



Examples

This chapter contains the following sections:

- [Managing Firmware, page 1](#)
- [Managing Platform Tasks, page 13](#)
- [Managing Server Tasks, page 17](#)
- [Managing Users and Groups, page 51](#)

Managing Firmware

Overview

The examples in this category consist of various firmware management tasks on Cisco IMC Supervisor. These include firmware image management in network locations, downloading them from cisco.com and also triggering a firmware upgrade operation on servers.

Creating a Firmware Network Image

Objective

Create a firmware image in a network location.

Prerequisites

The HUU Image must be available in a network location - NFS/CIFS/HTTP.

REST URL

```
/cloupia/api-v2/CreateNetworkImage
```

Components

The parameters of the NETWORK_IMAGE_CREATE API are:

- String profileName—The unique name of the profile.
- String platform—The name of the platform.
- String networkServerType—Network File System (NFS), Common Internet File System (CIFS) or HTTP/S server types.
- String locationLink—A valid HTTP/HTTPS URL link for the image location.
- String networkPath—The network path.
- String sharePath—The network share path.
- String remoteFileName—A remote filename.
- String nwPathUserName—Optional. The network path user name.
- String nwPathPassword—Optional. The network path password.
- String mountOptions—Optional. The valid mount options.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>NETWORK_IMAGE_CREATE</operationType>
  <payload>
    <![CDATA[
      <CreateNetworkImage>
        <profileName></profileName>

        <platform></platform>

        <networkServerType>NFS</networkServerType>

        <!-- Set this value only when networkServerType equals to HTTP -->
        <locationLink></locationLink>

        <!-- Set this value only when networkServerType not equals to HTTP -->
        <networkPath></networkPath>

        <!-- Set this value only when networkServerType not equals to HTTP -->
        <sharePath></sharePath>

        <!-- Set this value only when networkServerType not equals to HTTP -->
        <remoteFileName></remoteFileName>

        <nwPathUserName></nwPathUserName>

        <nwPathPassword></nwPathPassword>

        <!-- Set this value only when networkServerType equals to CIFS -->
        <mountOptions></mountOptions>

      </CreateNetworkImage>

    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

Profile Name is mandatory and must be unique. Platform, Server Type (NFS/CIFS/HTTP) is mandatory. Remote IP, Remote Share, Remote Filename are mandatory in case of NFS/CIFS. The HTTP Location must be reachable from the system.

See Also

[Updating Firmware Network Image](#), on page 3

[Deleting Firmware Image Profile](#), on page 8

Updating Firmware Network Image

Objective

Update a firmware image in a network location.

Prerequisites

The HUU Image must be available in a network location - NFS/CIFS/HTTP.

REST URL

```
/cloudpia/api-v2/UpdateNetworkImage
```

Components

The parameters of the NETWORK_IMAGE_UPDATE API are:

- String `imageId`—The unique ID of the image.
- boolean `platform`—The platform that manages a server.
- String `networkServerType`—Network File System (NFS), Common Internet File System (CIFS) or HTTP/S server types.
- String `locationLink`—A valid HTTP/HTTPS URL link for the image location.
- String `networkPath`—The network path.
- String `sharePath`—The network share path.
- String `remoteFileName`—A remote filename.
- String `nwPathUserName`—Optional. The network path user name.
- String `nwPathPasswprd`—Optional. The network path password.
- String `mountOptions`—Optional. The valid mount options.

Sample Input XML

```

<cuicOperationRequest>
  <operationType>NETWORK_IMAGE_UPDATE</operationType>
  <payload>
    <![CDATA[
      <UpdateNetworkImage>
        <imageId></imageId>

        <platform></platform>

        <networkServerType>NFS</networkServerType>

        <!-- Set this value only when networkServerType equals to HTTP -->
        <locationLink></locationLink>

        <!-- Set this value only when networkServerType not equals to HTTP -->
        <networkPath></networkPath>

        <!-- Set this value only when networkServerType not equals to HTTP -->
        <sharePath></sharePath>

        <!-- Set this value only when networkServerType not equals to HTTP -->
        <remoteFileName></remoteFileName>

        <nwPathUserName></nwPathUserName>

        <nwPathPassword></nwPathPassword>

        <!-- Set this value only when networkServerType equals to CIFS -->
        <mountOptions></mountOptions>

      </UpdateNetworkImage>
    ]]>
  </payload>
</cuicOperationRequest>

```

Implementation

Profile Name cannot be modified. Platform, Server Type (NFS/CIFS/HTTP) are mandatory. Remote IP, Remote Share, Remote Filename are mandatory in case of NFS/CIFS. The HTTP Location must be reachable from the system.

See Also

[Creating a Firmware Network Image, on page 1](#)

[Deleting Firmware Image Profile, on page 8](#)

Finding Firmware Image

Objective

Find a firmware image on cisco.com.

Prerequisites

The user must have a valid set of credentials to login to cisco.com and have access privileges for HUU ISO images.

REST URL

```
/cloupia/api-v2/FindFirmwareImage
```

Components

The parameters of the LOCAL_IMAGE_FIND API are:

- String platform—The name of the platform.
- String username—ISO share login user name.
- String password—ISO share login password.
- boolean enableProxy—Optional. Enable proxy configuration.
- String host—The host name for the proxy configuration.
- String port—Port for the proxy configuration.
- boolean enableProxyAuth—Optional. Enable proxy authentication.
- String proxyAuthUserName—Proxy username for the proxy authentication.
- String proxyAuthPassword—Password for the proxy username.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>LOCAL_IMAGE_FIND</operationType>
  <payload>
    <![CDATA[
      <FindFirmwareImage>
        <platform></platform>

        <username></username>

        <password></password>

        <enableProxy>>false</enableProxy>

        <!-- Set this value only when enableProxy equals to true -->
        <host></host>

        <!-- Set this value only when enableProxy equals to true -->
        <port>0</port>

        <!-- Set this value only when enableProxy equals to true -->
        <enableProxyAuth>>false</enableProxyAuth>

        <!-- Set this value only when enableProxyAuth equals to true -->
        <proxyAuthUserName></proxyAuthUserName>

        <!-- Set this value only when enableProxyAuth equals to true -->
        <proxyAuthPassword></proxyAuthPassword>

      </FindFirmwareImage>

    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

Username/Password for cisco.com and platform are mandatory. The platform of a server that is already added into the system.

See Also

[Creating a Firmware Local Image, on page 6](#)

Creating a Firmware Local Image

Objective

Create a firmware image in a local location inside the appliance.

Prerequisites

The user must have a valid set of credentials to login to cisco.com and have access privileges for HUU ISO images. The HUU Image must be downloadable from cisco.com, and must be found using the FindFirmwareImage API.

REST URL

```
/cloupia/api-v2/CreateLocalImage
```

Components

The parameters of the LOCAL_IMAGE_CREATE API are:

- String profileName—The unique name of the profile.
- String platform—The name of the platform.
- String username—ISO share login user name.
- String password—ISO share login password.
- String availableImage—The available .iso image.
- boolean enableProxy—Optional. Enable proxy configuration.
- String host—The host name for the proxy configuration.
- String port—Port for the proxy configuration.
- boolean enableProxyAuth—Optional. Enable proxy authentication.
- String proxyAuthUserName—Proxy username for the proxy authentication.
- String proxyAuthPassword—Password for the proxy username.
- boolean acceptLicense—Accept license agreement.
- boolean downloadNow—download the .iso image immediately after adding a profile.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>LOCAL_IMAGE_CREATE</operationType>
  <payload>
    <![CDATA[
      <CreateLocalImage>
        <profileName></profileName>

        <platform></platform>

        <username></username>

        <password></password>

        <availableImage></availableImage>

        <enableProxy>>false</enableProxy>

        <!-- Set this value only when enableProxy equals to true -->
        <host></host>

        <!-- Set this value only when enableProxy equals to true -->
        <port>0</port>

        <!-- Set this value only when enableProxy equals to true -->
        <enableProxyAuth>>false</enableProxyAuth>

        <!-- Set this value only when enableProxyAuth equals to true -->
        <proxyAuthUserName></proxyAuthUserName>

        <!-- Set this value only when enableProxyAuth equals to true -->
        <proxyAuthPassword></proxyAuthPassword>

        <acceptLicense>>false</acceptLicense>

        <downloadNow>>false</downloadNow>

      </CreateLocalImage>

    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

Profile Name is mandatory, must be unique. Username/Password for cisco.com and Platform are mandatory. The Platform must be that of a server already added into the system.

See Also

[Finding Firmware Image, on page 4](#)

Downloading Firmware Local Image

Objective

Download an image from cisco.com for an already configured firmware image profile, into a local location inside the appliance.

Prerequisites

The firmware image profile must be already configured.

REST URL

```
/clouppia/api-v2/DownloadLocalImage
```

Components

The parameter of the LOCAL_IMAGE_DOWNLOAD API is:

- String `profileName`—The unique name of the profile.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>LOCAL_IMAGE_DOWNLOAD</operationType>
  <payload>
    <![CDATA[
      <DownloadLocalImage>
        <profileName></profileName>
      </DownloadLocalImage>
    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

Profile Name is mandatory, must be a valid existing profile for a Local Image. The image should not be already downloading.

See Also

[Creating a Firmware Local Image, on page 6](#)

[Deleting Firmware Image Profile, on page 8](#)

Deleting Firmware Image Profile

Objective

Delete one or more existing firmware image profiles.

Prerequisites

None

REST URL

```
/clouppia/api-v2/CIMCFirmwareUpgradeConfig
```


Components

The parameters of the FIRMWARE_IMAGE_DELETE API are:

- String profileId—The unique ID of the profile.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>FIRMWARE_IMAGE_DELETE</operationType>
  <payload>
    <![CDATA[
      <DeleteFirmwareImage>
        <profileId></profileId>

      </DeleteFirmwareImage>

    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

Profile name is mandatory and must be unique. IP address search criteria is mandatory, but CSV File option is not supported through API.

See Also

[Creating a Firmware Local Image, on page 6](#)

[Creating a Firmware Network Image, on page 1](#)

[Updating Firmware Network Image, on page 3](#)

Running Firmware Upgrade

Objective

Run a firmware upgrade on one or more servers using an already configured firmware image profile.

Prerequisites

The firmware image profile must be already configured and must contain a valid HUU ISO Image.

REST URL

```
/cloupia/api-v2/UpgradeFirmWareConfig
```

Components

The parameters of the RUN_FIRMWARE_UPGRADE API are:

- String `profileName`—The unique name of the profile.
- String `servers`—Servers whose platform matches the one configured in the selected profile.
- boolean `enableSchedule`—Enable a schedule
- String `associatedScheduleName`—Name of the associate schedule.

Sample Input XML

```
<cuicOperationRequest>
<operationType>RUN_FIRMWARE_UPGRADE</operationType>
<payload>
<![CDATA[
<UpgradeFirmWareConfig>
<profileName></profileName>

<servers></servers>

<enableSchedule>>false</enableSchedule>

  <!-- Set this value only when enableSchedule not equals to false -->
<associatedScheduleName></associatedScheduleName>

</UpgradeFirmWareConfig>

]]>
</payload>
</cuicOperationRequest>
```

Implementation

Profile name is mandatory, must be a valid existing profile. For a local profile, the image should not be already downloading. The `serverIdKey` must consist of a comma-separated list of Id's. Each Id is of the format: `{AccountName};{ServerIPAddress}`. In case of schedule option, a valid schedule name must be provided.

See Also

[Reading Firmware Upgrade Status by Profile Name, on page 12](#)

[Reading Firmware Upgrade Status by IP Address, on page 13](#)

Reading Firmware Image by a Profile Name

Objective

Get Firmware Image By Profile Name

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCFirmwareUpgradeConfig/{CIMCFirmwareUpgradeConfigId}
```

Implementation

This task allows the user to query the firmware image details based on the profile name. The CIMCFirmwareUpgradeConfigId argument must be a valid profile name. If no argument is specified, all firmware images configured in the system will be returned.

See Also

[Reading Firmware Image by Platform, on page 11](#)

[Reading Firmware Image by Type, on page 11](#)

Reading Firmware Image by Type

Objective

Get firmware image by type.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCFirmwareImageByType/{CIMCFirmwareImageById}
```

Implementation

This task allows the user to query the firmware image details based on the type of location - NETWORK or LOCAL. The CIMCFirmwareImageById argument must be one of these values - NETWORK or LOCAL. If no argument is specified, all firmware images configured in the system will be returned.

See Also

[Reading Firmware Image by Platform, on page 11](#)

[Reading Firmware Image by a Profile Name, on page 10](#)

Reading Firmware Image by Platform

Objective

Get firmware image by platform.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCFirmwareImageByPlatform/{CIMCFirmwareImageByPlatformId}
```

Implementation

This task allows the user to query the firmware image details based on the platform. The CIMCFirmwareImageByPlatformId argument must be a valid platform name. If no argument is specified, all firmware images configured in the system will be returned.

See Also

[Reading Firmware Image by a Profile Name, on page 10](#)

[Reading Firmware Image by Type, on page 11](#)

Reading Download Status by Profile Name

Objective

Image download status by profile name.

Prerequisites

None

REST URL

```
/cloupia/api-v2/LocalImageDownloadStatusByProfileName/{LocalImageDownloadStatusByProfileNameId}
```

Implementation

This task allows the user to query the download status of a local firmware image based on the profile name. The LocalImageDownloadStatusByProfileNameId argument must be a valid profile name. If no argument is specified, an empty set of results will be returned.

See Also

[Downloading Firmware Local Image, on page 7](#)

Reading Firmware Upgrade Status by Profile Name

Objective

Firmware upgrade status by profile name.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCFirmwareUpgradeStatusbyProfileName/{CIMCFirmwareUpgradeStatusbyProfileNameId}
```

Implementation

This task allows the user to query the firmware upgrade status of one or more servers based on the profile name of the image. The CIMCFirmwareUpgradeStatusbyProfileNameId argument must be a valid profile name. If no argument is specified, all firmware upgrade operations' status will be returned.

See Also

[Running Firmware Upgrade, on page 9](#)

[Reading Firmware Upgrade Status by IP Address, on page 13](#)

Reading Firmware Upgrade Status by IP Address

Objective

Firmware upgrade status by server IP address.

Prerequisites

None

REST URL

```
>/cloupia/api-v2/CIMCFirmwareUpgradeStatusbyServerIP/{CIMCFirmwareUpgradeStatusbyServerIPId}
```

Implementation

This task allows the user to query the firmware upgrade status of one or more servers based on the profile name of the image. The CIMCFirmwareUpgradeStatusbyProfileNameId argument must be a valid profile name. If no argument is specified, all firmware upgrade operations' status will be returned. The dots in the IP address need to be substituted with an underscore.

See Also

[Running Firmware Upgrade, on page 9](#)

[Reading Firmware Upgrade Status by Profile Name, on page 12](#)

Managing Platform Tasks

Overview

The examples in this category consists of managing email alert rules on Cisco IMC Supervisor.

Creating an Email Alert Rule

Objective

Create an email alert rule for notification of faults.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCEmailAlertRuleConfig
```

Components

The parameters of the EMAIL_ALERT_RULE_CREATE API are:

- String name—The name for the email alert.
- String alertLevel—The alert level.
- String serverGroups—Optional. The server groups to which email alerts are sent.
- String emailAddress—The email address of the intended recipients of the email alert.
- String severity—Fault severity levels for which email alerts will be sent.
- Boolean enabled—Optional. Enable email alerts to the configured email address.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>EMAIL_ALERT_RULE_CREATE</operationType>
  <payload>
    <![CDATA[
      <CIMCEmailAlertRuleConfig>
        <name></name>

        <alertLevel>SYSTEM</alertLevel>

        <!-- Set this value only when alertLevel not equals to SYSTEM -->
        <serverGroups></serverGroups>

        <emailAddress></emailAddress>

        <severity>critical</severity>

        <enabled>false</enabled>

      </CIMCEmailAlertRuleConfig>

    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

Rule name is mandatory and must be unique. Email addresses are mandatory.

See Also

[Reading an Email Alert Rule](#)
[Updating an Email Alert Rule](#)
[Deleting Email Alert Rules](#)

Reading an Email Alert Rule

Objective

Get details of email alert rules.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCEmailAlertRuleConfig/{CIMCEmailAlertRuleConfigId}
```

Implementation

The Id argument must be a valid Rule name. If no argument is specified, all email alert rules configured in the system will be returned.

See Also

[Creating an Email Alert Rule](#)
[Updating an Email Alert Rule](#)
[Deleting Email Alert Rules](#)

Updating an Email Alert Rule

Objective

Update an existing email alert rule.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCEmailAlertRuleConfig
```

Components

The parameters of the EMAIL_ALERT_RULE_UPDATE API are:

- String emailAlertRule—The email alert rule.
- String alertLevel—The alert level.
- String serverGroups—Optional. The server groups to which email alerts are sent.
- String emailAddress—The email used to notify the group owner about the status of service requests and request approvals if necessary.
- String severity—Fault severity levels for which email alerts will be sent.
- Boolean enabled—Optional. Enable email alerts to the configured email address.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>EMAIL_ALERT_RULE_UPDATE</operationType>
  <payload>
    <![CDATA[
      <ModifyEmailAlertRuleConfig>
        <emailAlertRule></emailAlertRule>

        <alertLevel>SYSTEM</alertLevel>

        <!-- Set this value only when alertLevel not equals to SYSTEM -->
        <serverGroups></serverGroups>

        <emailAddress></emailAddress>

        <severity></severity>

        <enabled>false</enabled>

      </ModifyEmailAlertRuleConfig>

    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

Rule name cannot be modified.

See Also

- [Reading an Email Alert Rule](#)
- [Creating an Email Alert Rule](#)
- [Deleting Email Alert Rules](#)

Deleting Email Alert Rules

Objective

Delete one or more existing Email Alert Rules.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCEmailAlertRuleConfig
```

Components

String emailAlertRule—The email alert rule.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>EMAIL_ALERT_RULE_DELETE</operationType>
  <payload>
    <![CDATA[
      <DeleteEmailAlertRuleConfig>
        <emailAlertRule></emailAlertRule>
      </DeleteEmailAlertRuleConfig>
    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

Comma separated list of rule names, all of which must be of valid existing rules.

See Also

[Reading an Email Alert Rule](#)

[Creating an Email Alert Rule](#)

[Updating an Email Alert Rule](#)

Managing Server Tasks

Overview

The examples in this category consist of various server management tasks, such as discovery of servers through IP addresses, importing of discovered servers, power actions on servers and various methods to query server data, inventory data, and fault data.

Creating a Rack Group

Objective

Create a rack group to group servers logically in Cisco IMC Supervisor.

Prerequisites

None

REST URL

/cloupia/api-v2/CIMCRackGroup

Components

The parameters of the RACK_GROUP_CREATE API are:

- String groupName—The name of the group or the customer organization.
- String groupDescription—Optional. The description of the group or the customer organization, if required.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>RACK_GROUP_CREATE</operationType>
  <payload>
    <![CDATA[
      <CIMCRackGroup>
        <groupName></groupName>

        <description></description>
      </CIMCRackGroup>
    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

Group Name is mandatory and must be unique.

See Also

[Reading All Rack Groups, on page 18](#)

[Updating a Rack Group, on page 19](#)

[Deleting a Rack Group, on page 20](#)

Reading All Rack Groups

Objective

Get rack group details.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCRackGroup/{CIMCRackGroupId}
```

Components

None

Sample Input XML

```
<cuicOperationResponse><cuicOperationStatus>0</cuicOperationStatus>
<response><CIMCRackGroup><actionId>0</actionId><configEntryId>0</configEntryId>
<defaultGroup>true</defaultGroup><description>Default provided rack group
</description><groupName>Default Group</groupName></CIMCRackGroup><CIMCRackGroup>
<actionId>0</actionId><configEntryId>0</configEntryId><defaultGroup>>false
</defaultGroup><description>Test55</description><groupName>Test66</groupName>
</CIMCRackGroup><CIMCRackGroup><actionId>0</actionId><configEntryId>0
</configEntryId><defaultGroup>>false</defaultGroup><description>apitest
</description><groupName>apitest-ren</groupName></CIMCRackGroup><CIMCRackGroup>
<actionId>0</actionId><configEntryId>0</configEntryId><defaultGroup>>false
</defaultGroup><description></description><groupName>Test3-SumanthRen</groupName>
</CIMCRackGroup></response></cuicOperationResponse>
```

Implementation

The Id argument must be a valid Rack Group name. If no argument is specified, all Rack Groups configured in the system will be returned.

See Also

[Creating a Rack Group](#), on page 17

[Updating a Rack Group](#), on page 19

[Deleting a Rack Group](#), on page 20

Updating a Rack Group

Objective

Update an existing Rack Group.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCRackGroup
```

Components

The parameters of the RACK_GROUP_UPDATE API are:

- String groupName—The name of the group or the customer organization.
- String groupDescription—Optional. The description of the group or the customer organization, if required.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>RACK_GROUP_UPDATE</operationType>
  <payload>
    <![CDATA[
      <ModifyRackGroup>
        <groupID></groupID>

        <groupName></groupName>

        <description></description>
      </ModifyRackGroup>
    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

Group name is mandatory and must be unique.

See Also

[Creating a Rack Group, on page 17](#)

[Reading All Rack Groups, on page 18](#)

[Deleting a Rack Group, on page 20](#)

Deleting a Rack Group

Objective

Delete one or more existing rack groups.

Prerequisites

None

REST URL

/cloupia/api-v2/CIMCRackGroup

Components

The parameters of the RACK_GROUP_DELETE API are:

- String groupName—The name of the group or the customer organization.
- String groupDescription—Optional. The description of the group or the customer organization, if required.

Sample Input XML

```
<cuicOperationRequest>
<operationType>RACK_GROUP_DELETE</operationType>
<payload>
<![CDATA[
<DeleteRackGroup>
<groupID></groupID>

<forceDelete>>false</forceDelete>

</DeleteRackGroup>

]]>
</payload>
</cuicOperationRequest>
```

Implementation

Comma separated list of group names, all of which must be of valid existing rack groups.

See Also

[Creating a Rack Group, on page 17](#)

[Reading All Rack Groups, on page 18](#)

[Updating a Rack Group, on page 19](#)

Creating a Discovery Profile

Objective

Create a discovery profile to use for discovering servers based on IP address and importing them.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCDeviceDiscoveryConfig
```

Components

The parameters of the DISCOVERY_PROFILE_CREATE API are:

- String `profileName`—The name of the profile.
- boolean `isRange`—Optional. The range
- String `option`—The option.
- String `ipList`—List of IP addresses.
- String `startRange`—Valid beginning IP address.
- String `endRange`—Valid last IP address.
- String `networkAddress`—The network IP address.
- String `subnetMask`—The range of subnet mask.
- String `csvFile`—Search by csv file.
- boolean `credentialPolicy`—Optional. Create a credential policy.
- String `policy`—Optional. The policy name.
- String `username`—The server login name.
- String `password`—The server login password.
- String `protocol`—Optional. HTTP or HTTPS protocol.
- int `port`—The port number.

Sample Input XML

```

<cuicOperationRequest>
  <operationType>DISCOVERY_PROFILE_CREATE</operationType>
  <payload>
    <![CDATA[
      <CIMCDeviceDiscoveryConfig>
        <profileName></profileName>

        <option>IP</option>

        <!-- Set this value only when option equals to IPLIST -->
        <ipList></ipList>

        <!-- Set this value only when option equals to IP -->
        <startRange></startRange>

        <!-- Set this value only when option equals to IP -->
        <endRange></endRange>

        <!-- Set this value only when option equals to SUBNET -->
        <networkAddress></networkAddress>

        <!-- Set this value only when option equals to SUBNET -->
        <subnetMask></subnetMask>

        <!-- Set this value only when option equals to CSV -->
        <csvFile></csvFile>

        <credentialPolicy>>false</credentialPolicy>

        <!-- Set this value only when credentialPolicy not equals to false -->
        <policy></policy>

        <!-- Set this value only when credentialPolicy not equals to true -->
        <username></username>

        <!-- Set this value only when credentialPolicy not equals to true -->
        <password></password>

        <!-- Set this value only when credentialPolicy not equals to true -->
        <protocol>https</protocol>

        <!-- Set this value only when credentialPolicy not equals to true -->
        <port>443</port>

      </CIMCDeviceDiscoveryConfig>
    ]]>
  </payload>
</cuicOperationRequest>

```

Implementation

Profile Name is mandatory, must be unique. IP Address Search Criteria is mandatory, but CSV File option is not supported via API.

See Also

[Updating a Discovery Profile, on page 24](#)

[Deleting a Discovery Profile, on page 27](#)

Reading a Discovery Profile

Objective

Get discovery profiles details.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCDeviceDiscoveryConfig/{CIMCDeviceDiscoveryConfigId}
```

Implementation

The Id argument must be a valid profile name. If no argument is specified, all discovery profiles configured in the system will be returned.

See Also

[Creating a Discovery Profile, on page 21](#)

[Updating a Discovery Profile, on page 24](#)

[Deleting a Discovery Profile, on page 27](#)

Updating a Discovery Profile

Objective

Update an existing discovery profile.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCDeviceDiscoveryConfig
```


Components

The parameters of the DISCOVERY_PROFILE_UPDATE API are:

- String `profileName`—The unique name of the profile.
- String `option`—The option.
- String `ipList`—List of IP addresses.
- String `startRange`—Valid beginning IP address.
- String `endRange`—Valid last IP address.
- String `networkAddress`—The network IP address.
- String `subnetMask`—The range of subnet mask.
- String `csvFile`—Search by csv file.
- boolean `credentialPolicy`—Optional. Create a credential policy.
- boolean `policy`—Optional. The policy name.
- String `username`—The server login name.
- String `password`—The server login password.
- String `protocol`—Optional. HTTP or HTTPS protocol.
- int `port`—The port number.

Sample Input XML

```

<cuicOperationRequest>
  <operationType>DISCOVERY_PROFILE_UPDATE</operationType>
  <payload>
    <![CDATA[
      <ModifyCIMCDeviceDiscoveryProfile>
        <profileName></profileName>

        <option>IP</option>

        <!-- Set this value only when option equals to IPLIST -->
        <ipList></ipList>

        <!-- Set this value only when option equals to IP -->
        <startRange></startRange>

        <!-- Set this value only when option equals to IP -->
        <endRange></endRange>

        <!-- Set this value only when option equals to SUBNET -->
        <networkAddress></networkAddress>

        <!-- Set this value only when option equals to SUBNET -->
        <subnetMask></subnetMask>

        <!-- Set this value only when option equals to CSV -->
        <csvFile></csvFile>

        <credentialPolicy>>false</credentialPolicy>

        <!-- Set this value only when credentialPolicy not equals to false -->
        <policy></policy>

        <!-- Set this value only when credentialPolicy not equals to true -->
        <username></username>

        <!-- Set this value only when credentialPolicy not equals to true -->
        <password></password>

        <!-- Set this value only when credentialPolicy not equals to true -->
        <protocol>https</protocol>

        <!-- Set this value only when credentialPolicy not equals to true -->
        <port>443</port>

      </ModifyCIMCDeviceDiscoveryProfile>
    ]]>
  </payload>
</cuicOperationRequest>

```

Implementation

Profile Name cannot be modified.

See Also

[Creating a Discovery Profile, on page 21](#)

[Deleting a Discovery Profile, on page 27](#)

Deleting a Discovery Profile

Objective

Delete one or more existing discovery profiles.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCDeviceDiscoveryConfig
```

Components

The parameters of the DISCOVERY_PROFILE_DELETE API are:

- String `profileName`—Optional. The name of the profile.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>DISCOVERY_PROFILE_DELETE</operationType>
  <payload>
    <![CDATA[
      <DeleteCIMCDeviceDiscoveryProfile>
        <profileName></profileName>
      </DeleteCIMCDeviceDiscoveryProfile>
    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

Comma separated list of profile names, all of which must be of valid existing profiles.

See Also

[Creating a Discovery Profile](#), on page 21

[Updating a Discovery Profile](#), on page 24

[Reading a Discovery Profile](#), on page 24

Running Server Discovery

Objective

Run a Discovery operation to discovery servers based on IP addresses, using one or more configured Discovery Profiles.

Prerequisites

Discovery Profile must be configured.

REST URL

```
/clouppia/api-v2/CIMCAutoDiscoveryConfig
```

Components

The parameters of the RUN_SERVER_DISCOVERY API are:

- String profileNames—The name of the profile.
- boolean enableSchedule—Enable a schedule.
- String associatedScheduleName—Name of the associate schedule.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>RUN_SERVER_DISCOVERY</operationType>
  <payload>
    <![CDATA[
      <CIMCAutoDiscoveryConfig>
        <profileNames></profileNames>

        <enableSchedule>>false</enableSchedule>

        <!-- Set this value only when enableSchedule not equals to false -->
        <associatedScheduleName></associatedScheduleName>

      </CIMCAutoDiscoveryConfig>
    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

Comma-separated list of valid profile names. In case of schedule option, a valid schedule name must be provided.

See Also

[Importing Discovered Devices, on page 29](#)

Reading Discovered Devices

Objective

Get discovered device details.

Prerequisites

One or more servers must have been discovered using a discovery profile

REST URL

```
/cloupia/api-v2/CIMCDiscoveredDevice/{CIMCDiscoveredDeviceId}/State/{StateId}
```

Implementation

The CIMCDiscoveredDeviceId argument must be a valid profile name, and must be mandatorily specified. The StateId argument must be one of {All, Imported, NotImported}.

Importing Discovered Devices

Objective

Import one or more discovered devices.

Prerequisites

One or more servers must have been discovered using a Discovery Profile.

REST URL

```
/cloupia/api-v2/ImportRackServersConfig
```

Components

The parameters of the IMPORT_SERVER API are:

- String devices—The discovered devices.
- String userPrefix—Optional. The prefix for the user.
- String description—Optional. Description for the user.
- String contact—Optional. Contact details of the user.
- String location—Optional. Address of the user.
- String rackGroup—Create rack group.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>IMPORT_SERVER</operationType>
  <payload>
    <![CDATA[
      <ImportRackServersConfig>
        <devices></devices>

        <userPrefix></userPrefix>

        <description></description>

        <contact></contact>

        <location></location>

        <rackGroup>Default Group</rackGroup>

      </ImportRackServersConfig>
    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

Comma-separated list of one or more valid server IP addresses, which have been discovered. Group name of an existing rack group.

See Also

[Running Server Discovery, on page 27](#)

Hard Reset Server

Objective

Hard reset one or more servers.

Prerequisites

One or more Servers must be configured as Rack Accounts.

REST URL

```
/cloupia/api-v2/HardResetAction
```

Components

The parameters of the HARD_RESET_SERVER API are:

- String serverIdKey—The server Id key.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>HARD_RESET_SERVER</operationType>
  <payload>
    <![CDATA[
      <HardResetServer>
        <serverIdKey></serverIdKey>

      </HardResetServer>

    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

The serverIdKey must consist of a comma-separated list of Id's. Each Id is of the format: {AccountName};{ServerIPAddress }

See Also

- [Power Cycle Server, on page 31](#)
- [Power On Server, on page 33](#)
- [Power Off Server, on page 32](#)
- [Shutdown Server, on page 34](#)
- [Set Label on Server, on page 35](#)
- [Toggle Locator LED on Server, on page 36](#)

Power Cycle Server

Objective

Power cycle one or more servers.

Prerequisites

One or more servers must be configured as rack accounts.

REST URL

```
/cloupia/api-v2/PowerCycleAction
```

Components

The parameters of the POWER_CYCLE_SERVER API are:

- String serverIdKey—The server Id key.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>POWER_CYCLE_SERVER</operationType>
  <payload>
    <![CDATA[
      <PowerCycleServer>
        <serverIdKey></serverIdKey>

      </PowerCycleServer>
    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

The serverIdKey must consist of a comma-separated list of Id's. Each Id is of the format: {AccountName};{ServerIPAddress }

See Also

- [Hard Reset Server, on page 30](#)
- [Power On Server, on page 33](#)
- [Power Off Server, on page 32](#)
- [Shutdown Server, on page 34](#)
- [Set Label on Server, on page 35](#)
- [Toggle Locator LED on Server, on page 36](#)

Power Off Server

Objective

Power Off one or more Servers.

Prerequisites

One or more Servers must be configured as Rack Accounts

REST URL

```
/cloupia/api-v2/PowerOffAction
```

Components

The parameters of the POWER_OFF_SERVER API are:

- String serverIdKey—The server Id key.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>POWER_OFF_SERVER</operationType>
  <payload>
    <![CDATA[
      <PowerOffServer>
        <serverIdKey></serverIdKey>

      </PowerOffServer>

    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

The serverIdKey must consist of a comma-separated list of Id's. Each Id is of the format: {AccountName};{ServerIPAddress}

See Also

- [Hard Reset Server, on page 30](#)
- [Power Cycle Server, on page 31](#)
- [Power On Server, on page 33](#)
- [Shutdown Server, on page 34](#)
- [Set Label on Server, on page 35](#)
- [Toggle Locator LED on Server, on page 36](#)

Power On Server

Objective

Power On server.

Context

Power On one or more servers.

Prerequisites

One or more servers must be configured as rack accounts.

REST URL

```
/cloupia/api-v2/PowerOnAction
```

Components

The parameters of the POWER_ON_SERVER API are:

- String serverIdKey—The server Id key.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>POWER_ON_SERVER</operationType>
  <payload>
    <![CDATA[
      <PowerOnServer>
        <serverIdKey></serverIdKey>

      </PowerOnServer>

    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

The serverIdKey must consist of a comma-separated list of Id's. Each Id is of the format: {AccountName};{ServerIPAddress}.

See Also

- [Hard Reset Server, on page 30](#)
- [Power Cycle Server, on page 31](#)
- [Power Off Server, on page 32](#)
- [Shutdown Server, on page 34](#)
- [Set Label on Server, on page 35](#)
- [Toggle Locator LED on Server, on page 36](#)

Shutdown Server

Objective

Shut down one or more servers.

Prerequisites

One or more Servers must be configured as Rack Accounts.

REST URL

```
/cloupia/api-v2/ShutDownAction
```

Components

The parameters of the SHUT_DOWN_SERVER API are:

- String serverIdKey—The server Id key.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>SHUT_DOWN_SERVER</operationType>
  <payload>
    <![CDATA[
      <ShutDownServer>
        <serverIdKey></serverIdKey>

      </ShutDownServer>

    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

The serverIdKey must consist of a comma-separated list of Id's. Each Id is of the format: {AccountName};{ServerIPAddress}.

See Also

- [Power Cycle Server, on page 31](#)
- [Power On Server, on page 33](#)
- [Power Off Server, on page 32](#)
- [Hard Reset Server, on page 30](#)
- [Set Label on Server, on page 35](#)
- [Toggle Locator LED on Server, on page 36](#)

Set Label on Server

Objective

Set label for one or more servers.

Prerequisites

One or more Servers must be configured as Rack Accounts.

REST URL

```
/cloupia/api-v2/SetLabelAction
```

Components

The parameters of the SET_LABEL API are:

- String serverIdKey—The server Id key.
- String setLabel—The label name.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>SET_LABEL</operationType>
  <payload>
    <![CDATA[
      <SetLabelServer>
        <serverIdKey></serverIdKey>

        <setLabel></setLabel>
      </SetLabelServer>
    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

The serverIdKey must consist of a comma-separated list of Id's. Each Id is of the format: {AccountName};{ServerIPAddress}.

See Also

- [Power Cycle Server, on page 31](#)
- [Power On Server, on page 33](#)
- [Power Off Server, on page 32](#)
- [Shutdown Server, on page 34](#)
- [Hard Reset Server, on page 30](#)
- [Toggle Locator LED on Server, on page 36](#)

Toggle Locator LED on Server

Objective

Toggle Locator LED one or more Servers.

Prerequisites

One or more Servers must be configured as Rack Accounts.

REST URL

```
/cloupia/api-v2/LocatorLedAction
```

Components

The parameters of the LOCATOR_LED API are:

- String serverIdKey—The server Id key.
- String locatorLed—The locator LED.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>LOCATOR_LED</operationType>
  <payload>
    <![CDATA[
      <LocatorLedServer>
        <serverIdKey></serverIdKey>

        <locatorLed>ON</locatorLed>
      </LocatorLedServer>
    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

The serverIdKey must consist of a comma-separated list of Id's. Each Id is of the format: {AccountName};{ServerIPAddress}.

See Also

[Power Cycle Server, on page 31](#)

[Power On Server, on page 33](#)

[Power Off Server, on page 32](#)

[Shutdown Server, on page 34](#)

[Set Label on Server, on page 35](#)

[Hard Reset Server, on page 30](#)

Reading Servers by Tag Name

Objective

Get servers which are tagged with a specific name.

Prerequisites

One or more servers must be configured as Rack Accounts and be tagged.

REST URL

```
/cloupia/api-v2/ServersByTagName/{ServersByTagNameId}
```

Implementation

The ServersByTagValueId argument must be a valid tag value defined in the Tag Library.

See Also

- [Reading Servers by Account Name, on page 45](#)
- [Reading Servers by Rack Group, on page 47](#)
- [Reading Servers by Serial Number, on page 46](#)
- [Reading Servers by Server IP, on page 46](#)
- [Reading Servers by Tag Value, on page 38](#)
- [Reading Servers by UUID, on page 45](#)
- [Reading Servers by Product ID, on page 44](#)

Reading Servers by Tag Value

Objective

Get Servers which are tagged with a specific value.

Prerequisites

One or more servers must be configured as Rack Accounts and be tagged.

REST URL

```
/cloupia/api-v2/ServersByTagValue/{ServersByTagValueId}
```

Implementation

The ServersByTagValueId argument must be a valid tag value defined in the Tag Library.

See Also

- [Reading Servers by Tag Name, on page 37](#)
- [Reading Servers by Account Name, on page 45](#)
- [Reading Servers by Rack Group, on page 47](#)
- [Reading Servers by Serial Number, on page 46](#)
- [Reading Servers by Server IP, on page 46](#)
- [Reading Servers by UUID, on page 45](#)
- [Reading Servers by Product ID, on page 44](#)

Reading Server Faults by DN

Objective

Get Server Faults by affected DN.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCFaultsByDN/{CIMCFaultsByDNId}
```

Implementation

The CIMCFaultsByDNId argument must be a valid DN value. The RNs in the DN must be separated by an underscore instead of a forward slash.

See Also

[Reading Server Faults by Account Name, on page 40](#)

[Reading Server Faults by Fault Code, on page 41](#)

[Reading Server Faults by IP Address, on page 39](#)

[Reading Server Faults by Severity, on page 40](#)

Reading Server Faults by IP Address

Objective

Get Faults of a specific server by its IP address.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCFaultsByServerIP/{CIMCFaultsByServerIPId}
```

Implementation

The CIMCFaultsByServerIPId argument must be a valid IP Address. The dots in the IP address need to be substituted with an underscore.

See Also

[Reading Server Faults by DN, on page 38](#)

[Reading Server Faults by Fault Code, on page 41](#)

[Reading Server Faults by Account Name, on page 40](#)

[Reading Server Faults by Severity, on page 40](#)

Reading Server Faults by Account Name

Objective

Get Faults of a specific server by its Account Name.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCFaultsByAccountName/{CIMCFaultsByAccountNameId}
```

Implementation

The CIMCFaultsByAccountNameId argument must be a valid Account Name of a server being managed by IMCS.

See Also

[Reading Server Faults by DN, on page 38](#)

[Reading Server Faults by Fault Code, on page 41](#)

[Reading Server Faults by IP Address, on page 39](#)

[Reading Server Faults by Severity, on page 40](#)

Reading Server Faults by Severity

Objective

Get Server Faults by Severity level.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCFaultsBySeverity/{CIMCFaultsBySeverityId}
```

Implementation

The CIMCFaultsBySeverityId argument must be a valid Severity Level.

See Also

- [Reading Server Faults by DN, on page 38](#)
- [Reading Server Faults by Fault Code, on page 41](#)
- [Reading Server Faults by IP Address, on page 39](#)
- [Reading Server Faults by Account Name, on page 40](#)

Reading Server Faults by Fault Code

Objective

Get Server Faults by Fault Code.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCFaultsByCode/{CIMCFaultsByCodeId}
```

Implementation

The CIMCFaultsByCodeId argument must be a valid Fault Code.

See Also

- [Reading Server Faults by DN, on page 38](#)
- [Reading Server Faults by Account Name, on page 40](#)
- [Reading Server Faults by IP Address, on page 39](#)
- [Reading Server Faults by Severity, on page 40](#)

Reading Server Faults History by DN

Objective

Get Server Faults by affected DN.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCFaultsHistoryByDN/{CIMCFaultsHistoryByDNId}
```

Implementation

The CIMCFaultsHistoryByDNId argument must be a valid DN value. The RNs in the DN must be separated by an underscore instead of a forward slash.

See Also

[Reading Server Faults History by Fault Code](#), on page 43

[Reading Server Faults History by IP Address](#), on page 42

[Reading Server Faults History by Severity](#), on page 43

[Reading Server Faults History by Account Name](#), on page 42

Reading Server Faults History by IP Address

Objective

Get Faults History of a specific server by its IP address.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCFaultsHistoryByServerIP/{CIMCFaultsHistoryByServerIPId}
```

Implementation

The CIMCFaultsHistoryByServerIPId argument must be a valid IP address of a server being managed by IMCS. The dots in the IP address need to be substituted with an underscore.

See Also

[Reading Server Faults History by Fault Code](#), on page 43

[Reading Server Faults History by DN](#), on page 41

[Reading Server Faults History by Severity](#), on page 43

[Reading Server Faults History by Account Name](#), on page 42

Reading Server Faults History by Account Name

Objective

Get Faults History of a specific server by its Account Name.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCFaultsHistoryByAccountName/{CIMCFaultsHistoryByAccountNameId}
```

Implementation

The CIMCFaultsHistoryByAccountNameId argument must be a valid Account Name of a server being managed by Cisco IMC Supervisor.

See Also

[Reading Server Faults History by Fault Code](#), on page 43

[Reading Server Faults History by DN](#), on page 41

[Reading Server Faults History by Severity](#), on page 43

[Reading Server Faults History by IP Address](#), on page 42

Reading Server Faults History by Severity

Objective

Get Server Faults History by Severity level.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCFaultsHistoryBySeverity/{CIMCFaultsHistoryBySeverityId}
```

Implementation

The CIMCFaultsHistoryBySeverityId argument must be a valid Severity Level.

See Also

[Reading Server Faults History by Fault Code](#), on page 43

[Reading Server Faults History by DN](#), on page 41

[Reading Server Faults History by Account Name](#), on page 42

[Reading Server Faults History by IP Address](#), on page 42

Reading Server Faults History by Fault Code

Objective

Get Server Faults History by Fault Code.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCFaultsHistoryByCode/{CIMCFaultsHistoryByCodeId}
```

Implementation

The CIMCFaultsHistoryByCodeId argument must be a valid Fault Code.

See Also

[Reading Server Faults History by Severity, on page 43](#)

[Reading Server Faults History by DN, on page 41](#)

[Reading Server Faults History by Account Name, on page 42](#)

[Reading Server Faults History by IP Address, on page 42](#)

Reading Servers by Product ID

Objective

Get Server By Product ID.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCServerByProductID/{CIMCServerByProductIDId}
```

Implementation

The CIMCServerByProductIDId argument must be a valid Product ID of a server being managed by Cisco IMC Supervisor.

See Also

[Reading Servers by Tag Name, on page 37](#)

[Reading Servers by Account Name, on page 45](#)

[Reading Servers by Rack Group, on page 47](#)

[Reading Servers by Serial Number, on page 46](#)

[Reading Servers by Server IP, on page 46](#)

[Reading Servers by UUID, on page 45](#)

[Reading Servers by Tag Value, on page 38](#)

Reading Servers by Account Name

Objective

Get Servers By Account Name

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCServerByAccountName/{CIMCServerByAccountNameId}
```

Implementation

The CIMCServerByAccountNameId argument must be a valid Account Name of a server being managed by Cisco IMC Supervisor.

See Also

[Reading Servers by Tag Name, on page 37](#)

[Reading Servers by Tag Value, on page 38](#)

[Reading Servers by Rack Group, on page 47](#)

[Reading Servers by Serial Number, on page 46](#)

[Reading Servers by Server IP, on page 46](#)

[Reading Servers by UUID, on page 45](#)

[Reading Servers by Product ID, on page 44](#)

Reading Servers by UUID

Objective

Get Server By UUID

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCServerByUUID/{CIMCServerByUUIDId}
```

Implementation

The CIMCServerByUUIDId argument must be a valid UUID of a server being managed by Cisco IMC Supervisor.

See Also

- [Reading Servers by Tag Name, on page 37](#)
- [Reading Servers by Tag Value, on page 38](#)
- [Reading Servers by Account Name, on page 45](#)
- [Reading Servers by Rack Group, on page 47](#)
- [Reading Servers by Serial Number, on page 46](#)
- [Reading Servers by Server IP, on page 46](#)
- [Reading Servers by Product ID, on page 44](#)

Reading Servers by Server IP

Objective

Get Server By IP Address.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCServerByServerIP/{CIMCServerByServerIPId}
```

Implementation

The CIMCServerByServerIPId argument must be a valid IP address of a server being managed by Cisco IMC Supervisor. The dots in the IP address need to be substituted with an underscore.

See Also

- [Reading Servers by Tag Name, on page 37](#)
- [Reading Servers by Account Name, on page 45](#)
- [Reading Servers by Rack Group, on page 47](#)
- [Reading Servers by Serial Number, on page 46](#)
- [Reading Servers by Server IP, on page 46](#)
- [Reading Servers by UUID, on page 45](#)
- [Reading Servers by Product ID, on page 44](#)

Reading Servers by Serial Number

Objective

Get Server By Serial Number.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCServerBySerialNum/{CIMCServerBySerialNumId}
```

Implementation

The CIMCServerBySerialNumId argument must be a valid serial number of a server being managed by Cisco IMC Supervisor.

See Also

[Reading Servers by Tag Name](#), on page 37

[Reading Servers by Tag Value](#), on page 38

[Reading Servers by Account Name](#), on page 45

[Reading Servers by Rack Group](#), on page 47

[Reading Servers by Server IP](#), on page 46

[Reading Servers by Product ID](#), on page 44

[Reading Servers by UUID](#), on page 45

Reading Servers by Rack Group

Objective

Get Server By Rack Group.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCServerByRackGroup/{CIMCServerByRackGroupId}
```

Implementation

The CIMCServerByRackGroupId argument must be a valid Rack Group existing in Cisco IMC Supervisor.

See Also

- [Reading Servers by Tag Name, on page 37](#)
- [Reading Servers by Tag Value, on page 38](#)
- [Reading Servers by Account Name, on page 45](#)
- [Reading Servers by Server IP, on page 46](#)
- [Reading Servers by Serial Number, on page 46](#)
- [Reading Servers by Product ID, on page 44](#)
- [Reading Servers by UUID, on page 45](#)

Reading Server Inventory by Account Name

Objective

Get Server Inventory By Account Name.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCServerInventoryByAccountName/{CIMCServerInventoryByAccountNameId}
```

Implementation

The `CIMCServerInventoryByAccountNameId` argument must be a valid Account Name of a server being managed by Cisco IMC Supervisor.

See Also

- [Reading Server Inventory by Server IP, on page 48](#)

Reading Server Inventory by Server IP

Objective

Get server inventory by IP address.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCServerInventoryByServerIP/{CIMCServerInventoryByServerIPId}
```


Implementation

The CIMCServerInventoryByServerIPId argument must be a valid IP address of a server being managed by Cisco IMC Supervisor. The dots in the IP address need to be substituted with an underscore.

See Also

[Reading Server Inventory by Account Name, on page 48](#)

Reading Server Utilization by Account Name

Objective

Get Server Utilization By Account Name

Prerequisites

None

REST URL

```
/cloudpia/api-v2/CIMCServerUtilizationByAccountName/{CIMCServerUtilizationByAccountId}
```

Implementation

The CIMCServerUtilizationByAccountNameId argument must be a valid Account Name of a server being managed by Cisco IMC Supervisor.

See Also

[Reading Server Utilization by Server IP, on page 49](#)

Reading Server Utilization by Server IP

Objective

Get Server Utilization By IP Address.

Prerequisites

None

REST URL

```
/cloudpia/api-v2/CIMCServerUtilizationByServerIP/{CIMCServerUtilizationByServerIPId}
```

Implementation

The CIMCServerUtilizationByServerIPId argument must be a valid IP address of a server being managed by Cisco IMC Supervisor. The dots in the IP address need to be substituted with an underscore.

See Also

[Reading Server Utilization by Account Name, on page 49](#)

Reading Server Utilization History by Account Name

Objective

Get Server Utilization History By Account Name.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCServerUtilizationHistoryByAccountName/{CIMCServerUtilizationHistoryByAccountNameId}
```

Implementation

The CIMCServerUtilizationHistoryByAccountNameId argument must be a valid Account Name of a server being managed by Cisco IMC Supervisor.

See Also

[Reading Server Utilization History by Server IP, on page 50](#)

Reading Server Utilization History by Server IP

Objective

Get Server Utilization History By IP Address.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCServerUtilizationHistoryByServerIP/{CIMCServerUtilizationHistoryByServerIPId}
```

Implementation

The CIMCServerUtilizationHistoryByServerIPId argument must be a valid IP address of a server being managed by Cisco IMC Supervisor. The dots in the IP address need to be substituted with an underscore.

See Also

[Reading Server Utilization History by Account Name, on page 50](#)

Managing Users and Groups

Overview

The examples in this category consists of managing users and user groups to access Cisco IMC Supervisor.

Creating a User Group

Objective

Create a group of users in Cisco IMC Supervisor. This task allows a user to create a new group, which denotes a related set of users.

Prerequisites

None

REST URL

`/cloupia/api-v2/group`

Components

The parameters of the CREATE API are:

- String `groupName`—The name of the group or the customer organization.
- String `groupDescription`—Optional. The description of the group or the customer organization, if required.
- String `parentGroup`—Optional. The name of the parent group.
- String `groupCode`—Optional. A shorter name or code name for the group.
- String `groupContact`—The contact name for the group.
- String `firstName`—Optional. The first name of the group owner.
- String `lastName`—Optional. The last name of the group owner.
- String `phone`—Optional. The phone number of the group owner.
- String `address`—Optional. The address of the group owner.
- String `groupSharePolicyId`—Optional. The ID of group share policy for the users in this group.
- Boolean `allowPrivateUsers`—Optional. The option that allows creating users with exclusive access to their resources.

Sample Input XML

```
<AddGroupConfig>
  <groupName></groupName>

  <groupDescription></groupDescription>

  <parentGroup></parentGroup>

  <groupCode></groupCode>

  <groupContact></groupContact>

  <firstName></firstName>

  <lastName></lastName>

  <phone></phone>

  <address></address>

  <groupSharePolicyId>0</groupSharePolicyId>

  <allowPrivateUsers>>false</allowPrivateUsers>
</AddGroupConfig>
```

Implementation

The user group name is mandatory and must be unique. Contact Email is mandatory.

See Also

[Updating a User Group](#) , on page 52

[Deleting a User Group](#), on page 54

[Enabling All Users in a Group](#), on page 55

[Disabling All Users in a Group](#), on page 56

Updating a User Group

Objective

This task allows a user to update an existing group, which denotes a related set of users.

Prerequisites

None

REST URL

```
/cloupia/api-v2/group
```

Components

The parameters of the UPDATE API are:

- String `groupId`—The id of the group or the customer organization.
- String `groupDescription`—Optional. The description of the group or the customer organization, if required.
- String `parentGroup`—Optional. The name of the parent group.
- String `groupCode`—Optional. A shorter name or code name for the group.
- String `costCenter`—Optional. The cost center for the group.
- String `groupContact`—The contact name for the group.
- String `firstName`—Optional. The first name of the group owner.
- String `lastName`—Optional. The last name of the group owner.
- String `phone`—Optional. The phone number of the group owner.
- String `address`—Optional. The address of the group owner.
- String `groupSharePolicyId`—Optional. The ID of group share policy for the users in this group.
- Boolean `allowPrivateUsers`—Optional. The option that allows creating users with exclusive access to their resources.

Sample Input XML

```
<cuicOperationRequest>
  <payload>
    <![CDATA[
      <ModifyGroupConfig>
        <groupId></groupId>

        <groupDescription></groupDescription>

        <parentGroup></parentGroup>

        <groupCode></groupCode>

        <costCenter></costCenter>

        <groupContact></groupContact>

        <firstName></firstName>

        <lastName></lastName>

        <phone></phone>

        <address></address>

        <groupSharePolicyId>0</groupSharePolicyId>

        <allowPrivateUsers>>false</allowPrivateUsers>

      </ModifyGroupConfig>
    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

Name cannot be modified. The groupId tag is mandatory and must include the numeric ID of a valid existing group. Contact Email is mandatory.

See Also

[Creating a User Group, on page 51](#)

[Deleting a User Group, on page 54](#)

[Enabling All Users in a Group, on page 55](#)

[Disabling All Users in a Group, on page 56](#)

Deleting a User Group

Objective

This task allows a user to delete an existing group, which denotes a related set of users.

Prerequisites

None

REST URL

```
/cloupia/api-v2/group
```

Components

The parameters of the DELETE_USER API are:

String groupName—The name of the group or the customer organization.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>DELETE_GROUP</operationType>
  <payload>
    <![CDATA[
      <DeleteGroupConfig>
        <groupId></groupId>
      </DeleteGroupConfig>
    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

The groupId tag is mandatory and must include the numeric ID of a valid existing group.

See Also

- [Creating a User Group, on page 51](#)
- [Updating a User Group, on page 52](#)
- [Enabling All Users in a Group, on page 55](#)
- [Disabling All Users in a Group, on page 56](#)

Enabling All Users in a Group

Objective

This task allows a user to enable all users which are assigned to a group.

Prerequisites

None

REST URL

/cloupia/api-v2/group

Components

The parameter of the ENABLE_ALL_USERS_IN_GROUP API is:
String groupName—The name of the group or the customer organization.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>ENABLE_ALL_USERS_IN_GROUP</operationType>
  <payload>
    <![CDATA[
      <EnableAllUsersInGroupConfig>
        <groupId></groupId>
      </EnableAllUsersInGroupConfig>
    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

The groupId tag is mandatory and must include the numeric ID of a valid existing group.

See Also

- [Creating a User Group, on page 51](#)
- [Updating a User Group, on page 52](#)
- [Deleting a User Group, on page 54](#)
- [Disabling All Users in a Group, on page 56](#)

Disabling All Users in a Group

Objective

This task allows a user to disable all users which are assigned to a Group.

Prerequisites

None

REST URL

/cloupia/api-v2/group

Components

The parameter of the DISABLE_ALL_USERS_IN_GROUP API is:

String groupName—The name of the group or the customer organization.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>DISABLE_ALL_USERS_IN_GROUP</operationType>
  <payload>
    <![CDATA[
      <DisableAllUsersInGroupConfig>
        <groupId></groupId>
      </DisableAllUsersInGroupConfig>
    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

The groupId tag is mandatory and must include the numeric ID of a valid existing group.

See Also

[Creating a User Group, on page 51](#)

[Deleting a User Group, on page 54](#)

[Updating a User Group, on page 52](#)

[Enabling All Users in a Group, on page 55](#)

Creating a User

Objective

This task allows the user to create a new user.

Prerequisites

None

REST URL

`/cloupia/api-v2/user`

Components

The parameters of the CREATE API are:

- String `userType`—The type of user.
- String `userGroup`—Optional. The group of the user.
- String `mspOrganization`—Optional. MSP organization user.
- String `loginName`—The login name for the user.
- String `password`—The password for the user.
- String `confirmPassword`—Repeat the password from the previous field.
- String `userContactEmail`—The email address.
- String `firstName`—Optional. The first name of the group owner.
- String `lastName`—Optional. The last name of the group owner.
- String `phone`—Optional. The phone number of the group owner.
- String `address`—Optional. The address of the group owner.

Sample Input XML

```
<cuicOperationRequest>
  <payload>
    <![CDATA[
      <AddUserConfig>
        <userType>GroupAdmin</userType>

        <!-- Accepts value from the list: userGroupByType-->
        <userGroup>1</userGroup>

        <mspOrganization></mspOrganization>

        <loginName></loginName>

        <!-- Accepts value from the list: password-->
        <password></password>

        <!-- Accepts value from the list: password-->
        <confirmPassword></confirmPassword>

        <userContactEmail></userContactEmail>

        <firstName></firstName>

        <lastName></lastName>

        <phone></phone>

        <address></address>

      </AddUserConfig>
    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

Login Name is mandatory and must be unique. Password and Confirm Password are mandatory and the values must match. User Contact Email is mandatory. User Type is mandatory and must be an existing valid User Role. User Group Id is required only if the User Type is set to 'Group Admin', and it must denote the numeric Id of an existing User Group.

See Also

- [Reading a User, on page 59](#)
- [Updating a User, on page 59](#)
- [Deleting a User, on page 61](#)
- [Enabling a User, on page 62](#)
- [Disabling a User, on page 63](#)
- [Updating a User Expiry Date, on page 64](#)
- [Updating a User Password, on page 65](#)

Reading a User

Objective

This task allows the user to query the details of an existing user. The `userId` argument must be a valid login name of a user. If no argument is specified, no results will be returned.

Prerequisites

None

REST URL

```
/cloupia/api-v2/user/{userId}
```

Implementation

The `userId` argument must be a valid login name of a user. If no argument is specified, no results will be returned.

See Also

[Creating a User](#), on page 56

[Updating a User](#), on page 59

[Deleting a User](#), on page 61

[Enabling a User](#), on page 62

[Disabling a User](#), on page 63

[Updating a User Expiry Date](#), on page 64

[Updating a User Password](#), on page 65

Updating a User

Objective

This task allows to update an existing user.

Prerequisites

None

REST URL

```
/cloupia/api-v2/user
```

Components

The parameters of the UPDATE USER API are:

- String loginName—The login name for the user.
- String userType—The type of user.
- String userGroup—Optional. The group of the user.
- String mspOrganization—Optional. MSP organization user.
- String userContactEmail—The email address.
- String firstName—Optional. The first name of the group owner.
- String lastName—Optional. The last name of the group owner.
- String phone—Optional. The phone number of the group owner.
- String address—Optional. The address of the group owner.

Sample Input XML

```
<cuicOperationRequest>
<operationType>UPDATE_USER</operationType>
<payload>
<![CDATA[
<ModifyUserConfig>
<loginName></loginName>

<userType>GroupAdmin</userType>

<userGroup>1</userGroup>

<mspOrganization></mspOrganization>

<userContactEmail></userContactEmail>

<firstName></firstName>

<lastName></lastName>

<phone></phone>

<address></address>

</ModifyUserConfig>

]]>
</payload>
</cuicOperationRequest>
```

Implementation

Login Name is mandatory and must denote an existing valid user. It cannot be changed. User Contact Email is mandatory. User Type is mandatory and must be an existing valid User Role. User Group Id is required only if the User Type is set to 'Group Admin', and it must denote the numeric Id of an existing User Group.

See Also

- [Creating a User, on page 56](#)
- [Reading a User, on page 59](#)
- [Deleting a User, on page 61](#)
- [Enabling a User, on page 62](#)
- [Disabling a User, on page 63](#)
- [Updating a User Expiry Date, on page 64](#)
- [Updating a User Password, on page 65](#)

Deleting a User

Objective

This task allows to delete an existing User.

Prerequisites

None

REST URL

```
/cloupia/api-v2/user
```

Components

The parameters of the DELETE_USER API are:
String loginName—The login name for the user.

Sample Input XML

```
<cuicOperationRequest>  
  <operationType>DELETE_USER</operationType>  
  <payload>  
    <![CDATA[  
      <DeleteUserConfig>  
        <loginName></loginName>  
      </DeleteUserConfig>  
    ]]>  
  </payload>  
</cuicOperationRequest>
```

Implementation

Login Name is mandatory and must denote an existing valid user.

See Also

- [Creating a User, on page 56](#)
- [Reading a User, on page 59](#)
- [Updating a User , on page 59](#)
- [Enabling a User, on page 62](#)
- [Disabling a User, on page 63](#)
- [Updating a User Expiry Date, on page 64](#)
- [Updating a User Password, on page 65](#)

Enabling a User

Objective

This task allows to enable an existing user whose account has been disabled.

Prerequisites

None

REST URL

```
/cloupia/api-v2/user
```

Components

The parameter of the ENABLE_USER API is:
String loginName—The login name for the user.

Sample Input XML

```
<cuicOperationRequest>  
<operationType>ENABLE_USER</operationType>  
<payload>  
<![CDATA[  
<EnableUserConfig>  
<loginName></loginName>  
  
</EnableUserConfig>  
]]>  
</payload>  
</cuicOperationRequest>
```

Implementation

Login Name is mandatory and must denote an existing valid user.

See Also

- [Creating a User, on page 56](#)
- [Reading a User, on page 59](#)
- [Updating a User, on page 59](#)
- [Deleting a User, on page 61](#)
- [Disabling a User, on page 63](#)
- [Updating a User Expiry Date, on page 64](#)
- [Updating a User Password, on page 65](#)

Disabling a User

Objective

This task allows to disable an existing User whose account has been enabled.

Prerequisites

None

REST URL

/cloupia/api-v2/user

Components

The parameter of the DISABLE_USER API is:

String loginName—The login name for the user.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>DISABLE_USER</operationType>
  <payload>
    <![CDATA[
      <DisableUserConfig>
        <loginName></loginName>
      </DisableUserConfig>
    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

Login Name is mandatory and must denote an existing valid user.

See Also

- [Creating a User, on page 56](#)
- [Reading a User, on page 59](#)
- [Updating a User , on page 59](#)
- [Deleting a User, on page 61](#)
- [Enabling a User, on page 62](#)
- [Updating a User Expiry Date, on page 64](#)
- [Updating a User Password, on page 65](#)

Updating a User Expiry Date

Objective

This task allows to update the expiry date of an existing user.

Prerequisites

None

REST URL

```
/cloupia/api-v2/user
```

Components

The parameters of the DISABLE_DATE API are:

- String loginName—The login name for the user.
- Long userExpiryDate—The expiry date set for the user.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>DISABLE_DATE</operationType>
  <payload>
    <![CDATA[
      <ConfigureUserExpiryDateConfig>
        <loginName></loginName>

        <!-- Accepts value from the list: date_time-->
        <userExpiryDate>1460449200000</userExpiryDate>

      </ConfigureUserExpiryDateConfig>

    ]]>
  </payload>
</cuicOperationRequest>
```


Implementation

Login Name is mandatory and must denote an existing valid User. Expiry Date is mandatory and must be represented in a numeric form denoting the timestamp of the expiry date/time.

See Also

[Creating a User](#), on page 56

[Reading a User](#), on page 59

[Updating a User](#), on page 59

[Deleting a User](#), on page 61

[Enabling a User](#), on page 62

[Disabling a User](#), on page 63

[Updating a User Password](#), on page 65

Updating a User Password

Objective

This task allows to update an existing user password.

Prerequisites

None

REST URL

`/cloupia/api-v2/user`

Components

The parameters of the UPDATE_USER_PASSWORD API are:

- String loginName—The login name for the user.
- String password—The password for the user.
- String confirmPassword—Repeat the password from the previous field.

Sample Input XML

```
<cuicOperationRequest>
  <operationType>UPDATE_USER_PASSWORD</operationType>
  <payload>
    <![CDATA[
      <AddUserConfig>
        <loginName></loginName>

        <!-- Accepts value from the list: password-->
        <password></password>

        <!-- Accepts value from the list: password-->
        <confirmPassword></confirmPassword>

      </AddUserConfig>

    ]]>
  </payload>
</cuicOperationRequest>
```

Implementation

Login Name is mandatory and must denote an existing valid User. Password and Confirm Password are mandatory and values must match.

See Also

- [Creating a User, on page 56](#)
- [Reading a User, on page 59](#)
- [Updating a User , on page 59](#)
- [Deleting a User, on page 61](#)
- [Enabling a User, on page 62](#)
- [Disabling a User, on page 63](#)
- [Updating a User Expiry Date, on page 64](#)