1. RESTful API v4.0 User Guide ................................................................. 3
  1.1 Overview .......................................................... 4
  1.2 Domain Filters ............................................ 6
  1.3 Domain Filter details ................................... 11
  1.4 Event Filters .................................................. 18
  1.5 Event Filter details ....................................... 22
  1.6 Incident Filters ............................................ 27
  1.7 Incident Filter details ................................... 31
  1.8 Events ........................................................... 35
  1.9 Event types .................................................... 41
  1.10 Incidents ....................................................... 43
  1.11 Incident details ............................................ 47
  1.12 Incident types .............................................. 49
  1.13 Inventory Summary ....................................... 51
  1.14 Inventory details .......................................... 61
  1.15 Product Info ................................................... 71
  1.16 Servers .......................................................... 73
  1.17 Servers details .............................................. 75
  1.18 User Groups .................................................... 77
  1.19 User Group details ....................................... 81
  1.20 User Group permissions .................................. 83
  1.21 Tools ............................................................... 86
  1.22 Users ............................................................ 89
  1.23 User details .................................................... 92
  1.24 View List ....................................................... 98
  1.25 View details .................................................. 103
  1.26 View Objects .................................................. 107
  1.27 Object Attributes .......................................... 110
  1.28 Object Attribute details ............................... 113
  1.29 Object Ports ................................................... 117
  1.30 Object Associations ....................................... 123
  1.31 Association details ........................................ 126
  1.32 Configuration Management ................................ 129
  1.33 Port Management ............................................. 135
  1.34 Zones .............................................................. 137
  1.35 Zone details ................................................... 142
  1.36 IP SLA Management ........................................ 148
  1.37 IP SLA Creators ............................................. 157
  1.38 IP SLA Pollers ............................................... 164
  1.39 Data Access Interface ..................................... 167
  1.40 Data Access templates management – listing and creating .......................................................... 172
  1.41 Data Access templates management – operations on a single template .................................................. 175
  1.42 Data Access – export triggering ........................ 179
  1.43 Flow Data - listing devices with stored flow information .......................................................... 182
  1.44 Flow Data - listing applications supporting flow .......................................................... 185
Overview

- Introduction
- OPTIONS Method
- Multi-Server Resources
- Versioning
- Authentication

Introduction

This document describes the RESTful API provided by Entuity. The API is accessible via HTTP or HTTPS protocol and is exposed via URLs under the /api path. For example, you can access the 'info' resource via http://myserver/api/info or https://myserver/api/info depending on whether or not you have configured ENA for HTTPS access.

Each resource may support one or more of the following HTTP methods: GET, POST, PUT, DELETE, OPTIONS. Each resource may require input and may produce output. Please see the following documentation for each resource for details on supported methods and input/output details.

Resources can expect input in different forms:

- Query parameters, e.g. http://myserver/api/someResource?param=value. Note that ‘value’ in the example URL must be URL encoded (e.g. ‘hello world’ must be encoded as ‘hello%20world’). For the list of characters that must be encoded, see: http://www.w3schools.com/tags/ref_urlencode.asp.
- HTTP content (Entity). The Entity may be represented as either XML (content-type: application/xml) or JSON (content-type: application/json). Note that when sending entity in the request, you must specify the "content-type" header. If using the curl tool you can use the –H argument to specify the header, e.g. curl –H "content-type: application/json"

Most, but not all, resources will return resource representations in either XML or JSON. You can specify which representation by supplying a ‘media’ query parameter with a value of ‘xml’ or ‘json’, e.g. http://myserver/api/info?media=xml. Alternatively, you can specify this using a header field, e.g. Accept: application/xml or Accept: application/json.

Each response has a response code, indicating a success/failure of the request. These are standard HTTP specification response codes:

- 200-299: indicates success.
- 300-399: indicates redirection: clients should repeat request at redirected location.
- 400-499: indicates a problem with a client request.
- 500-599: indicates a problem on a server side.

OPTIONS Method

You can get some simple information on a resource’s supported methods by issuing an OPTIONS method against that resource, or you can find all available resources by issuing an OPTIONS method against a root resource. For example, by using curl:

- curl –X OPTIONS http://myserver/api/. This will output a lengthy output for all supported resources.
- curl –X OPTIONS http://myserver/api/info. This will output available methods for the specific resource only.
Multi-Server Resources

By default, HTTP methods operate on the resources local to the server you are speaking to. However, if the server you are speaking to has remote servers configured, you can work with any of them. You can qualify the server you want to be working with by using a query parameter ‘serverId’, e.g. http://mycentralserver/api/info?serverId=long-id-of-the-remote-server.

Versioning

All resources accessible via /api/* can also be accessed via /api/v2/*. Clients who wish to remain compatible with future versions of ENA should use resources under the specific version: /api/v2/*. Resources without a version specifier under /api/* will contain the latest resource implementations, and may be changed in future releases.

Please note that resources under the specific version /api/v2/* may still be changed with new ENA versions /patches. However, any changes will be limited to compatible changes that are unlikely to break any integrations making use of the API.

Compatible changes include:

- addition of new authentication methods.
- addition of new resources.
- addition of new fields in resource representations returned.
- requiring fewer inputs.
- applying fewer constraints on input.

Incompatible change will be added with a new version number, so the following URLs will be available:

- /api/v1/* will use version 1 of the API.
- /api/v2/* will use version 2 of the API which will include changes that are not compatible with previous versions.
- /api/* Will always track the latest version of the API.

Resources in this document are described using URLs relative to their version base, e.g. resource ‘info’ can be accessed as /api/info or /api/v2/info.

Authentication

ENA supports basic HTTP authentication method (RFC 2617). Basic HTTP authentication is widely supported. Please note that basic HTTP authentication is insecure when used without SSL, because passwords are sent as clear text). Therefore, Entuity recommends using the API over HTTPS.

If using the curl tool, you can supply ‘-u username:password’ arguments to provide authentication details.

For performance reasons, authentication results are cached on the server for 5 minutes after they are last used.

Note that you can authenticate with any Entuity user and the resources are protected by using the Entuity permission model: Users will only be able to access and modify resources that they have permission to access.
Domain Filters

Lists the filter id, name and serverId information of available domain filters.

URL: domainFilters (example http://localhost/api/domainFilters?media=json)

Methods Summary

- GET Method detail
  - Response
  - Examples
- POST Method details
  - Request
  - Response
  - Examples

GET Method detail

Lists available domain filters.

Response

Response includes a list of domain filters. Each domain filter has the following attributes:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>domain filter ID unique to the server.</td>
</tr>
<tr>
<td>name</td>
<td>domain filter name.</td>
</tr>
<tr>
<td>serverId</td>
<td>Entity Server ID on which resource resides.</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>INPUT</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>curl -u admin:admin <a href="http://localhost/api/domainFilters?media=json">http://localhost/api/domainFilters?media=json</a></td>
<td></td>
</tr>
</tbody>
</table>
POST Method details

Creates a new domain filter.

Request

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>filter name.</td>
</tr>
<tr>
<td>rules</td>
<td>the array of rules defining a filter.</td>
</tr>
</tbody>
</table>

Response

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>filter name.</td>
</tr>
</tbody>
</table>
### Examples

**INPUT**

```
curl -u admin:admin  http://localhost/api/domainFilters?media=json -X POST -H "content-type: application/json" -d '{
  
  "name" : "A",
  "rules" : [ { 
   "SRCTYPE" : "4",
   "DEVNAME" : "Two",
   "ZONENAME" : "None"
  } ]
}
```

**OUTPUT**

```
{
  "name" : "A",
  "systemFilter" : false,
  "rules" : [ 
    { 
      "SRCTYPE" : "4",
      "DEVNAME" : "Two",
      "ZONENAME" : "None"
    }
  ]
}
```
```sh
curl -u admin:admin http://localhost/api/domainFilters?media=xml -X POST -H "content-type: application/xml" -d
'<domainFilterInfo systemFilter="false" name="B">
  <rules>
    <entries>
      <key xsi:type="opCode" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">SRCTYPE</key>
    </entries>
    <entries>
      <key xsi:type="opCode" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">DEVNAME</key>
    </entries>
    <entries>
      <key xsi:type="opCode" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">ZONENAME</key>
    </entries>
  </rules>
</domainFilterInfo>'
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<domainFilterInfo systemFilter="false" name="B">
  <rules>
    <entries>
      <key xsi:type="opCode" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">DEVNAME</key>
    </entries>
    <entries>
      <key xsi:type="opCode" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">SRCTYPE</key>
    </entries>
  </rules>
</domainFilterInfo>
Domain Filter details

Represents a set of operations on a particular domain filter.

URL: domainFilters/{filterId}

- Methods summary
  - GET Method details
    - Response
    - Examples
  - PUT Method details
    - Request
    - Response
    - Examples
  - DELETE Method details
    - Request
    - Response
    - Examples

Methods summary

- GET Method - shows detailed information about the filter.
- PUT Method - modifies the parameters of the filter.
- DELETE Method - deletes the filter.

GET Method details

Shows detailed information about the selected domain filter.

Response

General parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>domain filter name.</td>
</tr>
<tr>
<td>systemFilter</td>
<td>whether the filter is a system filter.</td>
</tr>
<tr>
<td>rules</td>
<td>array of rules defining a filter, as per Rule definition section below.</td>
</tr>
</tbody>
</table>

Rule definitions - a single rule may contain at most one of the following parameters, the exception being that SRCTYPE has to be present in the rule definition:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>

SRCTYPE | source type, one of the following (case-insensitive):
PORT    
DEVICE  
VLAN    
APPLICATION
SERVICE

DEVTYPE | device type to which this filter refers.

DEVNAME | name of an object (port, device, application etc).

ZONENAME | name of a zone.

IPLE | IP low-end, used for defining IP ranges.

IPHE | IP high-end, used for defining IP ranges.

MANAGEMENT_ONLY | Management IP Only.

Examples

<table>
<thead>
<tr>
<th>INPUT</th>
<th>curl -u admin:admin <a href="http://localhost/api/domainFilters/1002?media=json">http://localhost/api/domainFilters/1002?media=json</a></th>
</tr>
</thead>
</table>
| OUTPUT   | {
|          |   "name" : "A",
|          |   "systemFilter" : false,
|          |   "rules" : [ {
|          |     "SRCTYPE" : "DEVICE",
|          |     "DEVNAME" : "Two",
|          |     "ZONENAME" : "All"
|          |   } ] |
|          | } |

<table>
<thead>
<tr>
<th>INPUT</th>
<th>curl -u admin:admin <a href="http://localhost/api/domainFilters/1002?media=xml">http://localhost/api/domainFilters/1002?media=xml</a></th>
</tr>
</thead>
</table>
PUT Method details

Modifies the selected domain filter.

Request

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Filter name</td>
</tr>
<tr>
<td>rules</td>
<td>The array of rules defining a filter, as per Rule Definition section above.</td>
</tr>
</tbody>
</table>

Response

The modified filter, as the detailed GET method would return it (see Rule Definition section above).

Examples
| **INPUT** | curl -u admin:admin http://localhost/api/domainFilters/1002?media=json -X PUT -H "content-type: application/json" -d

```
{
  "name" : "B",
  "rules" : [ {
    "SRCTYPE" : "APPLICATION",
    "DEVTYPE" : "158",
    "ZONENAME" : "None"
  } ]
}
```

| **OUTPUT** | {

```
  "name" : "B",
  "rules" : [ {
    "SRCTYPE" : "APPLICATION",
    "DEVTYPE" : "158",
    "ZONENAME" : "None"
  } ]
```

|  |  |
curl -u admin:admin http://localhost/api/domainFilters/1002?media=xml -X PUT -H "content-type: application/xml" -d
'<domainFilterInfo systemFilter="false" name="B">
    <rules>
        <entries>
            <key xsi:type="opCode" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">SRCTYPE</key>
        </entries>
        <entries>
            <key xsi:type="opCode" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">ZONENAME</key>
        </entries>
        <entries>
            <key xsi:type="opCode" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">DEVNAME</key>
        </entries>
    </rules>
</domainFilterInfo>'
**DELETE Method details**

Deletes the selected domain filter.

**Request**

No additional parameters needed.

**Response**

Gives the message “OK” if operation was successful. Otherwise, gives an error description.

**Examples**

```
| INPUT  | curl -u admin:admin http://localhost/api/domainFilters/1002?media=json -X DELETE |
```
| INPUT | curl -u admin:admin http://localhost/api/domainFilters/1002?media=xml -X DELETE |
| OUTPUT | <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<statusInfo>
    <message>OK</message>
</statusInfo> |

OUTPUT
{
    "message": "OK"
}
Event Filters

Lists minimal information about available event filters.

URL: eventFilters

- Methods Summary
- Get Method detail
  - Response
  - Examples
- POST Method details
  - Request
  - Response
  - Examples

Methods Summary

- GET Method - lists available event filters.
- POST Method - creates a new event filter.

Get Method detail

Lists available event filters.

Response

Response includes a list of event filters. Each event filter has the following attributes:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Event filter id unique to the server</td>
</tr>
<tr>
<td>name</td>
<td>Event filter name</td>
</tr>
<tr>
<td>serverId</td>
<td>Entuity Server Id on which resource resides</td>
</tr>
</tbody>
</table>

Examples

INPUT  |  curl -u admin:admin http://localhost/api/eventFilters?media=json |
POST Method details

Creates a new event filter.

Request

Event filter parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Filter name</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>name</td>
<td>Filter name</td>
</tr>
<tr>
<td>selectedNames</td>
<td>The array of filter names</td>
</tr>
<tr>
<td>systemFilter</td>
<td>Whether the filter is a system filter</td>
</tr>
<tr>
<td>passIP</td>
<td>Whether the filter should include devices not under management</td>
</tr>
</tbody>
</table>

**Response**

The newly created entry, as detailed GET method would return it.

**Examples**

**INPUT**

```
curl -u admin:admin http://localhost/api/eventFilters?media=json -X POST -H "content-type: application/json" -d
'
{
  "name" : "A",
  "selectedNames" : [
    "AvailMonitor Application Unavailable",
    "AvailMonitor High Latency Reaching Application Cleared"
  ],
  "passIP" : true
}
```

**OUTPUT**

```
{
  "name" : "A",
  "selectedNames" : [
    "AvailMonitor Application Unavailable",
    "AvailMonitor High Latency Reaching Application Cleared"
  ],
  "systemFilter" : false,
  "passIP" : true
}
```
| INPUT                  | curl -u admin:admin http://localhost/api/eventFilters?media=xml -X POST -H "content-type: application/xml" -d  
|                       |   '<<eventFilterInfo systemFilter="false" passIP="true" name="A">  
|                       |     <selectedNames>  
|                       |       <filterName>AvailMonitor Application Unavailable</filterName>  
|                       |       <filterName>AvailMonitor High Latency Reaching Application Cleared</filterName>  
|                       |     </selectedNames>  
|                       |   </eventFilterInfo>' |
| OUTPUT                | <?xml version="1.0" encoding="UTF-8" standalone="yes"?>  
|                       | <eventFilterInfo systemFilter="false" passIP="true" name="A">  
|                       |     <selectedNames>  
|                       |       <filterName>AvailMonitor Application Unavailable</filterName>  
|                       |       <filterName>AvailMonitor High Latency Reaching Application Cleared</filterName>  
|                       |     </selectedNames>  
|                       | </eventFilterInfo> |
Event Filter details

Represents a set of operations on a particular event filter.

URL: eventFilters/{filterId}

- Methods summary
- GET Method details
  - Response
  - Examples
- PUT Method details
  - Request
  - Response
  - Examples
- DELETE Method details
  - Request
  - Response
  - Examples

Methods summary

- GET Method - shows detailed information about the event filter.
- PUT Method - modifies the parameter of the event filter.
- DELETE Method - deletes the event filter.

GET Method details

Shows detailed information about the selected event filter.

Response

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>filter name.</td>
</tr>
<tr>
<td>selectedNames</td>
<td>array of filter names.</td>
</tr>
<tr>
<td>systemFilter</td>
<td>whether the filter is a system filter.</td>
</tr>
<tr>
<td>passIP</td>
<td>whether the filter should include devices not under management.</td>
</tr>
</tbody>
</table>

Examples

| INPUT | curl -u admin:admin http://localhost/api/eventFilters/1001?media=json |
**OUTPUT**

```json
{
    "name": "A",
    "selectedNames": [
        "AvailMonitor Application Unavailable",
        "AvailMonitor High Latency Reaching Application Cleared"
    ],
    "systemFilter": false,
    "passIP": true
}
```

**INPUT**

```
curl -u admin:admin http://localhost/api/eventFilters/1001?media=xml
```

**OUTPUT**

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<eventFilterInfo systemFilter="false" passIP="true" name="A">
  <selectedNames>
    <filterName>AvailMonitor Application Unavailable</filterName>
    <filterName>AvailMonitor High Latency Reaching Application Cleared</filterName>
  </selectedNames>
</eventFilterInfo>
```

**PUT Method details**

Modifies the parameters of the selected event filter.

**Request**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>filter name.</td>
</tr>
<tr>
<td>selectedNames</td>
<td>array of filter names.</td>
</tr>
<tr>
<td>passIP</td>
<td>whether the filter should include devices not under management.</td>
</tr>
</tbody>
</table>

**Response**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>filter name.</td>
</tr>
<tr>
<td><strong>selectedNames</strong></td>
<td>array of filter names.</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td><strong>systemFilter</strong></td>
<td>whether the filter is a system filter.</td>
</tr>
<tr>
<td><strong>passIP</strong></td>
<td>whether this filter should include devices not under management.</td>
</tr>
</tbody>
</table>

**Examples**

**INPUT**
```
curl -u admin:admin http://localhost/api/eventFilters/1001?media=json - X PUT -H "content-type: application/json" -d
'{
  "name" : "B",
  "selectedNames" : [
    "AvailMonitor Application Unavailable",
    "WAN Port Low Outbound Utilization Cleared",
    "AvailMonitor High Latency Reaching Application Cleared"
  ],
  "passIP" : false
}'
```

**OUTPUT**
```
{
  "name" : "B",
  "selectedNames" : [
    "WAN Port Low Outbound Utilization Cleared",
    "AvailMonitor Application Unavailable",
    "AvailMonitor High Latency Reaching Application Cleared"
  ],
  "systemFilter" : false,
  "passIP" : false
}
```
DELETE Method details

Deletes the selected event filter.

Request

No additional parameters needed.

Response

Gives the message “OK” if operation was successful. Otherwise, gives an error description.

Examples

<p>| INPUT | curl -u admin:admin <a href="http://localhost/api/eventFilters/1001?media=json">http://localhost/api/eventFilters/1001?media=json</a> -X DELETE |
| OUTPUT | { |
| | &quot;message&quot; : &quot;OK&quot; |
| } |</p>
<table>
<thead>
<tr>
<th>INPUT</th>
<th>curl -u admin:admin <a href="http://localhost/api/eventFilters/1001?media=xml">http://localhost/api/eventFilters/1001?media=xml</a> -X DELETE</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPUT</td>
<td>&lt;?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot; standalone=&quot;yes&quot;?&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;statusInfo&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;message&gt;OK&lt;/message&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/statusInfo&gt;</td>
</tr>
</tbody>
</table>
Incident Filters

List available incident filters, and create new filters.

URL: incidentFilters

- Methods Summary
- GET Method detail
  - Response
  - Examples
- POST Method details
  - Request
  - Response
  - Examples

Methods Summary

- GET Method - lists available incident filters.
- POST Method - creates new incident filter.

GET Method detail

Lists available incident filters.

Response

Response includes a list of incident filters. Each incident filter has the following attributes:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serverId</td>
<td>Entity Server Id on which resource resides.</td>
</tr>
<tr>
<td>id</td>
<td>incident filter id unique to the server.</td>
</tr>
<tr>
<td>name</td>
<td>incident filter name.</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>curl -u admin:admin <a href="http://boron/api/incidentFilters?media=json">http://boron/api/incidentFilters?media=json</a></td>
</tr>
</tbody>
</table>
INPUT
```
curl -u admin:admin http://boron/api/incidentFilters?media=xml
```

OUTPUT
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<items count="2">
  <item xsi:type="namedItem" name="All Incidents" id="1" serverId="821c3e87-bcdf-4fef-a5e8-7d2524928d96"/>
  <item xsi:type="namedItem" name="TestIncidents" id="2" serverId="821c3e87-bcdf-4fef-a5e8-7d2524928d96"/>
</items>
```

POST Method details

Creates new incident filter.

Request

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>filter name.</td>
</tr>
<tr>
<td>selectedNames</td>
<td>array of filter names.</td>
</tr>
<tr>
<td>passIP</td>
<td>whether the filter should include devices that are not under management.</td>
</tr>
</tbody>
</table>

Response

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>filter name.</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>selectedNames</td>
<td>array of filter names.</td>
</tr>
<tr>
<td>systemFilter</td>
<td>whether the filter is a system filter.</td>
</tr>
<tr>
<td>passIP</td>
<td>whether the filter should include devices that are not under management.</td>
</tr>
</tbody>
</table>

### Examples

**INPUT**

curl -u admin:admin http://localhost/api/incidentFilters?media=json -X POST -H "content-type: application/json" -d

```
{
    "name" : "A",
    "selectedNames" : [
        "AP Antenna Host Count Abnormality",
        "AP Not Associated With Controller"
    ],
    "passIP" : false
}
```

**OUTPUT**

```
{
    "name" : "A",
    "selectedNames" : [
        "AP Antenna Host Count Abnormality",
        "AP Not Associated With Controller"
    ],
    "systemFilter" : false,
    "passIP" : false
}
```

**INPUT**

curl -u admin:admin http://localhost/api/incidentFilters?media=xml -X POST -H "content-type: application/xml"

```
'<eventFilterInfo systemFilter="false" passIP="true" name="A">
    <selectedNames>
        <filterName>AvailMonitor High Latency</filterName>
        <filterName>AvailMonitor Application Problem</filterName>
    </selectedNames>
'</eventFilterInfo>'
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<eventFilterInfo systemFilter="false" passIP="true" name="A">
  <selectedNames>
    <filterName>AvailMonitor High Latency</filterName>
    <filterName>AvailMonitor Application Problem</filterName>
  </selectedNames>
</eventFilterInfo>
Incident Filter details

Show details about a particular incident filter, modify the parameters of a filter, and delete a filter.

URL: incident Filters/{filterId}

- Methods summary
- GET Method details
  - Response
  - Examples
- PUT Method details
  - Request
  - Response
  - Examples
- DELETE Method details
  - Request
  - Response
  - Examples

Methods summary

- GET Method - shows detailed information about the filter.
- PUT Method - modifies the parameters of the filter.
- DELETE Method - deletes the filter.

GET Method details

Shows detailed information about the selected incident filter.

Response

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>incident filter name.</td>
</tr>
<tr>
<td>systemFilter</td>
<td>whether this filter is a system filter.</td>
</tr>
<tr>
<td>rules</td>
<td>list of rules in this filter.</td>
</tr>
</tbody>
</table>

Examples

INPUT | curl -u admin:admin http://localhost/api/incidentFilters/2?media=json
### PUT Method details

Modifies the selected incident filter.

**Request**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>filter name.</td>
</tr>
<tr>
<td>rules</td>
<td>list of rules defining the filter.</td>
</tr>
</tbody>
</table>

**Response**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>filter name.</td>
</tr>
<tr>
<td>systemFilter</td>
<td>whether the filter is system one.</td>
</tr>
<tr>
<td>rules</td>
<td>list of rules defining a filter.</td>
</tr>
</tbody>
</table>
Examples

**INPUT**
curl -u admin:admin http://localhost/api/incidentFilters/2?media=json -X PUT $content-type: application/json$ -d
`{
  "name" : "B",
  "selectedNames" : [
    "AP Antenna Host Count Abnormality",
    "AP Antenna Power Change Frequency High",
    "AP Not Associated With Controller"
  ],
  "systemFilter" : false,
  "passIP" : true
}`

**OUTPUT**
{
  "name" : "B",
  "selectedNames" : [
    "AP Antenna Host Count Abnormality",
    "AP Antenna Power Change Frequency High",
    "AP Not Associated With Controller"
  ],
  "systemFilter" : false,
  "passIP" : true
}

**INPUT**
curl -u admin:admin http://localhost/api/incidentFilters/2?media=xml -X PUT $content-type: application/xml$ -d
`<eventFilterInfo systemFilter="false" passIP="false" name="B">
  <selectedNames>
    <filterName>AP Antenna Host Count Abnormality</filterName>
    <filterName>AP Antenna Power Change Frequency High</filterName>
    <filterName>AP Not Associated With Controller</filterName>
  </selectedNames>
</eventFilterInfo>`
### DELETE Method details

Deletes the selected incident filter.

**Request**

No additional parameters needed.

**Response**

“OK” message if operation was successful, and an error description otherwise.

**Examples**

<table>
<thead>
<tr>
<th>INPUT</th>
<th>curl -u admin:admin <a href="http://localhost/api/incidentFilters/2?media=json">http://localhost/api/incidentFilters/2?media=json</a> -X DELETE -H &quot;content-type: application/json&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPUT</td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>&quot;message&quot; : &quot;OK&quot;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INPUT</th>
<th>curl -u admin:admin <a href="http://localhost/api/incidentFilters/2?media=xml">http://localhost/api/incidentFilters/2?media=xml</a> -X DELETE</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPUT</td>
<td>&lt;?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot; standalone=&quot;yes&quot;?&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;statusInfo&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;message&gt;OK&lt;/message&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/statusInfo&gt;</td>
</tr>
</tbody>
</table>
Events

List available events, or raise a new event.

URL: events

- Methods Summary
- GET Method details
  - Request
  - Response
  - Example
- POST Method details
  - Request
  - Examples

Methods Summary

- GET Method - lists available events.
- POST Method - raises an event.

GET Method details

Lists available events.

Request

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>updateId</td>
<td>event update identifier, use the updateId in the last result set to get new events.</td>
</tr>
<tr>
<td>openedFrom</td>
<td>event opening time lower bound.</td>
</tr>
<tr>
<td>openedTo</td>
<td>event opening time upper bound.</td>
</tr>
<tr>
<td>closedFrom</td>
<td>event closing time lower bound.</td>
</tr>
<tr>
<td>closedTo</td>
<td>event closing time upper bound.</td>
</tr>
<tr>
<td>view</td>
<td>events filtered by view.</td>
</tr>
<tr>
<td>mask</td>
<td>event severity mask, severities can be added together to request multiple severities together.</td>
</tr>
<tr>
<td></td>
<td>1 = Info, 2 = Minor, 4 = Major, 8 = Sever, 16 = Critical</td>
</tr>
<tr>
<td></td>
<td>e.g. mask=24 would return both Sever and Critical events.</td>
</tr>
</tbody>
</table>
States

- event state, either “open”, “closed”, “finalized” and “all”. Multiple event states can be specified together.
  e.g. states=open&states=closed

Response

Response includes an array of events. Each event has the following attributes:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>event description.</td>
</tr>
<tr>
<td>details</td>
<td>event details.</td>
</tr>
<tr>
<td>objectKeyInfo</td>
<td>object key, consisting of:</td>
</tr>
<tr>
<td>swld</td>
<td>StormWorks Identifier (integer)</td>
</tr>
<tr>
<td>complid</td>
<td>Classic component identifier (4 integers)</td>
</tr>
<tr>
<td>severity</td>
<td>event severity where:</td>
</tr>
<tr>
<td>2</td>
<td>Information, 4 = minor, 6 = major, 8 = severe, 10 = critical</td>
</tr>
<tr>
<td>description</td>
<td>event Description.</td>
</tr>
<tr>
<td>sourceDescription</td>
<td>source description.</td>
</tr>
<tr>
<td>impactDescription</td>
<td>impact description.</td>
</tr>
<tr>
<td>timeStamp</td>
<td>event timestamp.</td>
</tr>
<tr>
<td>id</td>
<td>event unique identifier.</td>
</tr>
<tr>
<td>eventNumber</td>
<td>event number.</td>
</tr>
</tbody>
</table>

Example

<table>
<thead>
<tr>
<th>INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>curl -u admin:admin <a href="http://localhost/api/events?media=json">http://localhost/api/events?media=json</a></td>
</tr>
</tbody>
</table>
{  
  "events" : [ {  
    "description" : "Entuity Server Started",
    "details" : "Restarted",
    "objectKeyInfo" : {  
      "swld" : 0,
      "compId" : {  
        "ids" : [ 4096, 0, 0, 0 ]
      }
    },
    "severity" : 2,
    "sourceDescription" : "Entuity server London01",
    "impactDescription" : "Entuity service",
    "timeStamp" : 1540894535,
    "id" : 786437,
    "eventNumber" : 480
  },
  {  
    ...removed for brevity...
  }],
  "updateId" : 479,
  "count" : 459} 
}

curl -u admin:admin http://localhost/api/events?media=xml
POST Method details

 Raises an event.

**Request**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Event type</td>
</tr>
<tr>
<td>objectKeyInfo</td>
<td>Object key, consisting of:</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>swld</td>
<td>StormWorks Identifier (integer)</td>
</tr>
<tr>
<td>compId</td>
<td>Classic component identifier (4 integers)</td>
</tr>
</tbody>
</table>

| name | Event name |
| severity | Event Severity |
| 2 | Information |
| 4 | minor |
| 6 | major |
| 8 | severe |
| 10 | critical |

| reason | Reason for event |
| ImpactDescription | Impact description |
| sourceDescription | Source description |

**Examples**

**INPUT**
```
curl -u admin:admin http://localhost/api/events?media=json -X POST -H "content-type: application/json" -d \
'{
    "reason" : "API test",
    "name" : "New Event",
    "id" : 786441,
    "impactDescription" : "Entuity service",
    "severity" : 6,
    "objectKeyInfo" : { "swId" : 12345, "compId" : { "ids" : [4096, 0, 0, 0] } }
}
```

**OUTPUT**
```
{
    "message": "Event sent"
}
```
INPUT

```
curl -u admin:admin http://localhost/api/events -X POST -H "content-type: application/xml" -d
  
  <eventInfo>
  <reason>API test</reason>
  <name>New event</name>
  <details>New event details</details>
  <id>786441</id>
  <impactDescription>Entuity service</impactDescription>
  <severity>2</severity>
  <objectKeyInfo>
    <swId>0</swId>
    <compId>
      <ids>
        <ids>4096</ids>
        <ids>0</ids>
        <ids>0</ids>
        <ids>0</ids>
      </ids>
    </compId>
  </objectKeyInfo>
  </eventInfo>
```

OUTPUT

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<statusInfo>
  <message>Event sent</message>
</statusInfo>
```
Event types

List available event types.

URL: eventTypes

- Methods Summary
  - GET Method details
    - Response
    - Example

Methods Summary

- GET Method - lists available event types.

GET Method details

Lists available event types.

Response

Response includes an array of event types, each of them having the following format:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name of the event type.</td>
</tr>
<tr>
<td>severity</td>
<td>severity of the event type.</td>
</tr>
</tbody>
</table>

Example

| INPUT | curl -u admin:admin http://localhost/api/eventTypes?media=json |
**OUTPUT**

```
[ {
  "name" : "ATM VCC High Inbound Utilization",
  "severity" : 6
}, {
  ...removed for brevity...
}, {
  "name" : "WAN Port Low Outbound Utilization Cleared",
  "severity" : 2
}]
```

**INPUT**

```
curl -u admin:admin http://localhost/api/eventTypes?media=xml
```

**OUTPUT**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<items count="322">
  <item xsi:type="eventType" severity="6" name="ATM VCC High Inbound Utilization" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
  ...removed for brevity...
  <item xsi:type="eventType" severity="2" name="WAN Port Low Outbound Utilization Cleared" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
</items>
```
## Incidents

Lists available incidents.

URL: incidents

### Methods Summary

- GET Method - lists available incidents.

### GET Method details

Lists available incidents.

### Request

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>updateId</td>
<td>incident update identifier, use the updateId from the last result set to get new incidents.</td>
</tr>
<tr>
<td>view</td>
<td>incidents filtered by view.</td>
</tr>
<tr>
<td>mask</td>
<td>incident severity mask, severities can be added together to request multiple severities together 1 = Info, 2 = Minor, 4 = Major, 8 = Sever, 16 = Critical e.g. mask=24 would return both Sever and Critical incidents.</td>
</tr>
<tr>
<td>states</td>
<td>incident state, either “open”, “closed”, “expire” and “all”. Multiple incident states can be specified together e.g. states=open&amp;states=closed</td>
</tr>
<tr>
<td>openedFrom</td>
<td>incident opening time lower bound.</td>
</tr>
<tr>
<td>openedTo</td>
<td>incident opening time upper bound.</td>
</tr>
<tr>
<td>closedFrom</td>
<td>incident closing time lower bound.</td>
</tr>
</tbody>
</table>
closedTo | incident closing time upper bound.

**Response**

Response includes an array of event. Each event has following attributes:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>event description.</td>
</tr>
<tr>
<td>details</td>
<td>event details.</td>
</tr>
<tr>
<td>objectKeyInfo</td>
<td>object key, consisting of:</td>
</tr>
<tr>
<td></td>
<td>swld    StormWorks Identifier (integer)</td>
</tr>
<tr>
<td></td>
<td>complId Classic component identifier (4 integers)</td>
</tr>
<tr>
<td>severity</td>
<td>incident severity</td>
</tr>
<tr>
<td></td>
<td>2 = Information, 4 = Minor, 6 = Major, 8 = Severe, 10 = Critical</td>
</tr>
<tr>
<td>sourceDescription</td>
<td>source description.</td>
</tr>
<tr>
<td>impactDescription</td>
<td>impact description.</td>
</tr>
<tr>
<td>timeStamp</td>
<td>event timestamp.</td>
</tr>
<tr>
<td>id</td>
<td>incident Identifier.</td>
</tr>
<tr>
<td>state</td>
<td>incident state.</td>
</tr>
<tr>
<td>annotation</td>
<td>incident annotation.</td>
</tr>
<tr>
<td>eventCount</td>
<td>number of events contributing to the incident.</td>
</tr>
<tr>
<td>extraAttribs</td>
<td>attribute key value pairs for the incident.</td>
</tr>
</tbody>
</table>

**Example**

**INPUT**

```
curl -u admin:admin http://localhost/api/incidents?media=json
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<incidentsCollection count="2" updateId="799">
  <incidents>
    <incident id="69" eventCount="1" state="open" annotation="">
      <description>Entuity Server Component Problem</description>
      <details>Reason for event</details>
      <objectKeyInfo>
        <swId>815</swId>
        <compId>
          <ids>
            <ids>1</ids>
            <ids>2</ids>
            <ids>3</ids>
            <ids>0</ids>
          </ids>
        </compId>
      </objectKeyInfo>
      <sourceDescription>Huge</sourceDescription>
      <impactDescription></impactDescription>
      <extraAttribs>
        <entry>
          <key>attribKey</key>
          <value>Attrib value</value>
        </entry>
      </extraAttribs>
    </incident>
    ...removed for brevity...
  </incidents>
</incidentsCollection>
Incident details

Change state, annotation and attributes of an incident.

URL: incidents/{incidentID}

- Methods Summary
- PUT Method details
  - Request
  - Response
  - Examples

Methods Summary

- PUT Method - change incident state, annotation, attributes.

PUT Method details

Change incident state, annotation and attributes.

Request

All the fields in an incident request are optional so each can be updated independently.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>change state. Valid states are “open”, “closed” and “expired”.</td>
</tr>
<tr>
<td></td>
<td>Note: It is not possible to change the state of an incident if it has already expired.</td>
</tr>
<tr>
<td>annotation</td>
<td>change the annotation for an incident.</td>
</tr>
<tr>
<td>attribute</td>
<td>update an attribute for an incident. A key and value must be provided (see examples below).</td>
</tr>
<tr>
<td></td>
<td>Note: The attribute must have already been defined in the active event project.</td>
</tr>
</tbody>
</table>

Response

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>confirmation message.</td>
</tr>
</tbody>
</table>

Examples
    "annotation" : "New annotation", 
    "attribute" : {"key" : "attribKey", "value": "New value" } 
}' |
|-------|-----------------------------------------------------------------------------------------|
| OUTPUT | {  
    "message" : "Incident opened, Attribute 'attribKey' updated"  
} |

    <state>open</state>  
    <annotation>New annotation</annotation>  
    <attribute key="attrib1" value="New value"></attribute>  
</incidentInfo>' |
|-------|-----------------------------------------------------------------------------------------------------------------------------|
| OUTPUT | <?xml version="1.0" encoding="UTF-8" standalone="yes"?>  
<statusInfo>  
    <message>Incident opened, Attribute 'attribKey' updated</message>  
</statusInfo> |
Incident types

Lists incident types.
URL: incidentTypes

- Methods Summary
- GET Method details
  - Response
  - Example

Methods Summary

- GET Method - lists available incident types.

GET Method details

Lists available incident types.

Response

Response includes an array of event types, each of them having the following format:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name of the incident type.</td>
</tr>
</tbody>
</table>

Example

<table>
<thead>
<tr>
<th>INPUT</th>
<th>curl -u admin:admin <a href="http://localhost/api/incidentTypes?media=json">http://localhost/api/incidentTypes?media=json</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPUT</td>
<td>[ {</td>
</tr>
<tr>
<td></td>
<td>&quot;name&quot; : &quot;AP Antenna Channel Change Frequency High&quot;</td>
</tr>
<tr>
<td></td>
<td>} , {</td>
</tr>
<tr>
<td></td>
<td>...removed for brevity...</td>
</tr>
<tr>
<td></td>
<td>} , {</td>
</tr>
<tr>
<td></td>
<td>&quot;name&quot; : &quot;Wireless Controller High Number of Connected APs&quot;</td>
</tr>
<tr>
<td></td>
<td>} ]</td>
</tr>
<tr>
<td>INPUT</td>
<td>curl -u admin:admin <a href="http://localhost/api/incidentTypes?media=xml">http://localhost/api/incidentTypes?media=xml</a></td>
</tr>
<tr>
<td>-------</td>
<td>---------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| OUTPUT| ```XML
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<items count="256">
  <item xsi:type="incidentType" name="AP Antenna Channel Change Frequency High" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
  ...removed for brevity...
  <item xsi:type="incidentType" name="Wireless Controller High Number of Connected APs" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
</items>``` |
Inventory Summary

List the inventory on a server, or add a new device to the inventory.

URL: inventory

- Method Summary
- GET Method detail
  - Response data keys
  - Examples
- POST Method detail
  - URL parameters
  - Request Parameters
  - Response
  - Examples

Method Summary

- GET Method - list the contents of the inventory (XML, JSON).
- POST Method - queue a device to add to the inventory of a server (XML, JSON).

GET Method detail

List the contents of the inventory, in either XML or JSON formats.

Response data keys

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>number of devices in the inventory.</td>
</tr>
<tr>
<td>Item</td>
<td>list of device summary details:</td>
</tr>
<tr>
<td></td>
<td>- name: Display Name</td>
</tr>
<tr>
<td></td>
<td>- Id: Unique Identifier</td>
</tr>
<tr>
<td></td>
<td>- polledName: DNS name or IP address</td>
</tr>
<tr>
<td></td>
<td>- serverId: server identifier</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>curl -u admin:admin http://localhost/api/inventory?media=json</code></td>
</tr>
</tbody>
</table>
POST Method detail

Queue a device to add to the inventory of a server, via either XML or JSON formats.

URL parameters

Note: URL parameters are appended to the end of the URL and are delimited from the main part of the URL using a question mark (?) character, see the example below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>

INPUT

```
curl -u admin:admin http://localhost/api/inventory?media=xml
```

OUTPUT

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><items count="1">
  <item xsi:type="device" added="true" polledName="10.66.24.2" name="A" id="2" serverId="34564e92-3b4f-43ba-94a8-04309c0e48fe" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.w3.org/2001/XMLSchema"/>
</items>```
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authType</td>
<td>SNMP v3 authentication type: NONE, MD5, SHA.</td>
</tr>
<tr>
<td>cliMethod</td>
<td>Connection method: SSH, TELNET</td>
</tr>
<tr>
<td>cliPassword1</td>
<td>Password 1</td>
</tr>
<tr>
<td>cliPassword2</td>
<td>Password 2</td>
</tr>
<tr>
<td>cliPort</td>
<td>Connection port</td>
</tr>
<tr>
<td>cliUsername</td>
<td>CLI User name</td>
</tr>
<tr>
<td>deviceType</td>
<td>Device type, one of:</td>
</tr>
<tr>
<td></td>
<td>- Hub.</td>
</tr>
<tr>
<td></td>
<td>- Token Ring Switch.</td>
</tr>
<tr>
<td></td>
<td>- Ethernet Switch.</td>
</tr>
<tr>
<td></td>
<td>- ATM Switch.</td>
</tr>
<tr>
<td></td>
<td>- Router.</td>
</tr>
<tr>
<td></td>
<td>- Blade Center.</td>
</tr>
<tr>
<td></td>
<td>- User Created Node.</td>
</tr>
<tr>
<td></td>
<td>- VM Platform.</td>
</tr>
<tr>
<td></td>
<td>- Autonomous WAP.</td>
</tr>
<tr>
<td></td>
<td>- Firewall.</td>
</tr>
<tr>
<td></td>
<td>- VPN.</td>
</tr>
<tr>
<td></td>
<td>- Managed Host.</td>
</tr>
<tr>
<td></td>
<td>- Non-SNMP Device.</td>
</tr>
<tr>
<td></td>
<td>- Unclassified (Full).</td>
</tr>
<tr>
<td></td>
<td>- PoE Midspan Injector.</td>
</tr>
<tr>
<td></td>
<td>- Load Balancer.</td>
</tr>
<tr>
<td></td>
<td>- SSL Proxy.</td>
</tr>
<tr>
<td></td>
<td>- Unclassified.</td>
</tr>
<tr>
<td></td>
<td>- Wireless Controller.</td>
</tr>
<tr>
<td></td>
<td>- Uninterruptible Power Supply.</td>
</tr>
<tr>
<td></td>
<td>- Matrix Switch.</td>
</tr>
<tr>
<td></td>
<td>- Wide Area Application Service.</td>
</tr>
<tr>
<td></td>
<td>- Multiplexer.</td>
</tr>
<tr>
<td></td>
<td>- SDN Controller.</td>
</tr>
<tr>
<td></td>
<td>- Cloud Controller.</td>
</tr>
</tbody>
</table>

duplicateIps

allow duplicate IP addresses. By default, ENA will not manage a device that has IP addresses that have already been seen on an existing managed device. The duplicateIps flag can be used to override this behavior.

The flag will accept the values; "false", "no", "0", "true", "yes" or "1". To override ENA's default behavior, set this flag to true. For default behaviour set the flag to false or omit the flag altogether.
<table>
<thead>
<tr>
<th><strong>encrType</strong></th>
<th>SNMP v3 encryption type:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• NONE.</td>
</tr>
<tr>
<td></td>
<td>• DES.</td>
</tr>
<tr>
<td></td>
<td>• DES3.</td>
</tr>
<tr>
<td></td>
<td>• AES.</td>
</tr>
<tr>
<td></td>
<td>• AES192.</td>
</tr>
<tr>
<td></td>
<td>• AES256.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>managementLevel</strong></th>
<th>management level:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• FULL.</td>
</tr>
<tr>
<td></td>
<td>• FULL_NO_PORTS.</td>
</tr>
<tr>
<td></td>
<td>• BASIC.</td>
</tr>
<tr>
<td></td>
<td>• PING_ONLY.</td>
</tr>
<tr>
<td></td>
<td>• WEB.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>name</strong></th>
<th>device name.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>nameUsing</strong></th>
<th>device name is determined using one of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• CUSTOMNAME.</td>
</tr>
<tr>
<td></td>
<td>• IPADDRESS.</td>
</tr>
<tr>
<td></td>
<td>• POLLEDNAME.</td>
</tr>
<tr>
<td></td>
<td>• RESOLVABLENAME.</td>
</tr>
<tr>
<td></td>
<td>• RESOLVABLENAMEFQ.</td>
</tr>
<tr>
<td></td>
<td>• SYSTEMNAME.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>polledName</strong></th>
<th>DNS name or IP Address.</th>
</tr>
</thead>
</table>

|--------------|---------------------------------|

<table>
<thead>
<tr>
<th><strong>readCommunity</strong></th>
<th>SNMP v1/v2 read community string.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>writeCommunity</strong></th>
<th>SNMP v1/v2 write community string.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>snmpPDUSize</strong></th>
<th>maximum size of SNMP PDU.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 = system default</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>snmpRetry</strong></th>
<th>number of SNMP retries.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 = system default</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>snmpTimeout</strong></th>
<th>SNMP timeout in seconds.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 = system default</td>
</tr>
</tbody>
</table>
| **snmpType** | SNMP type:  
| | • v1.  
| | • v2c.  
| | • v3.  
| | • v1/2. |  
| **snmpPort** | SNMP port number (if omitted, the default value will be used). |  
| **username** | SNMP v3 user name. |  
| **webAccessKey** | access key (Amazon platform only). |  
| **webPassword** | virtual platform password (non-Amazon virtual platforms). |  
| **webPlatformType** | virtual platform type:  
| | • VMWARE_ESXi  
| | • ORACLE_VM_MANAGER  
| | • HYPER_V  
| | • AMAZON_WEB_SERVICE  
| | • AZURE  
| | • CISCO_APIC  
| | • MERAKI |  
| **webSecretKey** | secret key (Amazon platform only). |  
| **webTenantID** | password for Microsoft Azure platform. |  
| **webURL** | virtual platform URL (non-Amazon virtual platforms). |  
| **webUser** | virtual platform user name (non-Amazon virtual platforms). |  

**Response**

200 - OK (Device is queued to be added to the inventory.)

The status info message will say “Queued” or “Replaced”, depending on whether this was the first attempt to add a device, or if there was a previous failed attempt to add the device with the same **polledName**.

**Examples**
To add a device that is not a VM Platform, Cloud Controller, SDN Controller, Ping Only, or Custom Device:
To specify a Ping Only device:

| INPUT | curl -H 'Content-type: application/json' -u admin:admin -X POST 'http://localhost/api/inventory?media=json' -d '{
   nameUsing="POLLEDNAME"
   polledName="10.66.33.2"
   managementLevel="PING_ONLY"
   protocol="IPv4"
} />' |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPUT</td>
<td>{'&quot;message&quot;: &quot;Queued&quot;}</td>
</tr>
</tbody>
</table>

To specify a device type such as VM Platform, Cloud Controller, SDN Controller:

i). To add an Amazon Web Services platform:
ii). To add a Meraki Cloud Controller platform:

```
INPUT
    curl -H "Content-type: application/json" -u admin:admin -X POST
    'http://localhost/api/inventory?media=json' -d
    '{
        managementLevel="WEB"
        polledName="Meraki"
        deviceType="Cloud Controller"
        webPlatformType="MERAKI"
        webURL="https://api.meraki.com/api/v0/"
        webPassword="123456789"
    }'

OUTPUT
    {
        "message": "Queued"
    }
```

iii). To add an Azure platform:
### iv). To add a Cisco APIC SDN Controller:

```bash
 curl -H 'Content-Type:application/json' -u admin:admin -X POST http://localhost/api/inventory -d
' 
    
    managementLevel="WEB"
    
    polledName="APIC"
    
    deviceType="SDN Controller"
    
    webPlatformType="CISCO_APIC"
    
    webURL="https://myapicname.cisco.com/api/

    webUser="admin"
    
    webPassword="password"

/>'
```

<table>
<thead>
<tr>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
</tr>
<tr>
<td>&quot;message&quot;: &quot;Queued&quot;</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

```bash
 curl -H 'Content-Type:application/json' -u admin:admin -X POST http://localhost/api/inventory -d
' 
    
    managementLevel="WEB"
    
    polledName="AZURE"
    
    deviceType="VM Platform"
    
    webPlatformType="AZURE"
    
    webTenantID="TenantID"
    
    webSecretKey="secretKey"
    
    webAccessKey="vmAccessKey"

/>'
```

<table>
<thead>
<tr>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
</tr>
<tr>
<td>&quot;message&quot;: &quot;Queued&quot;</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>
v). To add a VMWare Platform:

<table>
<thead>
<tr>
<th>INPUT</th>
<th>curl -H 'Content-Type:application/json' -u admin:admin -X POST <a href="http://titanium/api/inventory">http://titanium/api/inventory</a> -d</th>
</tr>
</thead>
</table>
|       | '{
|       |   managementLevel="WEB"
|       |   polledName="vortex"
|       |   deviceType="VM Platform"
|       |   webPlatformType="VMWARE_ESXi"
|       |   webURL="https://vortex/sdk"
|       |   webUser="root"
|       |   webPassword="password"
|       |   protocol="IPv4"
|       | }' |

| OUTPUT | { |
|        |   "message": "Queued"
|        | } |
Inventory details

List, update or delete an inventory device.

URL: inventory/{deviceId}

- **Method Summary**
  - GET Method detail
    - Request Parameters
    - Response data keys
    - Examples
  - PUT Method detail
    - Request Parameters
    - Response data
    - Examples
  - DELETE Method details
    - Request
    - Response
    - Examples

**Method Summary**

- GET Method - list details of an inventory device (XML, JSON).
- PUT Method - change an inventory device’s details (XML, JSON).
- DELETE Method - remove a device from the inventory.

**GET Method detail**

List details of an inventory device, in either XML or JSON formats.

**Request Parameters**

None.

**Response data keys**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authType</td>
<td>SNMP v3 authentication type: NONE, MD5, SHA.</td>
</tr>
<tr>
<td>capabilities</td>
<td>device capabilities: routing, switching, switching &amp; routing.</td>
</tr>
<tr>
<td>certified</td>
<td>if device has been certified.</td>
</tr>
<tr>
<td>deviceType</td>
<td>device type.</td>
</tr>
<tr>
<td>dsObjectId</td>
<td>device’s stormworks identifier.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>encrType</td>
<td>SNMP v3 encryption type:</td>
</tr>
<tr>
<td></td>
<td>• none</td>
</tr>
<tr>
<td></td>
<td>• DES</td>
</tr>
<tr>
<td></td>
<td>• DES3</td>
</tr>
<tr>
<td></td>
<td>• AES</td>
</tr>
<tr>
<td></td>
<td>• AES192</td>
</tr>
<tr>
<td></td>
<td>• AES256</td>
</tr>
<tr>
<td>context</td>
<td>SNMP v3 context.</td>
</tr>
<tr>
<td>id</td>
<td>device’s unique identifier.</td>
</tr>
<tr>
<td>managementIP</td>
<td>management IP address.</td>
</tr>
<tr>
<td>managementLevel</td>
<td>management level:</td>
</tr>
<tr>
<td></td>
<td>• FULL</td>
</tr>
<tr>
<td></td>
<td>• FULL_NO_PORTS</td>
</tr>
<tr>
<td></td>
<td>• BASIC</td>
</tr>
<tr>
<td></td>
<td>• PING_ONLY</td>
</tr>
<tr>
<td></td>
<td>• WEB.</td>
</tr>
<tr>
<td>name</td>
<td>device name</td>
</tr>
<tr>
<td>nameUsing</td>
<td>device name is determined using either:</td>
</tr>
<tr>
<td></td>
<td>• CUSTOMNAME</td>
</tr>
<tr>
<td></td>
<td>• IPADDRESS</td>
</tr>
<tr>
<td></td>
<td>• POLLEDNAME</td>
</tr>
<tr>
<td></td>
<td>• RESOLVABLENAME</td>
</tr>
<tr>
<td></td>
<td>• RESOLVABLENAMEFQ</td>
</tr>
<tr>
<td></td>
<td>• SYSTEMNAME</td>
</tr>
<tr>
<td>polledName</td>
<td>DNS name or IP Address.</td>
</tr>
<tr>
<td>protocol</td>
<td>Transport protocol: IPv4 or IPv6.</td>
</tr>
<tr>
<td>readCommunity</td>
<td>SNMP read community string.</td>
</tr>
<tr>
<td>serverId</td>
<td>server identifier.</td>
</tr>
<tr>
<td>snmpPDUSize</td>
<td>maximum size of SNMP PDU, where 0 = system default.</td>
</tr>
<tr>
<td>snmpRetry</td>
<td>number of SNMP retries, where 0 = system default.</td>
</tr>
<tr>
<td>snmpTimeout</td>
<td>SNMP timeout in seconds, where 0 = system default.</td>
</tr>
<tr>
<td>snmpType</td>
<td>SNMP version: v1, v2, v3 or v1/2.</td>
</tr>
<tr>
<td>sysDescription</td>
<td>SNMP description field.</td>
</tr>
<tr>
<td>sysLocation</td>
<td>SNMP retrieved system Location field.</td>
</tr>
<tr>
<td>sysOid</td>
<td>SNMP system identifier field.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>username</td>
<td>SNMP v3 user name.</td>
</tr>
<tr>
<td>webAccessKey</td>
<td>access key (Amazon platform only).</td>
</tr>
<tr>
<td>webPassword</td>
<td>virtual platform password (Non Amazon virtual platforms).</td>
</tr>
<tr>
<td>webPlatformType</td>
<td>virtual platform type:</td>
</tr>
<tr>
<td></td>
<td>- VMWARE_ESXi</td>
</tr>
<tr>
<td></td>
<td>- ORACLE_VM_MANAGER</td>
</tr>
<tr>
<td></td>
<td>- HYPER_V</td>
</tr>
<tr>
<td></td>
<td>- AMAZON_WEB_SERVICE</td>
</tr>
<tr>
<td></td>
<td>- AZURE</td>
</tr>
<tr>
<td></td>
<td>- CISCO_APIC</td>
</tr>
<tr>
<td></td>
<td>- MERAKI</td>
</tr>
<tr>
<td>webSecretKey</td>
<td>secret key (Amazon platform only).</td>
</tr>
<tr>
<td>webTenantID</td>
<td>password for Microsoft Azure platform.</td>
</tr>
<tr>
<td>webURL</td>
<td>virtual platform URL (non-Amazon virtual platforms).</td>
</tr>
<tr>
<td>webUser</td>
<td>virtual platform username (non-Amazon virtual platforms).</td>
</tr>
<tr>
<td>zoneId</td>
<td>the ID of a zone.</td>
</tr>
</tbody>
</table>

**Examples**

| INPUT                | curl -u admin:admin http://localhost/api/inventory/2?media=json             |
{  
  "serverId": "34564e92-3b4f-43ba-94a8-04309c0e48fe",
  "id": "2",
  "name": "10.66.23.1",
  "dsObjectId": 814,
  "snmpTimeout": 0,
  "snmpRetry": 0,
  "snmpPDUSize": 0,
  "protocol": "IPv4",
  "snmpType": "v3",
  "nameUsing": "CUSTOMNAME",
  "certified": "Yes",
  "polledName": "10.66.23.1",
  "managementIP": "10.66.23.1",
  "sysOid": ".1.3.6.1.4.1.2.3.50",
  "sysDescription": "10/100 Mbps Ethernet Switch",
  "sysLocation": "Simulator",
  "deviceType": "Ethernet Switch",
  "readCommunity": "public",
  "userName": "entuity",
  "authType": "MD5",
  "encrType": "DES",
  "managementLevel": "FULL",
  "context": ",",
  "zoneId": 0,
  "cliUsername": ""
}
**PUT Method detail**

Change the details of an inventory device, in either XML or JSON formats. Please refer to the ENA Help Center for help and information on how to add devices to your network from the ENA UI.

**Request Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authPass</td>
<td>SNMP v3 authentication password.</td>
</tr>
<tr>
<td>authType</td>
<td>SNMP v3 authentication type: NONE, MD5, SHA.</td>
</tr>
<tr>
<td>cliMethod</td>
<td>Connection method: SSH, TELNET.</td>
</tr>
<tr>
<td>cliPassword1</td>
<td>password 1.</td>
</tr>
<tr>
<td>cliPassword2</td>
<td>password 2.</td>
</tr>
<tr>
<td>cliPort</td>
<td>connection port.</td>
</tr>
<tr>
<td>cliUsername</td>
<td>CLI User name.</td>
</tr>
<tr>
<td>context</td>
<td>SNMP v3 context.</td>
</tr>
<tr>
<td>encrPass</td>
<td>SNMP v3 encryption password.</td>
</tr>
<tr>
<td>encrType</td>
<td>SNMP v3 encryption type: NONE, DES, AES.</td>
</tr>
<tr>
<td>name</td>
<td>display name.</td>
</tr>
<tr>
<td>nameUsing</td>
<td>device name is determined using either:</td>
</tr>
<tr>
<td></td>
<td>• CUSTOMNAME</td>
</tr>
<tr>
<td></td>
<td>• IPADDRESS</td>
</tr>
<tr>
<td></td>
<td>• POLLEDNAME</td>
</tr>
<tr>
<td></td>
<td>• RESOLVABLENAME</td>
</tr>
<tr>
<td></td>
<td>• RESOLVABLENAMEFQ</td>
</tr>
<tr>
<td></td>
<td>• SYSTEMNAME.</td>
</tr>
<tr>
<td>protocol</td>
<td>Transport protocol: IPv4 or IPv6.</td>
</tr>
<tr>
<td>readCommunity</td>
<td>SNMP read community string.</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>snmpPDUSize</td>
<td>maximum size of snmp PDU, where 0 = system default.</td>
</tr>
<tr>
<td>snmpRetry</td>
<td>number of snmp retries, where 0 = system default.</td>
</tr>
<tr>
<td>snmpTimeout</td>
<td>SNMP timeout in seconds, where 0 = system default</td>
</tr>
<tr>
<td>snmpType</td>
<td>SNMP type:</td>
</tr>
<tr>
<td></td>
<td>• v1</td>
</tr>
<tr>
<td></td>
<td>• v2c</td>
</tr>
<tr>
<td></td>
<td>• v3</td>
</tr>
<tr>
<td></td>
<td>• v1/v2c.</td>
</tr>
<tr>
<td>userName</td>
<td>SNMP v3 user name.</td>
</tr>
<tr>
<td>webAccessKey</td>
<td>access key (Amazon platform only).</td>
</tr>
<tr>
<td>webPassword</td>
<td>virtual platform password (non-Amazon virtual platforms).</td>
</tr>
<tr>
<td>webPlatformType</td>
<td>virtual platform type:</td>
</tr>
<tr>
<td></td>
<td>• VMWARE_ESXi</td>
</tr>
<tr>
<td></td>
<td>• ORACLE_VM_MANAGER</td>
</tr>
<tr>
<td></td>
<td>• HYPER_V</td>
</tr>
<tr>
<td></td>
<td>• AMAZON_WEB_SERVICE</td>
</tr>
<tr>
<td></td>
<td>• AZURE</td>
</tr>
<tr>
<td></td>
<td>• CISCO_APIC</td>
</tr>
<tr>
<td></td>
<td>• MERAKI</td>
</tr>
<tr>
<td>webSecretKey</td>
<td>secret key (Amazon platform only).</td>
</tr>
<tr>
<td>webTenantID</td>
<td>password for Microsoft Azure platform.</td>
</tr>
<tr>
<td>webURL</td>
<td>virtual platform URL (non-Amazon virtual platforms).</td>
</tr>
<tr>
<td>webUser</td>
<td>virtual platform user name (non-Amazon virtual platforms).</td>
</tr>
</tbody>
</table>

**Response data**

200 OK

**Examples**
curl -u admin:admin http://localhost/api/inventory/2?media=json -X PUT -d
 '{
   "managementLevel" : "FULL",
   "protocol" : "IPv4",
   "snmpType" : "v3",
   "userName" : "entuity",
   "authType" : "MD5",
   "authPass" : "entuity123",
   "encrType" : "DES",
   "encrPass" : "entuity123"
}'
"serverId": "34564e92-3b4f-43ba-94a8-04309c0e48fe",
"id": "2",
"name": "10.66.23.1",
"dsObjectId": 814,
"snmpTimeout": 0,
"snmpRetry": 0,
"snmpPDUSize": 0,
"protocol": "IPv4",
"snmpType": "v3",
"nameUsing": "CUSTOMNAME",
"certified": "Yes",
"polledName": "10.66.23.1",
"managementIP": "10.66.23.1",
"sysOid": ".1.3.6.1.4.1.2.3.50",
"sysDescription": "10/100 Mbps Ethernet Switch",
"sysLocation": "Simulator",
"deviceType": "Ethernet Switch",
"readCommunity": "public",
"userName": "entuity",
"authType": "MD5",
"encrType": "DES",
"managementLevel": "FULL",
"context": "",
"zoneId": 0,
"cliUsername": ""
INPUT

```bash
curl -u admin:admin http://localhost/api/inventory/2?media=json -X PUT -d
'<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<inventoryDevice
    managementLevel="FULL"
    snmpType="v3"
    userName="entuity"
    authType="MD5"
    authPass="entuity123"
    encrType="DES"
    encrPass="entuity123"
/>'
```

OUTPUT

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<inventoryDevice cliUsername="" zoneId="0" snmpPDUSize="0" snmpRetry="0" snmpTimeout="0" context="" encrType="DES" authType="MD5" userName="entuity" readCommunity="public" snmpType="v3" protocol="IPv4" certified="Yes" deviceType="Ethernet Switch" managementLevel="FULL" capabilities="Switch" sysLocation="Simulator" sysDescription="10/100 Mbps Ethernet Switch" sysOid=".1.3.6.1.4.1.2.3.50" managementIP="10.66.23.1" polledName="10.66.23.1" nameUsing="CUSTOM" dsObjectId="922" name="10.66.23.1" id="6" serverId="34564e92-3b4f-43ba-94a8-04309c0e48fe"/>
```

DELETE Method details

Remove a device from the inventory.

**Request**

The ID of the object that is to be deleted is sent as a part of a URL.

**Response**

"OK" message if the device was removed successfully, an error message otherwise.

**Examples**

```bash
INPUT
curl -u admin:admin http://localhost/api/inventory/2?media=json -X DELETE
```

```json
{
    "message": "OK"
}
```
<table>
<thead>
<tr>
<th>INPUT</th>
<th><code>curl -u admin:admin http://localhost/api/inventory/2?media=xml -X DELETE</code></th>
</tr>
</thead>
</table>
| OUTPUT| `<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <statusInfo>
        <message>OK</message>
    </statusInfo` |
Product Info

Details information about the installed version of Entuity/ENA.

URL: info

- Method Summary
- GET Method detail
  - Response
  - Examples

Method Summary

- GET Method - returns information about the installed version of Entuity/ENA.

GET Method detail

Returns information about the installed version of Entuity/ENA.

Response

Response is an object with the following attributes:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostAddress</td>
<td>host name for accessing the product.</td>
</tr>
<tr>
<td>id</td>
<td>server ID.</td>
</tr>
<tr>
<td>sslAccess</td>
<td>specifies to use HTTP (false) or HTTPS (true)</td>
</tr>
<tr>
<td>version</td>
<td>product version.</td>
</tr>
<tr>
<td>versionDisplay</td>
<td>user-friendly version string of the product.</td>
</tr>
<tr>
<td>webPort</td>
<td>port number for accessing the product over HTTP(S).</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>INPUT</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>curl</td>
<td>-u admin:admin <a href="http://localhost/api/info?media=json">http://localhost/api/info?media=json</a></td>
</tr>
</tbody>
</table>
### INPUT
```
curl -u admin:admin http://localhost/api/info?media=xml
```

### OUTPUT
```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<serverInfo sslAccess="false" webPort="80" hostAddress="ENTLONDEV08" product="EYE" versionDisplay="Entuity 17.0" version="17.0.0.p0" id="34564e92-3b4f-43ba-94a8-04309c0e48fe"/>
```
Servers

List server or add a new server.

URL: servers

- Method Summary
- GET Method detail
  - Request Parameters
  - Response data keys
  - Examples

Method Summary

- GET Method - lists Entuity servers (XML, JSON).
- POST Method - adds a new server (XML, JSON).

GET Method detail

Lists Entuity servers, in either XML or JSON format.

Request Parameters

None

Response data keys

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>number of servers.</td>
</tr>
<tr>
<td>servers</td>
<td>list of servers.</td>
</tr>
<tr>
<td>server</td>
<td>Server summary details, with the following attributes:</td>
</tr>
<tr>
<td></td>
<td>• hostname: DNS name of the host</td>
</tr>
<tr>
<td></td>
<td>• Id: Unique server Identifier.</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>INPUT</th>
<th>Curl Command</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>curl -u admin:admin <a href="http://localhost/api/servers?media=json">http://localhost/api/servers?media=json</a></td>
</tr>
</tbody>
</table>
### OUTPUT

```json
{
    "items": [
        {
            "serverId": "34564e92-3b4f-43ba-94a8-04309c0e48fe",
            "id": "34564e92-3b4f-43ba-94a8-04309c0e48fe",
            "name": "ENTLONDEV08"
        }
    ],
    "count": 1
}
```

### INPUT

```
curl -u admin:admin http://localhost/api/servers?media=xml
```

### OUTPUT

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<items count="1">
    <item xsi:type="namedItem" name="ENTLONDEV08" id="34564e92-3b4f-43ba-94a8-04309c0e48fe" serverId="34564e92-3b4f-43ba-94a8-04309c0e48fe" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
</items>
```
Servers details

List, update or delete a server.

URL: servers/{serverId}

- Method Summary
- GET Method detail
  - Request Parameters
  - Response data keys
  - Examples

Method Summary

- GET Method - list details of a server (XML, JSON).
- PUT Method - change details of a server (XML, JSON).
- DELETE Method - remove a server (XML, JSON).

GET Method detail

List details of a server, in either XML or JSON format.

Request Parameters

None

Response data keys

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>centralServer</td>
<td>if the server is a Central Server: True or False.</td>
</tr>
<tr>
<td>hostname</td>
<td>name of host running the Entuity server.</td>
</tr>
<tr>
<td>included</td>
<td>if the server provides results in a multi server system: True or False.</td>
</tr>
<tr>
<td>licensed</td>
<td>if the server is licensed: True or False.</td>
</tr>
<tr>
<td>local</td>
<td>if this server is the one servicing the request: True or False</td>
</tr>
<tr>
<td>role</td>
<td>role of the server: Polling, FlowCollector, ESPServer.</td>
</tr>
<tr>
<td>serverId</td>
<td>unique server identifier.</td>
</tr>
<tr>
<td>ssl</td>
<td>if the server is configured to use Secure Socket Layer: True or False.</td>
</tr>
</tbody>
</table>
### Examples

<table>
<thead>
<tr>
<th>INPUT</th>
<th>curl -u admin:admin <a href="http://localhost/api/server/34564e92-3b4f-43ba-94a8-04309c0e48fe?media=json">http://localhost/api/server/34564e92-3b4f-43ba-94a8-04309c0e48fe?media=json</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPUT</td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>&quot;name&quot;: &quot;ENTLONDEV08&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;included&quot;: true,</td>
</tr>
<tr>
<td></td>
<td>&quot;role&quot;: &quot;Polling&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;webPort&quot;: 80,</td>
</tr>
<tr>
<td></td>
<td>&quot;ssl&quot;: false,</td>
</tr>
<tr>
<td></td>
<td>&quot;local&quot;: true,</td>
</tr>
<tr>
<td></td>
<td>&quot;serverId&quot;: &quot;34564e92-3b4f-43ba-94a8-04309c0e48fe&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;licensed&quot;: true,</td>
</tr>
<tr>
<td></td>
<td>&quot;centralServer&quot;: false</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INPUT</th>
<th>curl -u admin:admin <a href="http://localhost/api/servers/34564e92-3b4f-43ba-94a8-04309c0e48fe?media=xml">http://localhost/api/servers/34564e92-3b4f-43ba-94a8-04309c0e48fe?media=xml</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPUT</td>
<td>&lt;?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot; standalone=&quot;yes&quot;?&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;entuityServer licensed=&quot;true&quot; role=&quot;Polling&quot; local=&quot;true&quot; included=&quot;true&quot; centralServer=&quot;false&quot; ssl=&quot;false&quot; webPort=&quot;80&quot; name=&quot;ENTLONDEV08&quot; serverId=&quot;34564e92-3b4f-43ba-94a8-04309c0e48fe&quot;/&gt;</td>
</tr>
</tbody>
</table>
User Groups

This resource lists general information about user groups.

URL: userGroups

- Methods Summary
- GET Method detail
  - Response
  - Examples
- POST Method details
  - Request
  - Response
  - Examples

Methods Summary

- GET Method - lists available user groups.
- POST Method - creates a new user group.

GET Method detail

Lists available user groups. For non-administrators, the list is restricted to the groups to which the user currently belongs.

Response

Response includes a list of user groups. Each user group has the following attributes:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>user group id unique to the server.</td>
</tr>
<tr>
<td>name</td>
<td>user group name.</td>
</tr>
<tr>
<td>serverId</td>
<td>Entuity Server Id on which resource resides.</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>curl -u admin:admin <a href="http://localhost/api/userGroups?media=json">http://localhost/api/userGroups?media=json</a></td>
</tr>
</tbody>
</table>
POST Method details

Creates a new group of users.

**Request**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>name of the group.</td>
</tr>
</tbody>
</table>

**Response**

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<items count="2">
  <item xsi:type="namedItem" name="Administrators" id="1" serverId="ee934fe2-1f4d-4f1e-a0b5-a87b261eb30a" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
  <item xsi:type="namedItem" name="All Users" id="2" serverId="ee934fe2-1f4d-4f1e-a0b5-a87b261eb30a" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
</items>
```
The updated list of user groups, as the GET method would return them.

**Examples**

<table>
<thead>
<tr>
<th>INPUT</th>
<th>curl -u admin:admin <a href="http://localhost/api/userGroups?media=json">http://localhost/api/userGroups?media=json</a> -X POST -H &quot;content-type: application/xml&quot; -d</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>'{</td>
</tr>
</tbody>
</table>
|       |   "name" : "Support"
|       |}'                                                                                                      |
| OUTPUT| {                                                                                                       |
|       |   "items": [
|       |     {
|       |       "serverId": "ee934fe2-1f4d-4f1e-a0b5-a87b261eb30a",
|       |       "id": "1",
|       |       "name": "Administrators"
|       |     },
|       |     {
|       |       "serverId": "ee934fe2-1f4d-4f1e-a0b5-a87b261eb30a",
|       |       "id": "2",
|       |       "name": "All Users"
|       |     },
|       |     {
|       |       "serverId": "ee934fe2-1f4d-4f1e-a0b5-a87b261eb30a",
|       |       "id": "6",
|       |       "name": "Support"
|       |     }
|       |   },
|       |   "count": 3                                                                                           |
|       | }                                                                                                       |

<table>
<thead>
<tr>
<th>INPUT</th>
<th>curl -u admin:admin <a href="http://localhost/api/userGroups?media=xml">http://localhost/api/userGroups?media=xml</a> -X POST -H &quot;content-type: application/xml&quot; -d</th>
</tr>
</thead>
</table>
|       | '<userGroupInfo audit="false">
|       |   <name>Support</name>
|       | </userGroupInfo>'                                                                                     |
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<items count="3">
    <item xsi:type="namedItem" name="Administrators" id="1" serverId="ee934fe2-1f4d-4f1e-a0b5-a87b261eb30a" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>

    <item xsi:type="namedItem" name="All Users" id="2" serverId="ee934fe2-1f4d-4f1e-a0b5-a87b261eb30a" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>

    <item xsi:type="namedItem" name="Support" id="6" serverId="ee934fe2-1f4d-4f1e-a0b5-a87b261eb30a" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>

</items>
User Group details

Lists a set of operations on a particular group.

URL: userGroups/{groupId}

- Methods Summary
  - DELETE Method details
    - Response
    - Examples

Methods Summary

- DELETE Method - deletes the user group.

DELETE Method details

Deletes the user group.

Response

The updated list of user groups, as GET method would return them.

Examples

<table>
<thead>
<tr>
<th>INPUT</th>
<th>curl -u admin:admin <a href="http://localhost/api/userGroups?media=json">http://localhost/api/userGroups?media=json</a> -X DELETE</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPUT</td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>&quot;items&quot;: [</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>&quot;serverId&quot;: &quot;ee934fe2-1f4d-4f1e-a0b5-a87b261eb30a&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;id&quot;: &quot;1&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;name&quot;: &quot;Administrators&quot;</td>
</tr>
<tr>
<td></td>
<td>},</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>&quot;serverId&quot;: &quot;ee934fe2-1f4d-4f1e-a0b5-a87b261eb30a&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;id&quot;: &quot;2&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;name&quot;: &quot;All Users&quot;</td>
</tr>
<tr>
<td></td>
<td>}],</td>
</tr>
<tr>
<td></td>
<td>&quot;count&quot;: 2</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td>INPUT</td>
<td>curl -u admin:admin <a href="http://localhost/api/userGroups?media=xml">http://localhost/api/userGroups?media=xml</a> -X DELETE</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| OUTPUT| <?xml version="1.0" encoding="UTF-8" standalone="yes"?>  
|       | <items count="2">  
|       |   <item xsi:type="namedItem" name="Administrators" id="1" serverId="ee934fe2-1f4d-4f1e-a0b5-a87b261eb30a" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>  
|       |   <item xsi:type="namedItem" name="All Users" id="2" serverId="ee934fe2-1f4d-4f1e-a0b5-a87b261eb30a" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>  
|       | </items> |
User Group permissions

Lists and modifies the set of tools accessible to a user group.

URL: userGroups/{groupId}/tools

## Methods Summary

- GET Method details
  - Response
  - Examples
- PUT Method details
  - Request
  - Response
  - Examples

## GET Method details

The list of tools that this group has a permission to access.

**Response**

List of items, each one according to the following format:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>tool ID, unique per server.</td>
</tr>
<tr>
<td>name</td>
<td>tool name.</td>
</tr>
<tr>
<td>subgroup</td>
<td>the name of subgroup that given tool is a part of.</td>
</tr>
</tbody>
</table>

**Examples**

**INPUT**

curl -u admin:admin http://localhost/api/userGroups/6/tools?media=json

**OUTPUT**

```
{
   "items": [],
   "count": 0
}
```
**INPUT**
curl -u admin:admin http://localhost/api/userGroups/6/tools?media=xml

**OUTPUT**
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<items count="0"/>

---

**PUT Method details**

Modifies the list of tools to which the user group has permission to access.

**Request**
The list of tool IDs.

**Response**
The updated list of group permissions, as the GET method would return them.

**Examples**

**INPUT**
curl -u admin:admin http://localhost/api/userGroups/6?media=json X PUT -H "content-type: application/json" -d

'{
  "tools" : [
    {
      "id" : 15
    },
    {
      "id" : 89
    }
  ]
}'
OUTPUT
{
   "items" : [ {
      "id" : 15,
      "toolName" : "Ticker",
      "groupName" : "Tools"
   }, {
      "id" : 89,
      "toolName" : "Data Export",
      "groupName" : "Administrator Tools"
   } ],
   "count" : 2
}

INPUT
curl -u admin:admin http://localhost/api/userGroups/6?media=xml -X PUT -H "content-type: application/xml" -d
'<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<groupToolsWrapper>
   <tool id="15" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
   <tool id="89" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
</groupToolsWrapper>'

OUTPUT
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<items count="2">
   <item xsi:type="toolId" groupName="Tools" id="15" toolName="Ticker"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
   <item xsi:type="toolId" groupName="Administrator Tools" id="89"
      toolName="Data Export" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
</items>
Tools

Lists tools available on the server.

URL: tools

- Methods summary
- GET Method details
  - Response
  - Examples

Methods summary

- GET Methods - lists all tools available on the server.

GET Method details

Lists all tools available on the server.

Response

The list of tools are grouped into subgroups, where every entry has the following format:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>tool ID, unique per server.</td>
</tr>
<tr>
<td>name</td>
<td>the name of the tool.</td>
</tr>
</tbody>
</table>

Examples

| INPUT     | curl -u admin:admin http://localhost/api/tools?media=json |
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<items count="6">
    <item xsi:type="toolsGroup" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <subgroup>Administrator Tools</subgroup>
        <tools>
            <tools name="Event Administration" id="79"/>
            ...removed for brevity...
            <tools name="UD Polling" id="111"/>
        </tools>
    </item>
    ...removed for brevity...
    <item xsi:type="toolsGroup" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <subgroup>Tools</subgroup>
        <tools>
            <tools name="Ticker" id="15"/>
            ...removed for brevity...
            <tools name="Configuration Management" id="112"/>
        </tools>
    </item>
    ...removed for brevity...
</items>
Users

Lists information about users.

URL: users

- Methods Summary
- GET Method detail
  - Response
  - Examples
- POST Method details
  - Request
  - Response
  - Example

Methods Summary

- GET Method - lists available users.
- POST Method - creates a new user.

GET Method detail

The list of users returned is restricted for non-administrators: only the user object corresponding to the current user is returned.

Response

Response includes a list of users. Each user has following attributes.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>User id unique to the server</td>
</tr>
<tr>
<td>name</td>
<td>User name</td>
</tr>
<tr>
<td>serverId</td>
<td>Entuity Server Id on which resource resides</td>
</tr>
</tbody>
</table>

Examples

```
INPUT  | curl -u admin:admin http://localhost/api/users?media=json
```
POST Method details

Creates a new user.

Request

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>username.</td>
</tr>
<tr>
<td>password</td>
<td>user password.</td>
</tr>
</tbody>
</table>

Response

The list of users after an update, as GET method would return them.

Example
```
INPUT  
curl -u admin:admin http://localhost/api/users?media=json -X POST -H
"content-type: application/json" -d
'{'
   "name" : "John",
   "password" : "little"
}"

OUTPUT  
{
   "items": [ {
      "serverId": "ee934fe2-1f4d-4f1e-a0b5-a87b261eb30a",
      "id": "6",
      "name": "John"
   }, {
      "serverId": "ee934fe2-1f4d-4f1e-a0b5-a87b261eb30a",
      "id": "3",
      "name": "admin"
   } ],
   "count": 2
}
```

```
INPUT  
"content-type: application/xml" -d
'<'userPasswordWrapper>
   <name>John</name>
   <password>little</password>
</userPasswordWrapper>'

OUTPUT  
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
/items count="2">
   <item xsi:type="namedItem" name="John" id="6" serverId="ee934fe2-1f4d-4f1e-a0b5-a87b261eb30a" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
   <item xsi:type="namedItem" name="admin" id="3" serverId="ee934fe2-1f4d-4f1e-a0b5-a87b261eb30a" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
</items>
```
User details

List detailed user information, modify parameters of a user, and delete a user.

URL: users/{userID}

- **Methods summary**
- **GET Method details**
  - Response
  - Examples
- **PUT Method details**
  - Request
  - Response
  - Examples
- **DELETE Method details**
  - Response
  - Examples

**Methods summary**

- GET Method - lists detailed user info.
- PUT Method - modifies parameters of a user.
- DELETE Method - deletes user.

**GET Method details**

Returns detailed information about the given user.

**Response**

User info structure with the following format:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>user ID, unique per server.</td>
</tr>
<tr>
<td>name</td>
<td>name of the user.</td>
</tr>
<tr>
<td>lockAttempts</td>
<td>number of failed login attempts before locking the account.</td>
</tr>
<tr>
<td>expiryDays</td>
<td>number of days after which the account expires.</td>
</tr>
<tr>
<td>timeoutMinutes</td>
<td>number of minutes after which session becomes inactive.</td>
</tr>
<tr>
<td>pwChangeDays</td>
<td>days to password change.</td>
</tr>
<tr>
<td>forcePWChange</td>
<td>whether the user needs to change their password.</td>
</tr>
<tr>
<td>groups</td>
<td>list of groups that this user is a member of.</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>locked</td>
<td>“locked” status of the user.</td>
</tr>
<tr>
<td>admin</td>
<td>if the user is admin.</td>
</tr>
<tr>
<td>expired</td>
<td>if the user account expired.</td>
</tr>
</tbody>
</table>

**Examples**

```
curl -u admin:admin http://localhost/api/users/6?media=json
```

```
{
    "id" : 6,
    "name" : "John",
    "lockAttempts" : -3,
    "expiryDays" : -14,
    "pwChangeDays" : -14,
    "forcePWChange" : false,
    "groups" : [ "All Users" ],
    "locked" : false,
    "expired" : false,
    "admin" : false
}
```

```
curl -u admin:admin http://localhost/api/users/6?media=xml
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<userAccountInfo expired="false" locked="false" admin="false" id="6">
    <name>John</name>
    <forcePWChange>false</forcePWChange>
    <lockAttempts>-3</lockAttempts>
    <expiryDays>-14</expiryDays>
    <pwChangeDays>-14</pwChangeDays>
    <groups>All Users</groups>
</userAccountInfo>
```
PUT Method details

Modifies parameters of a user.

Request

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lockAttempts</td>
<td>number of failed login attempts before locking the account.</td>
</tr>
<tr>
<td>expiryDays</td>
<td>number of days after which the account expires.</td>
</tr>
<tr>
<td>timeoutMinutes</td>
<td>number of minutes after which session becomes inactive.</td>
</tr>
<tr>
<td>pwChangeDays</td>
<td>days to password change.</td>
</tr>
<tr>
<td>forcePWChange</td>
<td>if the user needs to change their password.</td>
</tr>
<tr>
<td>groups</td>
<td>list of groups that this user is a member of.</td>
</tr>
</tbody>
</table>

Response

Detailed information about the user after changes, as the GET method would return it.

Examples

```
curl -u admin:admin http://localhost/api/users/6?media=json -X PUT -H "content-type: application/json" -d "
  
  
  
  "name" : "John",
  "lockAttempts" : -3,
  "expiryDays" : 14,
  "forcePWChange" : true,
  "groups" : [ "Administrators", "Support" ]
```


```bash
curl -u admin:admin http://localhost/api/users/6?media=xml -X PUT -H "content-type: application/xml" -d '  
  <userAccountInfo expired="false" locked="false" admin="false">
    <name>John</name>
    <lockAttempts>3</lockAttempts>
    <expiryDays>14</expiryDays>
    <forcePWChange>true</forcePWChange>
    <groups>Administrators</groups>
    <groups>Support</groups>
  </userAccountInfo>'
```
DELETE Method details

Deletes user account.

Response

The updated list of users, as the GET method of URL "users" would return it.

Examples

```bash
curl -u admin:admin http://localhost/api/users/6?media=json -X DELETE
```

```
{
    "items": [
        {
            "serverId": "ee934fe2-1f4d-4f1e-a0b5-a87b261eb30a",
            "id": "3",
            "name": "admin"
        }
    ],
    "count": 1
}
```

```bash
curl -u admin:admin http://localhost/api/users/6?media=xml -X DELETE
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
items count="1">
    <item xsi:type="namedItem" name="admin" id="3" serverId="ee934fe2-1f4d-4f1e-a0b5-a87b261eb30a" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
</items>
View List

List available Views or create a new View.

URL: views

- Method Summary
- GET Method detail
  - Response
  - Examples
- POST Method detail
  - Request
  - Response
  - Examples

Method Summary

- GET Method - lists available Views.
- POST Method - create a new View.

GET Method detail

Lists available Views.

Response

Response includes a list of Views. Each View has the following attributes.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>displayName</td>
<td>View name.</td>
</tr>
<tr>
<td>id</td>
<td>View id unique to the server.</td>
</tr>
<tr>
<td>path</td>
<td>View path with forward slash as a sub-view separator.</td>
</tr>
<tr>
<td>serverId</td>
<td>Entuity Server Id on which resource resides.</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>curl -u admin:admin <a href="http://localhost/api/views?media=json">http://localhost/api/views?media=json</a></td>
</tr>
</tbody>
</table>
POST Method detail

Creates a new View.

**Request**

Request is an object which may contain a subset of following properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accessGroups</td>
<td>array of access objects, specifying view access permissions.</td>
</tr>
<tr>
<td>baseViewAggregation</td>
<td>A way to aggregate base views: One of NONE (default), UNION or INTERSECTION.</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>baseViewPaths</td>
<td>array of base views</td>
</tr>
<tr>
<td>domainFilterName</td>
<td>name of the domain filter to use</td>
</tr>
<tr>
<td>eventFilterName</td>
<td>name of the event filter to use</td>
</tr>
<tr>
<td>incidentFilterName</td>
<td>name of the incident filter to use</td>
</tr>
<tr>
<td>name</td>
<td>name of the View being created.</td>
</tr>
<tr>
<td>owner</td>
<td>username of the user who will be an owner of the View. This defaults to the user making a call. Only administrators may specify a user other than themselves.</td>
</tr>
<tr>
<td>parentViewPath</td>
<td>forward-slash separated path of the parent view. Leave out to create a top-level view.</td>
</tr>
</tbody>
</table>

**Response**

The updated list of Views, as when the GET response is used.

**Examples**

You can create a View using the following command:

curl -u admin:admin -H "content-type: application/json" -X POST --data @createView.json http://localhost/api/views

Following are examples for the createView.json file.

**Creates a top-level View with most information set to default.**

```
{
   "name" : "Simple View"
}
```

**Creates a top-level View with filter information provided.**

```
{
   "name" : "MyView",
   "domainFilterName" : "All Objects",
   "eventFilterName" : "All Events",
   "incidentFilterName" : "All Incidents"
}
```
Creates a Subview

```json
{
    "name": "SubView",
    "parentViewPath": "My Top View/Second-Level View"
}
```

Creates a View which unions contents from the base Views

```json
{
    "name": "Union View",
    "baseViewAggregation": "UNION",
    "baseViewPaths": [ "MyView", "My Top View/Second-Level View/SubView" ]
}
```

Creates a View which intersects contents from the base Views

```json
{
    "name": "Intersection View",
    "baseViewAggregation": "INTERSECTION",
    "baseViewPaths": [ "MyView", "My Top View/Second-Level View/SubView" ]
}
```

Creates a View with a specific owner and access rights
{
    "name" : "John's View",
    "owner" : "John",
    "accessGroups" : [ {
        "userGroupName" : "Support",
        "editable" : true
    }, {
        "userGroupName" : "All Users",
        "editable" : false
    } ]
}
View details

Inspect, update or delete a View.

URL: views/{id}

- Method Summary
- GET Method detail
  - Response
  - Examples
- PUT Method detail
  - Request
  - Response
  - Examples
- DELETE Method detail
  - Request
  - Response
  - Examples

Method Summary

- GET Method - inspect a View.
- PUT Method - update a View.
- DELETE Method - delete a View.

GET Method detail

Response

Response is the entity with the following attributes:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accessGroups</td>
<td>group access permissions.</td>
</tr>
<tr>
<td>baseViewAggregation</td>
<td>one of NONE, UNION, INTERSECTION.</td>
</tr>
<tr>
<td>baseViewPaths</td>
<td>array of base View paths.</td>
</tr>
<tr>
<td>displayName</td>
<td>View name.</td>
</tr>
<tr>
<td>domainFilterName</td>
<td>domain filter name.</td>
</tr>
<tr>
<td>eventFilterName</td>
<td>event filter name.</td>
</tr>
<tr>
<td>id</td>
<td>View id unique to the server.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>implicitAccessGroups</td>
<td>groups having implicit access by virtue of inheriting access through other group permissions, such as having access to the parent View. This may be indirect, e.g. access to the grandparent or great-grandparent.</td>
</tr>
<tr>
<td>implicitAccessUsers</td>
<td>users having implicit access by virtue of inheriting access through other user permissions, such as having access to the parent View. This may be indirect, e.g. access to the grandparent or great-grandparent.</td>
</tr>
<tr>
<td>incidentFilterName</td>
<td>incident filter name.</td>
</tr>
<tr>
<td>manuallyPopulated</td>
<td>if contents of the View are populated manually (true) or automatically (false).</td>
</tr>
<tr>
<td>owner</td>
<td>user owning a View.</td>
</tr>
<tr>
<td>path</td>
<td>View path.</td>
</tr>
<tr>
<td>serverId</td>
<td>Entuity Server Id on which resource resides.</td>
</tr>
</tbody>
</table>

**Examples**

**INPUT**
```
curl -u admin:admin http://localhost/api/views/48?media=json
```

**OUTPUT**
```
{
  "serverId" : "821c3e87-bcdf-4f6f-a5e8-7d2524928d96",
  "id" : "48",
  "path" : "Simple View",
  "displayName" : "Simple View",
  "baseViewAggregation" : "NONE",
  "baseViewPaths" : [ ],
  "domainFilterName" : "All Objects",
  "manuallyPopulated" : true,
  "eventFilterName" : "All Events",
  "incidentFilterName" : "All Incidents",
  "owner" : "admin",
  "accessGroups" : [ {
    "userGroupName" : "Administrators",
    "editable" : true
  } ],
  "implicitAccessGroups" : [ ],
  "implicitAccessUsers" : [ ]
}
```
### PUT Method detail

Update existing View.

#### Request

Request has the same structure as the POST request of views resource for adding a new view, except the following property must be absent: `parentViewPath`. You need to specify only properties you want to be changed.

#### Response

Response has the same structure as the GET response: shows View details after the update.

#### Examples

```bash
curl -u admin:admin -H "content-type: application/json" -X PUT --data @updateView.json http://localhost/api/views/48
```

```bash
INPUT
curl -u admin:admin -X PUT -H "content-type: application/json" -d '{
   "name" : "myView"
}' http://localhost/api/views/4
```

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<viewPathDetails owner="admin" incidentFilterName="All Incidents" eventFilterName="All Events" manuallyPopulated="true" domainFilterName="All Objects" baseViewAggregation="NONE" displayName="Simple View" path="Simple View" id="48" serverId="821c3e87-bcdf-4ef8-a5e8-7d2524928d96">
   <accessGroup editable="true" userGroupName="Administrators"/>
</viewPathDetails>
```
### DELETE Method detail

Delete existing View.

#### Request

No request expected.

#### Response

Empty response.

#### Examples

Delete View with id 48.

```
curl -u admin:admin -X DELETE http://localhost/api/views/48
```
View Objects

Inspect, add or delete View objects.

URL: views/{id}/objects

Method Summary

- GET Method detail
  - Request
  - Response
  - Examples
- PUT Method detail
  - Request
  - Response
  - Examples
- DELETE Method detail
  - Request
  - Response
  - Examples

Method Summary

- GET Method - list View objects.
- PUT Method - add objects to a View.
- DELETE Method - remove objects from a View.

GET Method detail

Method returns objects contained in a View.

Request

The following query parameter can be specified:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>indirect</td>
<td>set to a special value “include” to return objects from Subviews as well. By default, objects from Subviews are not included.</td>
</tr>
</tbody>
</table>

Response

List of items. Each item has the following properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>displayName</td>
<td>name of the object.</td>
</tr>
<tr>
<td>id</td>
<td>object id unique to the server.</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>serverId</td>
<td>Entuity Server Id on which resource resides.</td>
</tr>
<tr>
<td>typeDisplayName</td>
<td>user-friendly name of the object type.</td>
</tr>
<tr>
<td>typeName</td>
<td>name of the object type.</td>
</tr>
</tbody>
</table>

**Examples**

**INPUT**

```
curl -u admin:admin http://localhost/api/views/49/objects?media=json
```

**OUTPUT**

```
{
    "items" : [ {
        "serverId" : "821c3e87-bcdf-4fef-a5e8-7d2524928d96",
        "id" : 769,
        "typeName" : "ManagedHost",
        "typeDisplayName" : "Managed Host",
        "displayName" : "bvt"
    } ],
    "count" : 1
}
```

**INPUT**

```
curl -u admin:admin http://localhost/api/views/49/objects?media=xml
```

**OUTPUT**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<items count="1">
    <item xsi:type="viewContentItem" displayName="bvt" typeDisplayName="Managed Host" typeName="ManagedHost" id="769" serverId="821c3e87-bcdf-4fef-a5e8-7d2524928d96" />
</items>
```

**PUT Method detail**

**Request**
Following query parameters can be specified

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>item id. This parameter can appear multiple times.</td>
</tr>
</tbody>
</table>

**Response**

Response is the same as from a GET method after objects have been added.

**Examples**

Add two items (2024 and 3279) to a View:

curl -u admin:admin -X PUT "http://localhost/api/views/49/objects?media=json&id=2024&id=3279"

**DELETE Method detail**

**Request**

The following query parameters can be specified:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>object id. This parameter can appear multiple times.</td>
</tr>
<tr>
<td>include</td>
<td>Can have a special value “all” to empty a View completely. If specified any ‘id’ parameters are ignored.</td>
</tr>
</tbody>
</table>

**Response**

The updated list of View objects, as would be received from the GET method.

**Examples**

Remove two objects (2024 and 3279) from a View:

curl -u admin:admin -X DELETE "http://localhost/api/views/49/objects?media=json&id=2024&id=3279"

Remove all items from a View:

curl -u admin:admin -X DELETE "http://localhost/api/views/49/objects?media=json&include=all"
Object Attributes

Lists attributes of an object.

URL: objects/{swID}/attributes?includeDetails=true&name=name1&name=name2

- Methods summary
- GET Method details
  - Request
  - Response
  - Examples

Methods summary

- GET Method - lists attributes of the object.

GET Method details

Lists the attributes of the object.

Request

The request may contain a list of attribute names to retrieve in the format “name=value”. By default, all attributes are returned.

It is also possible to supply “includeDetails” flag. It is assumed to be true if this value is set to one of: “true”, “yes”, “t”, “y” or “1”. If it is set to any other value, it is false. If a value is absent, then it is automatically set to true if there is at least one “name=value” entry, otherwise it is false.

The “includeDetails” controls the level of details returned by this method. If false, only the attribute names will be returned. If true, attribute names together with their values will be returned.

Response

An array of entries, as described below.

If “includeDetails” is false, every entry has the following format:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>name of the attribute.</td>
</tr>
</tbody>
</table>

If “includeDetails” is true, every entry has the following format:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>value of “name” attribute of the device.</td>
</tr>
<tr>
<td>attributeName</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>displayName</td>
<td>value of “displayName” attribute of the device</td>
</tr>
<tr>
<td>userEditable</td>
<td>if the attribute can be modified by the user.</td>
</tr>
<tr>
<td>userOverriden</td>
<td>if the attribute is set to polled or user-defined value.</td>
</tr>
<tr>
<td>values</td>
<td>list of values for the attribute.</td>
</tr>
</tbody>
</table>

**Examples**

**INPUT**

```bash
curl -u admin:admin
http://localhost/api/objects/611/attributes?media=json&includeDetails=y
```

**OUTPUT**

```
{
    "items" : [ 
        {
            "name" : "devType",
            "displayName" : "Device Type",
            "userEditable" : false,
            "userOverriden" : false,
            "values" : [ "148" ]
        },
        ...
        {
            "name" : "category",
            "displayName" : "category",
            "userEditable" : false,
            "userOverriden" : false,
            "values" : [ "device" ]
        } ],
    "count" : 30
}
```

**INPUT**

```bash
curl -u admin:admin
http://localhost/api/objects/611/attributes?media=xml&includeDetails=y
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<items count="30">

  <item xsi:type="attribute" userOverriden="false" userEditable="false" displayName="Device Type" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <name>devType</name>
    <value>148</value>
  </item>

  ...removed for brevity...

  <item xsi:type="attribute" userOverriden="false" userEditable="false" displayName="category" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <name>category</name>
    <value>device</value>
  </item>

</items>
Object Attribute details

List values of an object attribute, and modify an object attribute.

URL: objects/{swID}/attributes/{attributeName}

Methods summary

- GET Method details
  - Response
  - Examples
- PUT Method details
  - Request
  - Response
  - Examples

GET Method details

Lists the values of an attribute on this object.

Response

The list of attributes, with entries according to the following format:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>value of “name” attribute of the device.</td>
</tr>
<tr>
<td>displayName</td>
<td>value of “displayName” attribute of the device.</td>
</tr>
<tr>
<td>userEditable</td>
<td>if the attribute can be user-modified.</td>
</tr>
<tr>
<td>userOverriden</td>
<td>if the attribute is set to polled or a user-defined value.</td>
</tr>
<tr>
<td>values</td>
<td>list of values for the attribute.</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>INPUT</th>
<th>curl -u admin:admin <a href="http://localhost/api/objects/611/attributes/typeName?media=json">http://localhost/api/objects/611/attributes/typeName?media=json</a></th>
</tr>
</thead>
</table>
### PUT Method details

Modifies an attribute of the object.

**Request**

An attribute definition:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name of attribute to be changed.</td>
</tr>
<tr>
<td>userOverridden</td>
<td>if the attribute has user-supplied or polled value. The fields &quot;userOverridden&quot; and &quot;values&quot; are mutually exclusive, meaning that &quot;values&quot; are only permitted if the &quot;userOverridden&quot; is false.</td>
</tr>
<tr>
<td>values</td>
<td>list of values for the attribute.</td>
</tr>
</tbody>
</table>

**Response**

An updated attribute:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>value of “name” attribute of the device.</td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>displayName</td>
<td>value of “displayName” attribute of the device.</td>
</tr>
<tr>
<td>userEditable</td>
<td>if the attribute can be modified by the user.</td>
</tr>
<tr>
<td>userOverriden</td>
<td>if the attribute is set to polled or a user-defined value.</td>
</tr>
<tr>
<td>values</td>
<td>list of values for this attribute.</td>
</tr>
</tbody>
</table>

**Examples**

**INPUT**
```
  '{
    "name" : "TransferServer",
    "displayName" : "Configuration Transfer Server",
    "userOverriden" : true,
    "values" : [ "10.44.2.101" ]
  }'
```

**OUTPUT**
```
{
    "name" : "TransferServer",
    "displayName" : "Configuration Transfer Server",
    "userEditable" : true,
    "userOverriden" : true,
    "values" : [ "10.44.2.101" ]
}
```

**INPUT**
```
  '<objectAttribute userOverriden="true" userEditable="true" displayName="Configuration Transfer Server"> 
    <name>TransferServer</name>
    <value>10.44.2.101</value>
  </objectAttribute>''
```
<objectAttribute userOverriden="true" userEditable="true" displayName="Configuration Transfer Server">
  <name>TransferServer</name>
  <value>10.44.2.101</value>
</objectAttribute>
Object Ports

Lists the ports on an object.

URL: objects/{swID}/ports?includeVirtual=BOOL&includeUnmanaged=BOOL

- Methods summary
- GET Method details
  - Response
  - Examples

Methods summary

- GET Method - lists ports on the object.

GET Method details

This method accepts one of two boolean parameters:

- one that equals "yes", "true", "y", "t" or "1". This is treated as 'true'.
- one that equals any other value, or the absence of a value. This is treated as ‘false’.

includeVirtual – controls whether virtual ports should be included in results.

includeUnmanaged – controls whether data for unmanaged ports should be included.

Response

The list of attributes, with entries according to the following format:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>StormWorks ID of the associated object.</td>
</tr>
<tr>
<td>typeName</td>
<td>type of the associated object.</td>
</tr>
<tr>
<td>displayName</td>
<td>display name.</td>
</tr>
<tr>
<td>displayType</td>
<td>display type.</td>
</tr>
<tr>
<td>hasServiceStatus</td>
<td>if object has a service status.</td>
</tr>
</tbody>
</table>

Examples
<table>
<thead>
<tr>
<th>INPUT</th>
<th>curl -u admin:admin <a href="http://localhost/api/objects/655/fans?includeVirtual=yes&amp;includeUnmanaged=yes&amp;media=json">http://localhost/api/objects/655/fans?includeVirtual=yes&amp;includeUnmanaged=yes&amp;media=json</a></th>
</tr>
</thead>
</table>
| OUTPUT | {  
|      |   "items": [ {  
|      |       "portAttributes": {  
|      |           "compId": {  
|      |               "ids": [  
|      |                   1,  
|      |                   2,  
|      |                   1,  
|      |                   0  
|      |               ],  
|      |               "type": 1,  
|      |               "invalid": false,  
|      |               "port": true,  
|      |               "networkPath": false,  
|      |               "device": false,  
|      |               "root": false,  
|      |               "dsObject": false,  
|      |               "view": false  
|      |           },  
|      |           "connectedHostIps": "",  
|      |           "connectedHostMacs": "",  
|      |           "connectedHostNames": "",  
|      |           "connectedHosts": "",  
|      |           "displayName": " [ Gi0/0 ] to 10.44 lan",  
|      |           "ifDescr": " [ Gi0/0 ] to 10.44 lan",  
|      |           "ifIndex": "1",  
|      |           "ifType": "Ethernet",  
|      |           "ipAddresses": "10.44.1.59",  
|      |           "objectId": 691,  
|      |           "portAdminStatus": "up",  
|      |           "portAlias": "to 10.44 lan",  
|      |           "portDescr": "GigabitEthernet0/0",  
|      |           "portDuplex": "Full Duplex",  
|      |           "portInSpeed": 100000000,  
|      |           "portMac": "00:07:0e:34:f4:00",  
|      |       }  
|      |   ]  
|      |} |
"portOperationalStatus": "up",
"portOutSpeed": 100000000,
"portShortDescr": "[ Gi0/0 ]",
"portSpare": "No",
"portVipStatus": "Router",
"portVirtualIndicator": "Physical",
"statusEventsEnabled": 1,
"statusFasterPolling": 0,
"timeOfLastChange": 1458210600,
"topoNodeState": 0,
"typeDisplayName": "Router Device",
"typeName": "portEx",
"utilFasterPolling": 0,
"vlans": ""
}
}, {
...removed for brevity...
}, {
"portAttributes": {
"compId": {
"ids": [
1,
2,
12,
0
],
"type": 1,
"invalid": false,
"port": true,
"networkPath": false,
"device": false,
"root": false,
"dsObject": false,
"view": false
},
"connectedHostIps": "",
}
"connectedHostMacs": "",
"connectedHostNames": "",
"connectedHosts": "",
"displayName": " [ Se0/1/0 ] Serial0/1/0-mpls layer",
...removed for brevity...
"typeDisplayName": "Router Device",
"typeName": "portEx",
"utilFasterPolling": 0,
"vlans": ""
}]
"count": 8
}
<ifIndex>1</ifIndex>
<ifType>Ethernet</ifType>
<ipAddresses>10.44.1.59</ipAddresses>
<objectId>691</objectId>
<portAdminStatus>up</portAdminStatus>
<portAlias>to 10.44 lan</portAlias>
<portDescr>GigabitEthernet0/0</portDescr>
<portDuplex>Full Duplex</portDuplex>
<portInSpeed>100000000</portInSpeed>
<portMac>00:07:0e:34:f4:00</portMac>
<portOperationalStatus>up</portOperationalStatus>
<portOutSpeed>100000000</portOutSpeed>
<portShortDescr>[ Gi0/0 ]</portShortDescr>
<portSpare>No</portSpare>
<portVipStatus>Router</portVipStatus>
<portVirtualIndicator>Physical</portVirtualIndicator>
<statusEventsEnabled>1</statusEventsEnabled>
<statusFasterPolling>0</statusFasterPolling>
<timeOfLastChange>1458210600</timeOfLastChange>
<topoNodeState>0</topoNodeState>
?typeDisplayName>Router Device</typeDisplayName>
<typeName>portEx</typeName>
<utilFasterPolling>0</utilFasterPolling>
<vlans></vlans>
</portAttributes>
</item>
...removed for brevity...

<item xsi:type="devicePortInfo" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<portAttributes>
<compId>
<ids>
<ids>1</ids>
<ids>2</ids>
<ids>12</ids>
<ids>0</ids>
</ids>
</compId>
</compId>
<connectedHostIps/>
<connectedHostMacs/>
<connectedHostNames/>
<connectedHosts/>
<displayName>[ Se0/1/0 ] Serial0/1/0-mpls layer</displayName>
...removed for brevity...
<typeDisplayName>Router Device</typeDisplayName>
<typeName>portEx</typeName>
<utilFasterPolling>0</utilFasterPolling>
<vlans/>
</portAttributes>
</item>
</items>
Object Associations

Lists an object's associations.

URL: objects/{swID}/associations?showEmpty=BOOL

- Methods summary
- GET Method details
  - Response
  - Examples

Methods summary

- Lists associations of the object.

GET Method details

Lists associations of the object.

This method accepts a “showEmpty” parameter, which controls whether empty associations should be listed. This method accepts one of two boolean parameters:

- one that equals "yes", "true", "y", "t" or "1". This is treated as 'true'.
- one that equals any other value, or the absence of a value. This is treated as 'false'.

Response

The list of associations.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Association name</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>INP</th>
<th>UT</th>
</tr>
</thead>
<tbody>
<tr>
<td>curl</td>
<td>-u admin:admin <a href="http://localhost/api/objects/655/associations?showEmpty=yes&amp;media=json">http://localhost/api/objects/655/associations?showEmpty=yes&amp;media=json</a></td>
</tr>
</tbody>
</table>
```json
{
    "items": [
        "AdaptorUnits",
        "Annotation",
        "ChassisList",
        "ComputeBlades",
        "DeviceMIBPolls",
        "FabricExtenders",
        "FanModules",
        "Fans",
        "HostEthernets",
        "IPSLABaseCreators",
        "IPSLABasePollers",
        "IpAddresses",
        "LocalDisks",
        ...removed for brevity...
        "uDComponents19",
        "uDComponents20"
    ],
    "count": 60
}
```

**Input**

```
curl -u admin:admin http://localhost/api/objects/655/associations?showEmpty=yes&media=xml
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<items count="60">
  ...removed for brevity...
</items>
Association details

Display details of an association.

URL: objects/{swID}/associations/{associationName}

Methods summary

- GET Method details
  - Response
  - Examples

GET Method details

Displays details of an association. Association names are not case sensitive.

Response

The list of items, with entries according to the following format:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>StormWorks ID of the associated object.</td>
</tr>
<tr>
<td>typeName</td>
<td>type of the associated object.</td>
</tr>
<tr>
<td>displayName</td>
<td>display name.</td>
</tr>
<tr>
<td>displayType</td>
<td>display type.</td>
</tr>
<tr>
<td>hasServiceStatus</td>
<td>if object has service status.</td>
</tr>
</tbody>
</table>

Examples

| INPUT       | curl -u admin:admin http://localhost/api/objects/655/associations/fans?media=json |
### OUTPUT

<table>
<thead>
<tr>
<th>items</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

```json
{
  "items": [
    {
      "objectId": 656,
      "typeName": "Fan",
      "displayName": "Fan 1",
      "displayType": "",
      "hasServiceStatus": true
    },
    {
      "objectId": 657,
      "typeName": "Fan",
      "displayName": "Fan 2",
      "displayType": "",
      "hasServiceStatus": true
    },
    {
      "objectId": 658,
      "typeName": "Fan",
      "displayName": "Fan 3",
      "displayType": "",
      "hasServiceStatus": true
    }
  ],
  "count": 3
}
```

### INPUT

```
curl -u admin:admin http://localhost/api/objects/655/associations/fans?media=xml
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<items count="3">

  <item xsi:type="associationInfo" hasServiceStatus="true"
       displayType="" displayName="Fan 1" typeName="Fan" objectId="656" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>

  <item xsi:type="associationInfo" hasServiceStatus="true"
       displayType="" displayName="Fan 2" typeName="Fan" objectId="657" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>

  <item xsi:type="associationInfo" hasServiceStatus="true"
       displayType="" displayName="Fan 3" typeName="Fan" objectId="658" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>

</items>
Configuration Management

List the configuration management settings of an object, and modify those settings.

URL: objects/{swID}/configManagement

- Methods summary
  - GET Method details
    - Response
    - Examples
  - PUT Method details
    - Request
    - Response
    - Examples

Methods summary

- GET Method - lists configuration management settings of the object.
- PUT Method - modifies configuration management attributes of the object.

GET Method details

Lists configuration management settings of the object.

Response

The list of configuration management settings of the object:

- NumberOfConfigsToArchive
- configMonitorRetrievalScript
- configMonitorPolicyRules
- ConfigRetrievalEnabled
- configMonitorTransferMethod
- SNMPChangeDetectionEnabled
- configMonitorExcludedDifference
- cliMethod
- cliUsername.

Every entry will follow the description below:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>value of “name” attribute of the device</td>
</tr>
<tr>
<td>displayName</td>
<td>value of “displayName” attribute of the device.</td>
</tr>
<tr>
<td>userEditable</td>
<td>if the attribute can be modified by the user.</td>
</tr>
<tr>
<td>userOverriden</td>
<td>if the attribute is set to polled or a user-defined value.</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>values</td>
<td>list of values for this attribute.</td>
</tr>
</tbody>
</table>

**Examples**

**INPUT**

curl -u admin:admin http://localhost/api/objects/605/ports?
includeVirtual=y&includeUnmanaged=y&media=json

**OUTPUT**

```json
{
    "items" : [ {
        "portAttributes" : {
            "compId" : {
                "ids" : [ 1, 2, 1, 0 ],
                "type" : 1,
                "invalid" : false,
                "root" : false,
                "networkPath": false,
                "device" : false,
                "port" : true,
                "view" : false,
                "dsObject" : false
            },
            "connectedHostIps" : "",
            "connectedHostMacs" : "",
            "connectedHostNames" : "",
            "connectedHosts" : "",
            "displayName" : " [ Vl1 ] Vlan1",
            "ifDescr" : " [ Vl1 ] Vlan1",
            "ifIndex" : "1",
            "ifType" : "Prop. Virtual/Internal",
            "ipAddresses" : "10.44.1.41",
            "objectId" : 723,
            "portAdminStatus" : "up",
            "portAlias" : "",
            "portDescr" : "Vlan1",
            "portDuplex" : "Unknown",
            "portInSpeed" : 0,
            "portMac" : "00:19:06:d2:1e:c0"
        }
    }
}
```
"portOperationalStatus" : "up",
"portOutSpeed" : 0,
"portShortDescr" : "[ Vl1 ]",
"portSpare" : "No",
"portVipStatus" : "",
"portVirtualIndicator" : "Virtual",
"statusEventsEnabled" : 0,
"statusFasterPolling" : 0,
"timeOfLastChange" : null,
"topoNodeState" : 292,
"typeName" : "portEx",
"utilFasterPolling" : 0,
"vlans" : ""
}
},

"portAttributes" : {
  "compId" : {
    "ids" : [ 1, 2, 6, 0 ],
    "type" : 1,
    "invalid" : false,
    "root" : false,
    "networkPath" : false,
    "device" : false,
    "port" : true,
    "view" : false,
    "dsObject" : false
  },
  "connectedHostIps" : "",
  "connectedHostMacs" : "",
  "connectedHostNames" : "",
  "connectedHosts" : "",
  "displayName" : "[ Fa0/4 ] FastEthernet0/4",
  "ifDescr" : "[ Fa0/4 ] FastEthernet0/4"
"portAttributes" : {
    "compId" : {
        "ids" : [ 1, 2, 18, 0 ],
        "type" : 1,
        "invalid" : false,
        "root" : false,
        "networkPath" : false,
        "device" : false,
        "port" : true,
        "view" : false,
        "dsObject" : false
    },
    "displayName" : " [ Fa0/16 ] FastEthernet0/16",
    "objectId" : 18,
    "typeName" : "UnmanagedPort"
}
} 
"count" : 29

**PUT Method details**

Modifies a single configuration management attribute of the device.

**Request**

An attribute definition:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>name of attribute that is to be modified. The only valid values are:</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>- NumberOfConfigsToArchive</td>
</tr>
<tr>
<td></td>
<td>- configMonitorRetrievalScript</td>
</tr>
<tr>
<td></td>
<td>- configMonitorPolicyRules</td>
</tr>
<tr>
<td></td>
<td>- ConfigRetrievalEnabled</td>
</tr>
<tr>
<td></td>
<td>- configMonitorTransferMethod</td>
</tr>
<tr>
<td></td>
<td>- SNMPChangeDetectionEnabled</td>
</tr>
<tr>
<td></td>
<td>- configMonitorExcludedDifference</td>
</tr>
<tr>
<td></td>
<td>- cliMethod</td>
</tr>
<tr>
<td></td>
<td>- cliUsername.</td>
</tr>
<tr>
<td>values</td>
<td>list of values for this attribute</td>
</tr>
</tbody>
</table>

**Response**

An updated attribute:

"OK" if the method succeeded, an error description otherwise.

**Examples**

**INPUT**

```bash

'{
  "name" : "cliMethod",
  "values" : [ "ssh" ]
}'
```

**OUTPUT**

```json
{
  "message" : "OK"
}
```

**INPUT**

```bash

'<objectAttribute>
  <name>cliMethod</name>
  <value>ssh</value>
</objectAttribute>'
```

```xml
<objectAttribute>
  <name>cliMethod</name>
  <value>ssh</value>
</objectAttribute>
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<statusInfo>
    <message>OK</message>
</statusInfo>
Port Management

Set the managed/unmanaged state of a port.

URL: portManagement/{devID}/{portID}?reManage=BOOL&unManage=BOOL

- Methods summary
- PUT Method details
  - Response
  - Examples

Methods summary

- PUT Method - controls the managed/unmanaged state of the port.

PUT Method details

This method can accept two boolean parameters, which can be equal to either of “yes”, “true”, “y”, “t” or “1”.

If both parameters are present, “reManage” takes over the “unManage”.

If none are present, then this method reports an error.

Response

A message, describing the current state of a port.

Examples

<table>
<thead>
<tr>
<th>INPUT</th>
<th>curl -u admin:admin <a href="http://localhost/api/portManagement/2/18?unManage=y&amp;media=json">http://localhost/api/portManagement/2/18?unManage=y&amp;media=json</a> -X PUT -H &quot;content-type: application/json&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPUT</td>
<td>{ &quot;message&quot; : &quot;Successfully marking port as unmanaged&quot; }</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INPUT</th>
<th>curl -u admin:admin <a href="http://localhost/api/portManagement/2/18?unManage=y&amp;media=xml">http://localhost/api/portManagement/2/18?unManage=y&amp;media=xml</a> -X PUT -H &quot;content-type: application/xml&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPUT</td>
<td>{ }</td>
</tr>
</tbody>
</table>
<xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <statusInfo>
    <message>Successfully marking port as unmanaged</message>
  </statusInfo>
Zones

Lists configured zones and creates new zones.

URL: zones

- Methods Summary
  - GET Method detail
    - Response
    - Examples
  - POST Method details
    - Request
    - Response
    - Examples

Methods Summary

- GET Method - lists configured zones.
- POST Method - creates a new zone.

GET Method detail

Lists configured zones.

Response

Response includes a list of zones. Each zone has following attributes.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>zone ID, unique to the server.</td>
</tr>
<tr>
<td>name</td>
<td>zone name.</td>
</tr>
<tr>
<td>serverId</td>
<td>Entuity Server ID on which resource resides.</td>
</tr>
</tbody>
</table>

Examples

| INPU   | curl -u admin:admin http://localhost/api/zones?media=json |
```
{  
    "items": [  
        {  
            "serverId": "ee934fe2-1f4d-4f1e-a0b5-a87b261eb30a",  
            "id": "0",  
            "name": "None"  
        }  
    ],  
    "count": 1  
}
```

**INPUT**
curl -u admin:admin http://localhost/api/zones?media=xml

**OUTPUT**
```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<items count="1">
    <item xsi:type="namedItem" name="None" id="0" serverId="ee934fe2-1f4d-4f1e-a0b5-a87b261eb30a" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
</items>
```

**POST Method details**

Creates a new zone.

**Request**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>zone ID, unique per server.</td>
</tr>
<tr>
<td>name</td>
<td>name of the zone.</td>
</tr>
<tr>
<td>flags</td>
<td>zone flags, should be set to 0.</td>
</tr>
<tr>
<td>description</td>
<td>description of the zone.</td>
</tr>
<tr>
<td>v4interface</td>
<td>IPv4 interface for the zone.</td>
</tr>
<tr>
<td>v6interface</td>
<td>IPv6 interface for the zone.</td>
</tr>
<tr>
<td>domainSuffix</td>
<td>domain suffix for the zone.</td>
</tr>
<tr>
<td>proxy</td>
<td>(this is unused.)</td>
</tr>
<tr>
<td><strong>devicePrefix</strong></td>
<td>specified device prefix that is included with each device name found under this zone, maximum of 5 characters.</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>hostFile</strong></td>
<td>host file.</td>
</tr>
<tr>
<td><strong>dnsServer</strong></td>
<td>list of DNS servers.</td>
</tr>
</tbody>
</table>

**Response**

The list of zones after an update, as GET method would return them.

**Examples**

```
curl -u admin:admin http://localhost/api/zones?media=json -X POST -H "content-type: application/json" -d
  '{
    "name" : "A",
    "flags" : 0,
    "description" : "",
    "v4Interface" : "10.44.2.103",
    "v6Interface" : "",
    "domainSuffix" : "",
    "proxy" : "",
    "devicePrefix" : "",
    "hostFile" : "",
    "dnsServers" : [ "a.b.c.d" ]
  }
```

`<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<zoneParametersInfo>
  <name>A</name>
  <flags>0</flags>
  <description/>
  <v4Interface>10.44.2.103</v4Interface>
  <v6Interface/>
  <domainSuffix/>
  <proxy/>
  <devicePrefix/>
  <hostFile/>
  <dnsServers>
    <server>a.b.c.d</server>
  </dnsServers>
</zoneParametersInfo>`
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<items count="2">
  <item xsi:type="namedItem" name="A" id="1" serverId="ee934fe2-1f4d-4f1e-a0b5-a87b261eb30a" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
  <item xsi:type="namedItem" name="None" id="0" serverId="ee934fe2-1f4d-4f1e-a0b5-a87b261eb30a" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
</items>
Zone details

This resource implements operations acting on a single zone.

URL: zones/{zoneID}

- Methods summary
- GET Method details
  - Response
  - Examples
- PUT Method details
  - Request
  - Response
  - Examples
- DELETE Method details
  - Response
  - Examples

Methods summary

- GET Method - details information about the zone.
- PUT Method - modifies parameters of the zone.
- DELETE Method - deletes the zone.

GET Method details

Returns a detailed information about the zone.

Response

User info structure with the following format:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>zone ID, unique per server</td>
</tr>
<tr>
<td>name</td>
<td>name of the zone</td>
</tr>
<tr>
<td>flags</td>
<td>zone flags.</td>
</tr>
<tr>
<td>description</td>
<td>description of the zone.</td>
</tr>
<tr>
<td>v4interface</td>
<td>IPv4 interface for the zone.</td>
</tr>
<tr>
<td>v6interface</td>
<td>IPv6 interface for the zone.</td>
</tr>
<tr>
<td>domainSuffx</td>
<td>Domain suffix for the zone.</td>
</tr>
</tbody>
</table>
### Examples

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>curl -u admin:admin <a href="http://localhost/api/zones/1?media=json">http://localhost/api/zones/1?media=json</a></td>
<td>`{</td>
</tr>
<tr>
<td></td>
<td>&quot;id&quot;: 1,</td>
</tr>
<tr>
<td></td>
<td>&quot;name&quot;: &quot;A&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;flags&quot;: 0,</td>
</tr>
<tr>
<td></td>
<td>&quot;description&quot;: &quot;&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;v4Interface&quot;: &quot;10.44.2.103&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;v6Interface&quot;: &quot;&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;domainSuffix&quot;: &quot;&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;proxy&quot;: &quot;&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;devicePrefix&quot;: &quot;&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;hostFile&quot;: &quot;&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;dnsServers&quot;: [</td>
</tr>
<tr>
<td></td>
<td>&quot;a.b.c.d&quot;</td>
</tr>
<tr>
<td></td>
<td>]</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

| INPUT                              | curl -u admin:admin http://localhost/api/zones/1?media=xml           |
PUT Method details

Modifies parameters of the zone.

Request

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>zone ID, unique per server</td>
</tr>
<tr>
<td>name</td>
<td>name of the zone</td>
</tr>
<tr>
<td>flags</td>
<td>zone flags</td>
</tr>
<tr>
<td>description</td>
<td>description of the zone</td>
</tr>
<tr>
<td>v4interface</td>
<td>IPv4 interface for this zone</td>
</tr>
<tr>
<td>v6interface</td>
<td>IPv6 interface for this zone</td>
</tr>
<tr>
<td>domainSuffix</td>
<td>domain suffix for this zone</td>
</tr>
<tr>
<td>proxy</td>
<td>// TODO</td>
</tr>
<tr>
<td>devicePrefix</td>
<td>// TODO</td>
</tr>
<tr>
<td>hostFile</td>
<td>host file</td>
</tr>
<tr>
<td>dnsServer</td>
<td>list of DNS servers</td>
</tr>
</tbody>
</table>
**Response**

Detailed information about the zone after changes, as GET method would return it.

**Examples**

| **INPUT** | curl -u admin:admin http://localhost/api/zones/1?media=json -X PUT -H "content-type: application/json" -d \`
|           | '{
|           |     "name": "B",
|           |     "v6Interface": "fe80:0:0:0:5efe:a2c:266",
|           |     "dnsServers": [
|           |         "t.x.y.z"
|           |     ]
|           |}' |
| **OUTPUT** | { "id": 1, "name": "B", "flags": 0, "description": "", "v4Interface": "10.44.2.103", "v6Interface": "fe80:0:0:0:5efe:a2c:266", "domainSuffix": "", "proxy": "", "devicePrefix": "", "hostFile": "", "dnsServers": [ "t.x.y.z" ] } |
### DELETE Method details

Deletes a zone.

**Response**

The current list of zones, as the GET method would return it.

**Examples**

#### INPUT

```bash
curl -u admin:admin http://localhost/api/zones/1?media=json -X DELETE
```
### OUTPUT
```
{
    "message" : "Zone deleted successfully"
}
```

### INPUT
```
curl -u admin:admin http://localhost/api/zones/1?media=xml -X DELETE
```

### OUTPUT
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<statusInfo>
    <message>Zone deleted successfully</message>
</statusInfo>
```
IP SLA Management

View information about, modify and delete an IP SLA.

URL: ipsla/{id}

Methods Summary

- GET Method details
  - Response
  - Examples
- PUT Method details
  - Request
  - Response
  - Examples
- DELETE Method details
  - Response
  - Examples

Methods Summary

- GET Method - displays information about the IP SLA.
- PUT Method - modifies a particular IP SLA creator.
- DELETE Method - deletes a particular IP SLA creator.

Request and Response

The following attributes must be present for every IP SLA type:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>IP SLA object ID</td>
</tr>
<tr>
<td>name</td>
<td>name</td>
</tr>
<tr>
<td>index</td>
<td>association index, used to match IP SLA creators and pollers</td>
</tr>
<tr>
<td>tag</td>
<td>zone flags</td>
</tr>
<tr>
<td>lifetime</td>
<td>creator's lifetime</td>
</tr>
<tr>
<td>frequency</td>
<td>frequency of polling</td>
</tr>
<tr>
<td>timeout</td>
<td>polling timeout</td>
</tr>
<tr>
<td>vrfName</td>
<td>VRF (VPN Routing and Forwarding) instance name</td>
</tr>
<tr>
<td>failureCause</td>
<td>cause of failure</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>changesPending</td>
<td>changes pending</td>
</tr>
<tr>
<td>type</td>
<td>creator type, one of the following:</td>
</tr>
<tr>
<td></td>
<td>• DHCP</td>
</tr>
<tr>
<td></td>
<td>• DNS</td>
</tr>
<tr>
<td></td>
<td>• ECHO</td>
</tr>
<tr>
<td></td>
<td>• ECHO_PATH</td>
</tr>
<tr>
<td></td>
<td>• HTTP</td>
</tr>
<tr>
<td></td>
<td>• HTTP_RAW</td>
</tr>
<tr>
<td></td>
<td>• JITTER</td>
</tr>
<tr>
<td></td>
<td>• VOIP</td>
</tr>
<tr>
<td></td>
<td>• TCP</td>
</tr>
<tr>
<td></td>
<td>• UDP</td>
</tr>
</tbody>
</table>

In addition, each type provides a specific set of attributes, as show:

<table>
<thead>
<tr>
<th>IP SLA type</th>
<th>Required attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DHCP</td>
<td>targetAddress</td>
</tr>
<tr>
<td></td>
<td>sourceAddress</td>
</tr>
<tr>
<td></td>
<td>sourcePort</td>
</tr>
<tr>
<td>DNS</td>
<td>targetAddress</td>
</tr>
<tr>
<td></td>
<td>nameserver</td>
</tr>
<tr>
<td>ECHO and ECHO_PATH</td>
<td>targetAddress</td>
</tr>
<tr>
<td></td>
<td>sourceAddress</td>
</tr>
<tr>
<td></td>
<td>sourcePort</td>
</tr>
<tr>
<td></td>
<td>typeOfService</td>
</tr>
<tr>
<td></td>
<td>packetSize</td>
</tr>
<tr>
<td>HTTP</td>
<td>targetURL</td>
</tr>
<tr>
<td></td>
<td>proxyAddress</td>
</tr>
<tr>
<td></td>
<td>sourceAddress</td>
</tr>
<tr>
<td></td>
<td>sourcePort</td>
</tr>
<tr>
<td></td>
<td>typeOfService</td>
</tr>
<tr>
<td></td>
<td>httpVersion</td>
</tr>
<tr>
<td></td>
<td>useCache</td>
</tr>
<tr>
<td>HTTP_RAW</td>
<td>targetURL</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td></td>
<td>proxyAddress</td>
</tr>
<tr>
<td></td>
<td>sourceAddress</td>
</tr>
<tr>
<td></td>
<td>sourcePort</td>
</tr>
<tr>
<td></td>
<td>typeOfService</td>
</tr>
<tr>
<td></td>
<td>httpVersion</td>
</tr>
<tr>
<td></td>
<td>useCache, adminStr1</td>
</tr>
<tr>
<td></td>
<td>adminStr2</td>
</tr>
<tr>
<td></td>
<td>adminStr3</td>
</tr>
<tr>
<td></td>
<td>adminStr4</td>
</tr>
<tr>
<td></td>
<td>adminStr5</td>
</tr>
<tr>
<td>JITTER</td>
<td>targetAddress</td>
</tr>
<tr>
<td></td>
<td>targetPort</td>
</tr>
<tr>
<td></td>
<td>sourceAddress</td>
</tr>
<tr>
<td></td>
<td>sourcePort</td>
</tr>
<tr>
<td></td>
<td>typeOfService</td>
</tr>
<tr>
<td></td>
<td>interval</td>
</tr>
<tr>
<td></td>
<td>numberOfPackets</td>
</tr>
<tr>
<td></td>
<td>packetSize</td>
</tr>
<tr>
<td>VOIP</td>
<td>targetAddress</td>
</tr>
<tr>
<td></td>
<td>targetPort</td>
</tr>
<tr>
<td></td>
<td>sourceAddress</td>
</tr>
<tr>
<td></td>
<td>sourcePort</td>
</tr>
<tr>
<td></td>
<td>typeOfService</td>
</tr>
<tr>
<td></td>
<td>codecType</td>
</tr>
<tr>
<td></td>
<td>codecInterval, codecPayload</td>
</tr>
<tr>
<td></td>
<td>codecNumOfPkts</td>
</tr>
<tr>
<td></td>
<td>icpIfFactor</td>
</tr>
<tr>
<td>Protocol</td>
<td>targetAddress</td>
</tr>
<tr>
<td>----------</td>
<td>---------------</td>
</tr>
<tr>
<td>TCP</td>
<td></td>
</tr>
<tr>
<td>UDP</td>
<td></td>
</tr>
</tbody>
</table>

**GET Method details**

Returns detailed information about a particular IP SLA creator.

**Response**

Refer to Request and Response section above.

**Examples**

<p>| INPUT     | curl -u admin:admin <a href="http://localhost/api/ipsla/772?media=json">http://localhost/api/ipsla/772?media=json</a> |</p>
<table>
<thead>
<tr>
<th>INPUT</th>
<th><code>curl -u admin:admin http://localhost/api/ipsla/772?media=xml</code></th>
</tr>
</thead>
</table>
| OUTPUT| ```xml version="1.0" encoding="UTF-8" standalone="yes"?>  
|       | <ipslaDetails>                                             |
|       |   <attributes>                                             |
|       |     <id>772</id>                                            |
|       |     <type>DNS</type>                                       |
|       |     <index>2</index>                                       |
|       |     <name>B</name>                                         |
|       |     <tag>a</tag>                                           |
|       |     <lifetime>forever</lifetime>                           |
|       |     <frequency>300</frequency>                             |
|       |     <timeout>1000</timeout>                               |
|       |     <vrfName>a</vrfName>                                   |
|       |     <targetAddress>127.0.0.1</targetAddress>               |
|       |     <nameserver>127.0.0.1</nameserver>                     |
|       |   </attributes>                                            |
|       | </ipslaDetails>                                            |
PUT Method details

Modifies parameters of an IP SLA creator.

NOTE: Parameters "id", "index" and "type" are not supposed to be modified and are therefore ignored.

Request

A set of attributes that define an IP SLA creator, according to the details described in Request and Response section above.

Response

Actual IP SLA creator parameters, as described in Request and Response section above.

Examples

<table>
<thead>
<tr>
<th>INPUT</th>
<th>curl -u admin:admin <a href="http://localhost/api/ipsla/1481?media=json">http://localhost/api/ipsla/1481?media=json</a> -X PUT -H &quot;content-type: application/json&quot; -d</th>
</tr>
</thead>
</table>
|          | `{>
|          |   "frequency" : 2000,>
|          |   "lifetime" : 0,>
|          |   "name" : "Y",>
|          |   "sourcePort" : 2666,>
|          |   "tag" : "x",>
|          |   "targetAddress" : "127.0.0.2",>
|          |   "timeout" : 1000,>
|          |   "vrfName" : "x">
|          | }` |
  <name>Y</name>
  <tag>x</tag>
  <lifetime>0</lifetime>
  <frequency>2000</frequency>
  <timeout>1000</timeout>
  <vrfName>x</vrfName>
  <targetAddress>127.0.0.2</targetAddress>
  <sourcePort>2666</sourcePort>
</ipslaCreator>"
DELETE Method details

The list of users returned is restricted for non-administrators: only the user object corresponding to the current user is returned.

Response

“OK” message if the IP SLA was properly deleted, an error message otherwise.

Examples

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>curl -u admin:admin <a href="http://localhost/api/ipsla/772?media=json">http://localhost/api/ipsla/772?media=json</a> -X DELETE</td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>&quot;message&quot; : &quot;OK&quot;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

INPUT                  | OUTPUT                  |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>curl -u admin:admin <a href="http://localhost/api/ipsla/772?media=xml">http://localhost/api/ipsla/772?media=xml</a> -X DELETE</td>
<td></td>
</tr>
<tr>
<td>OUTPUT</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td></td>
</tr>
</tbody>
</table>
| <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <statusInfo>
    <message>OK</message>
  </statusInfo> |


IP SLA Creators

Resource for managing IP SLA creators configured for a given device.

URL: objects/{swId}/ipslaCreators

Methods Summary

- GET Method details
  - Response
  - Examples
- POST Method details
  - Request
  - Response
  - Examples

GET Method details

Returns detailed information about IP SLA creators defined for the device.

Response

A list of IP SLA creators, with every entry conforming to the Request and Response description section of the IP SLA Management page.

Examples

| INPUT | curl --u admin:admin http://localhost/api/objects/600/ipslaCreators?media=json |
{
    "items": [
        {
            "attributes": {
                "failureCause": "",
                "frequency": "300",
                "id": "772",
                "index": "2",
                "lifetime": "forever",
                "name": "B",
                "nameserver": "127.0.0.1",
                "tag": "a",
                "targetAddress": "127.0.0.1",
                "timeout": "1000",
                "type": "DNS",
                "vrfName": "a"
            }
        }
    ],
    "count": 1
}
POST Method details

Creates an IP SLA creator for a particular device.

Request

A set of attributes that define an IP SLA creator, according to the details described in the Request and Response description section of the IP SLA Management page.

Response

An updated list of IP SLA creators, with every entry conforming to the Request and Response description section of the IP SLA Management page.

Examples
    "frequency" : 500,
    "index" : 0,
    "lifetime" : 0,
    "name" : "X",
    "sourcePort" : "2555",
    "tag" : "x",
    "targetAddress" : "127.0.0.1",
    "timeout" : 1000,
    "type" : "DHCP",
    "vrfName" : "x"
}'
OUTPUT
{
  "items" : [
    {
      "attributes" : {
        "frequency" : "300",
        "id" : "772",
        "index" : "2",
        "lifetime" : "forever",
        "name" : "B",
        "nameserver" : "127.0.0.1",
        "tag" : "a",
        "targetAddress" : "127.0.0.1",
        "timeout" : "1000",
        "type" : "DNS",
        "vrfName" : "a"
      }
    },
    {
      "attributes" : {
        "frequency" : "500",
        "id" : "1481",
        "index" : "0",
        "lifetime" : "0",
        "name" : "X",
        "sourceAddress" : "null",
        "sourcePort" : "2555",
        "tag" : "x",
        "targetAddress" : "127.0.0.1",
        "timeout" : "1000",
        "type" : "DHCP",
        "vrfName" : "x"
      }
    }
  ],
  "count" : 2
}
<table>
<thead>
<tr>
<th>INPUT</th>
</tr>
</thead>
</table>
| `<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
|     <ipslaCreator>
|         <type>DHCP</type>
|         <index>0</index>
|         <name>X</name>
|         <tag>x</tag>
|         <lifetime>0</lifetime>
|         <frequency>500</frequency>
|         <timeout>1000</timeout>
|         <vrfName>x</vrfName>
|         <targetAddress>127.0.0.1</targetAddress>
|         <sourcePort>2555</sourcePort>
|     </ipslaCreator>` |
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<items count="2">
  <item xsi:type="ipslaAttributesMap" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <attributes>
      <id>772</id>
      <type>DNS</type>
      <index>2</index>
      <name>B</name>
      <tag>a</tag>
      <lifetime>forever</lifetime>
      <frequency>300</frequency>
      <timeout>1000</timeout>
      <vrfName>a</vrfName>
      <targetAddress>127.0.0.1</targetAddress>
      <nameserver>127.0.0.1</nameserver>
    </attributes>
  </item>
  <item xsi:type="ipslaAttributesMap" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <attributes>
      <id>1481</id>
      <type>DHCP</type>
      <index>3</index>
      <name>X</name>
      <tag>x</tag>
      <lifetime>0</lifetime>
      <frequency>500</frequency>
      <timeout>1000</timeout>
      <vrfName>x</vrfName>
      <targetAddress>127.0.0.1</targetAddress>
      <sourceAddress>null</sourceAddress>
      <sourcePort>2555</sourcePort>
    </attributes>
  </item>
</items>
IP SLA Pollers

Displays information about IP SLA pollers present on a given device.

URL: objects/{swId}/ipslaPollers

- Methods Summary
- GET Method details
  - Response
  - Examples

Methods Summary

- GET Method - displays detailed information about the IP SLA creators defined for the device.

GET Method details

Returns the detailed information about the IP SLA pollers defined for the device.

Response

A list of entries, each containing a complete attribute set of a particular IP SLA poller.

Examples

| INPUT | curl -u admin:admin http://localhost/api/objects/600/ipslaPollers?media=json |
<items count="4">
    <item xsi:type="ipslaAttributesMap" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <attributes>
            <rttMonEchoAdminTOS>160</rttMonEchoAdminTOS>
            <rttMonEchoAdminTargetAddress>10.2.90.90</rttMonEchoAdminTargetAddress>
            <rttMonEchoAdminString4/>
            <rttMonEchoAdminString3/>
            <rttMonEchoAdminString2/>
            <rttMonEchoAdminVrfName/>
            <rttMonAdminCodecNumPackets/>
            <rttMonEchoAdminString1/>
            <type>IPSLABasePoller</type>
            ...removed for brevity...
            <rttMonEchoAdminHTTPVersion/>
            <rttMonScheduleAdminRttLife>forever</rttMonScheduleAdminRttLife>
            <rttMonEchoAdminProxy/>
            <rttMonCtrlAdminOwner>richard</rttMonCtrlAdminOwner>
            <rttMonAdminICPIFAAdvFactor/>
        </attributes>
    </item>
    <item xsi:type="ipslaAttributesMap" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        ...removed for brevity...
    </item>
    <item xsi:type="ipslaAttributesMap" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        ...removed for brevity...
    </item>
    <item xsi:type="ipslaAttributesMap" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        ...removed for brevity...
    </item>
</items>
Data Access Interface

- General description
- The format of the export template
  - Example template
  - Export template syntax

General description

Data Access is meant as a powerful, highly configurable interface for accessing ENA's internal data (like attribute values, stream data etc.)

The interface achieves this by processing the XML “templates” that describe sets of data to be exported. As these templates may be user-defined, this gives the customers an unprecedented level of control over the data itself, as well as their layout and format.

Currently, due to an extended set of possibilities in comparison to JSON, only XML template format is supported.

The interface consists of two parts (each described in detail later on): the “template management” which can be used to retrieve and modify templates residing on a server, and the “data access” part, which produces the output based on the early defined template.

The format of the export template

Example template

The following example defines a template that would traverse the list of views (supplied by the user as a part of data output) and, based on the value of “selector” element, would traverse every device. For each device found, the element DEVICE_OBJECT will be created and populated with these element (in the order corresponding to their order in the template):

1. The element name, with a value of the name attribute of a device.
2. The attribute id, with a value of executing the simple:id statement on the current device.
3. The stream data v_unifiedDeviceCPU appearing as a CPU element, with all the underlying attributes.
4. The list of PORT_OBJECT elements, corresponding to the result of executing the simple:ref.ports statement on the current device. Each of these element would consist of:
   1. The attribute descr, with the value of simple:ifDescr statement for the current port.
   2. The element status, consisting of a list of samples, each containing two attributes: adminStatus (presented as an adm element) and operStatus, presented as an oper attribute.
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dataAccessTemplate name="Example template">
  <description>This is a test description</description>
  <object type="DeviceEx" tag="DEVICE_OBJECT">
    <selector>simple;ref.devices</selector>
    <filter/>
    <attribute asElement="true" expression="name"/>
    <attribute expression="simple;id" tag="id"/>
    <stream allAttributes="true" name="v_unifiedDeviceCPU" tag="CPU">
      <object type="PortEx" tag="PORT_OBJECT">
        <selector>simple;ref.ports</selector>
        <filter/>
        <attribute asElement="true" expression="simple;this.ifDescr" tag="Descr"/>
        <stream name="portStatus" tag="status">
          <attribute asElement="true" expression="adminStatus" tag="adm"/>
          <attribute expression="operStatus" tag="oper"/>
        </stream>
      </object>
    </stream>
  </object>
</dataAccessTemplate>

Export template syntax

The following table describes all the XML nodes that can be a part of a Data Access Template. Unless stated otherwise, the order of elements is not important.

In the description below, some nodes have one of three symbols: ? (question mark), + (plus sign) or * (an asterisk) appended (in parenthesis) to their names.

These symbols are not part of the name itself, but serve to define how many times a node of this type may appear under the same parent:

- ? – the node is optional and unique (i.e. may appear one or zero times)
- * – the node is optional and not unique (i.e. may appear any number of times)
- + – the node is mandatory and not unique (i.e. must appear one or more times)

Nodes without any such symbol are implied to be mandatory and unique.

<table>
<thead>
<tr>
<th>Node Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| **dataAccesssTemplate** | XML element node that is the root node of every single Data Access Template.  
The list of attributes: **name**, **type(?)**  
The list of child elements: **description(?)**, **object** |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>name</strong></td>
<td>XML attribute node defining the name of the template. One server can only store templates with different names. Renaming of a template is not supported.</td>
</tr>
<tr>
<td><strong>description</strong></td>
<td>XML element node that serves as a placeholder for user-defined comment. It can only have one text node, with its body being the comment string.</td>
</tr>
</tbody>
</table>
| **object**              | XML element node that defines a group of objects.  
The list of attributes: **type**, **tag(?)**, **allAttributes(?)**, **asElement(?)**, **asAttribute(?)**, **asText(?)**, **allowMultibyte**  
Attributes **asElement**, **as Attribute** and **asText** are mutually exclusive.  
The list of child elements: **selector**, **filter**, **object(*)**, **stream(*)** and **attribute(*)**  
Children of type **object**, **stream** and **attribute** can be interleaved. During the processing step, these nodes are expanded in order of their definition and then replaced with data according to their types. |
| **attribute**           | XML element node that defines a StormWorks attribute to be exported.  
The list of attributes: **expression**, **tag(?)**, **asElement(?)**, **asAttribute(?)**, **asText(?)**, **allowMultibyte**  
Attributes **asElement**, **as Attribute** and **asText** are mutually exclusive.  
No child elements are permitted (attempt to do so, results in a processing failure). |
| **stream**              | XML element node that defines a StormWorks stream to be exported.  
The list of attributes: **name**, **tag(?)**, **allAttributes(?)**, **asElement(?)**, **asAttribute(?)**, **asText(?)**  
Attributes **asElement**, **as Attribute** and **asText** are mutually exclusive.  
The list of attributes: **attribute(?)** |
<table>
<thead>
<tr>
<th>asElement</th>
<th>asAttribute</th>
<th>asText</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>XML attribute</em> node that controls how data should be presented in a resulting XML file. Only one of these attributes can be set to “true” for any given node, and if none of them are set, they are inherited from parent. If still none of these attributes were set, the node is treated as if asText was set to “true”.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example: The node with key equal to “SomeTag” and value equal to “SomeData” will be presented as:</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;SomeTag&gt;</code>SomeData<code>&lt;/SomeTag&gt;</code>, if asElement is “true”</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SomeTag=”SomeData”, if asAttribute is “true”</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SomeData, if asText is “true”</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tag</th>
<th><em>XML attribute</em> defining the name of the type, under which all data related to this object will be anchored.</th>
</tr>
</thead>
<tbody>
<tr>
<td>This attribute may occur in object, stream and attribute nodes and is optional in all of them.</td>
<td></td>
</tr>
<tr>
<td>If omitted in an object, it will be defaulted to the type attribute of this node. For stream node, it will be defaulted to the name attribute of a stream. For attribute node, it will be defaulted to its expression attribute, but only if the expression does not begin with “simple;” string (in which case the whole template is invalid and its processing will fail).</td>
<td></td>
</tr>
<tr>
<td>Example: The following template fragment:</td>
<td></td>
</tr>
<tr>
<td><code>&lt;object asElement=&quot;true&quot; type=&quot;DeviceEx&quot; tag=&quot;DEVICE&quot;&gt;</code></td>
<td></td>
</tr>
<tr>
<td>will get mapped to:</td>
<td></td>
</tr>
<tr>
<td><code>&lt;DEVICE&gt;data depending on the children of object&lt;/DEVICE&gt;</code></td>
<td></td>
</tr>
<tr>
<td>If the template fragment were changed to:</td>
<td></td>
</tr>
<tr>
<td><code>&lt;object asElement=&quot;true&quot; type=&quot;DeviceEx&quot; tag=&quot;DeviceParameters&quot; &gt;</code></td>
<td></td>
</tr>
<tr>
<td>then the corresponding export fragments would have been changed to:</td>
<td></td>
</tr>
<tr>
<td><code>&lt;DeviceParameters&gt;…&lt;/DeviceParameters&gt;</code></td>
<td></td>
</tr>
</tbody>
</table>

| type | *XML attribute* node that defines the StormWorks type of object. The type of objects defines what attributes and streams may be access and exported. |

| selector | *XML element* node that is the StormWorks statement. This statement is evaluated to produce a list of children, each then undergoing subsequent processing according to their definition, etc. |
| Filter       | XML element node that is the StormWorks statement. This statement should evaluate to boolean value and can be used to restrict the list of children produced by a selector statement. |
Data Access templates management – listing and creating

Lists Data Access templates used to access the internal DsKernel data, and creates new Data Access templates.

URL: dataAccessTemplates

- Methods Summary
- GET Method Details
  - Response
  - Example
- POST Method Details
  - Request
  - Response
  - Example

Methods Summary

- GET Method - lists all Data Access templates existing on the server.
- POST Method - creates a new Data Access template.

GET Method Details

Lists all Data Access templates existing on the server.

Response

Lists information about each of templates defined on a server. Each entry has the following attributes:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name of the template.</td>
</tr>
<tr>
<td>description</td>
<td>description of the template.</td>
</tr>
<tr>
<td>type</td>
<td>type of the template (either “USER” or “SYSTEM”).</td>
</tr>
</tbody>
</table>

Example

```
```
POST Method Details

Creates a new Data Access Template.

Request

An XML definition of the template, defined according to the Export template syntax section of the Data Access Interface.

Response

A new template. If unsuccessful, a description of the error that occurred.

Example

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<items count="2">
  <item xsi:type="dataAccessTemplateSummary" description="" type="USER" name="B" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
</items>
```

```
  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <dataAccessTemplate type="USER" name="C">
    <description>Some description</description>
    <object type="DeviceEx" tag="device">
      <selector>simple;ref.devices</selector>
      <filter></filter>
      <attribute asElement="false" tag="name1" expression="name"/>
    </object>
    <object type="PortEx" tag="port">
      <selector>simple;ref.ports</selector>
      <filter></filter>
      <attribute asElement="false" tag="descr" expression="simple;this.ifdescr"/>
    </object>
  </dataAccessTemplate>
'```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dataAccessTemplate type="USER" name="C">
<description>Some description</description>
<object type="DeviceEx" tag="device">
  <attribute expression="name" tag="name1"/>
</object>
<object type="PortEx" tag="port">
  <attribute expression="simple;this.ifDescr" tag="descr"/>
  <selector>simple;ref.ports</selector>
  <filter></filter>
</object>
<selector>simple;ref.devices</selector>
<filter></filter>
</object>
</dataAccessTemplate>
Data Access templates management – operations on a single template

Manages a single Data Access template entity.

URL: dataAccessTemplates/{templateName}

- Methods Summary
- GET Method Details
  - Response
  - Example
- PUT Method Details
  - Request
  - Response
  - Example
- DELETE Method Details
  - Response
  - Example

Methods Summary

- GET Method - lists all Data Access templates existing on the server.
- PUT Method - modifies an existing Data Access template.
- DELETE Method - removes an existing Data Access template from a server.

GET Method Details

Response

Detailed information about the Data Access template, according to the Export template syntax section of the Data Access Interface page.

Example

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dataAccessTemplate type="USER" name="C">

<description>Some description</description>

<object type="DeviceEx" tag="device">
    <attribute expression="name" tag="name1"/>

<object type="PortEx" tag="port">
    <attribute expression="simple;this.ifDescr" tag="descr"/>
    <selector>simple;ref.ports</selector>
    <filter></filter>
</object>

<selector>simple;ref.devices</selector>
<filter></filter>
</object>
</dataAccessTemplate>

**PUT Method Details**

Modifies the existing Data Access Template.

**Request**

An XML definition of the template, defined according to a paragraph 2.38.2.2

**Response**

The modified Data Access Template. If unsuccessful, a description of the error.

**Example**
'<?xml version="1.0" encoding="UTF-8" standalone="yes"?><dataAccessTemplate type="USER" name="C">
  <description>Some description</description>
  <object type="DeviceEx" tag="device">
    <selector>simple;ref.devices</selector>
    <filter></filter>
    <attribute asElement="false" expression="name"/>
    <attribute asElement="false" expression="id"/>
  </object>
  <object type="PortEx" tag="port">
    <selector>simple;ref.ports</selector>
    <filter></filter>
    <attribute asElement="false" tag="descr" expression="simple;this.ifDescr"/>
  </object>
</dataAccessTemplate>'

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dataAccessTemplate type="USER" name="C">
  <description>Some description</description>
  <object type="DeviceEx" tag="device">
    <attribute expression="name" tag=""/>
    <attribute expression="id" tag=""/>
  </object>
  <object type="PortEx" tag="port">
    <attribute expression="simple;this.ifDescr" tag="descr"/>
    <selector>simple;ref.ports</selector>
    <filter></filter>
  </object>
  <selector>simple;ref.devices</selector>
  <filter></filter>
</dataAccessTemplate>
Removes the existing Data Access Template.

**Response**

The template after modifications on success, the error description otherwise.

**Example**

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPUT</td>
<td>&lt;?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot; standalone=&quot;yes&quot;?&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;statusInfo&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;message&gt;OK&lt;/message&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/statusInfo&gt;</td>
</tr>
</tbody>
</table>
Data Access – export triggering

Export user-defined Data Access templates.

URL: dataAccess/{templateName}

• Methods Summary
  • GET Method details
    • Request
    • Response
    • Examples

Methods Summary

• GET Method - exports user-defined Data Access templates.

GET Method details

Exports user-defined Data Access templates.

An export, generated according to the user-defined Data Access template.

Request

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>templateServer</td>
<td>ID of the server (see GET Method detail section of Servers page) from which to retrieve the template.</td>
</tr>
<tr>
<td>view</td>
<td>name of View of which objects will be processed. This argument may appear multiple times in which case any objects belonging to at least one of the Views will be processed.</td>
</tr>
<tr>
<td>startTime</td>
<td>start and end timestamp for exporting a stream data. Negative values are not allowed and the default value for these parameters is 0. As a special case, if both startTime and endTime are 0, then all the streams being exported will contain only the last sample.</td>
</tr>
<tr>
<td>endTime</td>
<td>maxResults</td>
</tr>
</tbody>
</table>
next element that the export should start from. It is optional, in which case it defaults to 0:0.

The result of each export will have the “position” element defined. This element consists of pairs of numbers that are separated from other pairs by commas, and the elements of a pair are separated by a colon, e.g:

1:2,3:4,5:6

After successful call that returns maximum number of elements (defined by maxResults parameter), the position will point to the next element to continue from.

To achieve continuation, the position returned from the last call should be passed along as an argument to the next call – as long as the returned value is an “END” string, which signals that no more data are available.

Response

An export, generated according to the user-defined Data Access template. If unsuccessful, description of the error will be returned.

Examples

Below are three examples showing the “continuation” mechanism, together with some explanation. Please note that normally the XML output generated by Data Access mechanism is not formatted – it has only been presented as formatted here for the purpose of readability.

The first example omits the position argument, which exports data from the very beginning:

**INPUT**
curl -u admin:admin http://localhost/api/dataAccess/C?maxResults=5&view=All%20Objects&media=xml -X GET -H "content-type: application/xml"

**OUTPUT**

```xml
<C serverName="ENTLONDEV08" serverId="dae26869-46e3-46df-91a0-28ce61b28d05" templateServerId="dae26869-46e3-46df-91a0-28ce61b28d05" startTime="0" endTime="0" elementsProcessed="5" position="565:2,575:0" processingTime="0.001">
    <device id="565" name="apiDevice">
        <port id="572" descr=" [ Vl1 ] Vlan1" />
        <port id="573" descr=" [ Fa0/1 ] FastEthernet0/1" />
        <port id="574" descr=" [ Fa0/2 ] FastEthernet0/2" />
        <port id="575" descr=" [ Fa0/3 ] FastEthernet0/3" />
    </device>
</C>
```
The second one is an example of the call just after the first one, which picks up from where the previous call left off:

| OUTPUT | <C serverName="ENTLONDDEV08" serverId="dae26869-46e3-46df-91a0-28ce61b28d05" templateServerId="dae26869-46e3-46df-91a0-28ce61b28d05" startTime="0" endTime="0" elementsProcessed="5" position="565:2,578:0" processingTime="0"> |
| | <device id="565" name="apiDevice"> |
| | <port id="576" descr=" [ Fa0/4 ] FastEthernet0/4" /> |
| | <port id="577" descr=" [ Fa0/5 ] FastEthernet0/5" /> |
| | <port id="578" descr=" [ Fa0/6 ] FastEthernet0/6" /> |
| | </device> |
| | </C> |

The third example is the last call, where the number of available elements were less than \texttt{maxResults} parameter (the \texttt{elementsProcessed} is less than 5, and the returned \texttt{position} is “END”):

| INPUT | curl -u admin:admin http://localhost/api/dataAccess/C?position=646:2,805:0&maxResults=5&view=All%20Objects&media=xml -X GET -H "content-type: application/xml" |
| OUTPUT | <C serverName="ENTLONDDEV08" serverId="dae26869-46e3-46df-91a0-28ce61b28d05" templateServerId="dae26869-46e3-46df-91a0-28ce61b28d05" startTime="0" endTime="0" elementsProcessed="4" position="END" processingTime="0"> |
| | <device id="646" name="e2821"> |
| | <port id="806" descr=" [ Gi0/1 ] GigabitEthernet0/1-mpls layer" /> |
| | <port id="807" descr=" [ Se0/1/0 ] Serial0/1/0-mpls layer" /> |
| | </device> |
| | </C> |
Flow Data - listing devices with stored flow information

Lists network devices for which flow is being collected and stored, and flow history for a specified device. Network devices must be configured to send flow information to ENA, meaning that ENA has no control over the flow information it receives. Further, flow information is not stored by default - this must be enabled for each device where desired.

URL: http(s):://{server hostname}/api/flowDevices

- Method summary
- GET Method detail
  - Request
  - Response
  - Examples

Method summary

- GET Method details - lists devices for which flow information is being collected.

GET Method detail

Lists the devices for which flow information is currently being collected.

Request

Call the resource URL with the GET method.

Response

The API will respond with a list of devices for which ENA has collected flow data. Within the list, each device is represented as an associative array, containing the following information about the devices:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flowServerld</td>
<td>ID of the flow server that is collecting flow for this device. Typically, each Entuity server manages a single flow server. In this case you would expect <code>flowServerld</code> to match <code>serverld</code>. However, an Entuity server can manage multiple flow servers and this attribute would allow you to determine which one.</td>
</tr>
<tr>
<td>zoneName</td>
<td>name of the zone that the device has been placed in. If zoning is not in use, the value of <code>zoneName</code> will be null.</td>
</tr>
<tr>
<td>deviceId</td>
<td>ID of the device as assigned when the device is taken under management by ENA. This ID is unique.</td>
</tr>
</tbody>
</table>
to a server but must be used in conjunction with flow ServerId to uniquely identify a device in a multi-server configuration.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IpAddress</td>
<td>string representation of the management IP address of the device. IPv4 and IPv6 are supported.</td>
</tr>
<tr>
<td>deviceName</td>
<td>string representation of the device's display name.</td>
</tr>
<tr>
<td>swObId</td>
<td>StormWorks Object ID. All objects, including devices, are assigned a stormworks object ID. As with deviceId, swObId must be used in conjunction with flowServerId to uniquely identify an object in a multi-server configuration.</td>
</tr>
<tr>
<td>vxlan</td>
<td>indicates that the flow record originates from a VMWare virtualized device. Either True or False.</td>
</tr>
<tr>
<td>ifIndexes</td>
<td>list of port ifIndexes for which flow is being collected. Note that ifIndeses are only unique within a device.</td>
</tr>
</tbody>
</table>

Examples

```
curl -u admin:admin https://localhost/api/flowDevices?media=json -X GET
```

```
{
  "devices": [
    {
      "flowServerId": "5ba829fa-86d9-4246-b4ce-24cc896180c0",
      "zoneName": null,
      "deviceId": 17,
      "ipAddress": "10.18.0.2",
      "deviceName": "ca01",
      "swObjId": 866,
      "vxlan": false,
      "ifIndexes": [ 1, 2, 3, 4, 5 ]
    },
    {
      "flowServerId": "5ba829fa-86d9-4246-b4ce-24cc896180c0",
      "zoneName": null,
      "deviceId": 1,
      "ipAddress": "10.1.2.2",
      "deviceName": "hq01",
      "swObjId": 730,
      "vxlan": false,
      "ifIndexes": [ 1, 2, 3, 4, 5 ]
    }
  ]
}
```
Flow Data - listing applications supporting flow

Lists the applications currently configured to support flow. ENA can characterize flow information by application, doing so by looking at the destination port at the transport layer (TCP/UDP).

URL: http(s)://{server hostname}/api/api/flowApplications

- Method summary
- GET Method detail
  - Request
  - Response
  - Examples

Method summary

- GET Method details - lists applications configured to support flow.

GET Method detail

List applications configured to support flow.

Request

Call the resource URL with the GET method.

Response

An associative array containing the single member called "applications". The value of applications is a list of application names.

Examples

This list contains 5665 applications by default, so the list has been truncated in this example.

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
</table>
| curl -u admin:admin https://testld.entuity.com/api/options /flowApplications?media=json -X GET | {
|                                            | {      |
|                                            |   "applications" : [ {   |
|                                            |     "name" : "1ci-smcs"   |
|                                            |   },   |
|                                            |     "name" : "3Com-nsd"   |
|                                            |   },   |
|                                            | }      |
|                                            |       |
"name" : "3PC"
}, {
   "name" : "3com-amp3"
}, {
...