ArubaOS 8.3.0.x

a Hewlett Packard
Enterprise company

API Guide

Open Source Code

This product includes code licensed under the GNU General Public License, the GNU Lesser General Public License, and/or certain other open source licenses. A complete machine-readable copy of the source code corresponding to such code is available upon request. This offer is valid to anyone in receipt of this information and shall expire three years following the date of the final distribution of this product version by Hewlett Packard Enterprise Company. To obtain such source code, send a check or money order in the amount of US $10.00 to:

Hewlett Packard Enterprise Company
Attn: General Counsel
3000 Hanover Street
Palo Alto, CA 94304
USA

Revision History

The following table lists the revisions of this document.

Table 1: *Revision History*

Revision	Change Description
Revision 01	Initial release.

This document describes the strategy and procedure for migrating existing Aruba controller deployments to Mobility Master managed deployment and the support for context APIs in Mobility Master.

This chapter includes the following sections:

Related Documents

The following guides are part of the documentation for Mobility Master:

- *ArubaOS Release Notes*
- *ArubaOS Getting Started Guide*
- *ArubaOS User Guide*
- *ArubaOS CLI Reference Guide*
- *ArubaOS Migration Guide*
- *Aruba Mobility Master Licensing Guide*
- *Aruba Virtual Appliance Installation Guide*
- *Aruba Wireless Access Point Installation Guide*
- *Aruba Mobility Master Hardware Appliance Installation Guide*

Contacting Support

Table 2: *Contact Information*

Main Site	arubanetworks.com
Support Site	support.arubanetworks.com
Airheads Social Forums and Knowledge Base	community.arubanetworks.com
North American Telephone	1-800-943-4526 (Toll Free) 1-408-754-1200
International Telephone	arubanetworks.com/support-services/contact-support/
Software Licensing Site	hpe.com/networking/support
End-of-life Information	arubanetworks.com/support-services/end-of-life/
Security Incident Response Team	Site: arubanetworks.com/support-services/security-bulletins/ Email: sirt@arubanetworks.com

Introduction

Prior to ArubaOS 8.0.0.0, the only way a controller was configured was using command line interface (CLI) or controller user interface (WebUI). This created hindrance to automation because the CLIs typically changed over time and WebUI could not be automated easily because most of the pages were hand crafted. The WebUI also used CLI to communicate to the back end, which was hard coded and not easily extensible.

Another issue with the old architecture was that the **show** command of configuration was used to display the config pages (**GET** request), which used to come from apps. The apps maintained the configuration presented to them in their own proprietary structures and the **show** command output was inconsistent across apps. So, user had to know which **show** command to use, how to parse it, and get the output. If this output changed over time, the scripts also had to change as not all outputs were generated using structured data. **GET** and **SET** in a structured format for all configuration was the main requirement of implementing the JSON model.

Structured Data- Schema and Data

One of the main reasons for providing JSON interface is that all the configuration can now be **GET** and **SET** using structured data Application Program Interfaces (APIs). Structured data means that all the data is organized in a structure format (there can be many structures) where all elements that belong to one data type follows the same data model. This is achieved by separating schema from data.

Schema is a data model representation (in JSON format), which tells the user the way to interpret the data. It lists the complete detail of each and every parameter or token that a particular configuration element can take. For example, the type (integer, short integer, character, string, IP address, IPv6 address, MAC address etc), minimum value, maximum value, default value (when the user doesn't provide any value), optional or mandatory.

Data is the representation of the configuration state of the Mobility Master in JSON format. It arranges the data in the same order as the schema and can be interpreted as schema tells it to be interpreted. There may be parameters or tokens, which are mandatory in schema to be omitted in data if their presence in schema is to only convey the relationships between various parameters or tokens.

Schema (also known as Metadata) and data complete the structured data representation.

Prerequisites

- Complete understanding of the configuration hierarchy.
- Knowledge of the CLIs is required for the first time as all objects are based on the equivalent CLIs.
- Complete documentation of various containers and objects that are supported in Mobility Master running ArubaOS 8.0.0.0 is available— The URL of this document is https://<MM-IP>:4343/api.

  The user can run **GET** or **SET** commands from API page on the Mobility Master or can run equivalent **curl** command from any machine supporting **curl** commands. **curl** commands in the document are examples to run the query.

Interface

The interface used to access the configuration elements on Mobility Master is **HTTPS**. HTTPS is used because it provides transport layer security, and hence the passwords and other secret information can be sent over in plain text without worrying about anyone interfering. The same interface is used for all managed devices irrespective of it being a hardware-based or Virtual Machine (VM) based platform. Also, the managed device's role does not change this interface. So, the same interface works for Mobility Master, managed device, and stand-alone controller.

Login

To access any configuration element -- weather it is **GET** or **SET** on the object, the user first has to login to the Mobility Master.

The following is a sample **CURL** command used by the user to log in to the Mobility Master:

```
curl --insecure -c "aruba-cookie" -d "username=<username>&password=<password>"
https://<controller-ip>:4343/v1/api/login
```

The **--insecure** option can be used with the curl command if the certificate of the Mobility Master cannot be validated.

Parameters

The following table shows the parameters used in the login command:

Table 3: *Login Command Parameters*

Parameters	Description
<username>	Username of the user.
<password>	Password of the user.
<controller-ip>	IPv4 address of the Mobility Master.

The following is an example response for a failed login:

```
{"_global_result": {"status":"1", "status_str": "Unauthorized request, authentication
failed"}}
```

The following is an example response for a successful login:

```
{"_global_result": {"status":"0", "status_str": "You've logged in successfully.",
"UIDARUBA":"8e9b0e1a-4de0-4ace-a0c1-007ef267fa4b"}}
```

The **UIDARUBA** token has to be used in all **GET** or **SET** queries after the login.

Once logged in, the user can run **GET** and **SET** requests on containers or objects.

Logout

To close all the interactions, you need to logout from the Mobility Master.

The following is a sample **CURL** command used by the user to log out of the Mobility Master:

```
curl -c "aruba-cookie" https://<controller-ip>:4343/v1/api/logout
```

The **--insecure** option can be used with the curl command if the certificate of the Mobility Master cannot be validated.

Parameters

The following table shows the parameters used in the logout command:

Table 4: *Logout Command Parameters*

Parameters	Description
<controller-ip>	IPv4 address of the Mobility Master.

The following is an example response for a successful logout:

```
{"_global_result": {"status":"0", "status_str": "You've been logged out successfully.",
"UIDARUBA":"(null)"}}
```

Once logged out, no GET or SET requests can be run on the Mobility Master.

Query Elements of GET

There two different elements which can be queried for GET are containers and objects.

Containers

A container defines a logical group of objects and sub-objects. The way these objects and sub-objects are grouped is based on the configuration they modify. To get the complete list of containers available to be queried, use the following URL:

```
curl –b "aruba-cookie" –i 'https://<controllerip>:
4343/v1/configuration/container?UIDARUBA=<session-cookie>'
```

The **UIDARUBA=<session-cookie>** parameter needs to be added for all the queries of GET/ SET to address XSRF vulnerabilities.

The **--insecure** option can be used with the curl command if the certificate of the Mobility Master cannot be validated.

Putting a "/" after container will assume that a container name is being queried and will return invalid data as it will not find any container with blank name.

Containers terminology is only available via JSON REST APIs and not via CLIs.

The following table shows the all containers available in system along with the objects which they return:

Table 5: *Containers*

Container Name	Description
AP-Provisioning	AP Provisioning, AP Whitelist, Provisioning Profile Objects
Authentication	Authentication Server, Authentication, Survivability, Captive Portal, 801.X, Guest Provisioning, Kerberos, NTLM, Radius, Server Group, TACACS, WISPR Objects.
Crypto	Certificate Management, IPsec maps, Site2Site maps, VIA Maps, VPN Objects
External-Services	External Syslog Interface, ClearPass Policy Manager, Palo Alto Networks Management Objects
Hierarchy	Device Add/Delete/Move, Folder Add/Delete/Move Objects
Interfaces	Physical/Logical/Loopback Interfaces, Tunnels and USB/Modem Interfaces Objects
L2L3-Protocols	IGMP/MLD Snooping, OSPF, STP Objects
LoadBal-Redun	Clustering, High Availability, VRRP Objects
Pools	DHCP Pools, NAT Pools, Tunnel Pools Objects
Roles-Policies	ACLs, AppRF, Bandwidth Contracts, Firewall, PBR, Roles, WebCC Objects

Table 5: *Containers*

Container Name	Description
Services	ALE, AirGroup, Lync, Openflow, SDN Objects
WAN	Compression, Health Check, Uplink Management Objects
WLAN	AP Group, Client Match, Hotspot, IDS, Mcell, Mesh, Mobility, RF, SSID, Virtual AP Objects

Objects

Objects are identified by a name commonly known as Object Name or *objname* in short. One object is a logical combination of several parameters and optionally sub-objects, which require the presence of a main object. To get the complete list of objects available to be queried, use the following URL:

```
curl -b "aruba-cookie" -i 'https://<controllerip>:
4343/v1/configuration/object?UIDARUBA=<session-id>'
```

There are over 1000 objects that can be queried, so this request may take some time to execute.

The **--insecure** option can be used with the curl command if the certificate of the Mobility Master cannot be validated.

Putting a "/" after object will assume that a object name is being queried and will return invalid data as it will not find any object with blank name.

Various keys, shared secrets, passwords (with exception of logon users of any Aruba device) that are part of object data will be presented in clear-text format when an **HTTP GET** request is issued for that object.

GET

GET request can be sent to get the configuration data for an object(s) or container(s).

Syntax

To get the list of various objects, use the following command:

```
curl -b "aruba-cookie" -X GET -i 'https://<controllerip>:
4343/v1/configuration/object?UIDARUBA=<session-id>'
```

To get the list of various containers, use the following command:

```
curl -b "aruba-cookie" -X GET -i 'https://<controllerip>:
4343/v1/configuration/container?UIDARUBA=<session-id>'
```

Once the object or the container for which GET query has to be sent is known, use the following GET request:

```
curl -b "aruba-cookie" -X GET -i 'https://<controller-
ip>:4343/v1/configuration/object/<objectname>?
config_path=<config-node>&UIDARUBA=<session-id>'
curl -b "aruba-cookie" -X GET -i 'https://<controllerip>:
4343/v1/configuration/container/<container-name>?config_path=<config-node>&
UIDARUBA=<session-id>'
```

The **--insecure** option can be used with the curl command if the certificate of the Mobility Master cannot be validated.

NOTE

If config_path is not specified, **/mm/mynode** will be assumed.

Parameters

Table 6: *Get Command Parameters*

Parameters	Description
<controller-ip>	IPv4 address of the Mobility Master where the configuration element should be got from.
<config-node>	The hierarchy (complete config-node or config-path) from which the information should be got from. On managed device this will be restricted to /mm/mynode, while on a stand-alone controller, this will be restricted to /mm and /mm/mynode
<container-name>	Name of the container which needs to be queried.
<object-name>	Name of the object which needs to be queried.
<session-id>	Session ID for this session.

Sample Output

This returns a JSON payload which looks like the following output. This output is for object "int_vlan", a data-type filter (see Data-Type Filters on page 31 for more details) of "meta-n-data" is used to put both schema and data in perspective:

```
{
"_meta": {
   "int_vlan": {
      "_mappings": {
         "root": "int_vlan",
         "key_list": {
            "id": "id",
            "int_vlan_shut": "int_vlan_shut",
            "int_vlan_ip.ipparams": "int_vlan_ip.ipparams",
            "int_vlan_ip.ipaddr": "int_vlan_ip.ipaddr",
            "int_vlan_ip.ipmask": "int_vlan_ip.ipmask",
            "int_vlan_ip.dhcp-client": "int_vlan_ip.dhcp-client",
            "int_vlan_ip.client-id": "int_vlan_ip.client-id",
            "int_vlan_ip.cid": "int_vlan_ip.cid",
            ...,
            "int_vlan_ip_ospf_msg_digest_key.value": "int_vlan_ip_ospf_msg_digest_key.value",
            "int_vlan_ip_ospf_msg_digest_key.passwd": "int_vlan_ip_ospf_msg_digest_
key.passwd",
            "int_vlan_ip_ospf_area.area-id": "int_vlan_ip_ospf_area.area-id"
         }
       },
       "_operations": [
         "GET",
         "SET"
       ],
       "_keys": "id",
       "_inst_key": "id",
       "id": {
          "_min": 1,
          "_type": "INT",
          "_help": "Vlan interface number",
```

```
      "_max": 4094
    },
    "int_vlan_shut": {},
    "int_vlan_ip": {
      "ipparams": {
         "_type": "enum",
         "_children": [
            "ipaddrmask",
            "dhcp_opt",
            "pppoe"
       ]
       "_enum_type": "mixed"
     },
     "ipaddrmask": {
     "_parent": "ipparams",
     "_children": [
       "ipaddr",
       "ipmask"
     ]
   },
   "ipaddr": {
     "_parent": "ipaddrmask",
     "_type": "IPADDR",
     "_help": "A.B.C.D  IP address"
   },
   "ipmask": {
     "_parent": "ipaddrmask",
     "_type": "IPADDR",
     "_help": "A.B.C.D  IP subnet mask"
   },
   "dhcp_opt": {
     "_parent": "ipparams",
     "_children": [
     "dhcp-client"
   ]
 },
   "dhcp-client": {
     "_type": "keyword",
     "_parent": "dhcp_opt",
     "_children": [
        "client-id"
     ]
   },
   "client-id": {
     "_type": "keyword",
     "_parent": "dhcp-client",
     "_optional": true,
     "_children": [
     "cid"
      ]
   },
   "cid": {
     "_min": 1,
     "_type": "STRING",
     "_max": 255,
     "_parent": "client-id",
     "_optional": true,
     "_help": "ASCII string to be sent in the options"
    },
   "pppoe": {
     "_type": "keyword",
     "_parent": "ipparams"
```

```
      }
     },
     "int_vlan_ipv6_addr": {
       "_keys": "eui-64,ipaddr",
       "_inst_key": "ipaddr",
       "eui-64": {
         "_type": "keyword",
         "_optional": true
       },
       "ipaddr": {
         "_type": "IP6PREFIX",
         "_help": "IPv6 prefix"
       }
       },
       ...,
       ...,
       "int_vlan_mtu": {
         "value": {
           "_min": 1280,
           "_type": "INT",
           "_max": 1500,
           "_default_val": 1500,
           "_help": "MTU value"
         }
       }
   }
},
"_data": {
   "int_vlan": [
     {
        "id": 95,
        "int_vlan_ip": {
           "ipaddr": "95.95.1.1",
           "ipparams": "ipaddrmask",
           "ipmask": "255.255.255.0"
         },
         "int_vlan_routing": {
           "_present": true,
           "_flags": {
              "default": true
           }
         },
         "int_vlan_ndra_hlimit": {
           "_flags": {
             "default": true
           },
         "value": 64
           },
         "int_vlan_ndra_interval": {
           "_flags": {
             "default": true
           },
           "value": 600
         },
         "int_vlan_ndra_ltime": {
           "_flags": {
             "default": true
           },
           "value": 1800
         },
         "int_vlan_ndra_mtu": {
           "_flags": {
```

```
            "default": true
          },
          "value": 1500
        },
        "int_vlan_nd_reachtime": {
          "_flags": {
           "default": true
          },
          "value": 0
        },
        "int_vlan_nd_rtrans_time": {
          "_flags": {
            "default": true
          },
          "value": 0
         },
        "int_vlan_mtu": {
          "value": 1390
          },
        "int_vlan_suppress_arp": {
          "_present": true,
          "_flags": {
            "default": true
          }
        }
      },
     {
     "id": 96,
     "int_vlan_ip": {
       "dhcp-client": true,
       "ipparams": "dhcp_opt"
     },
     ...,
     "int_vlan_mtu": {
       "value": 1400
     }
   },
   {
   "id": 97,
   ...,
   "int_vlan_mtu": {
     "value": 1290
   }
   },
 {
 "id": 98,
 ...,
 "int_vlan_ip": {
   "cid": "123",
   "ipparams": "dhcp_opt",
   "client-id": true,
   "dhcp-client": true
  }
},
{
"id": 99,
...,
"int_vlan_mtu": {
  "value": 1360
},
"int_vlan_suppress_arp": {
  "_present": true,
```

```
      "_flags": {
         "default": true
      }
     }
    }
   ]
  }
}
```

Understanding the Output

The following is the explanation for each element:

- The initial elements in the response have heading called "_meta" and "_data". Before delving into what each one of these headings mean (which is self explanatory), know that any name starting with underscore "_" is system generated and has a special meaning, which is global rather than per command basis.
- "_meta" specifies the meta or schema for this section. The following is a detailed explanation of each schema element.
  - The first order elements are all objects which belong to the container name. In case of single object, it will just list the object name there. In this example, we have only one object named "int_vlan" -- which gets all configuration elements for "interface vlan" object.
  - "_mappings" field specifies all the objects and their parameters inside this particular object. The "root" is always the object itself and "key_list" carries the rest of elements and how to access them from Javascript objects.
  - "_inst_key" specifies which parameters inside this object are used to form a unique key identifying one instance of an object. This field is present only for objects which can have multiple instances -- e.g., VLANs, interfaces, aaa profiles etc ...
  - "_keys" tells what all fields are significant in this object. This is only meant for working of WEBUI (which also uses JSON Interface) and can be safely ignored for pure API use.
  - “_operations” specifies all operations that are permitted on that object. It can have a value of “GET” and/or “SET”
  - Other fields which do not start with underscore "_" are parameters or sub-objects of this main object. Each object has one or more of the following attributes:
    - "_type": This tells the type of the attribute of the parameter. If this attribute is missing for any parameter or subobject, it means that this is just for structuring data. This need not be specified by the user or may or may not come from the Mobility Master.
    - "_max", "_min": These gives the limits of the parameter. If the type is Integer or anything with a number, it specifies maximum and minimum numbers which can be entered. If it's type is "string", it specifies max and min length of string respectively.
    - "_optional": Specifies that this parameter is not a required parameter. All parameters with a "_type" which are not marked "_optional" are all required parameters.
    - "_parent": Specifies the parent "parameter" of this parameter. Used only for nested parameters. If this field is not present means that the object parameter is not a nested one.
    - "_children": Specifies that this parameter is a nested one and contains multiple children parameters specified by the value of this tag. "_type" argument will help in determining if all the children parameters need to be specified or only one of them (in case it's of type enum which is same as OneOf type in Netconf model.
    - "_enum_type": Specifies if the parameter is of "enum" type, then are all the values of the enum of same type. For example, the values are "ui_dropbox" (for static strings) and "mixed" (for dynamic strings).

- If the parameter contains another complex object(s) inside it without a "_type" or "_parent" or "_children" tokens, then it's a subobject which is different from nested parameter. All nested parameters have to be specified together but sub-object is totally independent and can be specified or not.

- "_data" specifies the data or configuration elements from the Mobility Master. Now lets go into details about each data element.
  - As with the "_meta", the first thing is the object name for which the values are present below.
  - If the object can have multiple instances, then it's value is an array of single objects -- each entry in the array is an individual instance. If it's a single instance object, then all parameters are present right there.
  - For sub-object inside an object, it is present like another object inside the main object. Each sub-object has its own elements but cannot exist without the main or top-level object.
  - Each object's parameters are then specified using "Tag" and "Value" nomenclature. No hierarchy exists (flat tag-value kind of parameters) between various parameters -- not even nested parameters -- One caveat is that nested Objects' parameters are specified as a separate object in the main object. "_meta" specifies how the nesting should be done.
  - "_flags": This is a special field in each object/sub-obj level which specifies details about the data. If none of these flag types are specified (flags are empty), then this structure may be missing. The various values possible inside are:
    - "inherited": Specifies that this configuration element has been inherited from the hierarchy above. It means that the configuration present here has been configured on a node above this and not at this node.
    - "readonly": Species that this configuration element is readonly -- it can't be edited. It can only be deleted.
    - "undeletable": Specifies that this configuration element can't be deleted -- it can however be edited.
    - "pending": Specifies that this configuration hasn't been saved in flash and has not yet been pushed to apps.
    - "default": Specifies that the configuration is part of factory default configuration or is generated as default by the system.
    - "system": Specifies that the configuration is system generated.

Additionally, "Filters" can also be sent to filter data or url parameter options can be used to sort, count and paginate the data in case there is too much data. This will be explained in later sections.

SET

A SET request is sent in case a new data has to be added, or the old data has to be either modified or deleted (we currently do not support HTTP DELETE and HTTP PUT operations). The same is achieved by sending HTTP POST request using the **curl** command.

Syntax

```
curl -b "aruba-cookie" -X POST -i 'https://<controller-
ip>:4343/v1/configuration/object?config_
path=/md/11:22:33:aa:bb:cc&UIDARUBA=<session-cookie>' -d @<set-payload-file>
```

The **--insecure** option can be used with the curl command if the certificate of the Mobility Master cannot be validated.

You can send as many objects as you want in a request as long as the complete request does not exceed 1 MB.

Set request is best effort. It is not all or none. It can have a partially applied configuration. Look at the payload result for details on what succeeded and what failed.

Parameters

Table 7: *Set Command Parameters*

Parameters	Description
<controller-ip>	IPv4 address of the Mobility Master where the configuration element should be got from.
<session cookie>	Session cookie for this session.
<set-payload-file>	File containing the JSON payload which can be set to.

Sample Configuration or Payload File

The following is the sample file for setting the "int_vlan" object:

```
{
   "int_vlan": [
     {
       "id": "199",
       "_action" : "add",
       "int_vlan_ip": {
       "ipaddr": "100.1.1.1",
       "ipmask": "255.0.0.0",
       "_action" : "add"
      },
     "int_vlan_mtu": {
       "value": "1300",
       "_action": "add"
     }
    },
   {
    "id": 198,
    "_action" : "add",
    "int_vlan_ip": {
      "ipaddr": "111.1.1.1",
      "ipmask": "255.0.0.0",
      "_action": "add"
   },
   "int_vlan_mtu": {
     "value": "1400",
     "_action": "add"
     }
   },
   {
     "id": 197,
     "_action" : "add",
     "int_vlan_ip_helper": [
     {
       "address": "101.1.1.1",
       "_action": "add"
     },
     {
       "address": "102.1.1.1",
       "_action": "add"
     },
     {
```

```
        "address": "103.1.1.1",
        "_action": "add"
      },
      {
        "address": "104.1.1.1",
        "_action": "add"
      }
     ]
    },
    {
      "id": "196",
      "_action" : "add",
      "int_vlan_ip": {
        "dhcp-client": true,
        "_action": "add"
       }
     },
    {
      "id": "95",
      "_action": "add"
    }
 ]
}
```

Understanding SET Request and Response

- "config-path": This specifies the node path in the configuration hierarchy at which this configuration should go to. In this example it says: "/md/11:22:33:aa:bb:cc" is the configuration node path. That means device name is "11:22:33:aa:bb:cc" and it is located under "/" node in the configuration hierarchy.
- Next comes the "data", which contains the data to be set. The details on data are below:
  - The "data" contains a bunch of objects, which needs to be set. Each object's name comes first.
  - Each object contains either an array of instances (very similar to GET request) if there can be multiple instances of an object or directly an object for single instance object or you want to configure only one instance of a multi-instance object.
  - Every object and sub-object can optionally contain another field called "_action". It tells about what should be done with the object. Absence of this parameter means that the user wants to add/modify the configuration. This field is mandatory if trying to delete any configuration. The various values it can take is:
    - "add" - add this new object and if already present, it will replace the old object with new.
    - "delete" - delete the instance of a multi-instance object or delete the full object for a single instance object.
    - "noop" - No action is required for this object. This value should ideally never be used.

This returns the same JSON payload with result value and global result flag indicating what succeeded and what didn't.

```
{
   "int_vlan": [
     {
       "id": "199",
       "_action": "add",
       "int_vlan_ip": {
         "ipaddr": "100.1.1.1",
         "ipmask": "255.0.0.0",
         "_action": "add",
         "_result": {
           "status": 0,
           "status_str": ""
```

```
      }
    },
    "int_vlan_mtu": {
      "value": "1300",
      "_action": "add",
      "_result": {
        "status": 0,
        "status_str": ""
      }
    },
    "_result": {
    "status": 0,
    "status_str": ""
      }
  },
{
  "id": 198,
  "_action": "add",
  "int_vlan_ip": {
    "ipaddr": "111.1.1.1",
    "ipmask": "255.0.0.0",
    "_action": "add",
    "_result": {
      "status": 0,
      "status_str": ""
    }
 },
    "int_vlan_mtu": {
      "value": "1400",
      "_action": "add",
      "_result": {
        "status": 0,
        "status_str": ""
      }
  },
    "_result": {
      "status": 0,
      "status_str": ""
      }
    },
 {
    "id": 197,
    "_action": "add",
    "int_vlan_ip_helper": [
     {
       "address": "101.1.1.1",
       "_action": "add",
       "_result": {
         "status": 1,
         "status_str": "IP Address not set on the Vlan Interface.\n"
       }
     },
     {
       "address": "102.1.1.1",
       "_action": "add",
       "_result": {
         "status": 2,
         "status_str": "Error detected in previous object. Bypassing this object."
       }
     },
     {
       "address": "103.1.1.1",
```

```
            "_action": "add",
            "_result": {
              "status": 2,
              "status_str": "Error detected in previous object. Bypassing this object."
            }
          },
          {
            "address": "104.1.1.1",
            "_action": "add",
            "_result": {
              "status": 2,
              "status_str": "Error detected in previous object. Bypassing this object."
            }
          }
        ],
           "_result": {
             "status": 0,
             "status_str": ""
        }
       },
       {
         "id": "196",
         "_action": "add",
         "int_vlan_ip": {
           "dhcp-client": true,
           "_action": "add",
           "_result": {
             "status": 2,
             "status_str": "Error detected in previous object. Bypassing this object."
           }
         },
           "_result": {
             "status": 2,
             "status_str": "Error detected in previous object. Bypassing this object."
           }
        },
       {
         "id": "95",
         "_action": "add",
         "_result": {
           "status": 2,
           "status_str": "Error detected in previous object. Bypassing this object."
          }
      }
     ],
     "_global_result": {
       "status": "1",
       "status_str": "IP Address not set on the Vlan Interface.\n"
       }
     "_global_result": {
         "status": "1",
         "status_str": "IP Address not set on the Vlan Interface.\n",
       "_pending" : 0
       }
}
```


The response payload is very identical to the SET payload with an additional field with each object and sub-object called "_result", which carries the result information. This is composed of two sub-objects: "status" and "status_str". If the "status" is 0, it is successful execution of the action specified. Any notice/output from the result is provided in the "status_str" which is mostly empty in cases of SUCCESS. In case the "status" is non-zero, it carries an error message in the "status_str" for the executioner of the API. Similarly to get the status of

the complete query, there is a field called "_global_result" which returns the first error which happened or SUCCESS. There is another field, which is present in global result but not present in per object results – "_ pending", the value of which can be 1 or 0. 1 means that the config node for which set request was run, is now in pending state. 0 means that it is not in pending state (in committed state).

The SET request is best effort and in case of first failure, others in the same block are not even tried. The error in such case will be "Error detected in previous object. Bypassing this object".

Multi-part SET

Multiple SET/POST requests can also be concatenated in one message. Each block is treated as independent request. So, even though setting of one object fails in one block, the other blocks will still continue to be processed.

Here is a sample multi-part SET Request:

```
{
   "_list":[
    [
      {
        "ids_dos_prof":[
          {
            "_action" : "add",
            "profile-name":"AAA"
          },
          {
            "_action" : "add",
            "profile-name" : "pA"
          }
        ]
      },
      {
        "ids_impersonation_prof": [
          {
             "_action" : "add",
             "profile-name":"AAA"
          }
        ]
      },
      {
        "ids_signature_matching_prof":[
         {
           "_action" : "add",
           "profile-name":"AAA"
         },
         {
           "_action": "add",
           "profile-name" : "pA"
        }
       ]
    }
   ],
   [
     {
       "ids_dos_prof":[
         {
           "_action" : "add",
           "profile-name":"AAA2"
         },
         {
           "_action" : "add",
```

```
            "profile-name" : "pA2"
          }
        ]
      },
      {
        "ids_impersonation_prof": [
          {
            "_action" : "add",
            "profile-name":"AAA2"
          }
         ]
      },
      {
        "ids_signature_matching_prof":[
          {
            "_action" : "add",
            "profile-name":"AAA2"
          },
          {
            "_action": "add",
            "profile-name" : "pA2"
           }
        ]
       }
    ]
  ]
}
```

In above payload each element in "_list" is an independent SET request. The following is the response to this API:

```
{
    "_list": [
      [
        {
          "ids_dos_prof": [
            {
              "_action": "add",
              "profile-name": "AAA",
              "_result": {
                "status": 0,
                "status_str": ""
              }
             },
             {
              "_action": "add",
              "profile-name": "pA",
              "_result": {
                "status": 0,
                "status_str": ""
              }
            }
          ]
        },
        {
           "ids_impersonation_prof": [
            {
              "_action": "add",
              "profile-name": "AAA",
              "_result": {
                "status": 0,
                "status_str": ""
              }
```

```
        }
       ]
     },
     {
       "ids_signature_matching_prof": [
        {
          "_action": "add",
          "profile-name": "AAA",
          "_result": {
          "status": 0,
          "status_str": ""
         }
       },
       {
         "_action": "add",
         "profile-name": "pA",
         "_result": {
         "status": 0,
         "status_str": ""
         }
        }
       ]
     },
     {
       "_global_result": {
         "status": "0",
         "status_str": "Success"
       }
     }
  ],
 [
 {
    "ids_dos_prof": [
     {
       "_action": "add",
       "profile-name": "AAA2",
       "_result": {
         "status": 0,
         "status_str": ""
       }
       },
     {
       "_action": "add",
       "profile-name": "pA2",
       "_result": {
         "status": 0,
         "status_str": ""
       }
      }
     ]
   },
 {
   "ids_impersonation_prof": [
     {
        "_action": "add",
        "profile-name": "AAA2",
        "_result": {
           "status": 0,
           "status_str": ""
         }
       }
     ]
```

```
        },
        {
           "ids_signature_matching_prof": [
             {
               "_action": "add",
               "profile-name": "AAA2",
               "_result": {
                 "status": 0,
                 "status_str": ""
               }
             },
             {
               "_action": "add",
               "profile-name": "pA2",
               "_result": {
                 "status": 0,
                 "status_str": ""
               }
             }
           ]
         },
         {
            "_global_result": {
              "status": "0",
              "status_str": "Success"
            }
          }
        ],
           "_global_result": {
             "status": "0",
             "status_str": "Success"
    }
}
```


GET Modifiers

The output data which is retrieved with GET query may be huge and may require some actions on it. For this reason, modifiers are designed on the GET query. This can also be achieved by putting URL parameters (query-params) in the URL, as shown below:

```
curl -b "aruba-cookie" -X GET -i 'https://<controllerip>:
4343/v1/configuration/container/<container-name>?config_path=<config-
node>&<queryparams>&UIDARUBA=<session-id>'
```

The **--insecure** option can be used with the curl command if the certificate of the Mobility Master cannot be validated.

Basic Filters (Key/Value Filters)

Basic filters are one the main modifiers which reduces the amount of data from GET Query. There are two types of basic filters: object and data. This filter is also known as key/value filter or filter, as it takes keys and/or values for applying the filter.

Object filters can filter schema and data information based on the objects specified.

Data filter only filters the data being returned based on certain values for the various fields.

The following is an example of a basic filter:

```
curl -b "aruba-cookie" -X GET -i
'https://10.4.248.227:4343/v1/configuration/object/<object>?config_path=<config-
node>&filter=<list_of_filters>&UIDARUBA=<session-id>'
```

Object Filter

An object filter limits which objects or which sub_objects should be present in the response.

This filter cannot be applied on individual parameters within an object or a sub-object. Also, there can only be one object filter for every request. These cannot be concatenated.

The following is an example of an object filter:

```
curl -b "aruba-cookie" -X GET -i
'https://10.4.248.227:4343/v1/configuration/object/<object>?config_path=<config-node>&filter=
[{"OBJECT" : { "<oper>" : <list-of-parameters> } }  ]&UIDARUBA=<session-id>'
```

Parameters

Table 8: *Object Filter Parameters*

Parameters	Description
"OBJECT"	This specifies that this filter is of type object.
<oper>	This specifies what operation should be applied on the values following this. Currently, it can carry the following two values: ■ $eq: matches one of the values ■ $neq: does not match any of the values
<list_of_parameters>	The list following <oper> specifies the values which need to be filtered based on the operation.
<session-id>	Session ID for this session.

The following sample request shows how to return sub-objects of "ip address" and "mtu" value configured for all interface VLAN at this particular node (or above in hierarchy):

```
curl -b "aruba-cookie" -X GET -i
'https://10.4.248.227:4343/v1/configuration/object/<object>?config_path=<config-node>&filter=
[{"OBJECT" : { "$eq" : ["int_vlan.int_vlan_ip", "int_vlan.int_vlan_mtu"] } }
]&UIDARUBA=<session-id>'
```

The following is the response to the above request:

```
{
     "_data": {
       "int_vlan": [
       {
         "id": 98,
         "int_vlan_ip": {
           "ipaddr": "98.1.1.1",
           "ipparams": "ipaddrmask",
           "ipmask": "255.0.0.0"
          },
       "int_vlan_mtu": {
         "value": 1400
       }
     },
     {
       "id": 95,
       "int_vlan_mtu": {
         "_flags": {
```

```
            "default": true
          },
          "value": 1500
        }
      },
      {
        "id": 99,
        "int_vlan_ip": {
          "ipaddr": "99.1.1.1",
          "ipparams": "ipaddrmask",
          "ipmask": "255.0.0.0"
        },
          "int_vlan_mtu": {
          "value": 1300
        }
      },
      {
        "id": 97,
        "int_vlan_ip": {
          "ipaddr": "97.1.1.22",
          "ipparams": "ipaddrmask",
          "ipmask": "255.255.255.0"
        },
        "int_vlan_mtu": {
          "_flags": {
            "default": true
        },
        "value": 1500
      }
     },
     {
       "id": 96,
       "int_vlan_ip": {
         "dhcp-client": true,
         "ipparams": "dhcp_opt"
      },
         "int_vlan_mtu": {
        "value": 1280
       }
      }
     ]
   }
}
```


Data Filters

Any filter which is not an object filter is a data filter by default. It filters out the configuration elements configured on the system. You can concatenate multiple data filters in one request.

The following is a sample data filter:

```
curl -b "aruba-cookie"  -X GET -i
'https://10.4.248.227:4343/v1/configuration/object/<object>?config_path=<config-node>&filter=
[{"<param-name" : { "<oper>" : <list-of-values> } }  ]&UIDARUBA=<session-id>'
```

Parameters

Table 9: *Data Filter Parameters*

Parameters	Description
<param-name>	Fully qualified name of the parameter on values of which filter needs to be applied.
<oper>	This specifies what operation should be applied on the values following this. Currently, it can carry the following two values: ■ $eq: matches one of the values ■ $neq: does not match any of the values ■ $gt: matches a value which is greater than the filter ■ $gte: matches a value which is greater than or equal to the filter ■ $lt: matches a value which is less than the filter ■ $lte: matches a value which is less than or equal to the filter ■ $in: pattern matches the filter value. E.g., if filter says “ap”, “default-ap” and “ap-grp1” will both match ■ $nin: pattern does not match the filter. Opposite of $in
<list_of_values>	The list following <oper> specifies the values which need to be filtered based on the operation.
<session-id>	Session ID for this session.

The following sample request shows a data filter where we want to get details of interface VLANs 95 and 96:

```
curl -b "aruba-cookie" -X GET -i
'https://10.4.248.227:4343/v1/configuration/object/<object>?config_path=<config-node>&filter=
[{"int_vlan.id" : { "$eq" : [95, 96] } }&UIDARUBA=<session-id>'
```

The following is the response to the above request:

```
{
    "_data": {
      "int_vlan": [
    {
      "id": 95,
      "int_vlan_routing": {
        "_present": true,
        "_flags": {
          "default": true
      }
    },
      "int_vlan_ndra_hlimit": {
        "_flags": {
          "default": true
        },
        "value": 64
       },
      "int_vlan_ndra_interval": {
         "_flags": {
           "default": true
         },
         "value": 600
       },
       "int_vlan_ndra_ltime": {
         "_flags": {
           "default": true
         },
         "value": 1800
       },
       "int_vlan_ndra_mtu": {
```

```
      "_flags": {
        "default": true
      },
      "value": 1500
    },
    "int_vlan_nd_reachtime": {
      "_flags": {
        "default": true
      },
      "value": 0
    },
    "int_vlan_nd_rtrans_time": {
       "_flags": {
         "default": true
      },
      "value": 0
    },
    "int_vlan_mtu": {
      "_flags": {
        "default": true
      },
      "value": 1500
    },
    "int_vlan_suppress_arp": {
      "_present": true,
      "_flags": {
        "default": true
      }
    }
  },
  {
    "id": 96,
    "int_vlan_ip": {
      "dhcp-client": true,
      "ipparams": "dhcp_opt"
    },
    "int_vlan_routing": {
      "_present": true,
      "_flags": {
        "default": true
      }
    },
    "int_vlan_ndra_hlimit": {
      "_flags": {
        "default": true
      },
      "value": 64
    },
    "int_vlan_ndra_interval": {
      "_flags": {
        "default": true
      },
      "value": 600
    },
    "int_vlan_ndra_ltime": {
      "_flags": {
        "default": true
      },
      "value": 1800
    },
    "int_vlan_ndra_mtu": {
      "_flags": {
```

```
            "default": true
          },
          "value": 1500
        },
        "int_vlan_nd_reachtime": {
          "_flags": {
            "default": true
          },
          "value": 0
        },
        "int_vlan_nd_rtrans_time": {
          "_flags": {
            "default": true
          },
          "value": 0
        },
        "int_vlan_mtu": {
          "value": 1280
        },
        "int_vlan_suppress_arp": {
          "_present": true,
          "_flags": {
            "default": true
           }
         }
       }
    ]
  }
}
```


Data-Type Filters

Data-type filters is just an extension to the BASIC filters and applies to whole data. This specifies what type of response is required.

The following sample shows the way to specify the type of data requested:

```
curl -b "aruba-cookie" -X GET -i
'https://10.4.248.227:4343/v1/configuration/object/<object>?config_path=<config-
node>&type=<data-type>&UIDARUBA=<session-id>'
```

The **--insecure** option can be used with the curl command if the certificate of the Mobility Master cannot be validated.

The following table shows the list of various data-types that can be specified in the list (the data-types can be concatenated using "commas" in between with no spaces):

Table 10: *List of Data-Types*

Data-Type	Description
"non-default"	Send configuration which is not factory default. It can be system generated or user configured.
"default"	Send only factory default configuration.
"local"	Send the configuration done at this particular node by user. It can still be in pending state or committed — both are present.

Data-Type	Description
"user"	All configuration done by user (non factory default and non system generated). It can be done on any node present in this node's hierarchy (all inherited user configuration is also present)
"system"	A user configuration may trigger system to internally generate some configurations—e.g., ACE generation on net-destination use. This type returns all such system generated configurations. It contains configurationgenerated at this node or higher in hierarchy.
"pending"	Send only the configuration at this node which is not written to memory yet.
"committed"	Send only committed configuration at this node. This includes the inherited configuration.
"inherited"	Send all configuration inherited at this node. It includes all user, system, default configuration which is coming from hierarchy above.
"meta-n-data"	Special Flag to indicate that get both metadata and data in the same request. This cannot be combined with "meta-only"; the two are mutually exclusive.
"meta-only"	Special Flag to indicate that only metadata is desired, skip sending any configuration elements (or data). This cannot be combined with the "meta-n-data" option; the two are mutually exclusive.
<session-id>	Session ID for this session.

Sort

The data from the GET request can be sorted based on a single field (currently, multi-parameter or nested sorts are not supported). There can only be one sort filter per request.

The following sample shows how to sort:

```
curl -b "aruba-cookie" -X GET -i
'https://10.4.248.227:4343/v1/configuration/object/<object>?config_path=<config-
node>&sort=<oper><key>&UIDARUBA=<session-id>'
```

Parameters

Table 11: *Sort Parameters*

Parameters	Description
<oper>	Shows the order in which you want the output. It can have only two values, "+" for ascending order and "-" for descending order. If this is not specified, ascending order is assumed by default.
<key>	Key is the parameter name on which sort must be applied. It is always of the form: <objname>.<param_name> or in case of sub-objs, it should be <objname>.<subobj_ name>.<param_name>
<session-id>	Session ID for this session.

Points to Remember

- If the top-level object is multi-instance, you can apply filter on any parameter of that object. The following is an example for ascending order sort:

```
curl -b "aruba-cookie" -X GET -i
'https://10.4.248.227:4343/v1/configuration/object/<object>?config_path=<config-
node>&sort=+int_vlan.id&UIDARUBA=<session-id>'
```

- If the top-level object is multi-instance, you can apply filter on any parameter of the sub-object as long as sub-object is not multi-instance. The following is an example for descending order sort:

  ```
  curl -b "aruba-cookie" -X GET -i
  'https://10.4.248.227:4343/v1/configuration/object/<object>?config_path=<config-
  node>&sort=-int_vlan.int_vlan_mtu.value&UIDARUBA=<session-id>'
  ```

- If the top-level object is multi-instance, you can apply filter on any parameter of the sub-object which is multi-instance as long as only one instance exists for top object (e.g., applying sort on port number of an ACE for a single session ACL). The following example sorts the "IP Helper ipv4 addresses" of interface vlan 95 object while returning all the sub-objects of "interface vlan 95":

  ```
  curl -b "aruba-cookie" -X GET -i
  'https://10.4.248.227:4343/v1/configuration/object/<object>?config_path=<config-
  node>&sort=-int_vlan.int_vlan_ip_helper.address&filter=[ {"int_vlan.id" : { "$eq" : [95] }
  }]&UIDARUBA=<session-id>'
  ```

Paginate

As the name suggests, paginate slices the data into pages and returns the amount of data the user is interested in. The following shows a sample paginate modifier:

```
curl -b "aruba-cookie" -X GET -i
'https://10.4.248.227:4343/v1/configuration/object/<object>?config_path=<config-
node>&offset=<off>&limit=<lim>&total=<count>&UIDARUBA=<session-id>'
```

Parameters

Table 12: *Paginate Parameters*

Parameters	Description
<off>	This conveys the number of the entry from which we should start the next data set. For example, offset value of 21 means that out of all the instances of an object, give me data from 21st object. Currently, we support <off> value to be multiples of <lim> field described below + 1. For example, if <lim> is 20, the <off> can take values of 1, 21, 41, 61, 81 etc.
<limit>	The maximum number if instances of an object that should be put in a single request.
<count>	The total number of instances of that object existing in system at that given configuration node. It should be set to 0 when first query is done and the count field specified in the result should be put here (optionally) for subsequent queries. This is an optional field and is explained below.
<session-id>	Session ID for this session.

For example, let's assume that there are 260 VLANs configured on a system and you want to query 50 VLANs at one time. So, for first request, you will set <off> to 1 and <limit> to 50. The query will return first 50 VLANs to you. Now, for second query, the <off> will be 51 and <count> (which we would have returned in the first request) will be set to 260. If before the second request, someone adds a VLAN which makes the total number 261, we will check this number against one in the request which is 260. Since these do not match, we will send a "_status" field at the same level as "_meta" and "_data" and it's value will be set to "Data Dirty". This indicates that even though we returned next 50 records, data elements may be repeated or missing (in case someone deletes one of first 50 VLANs, we will return from 52nd VLAN the next time as that becomes the new 51st VLAN).

The last query will return only 10 VLANs which signals to user that no more VLANs are available. If user requests for <off> of 401, no data will be returned as we don't have 400+ VLANs.

- If top-level object is multi-instance and more than one instance exists in query data, paginate will apply to it.
- If top-level object is multi-instance, and you have only one instance in query data, paginate will run on multi-instances of the sub_objects.
- If top-level command is single-instance, paginate will run on multi-instance sub_objects only.

The following is a sample paginate request:

Request on Section "vlans", where we query for all sub-objs of vlan 97 and any multi-instance subobject (like int_vlan_ip_helper) should only return 3 total objects:

```
curl -b "aruba-cookie" -X GET -i
'https://10.4.248.227:4343/v1/configuration/object/<object>?config_path=<config-
node>&offset=1&limit=3&filter=[{"int_vlan.id" : { "$eq" : [97] } }, {"OBJECT" : { "$eq" :
["int_vlan.int_vlan_ip_helper"]}}]&UIDARUBA=<session-id>'
```

The following is the response to the above request:

```
{
    "_data": {
      "int_vlan": [
      {
        "id": 97,
        "int_vlan_ip_helper": [
         {
           "address": "1.1.1.1"
         },
         {
           "address": "2.1.1.1"
         },
         {
           "address": "3.1.1.1"
         }
        ]
       }
      ]
    },
    "_status": "Data Dirty",
    "_count": {
      "int_vlan.int_vlan_ip_helper": 4
   }
}
```

So, you see that the total count of "int_vlan.int_vlan_ip_helper" sub-object is 4 but only 3 records are returned. The status is marked "Data Dirty" as we didn't send any <count> in initial request.

COUNT

The count modifier just returns the total count of an object for multi-instance object or for multi-instance sub-object rather than the actual details of the objects. This is particularly useful when you want to get only the number of instances in an object rather than the object details. For example, querying for number of ACEs in an ACL.

The following shows a sample COUNT modifier:

```
curl -b "aruba-cookie" -X GET -i
'https://10.4.248.227:4343/v1/configuration/object/<object>?config_path=<config-
node>&count=<count_keys_list>&UIDARUBA=<session-id>'
```

Parameters

Table 13: *Count Modifier Parameters*

Parameters	Description
<count_keys_list>	List of fully qualified parameter name for which count operation needs to be performed.
<session-id>	Session ID for this session.

The following is a sample request on section "vlans", where we query for all interface vlans with count modifier set on int_vlan_ip_helper's address parameter:

```
curl -b "aruba-cookie" -X GET -i
'https://10.4.248.227:4343/v1/configuration/object/<object>?config_path=<config-
node>&count=int_vlan.int_vlan_ip_helper&filter=[{"OBJECT" : { "$eq" : ["int_vlan.int_vlan_ip_
helper"]}}]&UIDARUBA=<session-id>'
```

The following is the response of the above request. Notice that **int_vlan_ip_helper** just has count field inside it rather than full details of the **int_vlan_ip_helper**.

```
{
    "_data": {
      "int_vlan": [
      {
        "id": 98,
        "int_vlan_ip_helper": {
          "_count": 0
        }
     },
     {
       "id": 299,
       "int_vlan_ip_helper": {
         "_count": 2
       }
    },
    {
      "id": 97,
      "int_vlan_ip_helper": {
        "_count": 4
      }
    }
  ]
}
```

Special GET Queries

Configuration Hierarchy

The configuration hierarchy along with all the devices information can be fetched using a specialized query listed below:

```
curl -b "aruba-cookie" -X GET -i 'https://<controller-ip>:4343/v1/configuration/object/node_
hierarchy?UIDARUBA=<session-id>'
```

The **--insecure** option can be used with the curl command if the certificate of the Mobility Master cannot be validated.

The following is the response to the above query:

```
{
   "name": "/",
   "devices": [],
   "num_ports": 3,
   "device_count": 5,
   "childnodes": [
   {
      "name": "mm",
      "devices": [],
      "num_ports": 3,
      "device_count": 0,
      "childnodes": [
      {
         "name": "mynode",
         "devices": [],
         "num_ports": 3,
         "device_count": 0,
         "childnodes": [],
         "type": "group"
      }
   ],
   "type": "group"
},
{
   "name": "md",
   "devices": [
      {
         "name": "first-device",
         "longitude": "44",
         "mac": "00:11:22:33:44:55",
         "num_ports": 8,
         "latitude": "33",
         "type": "A7008"
      },
      {
         "name": "66:66:66:66:66:66",
         "longitude": "",
         "mac": "66:66:66:66:66:66",
         "num_ports": 4,
         "latitude": "",
         "type": "A7005"
      },
      {
         "name": "77:77:77:77:77:77",
         "longitude": "",
         "mac": "77:77:77:77:77:77",
         "num_ports": 4,
         "latitude": "",
         "type": "A7005"
      }
   ],
   "num_ports": 4,
   "device_count": 5,
   "childnodes": [
   {
      "name": "us",
      "devices": [
         {
         "name": "second-device",
         "longitude": "",
         "mac": "22:11:22:11:22:11",
         "num_ports": 6,
```

```
            "latitude": "",
            "type": "A7210"
            }
         ],
         "num_ports": 6,
         "device_count": 2,
         "childnodes": [
         {
            "name": "nevada",
            "devices": [
            {
            "name": "11:11:11:33:44:55",
            "longitude": "",
            "mac": "11:11:11:33:44:55",
            "num_ports": 8,
            "latitude": "",
            "type": "A7008"
            }
         ],
         "num_ports": 8,
         "device_count": 1,
         "childnodes": [],
         "type": "group"
         }
      ],
      "type": "group"
      }
      ],
      "type": "group"
      }
],
"type": "root"
}
```


System Information

System API is a special query which returns details about the system on which the query is being sent to. The API is node-specific. If no path is specified, the query is run for that device and it's data returned.

The following is a sample system information request:

```
curl -b "aruba-cookie" -X GET -i 'https://<controller-ip>:4343/v1/configuration/object/sys_
info?config_path=<config-node>&UIDARUBA=<session-id>'
```

The **--insecure** option can be used with the curl command if the certificate of the Mobility Master cannot be validated.

The following is the response to the above request:


```
{
   "_global": {
      "_version": {
         "_image_version": "8.0.0.0-svcs-ctrl",
         "_supported_image_version": [
          "8.0.0.0-svcs-ctrl"
      ]
   },
   "_switch_role": "master",
   "_hostname": "User-VM1",
   "_model": "ArubaMM",
   "_user_info": {
      "_role": "root",
      "_name": "admin"
```

```
    },
    "_clock": {
       "clock_set_timezone": {
          "minutes": 0,
          "name": "PST",
          "hours": -8
       },
       "clock_set_summer_time": null
       }
    },
    "_local": {
       "_type": "group",
       "_feature": {
          "airgroup": "Enabled",
          "firewall_visibility": "Disabled",
          "firewall_dpi": "Disabled",
          "mobility_manager": "Disabled",
          "uplink": "Disabled",
          "stat_update": "Disabled",
          "service_termination": "Disabled"
       },
       "_hardware": {
          "_capabilities": "scm,ipv6_np,wlan,fastethernet,cluster,iapmgr,loopback,dds,intf_
          bwm,extifmgr,dns_np,hcm,interface_non_profile_based,platform_lcd_cli,ap-
          cdump,airgroup,tunnel,pan_gp,routing_non_profile_based,auth_survivability,openflow_
          agent,ip_flow_export,layer2_non_profile_based,jumbo_frames,infra_np,web_cc,intf_
          sched,out_of_band_mgmt,l2l3-future,layer2_profile_based,traceoptions_profile_
          based,qos_profile_based,stacking,stateless-acl,routing_profile_based,l3auth,interface_
          profile_based,l2l3,seamless_logon,lsm,airgroup_app,openflow_controller,nbapi_non_pro-
          file_based,ssh,license_profile_based,sc_mon",
          "_model": "",
          "_name": "",
          "_mac": "",
          "_port_info": {
          "_num_ports": 4
       }
    },
    "_pending": {
       "write_mem_reqd": false,
       "last_save_time": "--",
       "last_save_user": "--",
       "last_config_time": "--",
       "last_config_user": "--"
       },
       "_user_info": {
          "_permission": "read-write"
       }
    }
}
```


Running Show Commands

Any **show** command can be run using the API model and the JSON response will be available for the same.

The following is a sample request for **show** command:

```
curl -b "aruba-cookie" -X GET -i 'https://<controller-
ip>:4343/v1/configuration/showcommand?command=show+local-userdb&UIDARUBA=<session-id>'
```

The **--insecure** option can be used with the curl command if the certificate of the Mobility Master cannot be validated.

The following is the response for the above request:

```
{
    "User Summary": [
      {
        "E-Mail": "test@aruba.com",
        "Enabled": "Yes",
        "Expiry": null,
        "Grantor-Name": "admin",
        "Name": "test",
        "Password": "aruba123",
        "Remote-IP": "0.0.0.0",
        "Role": "logon",
        "Sponsor-Name": null,
        "Status": "Active"
      }
    ],
    "_data": [
       "User Entries: 1"
     ],
     "_meta": [
      "Name",
      "Password",
      "Role",
      "E-Mail",
      "Enabled",
      "Expiry",
      "Status",
      "Sponsor-Name",
      "Remote-IP",
      "Grantor-Name"
    ]
}
```


Getting Full Configuration of a Node

Full configuration of a particular configuration node can be queried using APIs. The following is an example:

```
curl -b "aruba-cookie" -X GET -i 'https://<controller-
ip>:4343/v1/configuration/object/config?config_path=<config-
node>&type=<typ>&UIDARUBA=<session-id>'
```

The following table shows the values of the <typ> parameter:

Table 14: *Values of the <typ> Parameter*

Value	Description
pending	Shows the pending configuration at a node.
committed	Shows the committed configuration (including inherited configuration at a node)
local	Shows the configuration done only at this node (no inheritance). Also, includes pending configuration at this node, if any
committed,local	Shows committed configuration only at this node only (no inheritance)

The caveat to the **pending** configuration being shown is that, if the user deletes any configuration which is **pending**, it is not seen in this API call. Only added or modified configurations are seen. For seeing deleted configuration, you have to rely on the **show configuration pending** command.

Action Objects

Action commands are SET requests for which there is no get. The only way to see the configuration is using **show** commands. The **show** command output is made compatible with JSON.

Handling Write Memory

The following is a sample write memory request:

```
curl -b "aruba-cookie" -X POST -i 'https://<controller-ip>:4343/v1/configuration/object/write_memory?config_path=<config-node>&UIDARUBA=<session-id>' -d "{}"
```

The following is the response for the above request:

```
{
    "write_memory": {
       "_result": {
          "status": 0,
          "status_str": "Command executed."
        }
      },
      "_global_result": {
        "status": 0,
        "status_str": "Success"
    }
}
```

The write memory API request should be called via it's own message and no other SET requests for any other payload should be present in this payload. This is because processing of this object can take time.

Generic Action Object

The following is a sample generic action object request:

```
curl -b "aruba-cookie" -X POST -i 'https://<controller-ip>:4343/v1/configuration/object?config_path=<config-node>&UIDARUBA=<session-id>' -d @<set-payload-file>
```

The following is an example of the payload file:

```
{
    "local_userdb_add": {
      "_action":"modify",
      "user-role":"logon",
      "name":"test",
      "passwd":"aruba123",
      "user-email":"test@aruba.com"
    }
}
```

The following is the response for the above request:

```
{
"local_userdb_add": {
   "_action": "modify",
   "user-role": "logon",
   "name": "test",
   "passwd": "aruba123",
   "user-email": "test@aruba.com",
   "_result": {
      "status": 0,
      "status_str": "Command executed."
    }
  },
  "_global_result": {
```

```
    "status": 0,
    "status_str": "Success"
  }
}
```

Overview

The NBAPI is part of Analytics and Location Engine (ALE) solution that is integrated with Mobility Master.

The Analytics and Location Engine supports two types of APIs: a polling-based REST API, and a publish/subscribe API based on Google Protobuf and ZeroMQ.

Types of Context/Location APIs

Polling APIs

The Representational State Transfer (REST) polling-based API supports HTTPS GET operations by providing a specific URL for each query. Outputs are displayed in JSON format. The following Polling APIs are supported by ALE in the Mobility Master:

- Access Point API
- Application API
- Building API
- Campus API
- Controller API
- Cluster Info API
- Destination API
- Floor API
- GeoFence API
- Location API
- Presence API
- Proximity API
- Station API
- System Information API
- Topology API
- Virtual Access Point API
- WebCC Category API

For more information on Polling APIs, refer to latest *Analytics and Location Engine API Guide*.

Starting from ArubaOS 8.0.1.0, authentication and authorization is enabled for REST API. For more information, see Login on page 9.

The following example shows the usage of Polling API after login:

```
curl --insecure -b "aruba-cookie" -i "https://<ip address>/api/v1/context/application"
```

Publish/Subscribe APIs

The publish/subscribe API is based on the ØMQ transport. A subscriber uses ØMQ client libraries to connect to ALE and receive information from ALE asynchronously. This information is delivered in the Google Protobuf format. The following publish/subscribe APIs are supported by ALE in the Mobility Master:

- Access Point API
- Access Point State API
- Air Monitor Info API
- Application API
- Building API
- Campus API
- Client URL API
- Controller Info API
- Destination API
- Floor API
- Geofence Notify API
- Location API
- Modem Statistics API
- Presence API
- Proximity API
- Radio API
- Radio Statistics API
- Radio Utilization/Histogram Statistics API
- Station RSSI API
- Security API
- Station API
- Station Statistics API
- State Station API
- WebCC API
- Rogue Info API
- Spectrum Info API
- Uplink Bandwidth API
- Uplink Info API
- Uplink Statistics API
- Uplink WAN Compression API
- Uplink IP Probe Statistics API
- Virtual Access Point (VAP) API
- VAP statistics API
- Visibility Record API

For more information on publish/subscribe APIs, refer to latest *Analytics and Location Engine API Guide*.

NBAPI Helper Process

The NBAPI helper acts as a proxy to collect the feeds of five ALE servers and presents those five feeds as a single feed from the Mobility Master. The advantage of using NBAPI helper is that it creates a single feed to subscribe to, for devices managed by a Mobility Master. However, the location API information will not be available in the NBAPI helper because the quantity of location API data is very high and cannot support five ALEs worth of information.

You can set up ALE by following the steps mentioned below:

1. Add ALE as a management server on the Mobility Master.
2. Turn on the device location.
3. Enter the Mobility Master credentials in to the ALE so that the ALE can perform CLI tasks every five minutes, to verify user counts and other information.
4. Configure the NBAPI helper process to allow the aggregation of non-location API information directly from the Mobility Master.

Configuration

The following command is used to configure and manage NBAPI helper on Mobility Master:

```
ale-configuration
   ale_sta_associated
   anonymize
   ip <ip-addr> username <uname> password <passwd>
   nbapi_publish
```

The following command is used to configure an ALE IP address with login information. A maximum of five ALE IP addresses can be configured on the Mobility Master:

```
(host) [mynode] (config) #ale-configuration
(host) [mynode] (config-submode) # ip <IP address> username <username> password <password>
```

The following command is used to configure anonymization on the Mobility Master REST API:

```
(host) [mynode] (config) #ale-configuration
(host) [mynode] (config-submode) #anonymize
```

The following command is used to enable REST APIs on the Mobility Master to publish data available via ZMQ, including station, virtual AP, AP, radio, RSSI, visibility record, destination. By default, this parameter is false.

```
(host) [mynode] (config) #ale-configuration
(host) [mynode] (config-submode) #nbapi_publish
```

VRRP Support

The NBAPI helper process supports VRRP. The configuration is synced across Mobility Master and managed devices. VRRP has to be configured only in the Mobility Master hierarchy.