returned.

Credentials

The Credentials API allows WFA API clients to:

- Get information about existing credential(s)
- Create and delete new credentials
- Test credentials

Query parameters to filter collection of credentials

Parameter	Value	Description
Type	ConnectionType	Filter credential collection based on a connection type. Connection type can be any one of the following: • ONTAP • DFM • VIRTUAL_CENTER • OTHER This field is deprecated and query parameter 'remoteSystemType' must be used.
RemoteSystemType	String	Filter credential collection based on the given remote system type name. Remote system type name must be one of the valid remote system types existing in the WFA system.

Query parameter example

```
An HTTP GET on the following URL returns all credentials.
https://localhost/rest/credentials
An HTTP GET on the following URL returns all credentials for ONTAP systems
https://localhost/rest/credentials?type=ONTAP
An HTTP GET on the following URL returns all credentials with remote system
type Data ONTAP Systems
https://localhost/rest/credentials?remoteSystemType=Data%20ONTAP%20Systems
```

Security

Only users with the admin role or the architect role can use the credentials API. If operators or guest users attempt to use this API, the server returns a '403 Forbidden'.

Any 'unauthorized access' (i.e., incorrect or missing credentials) will be challenged with a '401 Unauthorized' by the WFA server.

Since credentials are very sensitive data, it is recommended that an HTTPS connection is configured before using this API, especially if the 'create' credential API is used.

Credential data structures

This section provides a description of all the major data structures exposed through the credentials API.

Note: To obtain the corresponding XML schema definition and sample XML documents representing these data structures, see the *Reference Manual* that is available in the product.

Credential

The Credential object represents access information to a single device. The following table describes the information encapsulated within this object.

Name	Type	Mandatory	Description
IP	String	True	IP address of the host. The host can be an ONTAP system, VMware vCenter or any other data center resource.
Connection Type	Enum	True	Type of connection. Can be any one of the following: ONTAP • DFM • VIRTUAL_CENTER • OTHER This field is deprecated with the introduction of remote system type.
Match Type	Enum	True	Match type for the IP address. Could be any one of the following: • EXACT • PATTERN EXACT means the IP is an exact match. PATTERN means IP is a regular expression that can be used to match a range of IP addresses.
Remote system type	String	True	Name of the remote system type attached to the Credential. Remote system type provides connection details (port, protocol, and timeout) for the given remote system.
Connection protocol details	Array of connection protocol detail entries.	True	This field includes the connection details for the credential object as per the corresponding remote system type. A connection protocol detail entry includes protocol, connectionPort, and connectionTimeout values.
Name	String	False	Host name of the entity.
User Name	String	False	User name of the account that will be accessed by WFA to access the device or host.
Hyper media Links (only for XML)	Array of Atom links	False	A collection of atom links that specifies available resources and their links. Allows clients to do hypermedia traversal. Each of these URLs and the associated methods are described in this document. This entry is read-only for the client.

The following is sample XML containing the credential object.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<credential xmlns:atom="http://www.w3.org/2005/Atom">
    <ip>1.1.1.1</ip>
<connectionType>ONTAP</connectionType><remoteSystemType>Data ONTAP
Systems</remoteSystemType>

    <connection-protocol-details>
        <connection-protocol-detail>
               <connectionPort>443</connectionPort>
               <connectionTimeout>60</connectionTimeout>
               <protocol>HTTPS</protocol>
         </connection-protocol-detail>
         <connection-protocol-detail>
               <connectionPort>80</connectionPort>
               <connectionTimeout>60</connectionTimeout>
               <protocol>HTTP</protocol>
         </connection-protocol-detail>
  </connection-protocol-details>
    <matchType>EXACT</matchType>
    <name>name_1</name>
    <userName>user_name_1</userName>     <atom:link rel="test"
href="http://localhost:9095/rest/credentials/1.1.1.1/test"/>
    <atom:link rel="self"
href="http://localhost:9095/rest/credentials/1.1.1.1"/>
</credential>
<atom:link rel="remote-system-type"
href="http://localhost:9095/rest/remote_system_types/3e4e2811-1b60-42ad8eb8-
d8250d182166"/>
```

The following is sample JSON containing the credential object:

```
{
  "ip": "1.1.1.1",
  "connectionType": "ONTAP",
  "remoteSystemType": "Data ONTAP Systems",
  "matchType": "EXACT",
  "name": "name_1",
  "userName": "user_name_1"
  "connection-protocol-details": [
    {
      "protocol": "HTTPS",
      "connectionPort": 443,
      "connectionTimeout": 60
    },
    {
      "protocol": "HTTP",
      "connectionPort": 80,
      "connectionTimeout": 60
    }
  ]
}
```

CredentialWithPassword

The ‘CredentialWithPassword’ is a credential object described above along with an additional string field that represents the password of the device or host. It is only used as input when creating new credentials. It is not used as output, as the API will not let out password of any credentials that are already stored within the system.

Links and relations (only for XML)

Relation	Description	HTTP Request	Request Type	Response Type

list	Lists all credential objects. Query parameters can be used to restrict the list based on a particular connection type.	GET	NA	*Credential[]*
self	Returns this specific credential only.	GET	NA	*Credential*
test	Tests the credential and returns the credential objects if the test was successful. **Query Parameters** testIP=<value>: The IP address of the device or host against which the credential has to be tested.	GET	NA	*Credentiall*
	http://localhost/rest/10.72.76./test?testIp= 10.72.76.76*			
add	Creates a new credential object.	POST	*Credential WithPassw ord*	*Credential*
Remove	Removes the credential specified by name or IP address.	DELETE	NA	NA
remotesystemtype	Lists the remote system type for the given credential.	GET	NA	*Remote System Type*

HTTP error codes

Status Code	Response
Valid codes Success Codes: 200 OK Error Codes 400 Bad request 401 Unauthorized 403 Forbidden 404 Not found 405 Method Not Allowed 415 Unsupported Media Type 500 Server error	See *Appendix B* on HTTP status codes and detailed error messages for a detailed description of these error codes and when they will be returned.

Data Source and Data Source Types

The Data source API allows the client to perform the following:

- Get information about existing data source types on the server
- Get information about existing data sources on the server
- Perform an immediate acquisition on a specific data source
- Get information on the status of a specific acquisition job

```
The root URIs for these resources is /rest/data_sources and
/rest/data_source_types.
```

Security

All the resources under /rest/data_sources and /rest/data_source_types are visible only for admins and architects.

An HTTP '403 Forbidden' error code is returned for operators and guest users.

Any unauthorized access (i.e., incorrect or missing credentials) will be challenged with a '401 Unauthorized' by the WFA server.

Data structures

Data source type

Name	Type	Description
Uuid	String	ID that uniquely identifies this resource instance.
Certification	Enum	NONE/PS/COMMUNITY/USER_LOCKED/NETAPP
productType	String	for example, "onCommand unified manager (DFM)"
productVersion	String	for example, 5.1.X Cluster-Mode
dataSourceDriver	String	for example, Sybase jConnect 3.0 (if not null)
Links (only for XML)	Array of atom links	Links to relations and actions that are navigable from this resource.
Scheme	String	Scheme of the data source type.
Method	String	Indicates whether the data source type is of SQL or SCRIPT type.
DefaultDatabaseName	String	Name of the default database.
Version	Version	Version of the data source type. Provides major, minor, and revision numbers.
DefaultPort	int	NA
Instructions	String	Setup instructions to configure the data source.

The following sample XML illustrates the dataSourceType.

```
<dataSourceType uuid="6ee5e2a9-dacf-445d-a700-37d346526809">
 <certification>NETAPP</certification>
 <productType>OnCommand Unified Manager</productType>
 <productVersion>5.2.X for Clustered Data ONTAP</productVersion>
<dataSourceDriver>Sybase jConnect3</dataSourceDriver>
 <version>
        <major>1</major>
        <minor>0</minor>
        <revision>0</revision>
 </version>
 <defaultPort>2638</defaultPort>
 <scheme>cm_storage</scheme>
 <method>SQL</method>
 <instructions>For Windows 1. Connect to the OnCommand Unified Manager host
using Remote Desktop Connection as a local admin. 2. On the OnCommand
Unified Manager host, perform the following: a. Download wfa_ocsetup.exe
from the following address to a temp folder:
http://localhost/download/wfa_ocsetup.exe b. Run wfa_ocsetup.exe while
providing the following details: c. Select a valid path to a Java Runtime
Environment (You may approve the suggested one if applicable) d. Provide
the username and password that will be created. e. Verify setup finished
successfully. For Linux 1. Connect to the OnCommand Unified Manager host
using SSH, logged in as root. 2. On the OnCommand Unified Manager host,
perform the following: a. Download wfa_ocsetup.sh from the following
address to a temp directory: # wget --user=<admin/architect user> -
password=<password> http://localhost/download/wfa_ocsetup.sh b. Modify the
permissions of wfa_ocsetup.sh to be executable:chmod +x wfa_ocsetup.sh c.
Run wfa_ocsetup.sh, providing a valid path to a Java Runtime Environment
(1.6 and above).Example: # ./wfa_ocsetup.sh /usr/bin/java d. Supply the
username and password when prompted to override the default credentials. e.
Verify setup finished successfully.</instructions>
 <atom:link rel="add-data-source"
href="https://localhost/rest/data_sources/type/6ee5e2a9-dacf-445d-
a70037d346526809"/>
 <atom:link rel="data-sources"
href="https://localhost/rest/data_sources/type/6ee5e2a9-dacf-445d-
a70037d346526809"/>
 <atom:link rel="list" href="https://localhost/rest/data_source_types"/>
 <atom:link rel="self"
href="https://localhost/rest/data_source_types/6ee5e2a9-dacf-445d-
a70037d346526809"/>
</dataSourceType><dataSourceType uuid=”6dee7cb0-c411-4aa1-81b1-
70ac32d2af8a”>
  <cetification>WFA</certification>
  <productType>OnCommand Unified Manager (DFM)</productType>
  <productVersion>5.1.X Cluster-Mode</productVersion>
  <dataSourceDriver>Sybase jConnect 3.0</dataSourceDriver>
  <ns2:link href="http://localhost/rest/data_source_types" rel="list
<ns2:link href="http://localhost/rest/data_source_types/{uuid}" rel="self"/>
  <ns2:link href=".../rest/data_source/type/6dee7cb0-c411-4aa1-81b1-
70ac32d2af8a rel="data-sources"/></dataSourceType>
```

The following sample JSON illustrates the dataSourceType:

```
{
"uuid": "6ee5e2a9-dacf-445d-a700-37d346526809", "certification": "NETAPP",
"productType": "OnCommand Unified Manager", "productVersion": "5.2.X for Clustered
Data
ONTAP</productVersion>   <dataSourceDriver>Sybase jConnect3",
"dataSourceDriver": "Sybase jConnect 3.0", "version": {
"major": 1,
"minor": 0,
"revision": 0
},
"defaultPort": 2638, "scheme": "cm_storage", "method": "SQL",
"instructions": "For Windows 1. Connect to the OnCommand Unified Manager host using
Remote Desktop Connection as a local admin. 2. On the OnCommand Unified Manager host,
perform the following: a.
Download wfa_ocsetup.exe from the following address to a temp folder:
http://localhost/download/wfa_ocsetup.exe b. Run wfa_ocsetup.exe while providing the
following details: c. Select a valid path to a Java Runtime Environment (You may
approve the suggested one if applicable) d. Provide the username and password that
will be created. e. Verify setup finished successfully. For Linux 1. Connect to the
OnCommand Unified Manager host using SSH, logged in as root.
2. On the OnCommand Unified Manager host, perform the following: a. Download
wfa_ocsetup.sh from the following address to a temp directory: # wget --
user=<admin/architect user> -password=<password>
http://localhost/download/wfa_ocsetup.sh b. Modify the permissions of wfa_ocsetup.sh
to be executable:chmod +x wfa_ocsetup.sh c. Run wfa_ocsetup.sh, providing a valid path
to a Java Runtime Environment
(1.6 and above).Example: # ./wfa_ocsetup.sh
/usr/bin/java d. Supply the username and password when prompted to override the
default credentials. e.
Verify setup finished successfully."
}
```

Data source

Name	Type	Description
Name	String	Name of the data source
Schema	String	Storage/cm_storage
Type	String	String representation of the data source type that the data source is using
typeUuid	String	UUId of the data source type
Ip	String	Data source ip
Port	Int	Data source port
Interval	Int	Acquisition Interval (minutes)
links (only for XML)	Array of atom links	Links specifying relations and actions that are possible for this resource.

The combination of the name and the schema is the data source ID.
The following is a sample XML data source.

```
 <dataSource name="DFM 234" schema=”storage”>
   <type>OnCommand Unified Manager (DFM) - 4.0 , 5.0.X,5.1.X 7 Mode
 (SYBASE)</type>
   <typeUuid>6dee7cb0-c411-4aa1-81b1-70ac32d2af8a</typeUuid>
   <ip>10.68.66.234</ip>
   <port>2638</port>
   <interval>30</interval>
   <ns2:link href="http://localhost/rest/data_sources" rel="list"/>
 <ns2:link href="http://localhost/rest/data_sources/DFM%20234/storage" rel="
 self"/>
<ns2:link href="http://localhost/rest/data_sources/DFM%20234/storage/jobs"
rel="acquire"/>
   <ns2:link href="http://localhost/rest/data_source_type/6dee7cb0-c411-
 4aa1-81b1-70ac32d2af8a" rel="data-source-type"/>
 </dataSource>
```

Note: “self” and “acquire” have the same URL. “self” uses GET and “acquire” uses POST.

The following is a sample JSON data source:

```
{
  "name": " DFM 234",
  "type": " OnCommand Unified Manager (DFM) - 4.0 , 5.0.X,5.1.X 7 Mode
(SYBASE)",
  "ip": "10.68.66.234",
  "port": 2638,
  "schema": " storage",
  "interval": 30
}
```

Acquisition job

Name	Type	Description
Id	Int	The job id
Data Source	Data Source Object	The data source against which the acquisition job is performed.
plannedExecution	Date	Job planned execution time
startTime	Date	Job start time
Duration	Int	Job duration in seconds
scheduleType	String	Immediate/Recurring
Status	Enum	Failed/Canceled/…
errorMessage	String	Error message if exists

Note: “startTime”,” duration”,” plannedExecution”,” scheduleType”,”status”, and “message” will be warped by “jobStatus” element.

The following shows a sample XML of acquisition job.

```
<acquisitionJob xmlns:atom="http://www.w3.org/2005/Atom" jobId="248">
<jobStatus>
```

```
<plannedExecution>Jan 28, 2016 12:09:35 PM</plannedExecution>
<scheduleType>Immediate</scheduleType>
<status>SCHEDULED</status>
</jobStatus>
<dataSource name="wfa-ocum-4" schema="cm_storage">
<type>OnCommand Unified Manager - 6.3 (MYSQL)</type>
<ip>wfa-ocum-4</ip>
<port>3306</port>
<interval>30</interval>
<atom:link rel="acquire"
href="http://localhost/rest/data_sources/wfa-ocum-4/cm_storage/jobs"/>
<atom:link rel="acquire-data-source-by-name"
href="http://localhost/rest/data_sources/wfa-ocum-4/jobs"/>
<atom:link rel="list" href="http://localhost/rest/data_sources"/>
<atom:link rel="edit-data-source" href="http://localhost/rest/data_sources/wfa-ocum-
4"/>
<atom:link rel="remove-data-source" href="http://localhost/rest/data_sources/wfa-ocum-
4"/>
<atom:link rel="last_acquisition_jobs_by_type"
href="http://localhost/rest/data_sources/type/ddc7d0f2-2474-44f0-
b89bf26902b4872a/jobs"/>
<atom:link rel="self" href="http://localhost/rest/data_sources/wfaocum-4/cm_storage"/>
<atom:link rel="data-source-type"
href="http://localhost/rest/data_source_types/ddc7d0f2-2474-44f0- b89bf26902b4872a"/>
     </dataSource>
<atom:link rel="acquisition-job-by-name" href="http://localhost/rest/data_sources/wfa-
ocum-4/jobs/248"/>
<atom:link rel="self" href="http://localhost/rest/data_sources/wfaocum-
4/cm_storage/jobs/248"/>
</acquisitionJob>
```

The following shows a sample JSON of acquisition job:

```
{
  "jobId": "248",
  "jobStatus": {
    "plannedExecution": "Jan 28, 2016 12:09:35 PM", "scheduleType":
    "Immediate",
    "status": "SCHEDULED"
  },
  "dataSource": {
    "name": "wfa-ocum-4",
    "type": "OnCommand Unified Manager - 6.3 (MYSQL)", "ip":
    "wfa-ocum-4",
    "port": 3306,
    "schema": "cm_storage", "interval":
    30
  }
}
```

Links, relations, and actions (only for XML)

The following table explains links, relations, and actions corresponding to data sources and data source types.

Relation	Description	HTTP Request	Request Type	Response Type

List	Lists all the data sources (/rest/data_sources) or data source types (/rest/data_source_types)	GET	NA	*Data* source*[]* OR *Data* source
Self	Returns this specific resource only.	GET	NA	*Data* source*[]* OR *Data* source
Acquire	Performs an immediate acquisition from the specified data source. On success, returns the acquisition job details that was started. The body of the request is ignored by the server. URL request: Using data source name and schema: (/data_sources/{name}/{schema}/jobs) or Using only data source name: (/data_sources/{name}/jobs)	POST	None	*Acquisition* job
data-sources	Lists all data sources for a given data source type. Relates data sources to data source types.	GET	NA	*Data* source*[]*
data-source-type	Lists the data source type of a given data source. Relates data source type to the data source.	GET	NA	*Data* source type

The following table shows the additional links returned in the 'AcquisitionJob' by the server to enable the client to get the status of the job.r

Links, relations, and actions (only for XML)

Relation	**Description**	**HTTP Request**	**Request Type**	**Response Type**
Self	Get the latest status of the acquisition job.	GET	NA	*Acquisition* job

Note: The status of the acquisition job can be obtained using either of the following URL requests:

- Using data source name and schema: /data_sources/{name}/{schema}/jobs/{job_id}

- using only data source name: /data_sources/{name}/jobs/{job_id}

HTTP error codes

Status Code	**Response**
Valid codes Success Codes: 200 OK Error Codes 400 Bad request 401 Unauthorized 403 Forbidden 404 Not found 405 Method Not Allowed 415 Unsupported Media Type 500 Server error	Refer to *Appendix B* on HTTP status codes and detailed error messages for a detailed description of these error codes and when they will be returned.

Schedules and recurring schedules

The Schedule API allows the client to perform the following tasks:

- Get information about existing schedules on the server
- Get information about existing schedule instances on the server
- Create, modify, and delete a schedule
- Suspend, resume, and delete a given schedule instance

The root URIs for these resources are `/rest/schedules` and `/rest/schedules/instances`.

Security

```
All resources under the
/rest/schedules and
/rest/schedules/instances root
```

URIs are visible to admins, architects, approvers, and operators.

An HTTP ‘403 Forbidden’ error code returns for guest users.

Any unauthorized access (that is, incorrect or missing credentials) is challenged with a ‘401 Unauthorized’ error code by the WFA server.

Query parameters to filter collection of schedule instances

Parameter	**Value**	**Description**
schedule_id	Schedule ID	Filter the schedule instances by the given schedule ID.

Data structures

Schedules

This section provides a description of all the major data structures exposed through the credentials API.

Note: To obtain the corresponding XML schema definition and sample XML documents representing these data structures, see the Reference Manual that is available with the product.

The following table describes the information encapsulated within this object:

Name	**Type**	**Mandatory**	**Description**
Id	String	False	Schedule ID
Name	String	True	Name of the recurring schedule
description	String	False	Description of the schedule.
minutes	String	False	Minutes in the range of 0 through 59.
hours	String	False	Hours in the range of 0 through 23
daysOfMonth	String	False	Days of a month in the range of 1 through 31

daysOfWeek	String	False	Days of a week in the range of 1 through 7

The following is a sample XML containing the schedule object:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<schedule xmlns:atom="http://www.w3.org/2005/Atom" id="376">
  <name>sch0001</name>
  <description>This is a schedule</description>
  <minutes>1</minutes>
  <hours>2</hours>
  <daysOfMonth>2</daysOfMonth>
  <daysOfWeek>?</daysOfWeek>
  <atom:link rel="list" href="http://localhost:90/rest/schedules"/>
  <atom:link rel="add" href="http://localhost:90/rest/schedules"/>
  <atom:link rel="edit" href="http://localhost:90/rest/schedules/376"/>
  <atom:link rel="remove" href="http://localhost:90/rest/schedules/376"/>
  <atom:link rel="self" href="http://localhost:90/rest/schedules/376"/>
  </schedule>
```

The following is a sample JSON containing the schedule object:

```
{
  "id": 376,
  "name": "sch0001",
  "description": "This is a schedule",
  "minutes": "1",
  "hours": "2",
  "daysOfMonth": "2",
  "daysOfWeek": "?"
  }
```

Links, relations, and actions (only for XML)

The following table describes links, relations, and actions corresponding to schedules:

Relation	**Description**	**HTTP request**	**Request**	**Response type**
List	Lists all the schedules /rest/schedules	GET	NA	*Schedules[ ]*
Add	Add a schedule /rest/schedules	POST	NA	*Schedules[]*
Edit	Edit a schedule /rest/schedules/<id>	PUT	NA	*Schedules[]*
Remove	Delete a schedule /rest/schedules/<id>	DELETE	NA	No content
Self	The given schedule /rest/schedules/<id>	GET	NA	*Schedules[ ]*

Recurring schedules

The following table describes links, relations, and actions corresponding to schedules:

Name	**Type**	**Mandatory**	**Description**
scheduleInstance	String	True	Instance ID
scheduleName	String	True	Name of the recurring schedule
scheduleId	String	False	Schedule ID
workflowName	String	False	Workflow name
workflowUuid	String	False	Workflow UUID
userInputValues	Array	False	User-input values in the given schedule instance
status	String	False	Status of the workflow: ACTIVE SUSPENDED

The following is a sample XML schedule instance:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<scheduleInstance xmlns:atom="http://www.w3.org/2005/Atom" id="3">
  <scheduleName>sch-hourly</scheduleName>
  <scheduleId>2</scheduleId>
  <workflowName>Workflow to Print a Footballer Name - delete it1 -
copy</workflowName>
  <workflowUuid>9f2b07fb-a0c1-4075-b660-5f6ab152630b</workflowUuid>
  <userInputValues>
    <userInputEntry key="$Country" value="Sweden"/>
    <userInputEntry key="$const" value="Adaptive Qos Service Manager"/>
  </userInputValues>
  <status>ACTIVE</status>
  <atom:link rel="resume"
href="http://localhost:90/rest/schedules/instances/3/resume"/>
  <atom:link rel="self"
href="http://localhost:90/rest/schedules/instances/3"/>
  <atom:link rel="remove"
href="http://localhost:90/rest/schedules/instances/3"/>
  <atom:link rel="list"
href="http://localhost:90/rest/schedules/instances"/>
  <atom:link rel="suspend"
href="http://localhost:90/rest/schedules/instances/3/suspend"/>
</scheduleInstance>
```

The following is a sample JSON schedule instance:

```
{
  "id": 3,
  "scheduleName": "sch-hourly",
  "scheduleId": 2,
  "workflowName": "Workflow to Print a Footballer Name - delete it1 - copy",
  "workflowUuid": "9f2b07fb-a0c1-4075-b660-5f6ab152630b",
  "userInputValues": [
    {
      "key": "$Country",
      "value": "Sweden"
    },
    {
      "key": "$const",
      "value": "Adaptive Qos Service Manager"
    }
  ],
  "status": "ACTIVE"
}
```

Links, relations, and actions (only for XML)

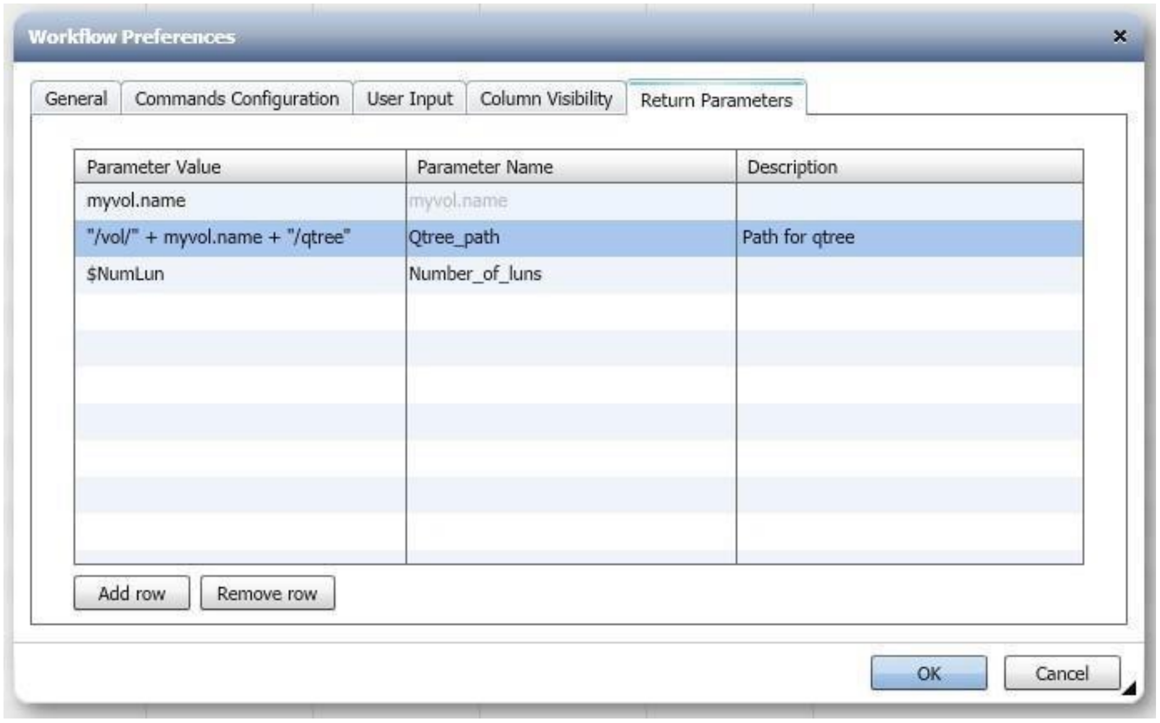
The following table describes links, relations, and actions corresponding to recurring schedules:

Relation	**Description**	**HTTP request**	**Requ est**	**Response type**
List	Lists all the schedule instances	GET	NA	ScheduleInstance []
Self	The schedule instance /rest/schedules/instances/<id>	GET	NA	ScheduleInstance[]

Suspend	Suspend a given schedule instance /rest/schedules/instances/<id>/suspend	POST	NA	No Content
Resume	Resume a given schedule instance /rest/schedules/instances/<id>/resume	POST	NA	No Content
Delete	Delete a given schedule instance /rest/schedules/instances/<id>	DELETE	NA	No Content

Appendix A – Return Parameters – Feature Description

This feature allows designating a set of parameters, such as Variable attributes, expressions, user input values in a workflow, and to retrieve the values for the defined parameters on request.
The designation is done in the Return Parameters tab in workflow preferences:



Parameter Value	Parameter Name	Description
myvol.name	myvol.name	
"/vol/" + myvol.name + "/qtree"	Qtree_path	Path for qtree
$NumLun	Number_of_luns	

Start by adding a row and determine which value you wish to be returned. It receives a label automatically, identical to the parameter value column, which appears in light gray. You might change it to match any label you like and even add a short description.

Note: When executing the workflow, the values are populated as soon as the planning phase is completed and execution commences. It is crucial to test the workflow execution status and confirm its completion before addressing the values of the return parameters.

These values are set per execution. If, after several executions, another parameter is added, that parameter value would be available from that point onwards only, and not in any execution prior to the addition of new parameter.

Appendix B – Detailed Error Messages and Status Codes

Error	Description	Remarks
200 OK	This is returned by WFA service if the requested verb on the requested resource was successfully processed by WFA.	The response contains the representation of the requested resource in.
201 CREATED	This is returned by WFA if a new child resource was created. This is typically returned in response to a POST, which is the only verb that represents a non-idempotent operation in HTTP. This is returned only for POST operations that actually create a child resource and return its representation.	The response contains the resource representation of the created resource.
204 NO CONTENT	This will be returned by WFA for operations that were successful but a matching resource was not found for the specific URI and query parameter combination.	
401 Unauthorized	This is returned if the client is not authenticated OR if the client does not have the privilege to perform the requested operation on the requested resource (job or workflow).	
400 Bad Request	This is returned if the input specified by the client is invalid. Ex. specifying a non-existing workflow UUID OR specifying a non-existing row in a plan.	
500 Server error	This is returned if an internal server error caused the server to abort the requested operation on the resource.	The internal error could include but is not limited to server side problems, such as network errors, DB connectivity issues, lack of resources to complete the requested operation, and so on.

STATUS CODE	Resource URI and method	Error Message	Description	Example
	Workflow Collection			
400	GET /rest/workflows?categories=examples	Category name <name> does not exist.	The provided category name does not exist.	Category name: An example does not exist.
400	GET /rest/workflows?categories=examples	Current user is not authorized for any of the given categories.	The user is an operator and is not authorized for any of the categories given as parameter.	Current user is not authorized for any of the given categories.
404	All operations on resources that match the following url: /rest/workflows/{workflow_uuid}/*	No workflow found for uuid: <uuid>	The provided workflow uuid does not exist.	No workflow found for id: some_uuid
403	POST /rest/workflows/{workflow_uuid}/jobs	Current user is not allowed to execute workflow.	The user is an operator and not assigned to the workflow category.	Current user David is not allowed to execute workflow 6.
403	POST /rest/workflows/{workflow_uuid}/jobs	Current user is not allowed to execute workflow since it is not ready for production yet.	The user is an operator and this workflow is not “ready for production”.	Current user David is not allowed to execute workflow 6 as it is not ready for production yet.
400	POST /rest/workflows/{workflow_uuid}/jobs	Got incorrect date and time format input.	The date format is not valid.	Got incorrect date and time format '234'. Correct format 'M/d/yy h:mm a', for example: 2/14/12 3:28 PM.
400	POST /rest/workflows/{workflow_uuid}/jobs	Cannot schedule a workflow in the past.	The date for the workflow is in the past.	Cannot schedule a workflow in the past.

400	POST /rest/workflows/{workflow_uuid}/jobs POST /rest/workflows/{workflow_uuid}/preview	Invalid user-input and value, should be: name=value	The user input value was not provided in the right format.	Invalid user-input and value volumeName, should be: name=value
400	POST /rest/workflows/{workflow_uuid}/jobs POST /rest/workflows/{workflow_uuid}/preview	The values for <user_input_key> have to fit <query>	When a user input is defined as a locked query and the value does not match the query.	The values for array_ip have to fit select array_ip from wfa.array
400	POST /rest/workflows/{workflow_uuid}/jobs POST /rest/workflows/{workflow_uuid}/preview	The values for <user_input_key> have to be within <enum list>	When a user input is defined as a locked Enum and the value is not one of its values.	The values for size have to be within 30, 40, and 50.
400	POST /rest/workflows/{workflow_uuid}/jobs POST /rest/workflows/{workflow_uuid}/preview	The values for <user_input_key> have to be between <range>	When a user input is defined as a Number and the value is not in the defined range.	The values for age should be between 10 and 50.
400	POST /rest/workflows/{workflow_uuid}/jobs POST /rest/workflows/{workflow_uuid}/preview	The values for <user_input_key> must match the regular expression: <expression>	When a user input is defined as a String and the value does not match the regular expression.	The values for ip must match the regular expression: [^]
400	POST /rest/workflows/{workflow_uuid}/jobs POST /rest/workflows/{workflow_uuid}/preview	User input <user_input_key> is not defined in workflow <workflow name>	When a provided user input is not defined for the workflow.	User input name is not defined in workflow Create Vfiler.

404	All operations on resources with URI /rest/workflows/{workflow_uuid}/jobs/{job_id}/*	Workflow execution Id <job_id> was not found.	The provided job Id was not found.	Workflow execution Id 12 was not found.
403	POST /rest/workflows/{workflow_uuid}/jobs/{job_id}/resume	current user <user_name> is not allowed to resume workflow <job_id>	The user is an operator and not assigned to the workflow category.	Current user David is not allowed to resume workflow 6.
400	POST /rest/workflows/{workflow_uuid}/jobs/{job_id}/resume	Could not resume workflow execution with id <job_id>. Resume is only allowed from statuses: PAUSED, CANCELED, FAILED, and SCHEDULED.	Resume is only possible for jobs that are in statuses: PAUSED, CANCELED, FAILED, and SCHEDULED.	Could not resume workflow execution with id 213. Resume is only allowed from statuses: PAUSED, CANCELED, FAILED, and SCHEDULED.
400	POST /rest/workflows/{workflow_uuid}/jobs/{job_id}/resume	Resume of workflow execution with id is not allowed. Workflow execution is expired - more than the days have passed since it stopped running.	This job cannot be resumed since its reservation has already expired and the results are not predictable.	Resume of workflow execution with id 105 is not allowed. Workflow execution has expired - more than 2 days have passed since it stopped running.
403	POST /rest/workflows/{workflow_uuid}/jobs/{job_id}/resume	Users with role <role_name> are not allowed to resume workflow executions.	When an operator tries to resume a job which is in status paused and the configuration forbids it.	Users with operator role are not allowed to resume workflow executions.
400	POST /rest/workflows/{workflow_uuid}/jobs/{job_id}/resume	Workflow execution with id cannot be resumed because it was canceled before it completed its planning.	If the workflow was canceled before the planning was done then the workflow cannot be resumed.	Workflow execution with id 213 cannot be resumed because it was canceled before it completed its planning.

403	POST /rest/workflows/{workflow_uuid}/jobs/{job_id}/cancel	current user <user_name> is not allowed to reject workflow.	The user is an operator and not assigned to the workflow category.	Current user David is not allowed to reject workflow 6.
403	POST /rest/workflows/{workflow_uuid}/jobs/{job_id}/cancel	Users with role <role_name> are not allowed to reject workflow executions.	When an operator tries to reject a job which is in status paused and the configuration forbids it.	Users with operator role are not allowed to reject workflow executions.
400	GET /rest/workflows/{workflow_uuid}/jobs/{job_id}/plan/out	The job status data can be retrieved only in the following statuses: COMPLETED, FAILED, PARTIALLY_SUCCESSFUL, and CANCELED	Return parameters can be viewed only after the job finished running that is in: COMPLETED, FAILED, PARTIALLY_SUCCESSFUL, and CANCELED.	The job status is RUNNING, data can be retrieved only in the following statuses: COMPLETED, FAILED, PARTIALLY_SUCCESSFUL, and CANCELED.
400	GET /rest/workflows/{workflow_uuid}/jobs/{job_id}/plan/out	Parameter <parameter_name> was not yet calculated, check the execution result.	One of the parameters was not yet calculated (should not happen).	Parameter volumeName was not yet calculated, check the execution result.
400	GET /rest/workflows/{workflow_uuid}/jobs/{job_id}/plan/out?paramterName=<parameter_name>	Parameter not found. It's not defined as a return parameter.	The given return parameter was not defined for this workflow.	Parameter volumeName not found. It's not defined as a return parameter.
400	GET /rest/workflows/{workflow_uuid}/jobs/{job_id}?waitInterval=-1	Cannot wait on job with status: <status>	The job is currently not in a status that allows waiting for it to complete.	Cannot wait on job with status: COMPLETED.

400	GET /rest/workflows/{workflow_uid}/jobs/{job_id}?waitInterval=-1	Cannot wait for executions that are not scheduled to run in the next 24 hours.	The job is scheduled to run in the near future (24 hours).	Executing 'waitForCompletion' with a jobId that is scheduled to run in a week from today will return the following error: 'Cannot wait for executions that are not scheduled to run in the next 24 hours'.
	User Collection			
403	/rest/users	Current user is not allowed to retrieve all users.	The current user is not of role 'Admin' and therefore not allowed to retrieve all users.	User 'operator' is not allowed to retrieve all users.
400	/rest/users/{user_name}/password?oldPassword={oldPassword}&newPassword={newPassword}	Unable to change password because old password is empty.	Old password given as parameter is either null or an empty string.	Unable to change password because old password is empty.
400	/rest/users/{user_name}/password?oldPassword={oldPassword}&newPassword={newPassword}	Unable to change password because new password is empty.	New password given as parameter is either null or an empty string.	Unable to change password because new password is empty.
400	/rest/users/{user_name}/password?oldPassword={oldPassword}&newPassword={newPassword}	Unable to change password because old password does not match existing password.	Old password given as parameter does not match the password in the database.	Unable to change password because old password does not match existing password.
403	/rest/users/{user_name}/password?oldPassword={oldPassword}&newPassword={newPassword}	LDAP users are not allowed to change their passwords.	The current user that is trying to change his password is an LDAP user.	LDAP users are not allowed to change their passwords.

Copyright

Trademark

How to send comments about documentation and receive update notifications

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

If you have suggestions for improving this document, send us your comments by email to *doccomments@netapp.com*.

To help us direct your comments to the correct division, include in the subject line the product name, version, and operating system.

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

You can also contact us in the following ways:

- NetApp, Inc., 1395 Crossman Ave., Sunnyvale, CA 94089 U.S.
- Telephone: +1 (408) 822-6000
- Fax: +1 (408) 822-4501
- Support telephone: +1 (888) 463-8277