Cisco IMC Supervisor REST API Cookbook, Release 2.4

First Published: 2024-05-07

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

CONTENTS

Preface

This preface contains the following sections:

Audience

This guide is intended primarily for data center administrators who use Cisco IMC Supervisor and who have responsibilities and expertise in server administration.

Conventions

Text Type	**Indication**
GUI elements	GUI elements such as tab titles, area names, and field labels appear in **this font**. Main titles such as window, dialog box, and wizard titles appear in **this font**.
Document titles	Document titles appear in *this font*.
TUI elements	In a Text-based User Interface, text the system displays appears in `this font`.
System output	Terminal sessions and information that the system displays appear in `this font`.
CLI commands	CLI command keywords appear in **this font**. Variables in a CLI command appear in *this font*.
[ ]	Elements in square brackets are optional.
{x | y | z}	Required alternative keywords are grouped in braces and separated by vertical bars.

Text Type	Indication
[x | y | z]	Optional alternative keywords are grouped in brackets and separated by vertical bars.
string	A nonquoted set of characters. Do not use quotation marks around the string or the string will include the quotation marks.
< >	Nonprinting characters such as passwords are in angle brackets.
[ ]	Default responses to system prompts are in square brackets.
!, #	An exclamation point (!) or a pound sign (#) at the beginning of a line of code indicates a comment line.

Note Means *reader take note*. Notes contain helpful suggestions or references to material not covered in the document.

Caution Means *reader be careful*. In this situation, you might perform an action that could result in equipment damage or loss of data.

Tip Means *the following information will help you solve a problem*. The tips information might not be troubleshooting or even an action, but could be useful information, similar to a Timesaver.

Timesaver Means *the described action saves time*. You can save time by performing the action described in the paragraph.

Warning IMPORTANT SAFETY INSTRUCTIONS

This warning symbol means danger. You are in a situation that could cause bodily injury. Before you work on any equipment, be aware of the hazards involved with electrical circuitry and be familiar with standard practices for preventing accidents. Use the statement number provided at the end of each warning to locate its translation in the translated safety warnings that accompanied this device.

SAVE THESE INSTRUCTIONS

Documentation Feedback

To provide technical feedback on this document, or to report an error or omission, please send your comments to ucs-director-docfeedback@cisco.com. We appreciate your feedback.

Obtaining Documentation and Submitting a Service Request

For information on obtaining documentation, submitting a service request, and gathering additional information, see the monthly What's New in Cisco Product Documentation, which also lists all new and revised Cisco technical documentation.

Subscribe to the What's New in Cisco Product Documentation as a Really Simple Syndication (RSS) feed and set content to be delivered directly to your desktop using a reader application. The RSS feeds are a free service and Cisco currently supports RSS version 2.0.

Related Documentation

Cisco IMC Supervisor Documentation Set

Following are the documents that are available for Cisco IMC Supervisor:

- Cisco IMC Supervisor Release Notes
- Cisco IMC Supervisor Installation and Upgrade on VMware Vsphere Guide
- Cisco IMC Supervisor Rack-Mount Servers Management Guide
- Cisco IMC Supervisor Shell Guide
- Cisco IMC Supervisor REST API Getting Started Guide
- Cisco IMC Supervisor REST API Cook Book

Other Documentation

For a complete list of all C-Series documentation, see the *Cisco UCS C-Series Servers Documentation Roadmap* available at the following URL: http://www.cisco.com/go/unifiedcomputing/c-series-doc.

Note The *Cisco UCS C-Series Servers Documentation Roadmap* includes links to documentation for Cisco Integrated Management Controller.

CHAPTER 1

New and Changed Information in Release 2.4(x.x)

The following table provides an overview of the significant changes to this guide made in versions 2.4(x.x). The table does not provide an exhaustive list of all changes, or of all new features in this release.

Table 1: New and Modified APIs in Release 2.4(0.0)

Feature	What is New	Where Documented
Managing Firmware	This release introduces changes in the APIs for the following: You must activate your device using the **Activate Device** action under the **Images-Local** screen first to find, create and download firmware images from Cisco.com to the local appliance	Finding Firmware Image, on page 8 Creating a Firmware Local Image, on page 10 Downloading Firmware Local Image, on page 11

CHAPTER 2

Overview

This chapter contains the following sections:

Structure of an Example

Under a descriptive title, each example comprises the following sections:

Objective

When you would use the example.

Prerequisites

What conditions have to exist for the example to work.

REST URL

What is the REST URL to pass the REST API.

Components

Which objects and methods are used in the example, and what the input variables represent.

Sample Input XML

The input code sample.

Implementation

Notes on implementing the example, including what modifications might be necessary to implement it.

See Also

Related examples

How to Use the Examples

This document is a collection of examples-recipes, if you will-for using REST API, a server-side scripting solution for use with Cisco IMC Supervisor. Like a cookbook, you can use this document in at least three ways:

- You can follow the examples as written (substituting your own variables, of course) to complete tasks without necessarily knowing everything about the steps you are following.
- You can use the examples as templates and adapt them to similar tasks in your work.
- You can study the examples to figure out “how things are done” in REST API and generalize to using different methods for other tasks you need to script.

The examples are chosen to illustrate common use cases and are intended to facilitate all three of these modes of use.

Note An API uses either HTTP POST or GET. In the following examples, all the READ APIs are GET and others are POST.

CHAPTER 3

Examples

This chapter contains the following sections:

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/NetworkImage
```

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.
- Boolean Configure Graceful—Optional. Choose to configure graceful timeout.
- String GracefulTimeout—The timeout in minutes.
- Boolean DoForceDown—Enable to forcefully shutdown the server after graceful timeout is expired.

Sample Input XML

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

<platform>C220 M4</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>1.1.1.1</networkPath>
<!-- Set this value only when networkServerType not equals to HTTP  -->
<sharePath>/var/www/test</sharePath>
<!-- Set this value only when networkServerType not equals to HTTP  -->
<remoteFileName>sample_fileName</remoteFileName>
<nwPathUserName></nwPathUserName>
<nwPathPassword></nwPathPassword>
<!-- Set this value only when networkServerType equals to CIFS  -->
<mountOptions></mountOptions>
<configureGraceful>true</configureGraceful>
<!-- Set this value only when configureGraceful not equals to false  -->
<gracefulTimeOut>12</gracefulTimeOut>
<doForceDown>true</doForceDown>
</NetworkImage>
]]>
</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. Graceful Timeout is optional, to configure graceful timeout. Timeout (in mins), a graceful timeout period. Valid range is [0-60]. Force Shutdown Server, enable to forcefully down the server after Graceful timeout is expired.

See Also

Updating Firmware Network Image, on page 7

Deleting Firmware Image Profile, on page 12

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

```
/cloupia/api-v2/NetworkImage
```

Components

The parameters of the NETWORK_IMAGE_UPDATE API are:

- String profileName—Unique name of the profile.
- 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.
- Boolean Configure Graceful—Optional. Choose to configure graceful timeout.
- String GracefulTimeout—The timeout in minutes.
- Boolean DoForceDown—Enable to forcefully shutdown the server after graceful timeout is expired.

Sample Input XML

```
<cuicOperationRequest>
<operationType>NETWORK_IMAGE_UPDATE</operationType>
<payload>
<![CDATA[
<NetworkImage>
<profileName>sample</profileName>
<platform>C220 M4</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>1.1.1.1</networkPath>

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

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

<nwPathUserName></nwPathUserName>

<nwPathPassword></nwPathPassword>

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

<configureGraceful>true</configureGraceful>

   <!-- Set this value only when configureGraceful not equals to false  -->
<gracefulTimeOut>10</gracefulTimeOut>

<doForceDown>true</doForceDown>

</NetworkImage>

]]>
</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. Graceful Timeout is optional, to configure graceful timeout. Timeout (in mins), a graceful timeout period. Valid range is [0-60]. Force Shutdown Server, enable to forcefully down the server after Graceful timeout is expired.

See Also

Creating a Firmware Network Image, on page 5

Deleting Firmware Image Profile, on page 12

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.

The user must activate their device first using the **Activate Device** action under the **Images – Local** screen.

Note Device Activation done once stays active for an hour. So, users must re-activate their device every one hour once to access images from Cisco.com for security reasons.

REST URL

```
/cloupia/api-v2/LocalImage
```

Components

The parameters of the LOCAL_IMAGE_FIND API are:

- String platform—The name of the platform.
- 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[
<LocalImage>
<platform></platform>

<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>

</LocalImage>

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

Implementation

The platform of a server that is already added into the system is mandatory.

See Also

Creating a Firmware Local Image, on page 10

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 LocalImage API.

The user must activate their device first using the **Activate Device** action under the **Images – Local** screen.

Note Device Activation done once stays active for an hour. So, users must re-activate their device every one hour once to access images from Cisco.com for security reasons.

REST URL

```
/cloupia/api-v2/LocalImage
```

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 availableImage—The available .iso image.
- boolean acceptLicense—Accept license agreement.
- boolean downloadNow—download the .iso image immediately after adding a profile.
- Boolean Configure Graceful—Optional. Choose to configure graceful timeout.
- String GracefulTimeout—The timeout in minutes.
- Boolean DoForceDown—Enable to forcefully shutdown the server after graceful timeout is expired.

Sample Input XML

```
<cuicOperationRequest>
<operationType>LOCAL_IMAGE_CREATE</operationType>
<payload>
<![CDATA[
<LocalImage>
<profileName>sample</profileName>
<platform>C220 M4</platform>

<availableImage>sampleImage.iso</availableImage>

<downloadNow>false</downloadNow>

<configureGraceful>true</configureGraceful>

   <!-- Set this value only when configureGraceful not equals to false  -->
<gracefulTimeOut>10</gracefulTimeOut>

<doForceDown>true</doForceDown>

</LocalImage>

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

Implementation

Profile Name is mandatory, must be unique. Platform is mandatory. The Platform must be that of a server already added into the system. Graceful Timeout is optional, to configure graceful timeout. Timeout (in mins), a graceful timeout period. Valid range is [0-60]. Force Shutdown Server, enable to forcefully down the server after Graceful timeout is expired.

See Also

Finding Firmware Image, on page 8

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.

The user must activate their device first using the **Activate Device** action under the **Images – Local** screen.

Note Device Activation done once stays active for an hour. So, users must re-activate their device every one hour once to access images from Cisco.com for security reasons.

REST URL

```
/cloupia/api-v2/LocalImage
```

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[
<LocalImage>
<profileName></profileName>

</LocalImage>

]]>
</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 10

Deleting Firmware Image Profile, on page 12

Deleting Firmware Image Profile

Objective

Delete one or more existing firmware image profiles.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCFirmwareUpgradeConfig
```

Components

The parameters of the FIRMWARE_IMAGE_DELETE API are:

- String profileNames—The unique name 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 10

Creating a Firmware Network Image, on page 5

Updating Firmware Network Image, on page 7

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/RunFirmwareUpgrade
```

Components

The parameters of the RUN_FIRMWARE_UPGRADE API are:

- String profileName—The unique name of the profile.
- String platform—The server platform name.
- String imageVersion—The version of the image.
- String imagePath—The path of the image.
- 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[
<RunFirmwareUpgrade>
<profileName></profileName>

<servers></servers>

<enableSchedule>false</enableSchedule>

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

</RunFirmwareUpgrade>

]]>
```

```
</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 15

Reading Firmware Upgrade Status by IP Address, on page 16

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.

Reading Firmware Image by Type

Objective

Get firmware image by type.

Prerequisites

None

REST URL

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

Implementation

This task allows the user to query the firmware image details based on the type of location - NETWORK or LOCAL. The CIMCFirmwareImageByTypeId 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 15

Reading Firmware Image by a Profile Name, on page 14

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 14

Reading Firmware Image by Type, on page 14

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 11

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 13

Reading Firmware Upgrade Status by IP Address, on page 16

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 13

Reading Firmware Upgrade Status by Profile Name, on page 15

Creating a Host Image Profile

Objective

Create a Host Image in a Network Location.

Prerequisites

The Host Image must be present in the network location.

REST URL

```
/cloupia/api-v2/HostImageNetworkImage
```

Components

The parameters of the HostImageNetworkImage API are:

- String Profile Name—The unique name of the profile.

- String Platform—The platform that manages the server.
- String Option Download Image From—The location from where the image must be downloaded from.
- String Server—The IP address of the server.
- String File Path Name—The file path
- String File Type—The file type.
- String File Name—The name of the file.
- String User Name—The user name.
- String Password—The password
- Boolean Map After Download—Map the .iso image after download
- Boolean Delete All Images—Deletes all images on the server.
- Boolean Run Upgrade After Download—Run upgrade immediately after downloading the image.

Sample Input XML

```
<cuicOperationRequest>
<operationType>CREATE_HOST_IMAGE_PROFILE</operationType>
<payload>
<![CDATA[
<HostImageNetworkImage>
<profileName>sample</profileName>
<platform>EN120S M2</platform>
<option>FTP Server</option>
<server>100.10.10.10</server>
<pathFileName>/var/www/test</pathFileName>
<fileType>ISO</fileType>
<fileName>sample</fileName>
<!-- Set this value only when option not equals to any of {HTTP Server,HTTPS Server,}
-->
<username>admin</username>

<!-- Set this value only when option not equals to any of {HTTP Server,HTTPS Server,}
->
<password>YWRtaW4=</password>
<!-- Set this value only when fileType not equals to any of {CIMC,BIOS,}  -->
<mapAfterDownload>true</mapAfterDownload>
<deleteAllImages>true</deleteAllImages>
<upgradeNow>true</upgradeNow>
</HostImageNetworkImage>
]]>
</payload>
</cuicOperationRequest>
```

Implementation

Profile is a mandatory field and it must be unique. Platform, Download Image From, Server IP Address, File Path and File Name are also mandatory fields.

Applying a Host Image Profile

Objective

Apply a host image profile on an E-Series server.

Prerequisites

One or more E -series servers must be configured as Rack Accounts.

REST URL

```
/cloupia/api-v2/ApplyHostImageMap
```

Components

The parameters of the ApplyHostImageMap API are:

- String Server—The server on which the host image map must be applied
- String Profile Name—The unique name of the profile.
- Schedule Later—The option to apply the host image profile at a later point in time.
- Schedule Name—The name of the schedule.

Sample Input XML

```
<cuicOperationRequest>
<operationType>APPLY_HOST_IMAGE_PROFILE</operationType>
<payload>
<![CDATA[
<ApplyHostImageMap>
<serverIdKey>100.100.xx.xxx;100.2x.4x.xxx</serverIdKey>

<profileName>sample</profileName>

</ApplyHostImageMap>

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

Implementation

ServerIdKey is comma(,) separated value. ServerIdKey is of the format: {AccountName};{ServerIPAddress} and it is a mandatory field. Profile Name is mandatory field.

Creating a Cisco.Com Image Profile

Objective

This task allows the user to create a CCO Image Profile that stores the downloaded file (from cisco.com) 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 BIN, SPA and ISO Images.

REST URL

```
/cloupia/api-v2/CIMCHIMCCOImage
```

Components

The parameters of the CIMCHIMCCOImage API are:

- String Profile Name—The unique name of the profile.
- String Platform—The platform that manages the server.
- Boolean Download Now—Download the image immediately after adding a profile.
- String Available Image—The available image.
- Boolean Map After Download—Map the .iso image after download
- Boolean Delete All Images—Deletes all images on the server.
- Boolean Run Upgrade After Download—Run upgrade immediately after downloading the image.
- String License Text—License text.

Sample Input XML

```
<cuicOperationRequest>
<operationType>CCO_IMAGE_CREATE</operationType>
<payload>
<![CDATA[
<CIMCHIMCCOImage>
<profileName>sample</profileName>
<platform>EN120S M2</platform>
<downloadNow>true</downloadNow>
<availableImage>sample.iso</availableImage>
<!-- Set this value only when fileType not equals to any of {CIMC,BIOS,}  -->
<mapAfterDownload>true</mapAfterDownload>
<deleteAllImages>true</deleteAllImages>
<upgradeNow>true</upgradeNow>
<licenseText></licenseText>
</CIMCHIMCCOImage>
]]>
</payload>
</cuicOperationRequest>
```

Implementation

Profile Name is mandatory, must be unique. Platform are mandatory. The platform must be that of a server already added into the system.

Deleting a Host Image Mapping Profile

Objective

Delete one or more existing Host Image Mapping Profiles.

Prerequisites

None

REST URL

```
/cloupia/api-v2/DeleteHostImageProfile
```

Components

The parameter of the DeleteHostImageProfile API is:

- String Profile Name—One or more firmware image profiles to delete.

Sample Input XML

```
<cuicOperationRequest>
<operationType>DELETE_HOST_IMAGE_PROFILE</operationType>
<payload>
<![CDATA[
<DeleteHostImageProfile>
<profileNames>sample_profile_name</profileNames>
</DeleteHostImageProfile>
]]>
</payload>
</cuicOperationRequest>
```

Implementation

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

Downloading a Cisco.Com Image

Objective

This task allows the user to download a CCO Image from cisco.com into a local location inside the appliance.

Prerequisites

The CCO Image Profile must be already configured.

REST URL

```
/cloupia/api-v2/CIMCHIMCCOImage
```

Components

The parameter of the CIMCHIMCCOImage API is:

- String Profile Name—The unique name of the profile.

Sample Input XML

```
<cuicOperationRequest>
<operationType>CCO_IMAGE_DOWNLOAD</operationType>
<payload>
<![CDATA[
<CIMCHIMDownloadCCOImage>
<profileName>sampleCCOProfile</profileName>

</CIMCHIMDownloadCCOImage>

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

Implementation

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

Finding a Cisco.com Image

Objective

This task allows the user to find a CCO Image (BIN,SPA or ISO Image) on cisco.com for only E-Series server platforms.

Prerequisites

The user must have a valid set of credentials to login to cisco.com and have access privileges for BIN, SPA and ISO Images.

REST URL

```
/cloupia/api-v2/CIMCHIMCCOImage
```

Components

The parameter of the CIMCHIMCCOImage API is:

- String Platform—The platform that manages the server.

Sample Input XML

```
<cuicOperationRequest>
<operationType>CCO_IMAGE_FIND</operationType>
<payload>
<![CDATA[
<CIMCHIMCCOImage>
<platform>EN120S M2</platform>
</CIMCHIMCCOImage>
]]>
</payload>
</cuicOperationRequest>
```

Implementation

Platform is mandatory field. The Platform must be that of a server already added into the system.

Reading Host Image Mapping Profile by a Profile Name

Objective

This task allows the user to query the Host Image Mapping details based on the profile name

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCHostImageProfileConfig/{CIMCHostImageProfileConfigId}
```

Components

The parameters of the CIMCHostImageProfileConfig API are:

- String Profile Name—The unique name of the profile.
- String Platform—The platform that manages the server.
- String Option Download Image From—The location from where the image must be downloaded from.

- String Server—The IP address of the server.
- String File Path Name—The file path
- String File Type—The file type.
- String File Name—The name of the file.
- String User Name—The user name.
- String Password—The password
- Boolean Map After Download—Map the .iso image after download
- Boolean Delete All Images—Deletes all images on the server.
- Boolean Run Upgrade After Download—Run upgrade immediately after downloading the image.

Implementation

The CIMCHostImageProfileConfigId argument must be a valid profile name. If no argument is specified, all Host Image Mapping Profile configured in the system will be returned.

Modifying a Host Image Mapping Profile

Objective

Modify Host Image Profile using an image that is present on a network location

Prerequisites

The Host Image must be present in the network location.

REST URL

```
/cloupia/api-v2/HostImageNetworkUpdateImage
```

Components

The parameters of the HostImageNetworkUpdateImage API are:

- String Profile Name—The unique name of the profile.
- String Platform—The platform that manages the server.
- String Option Download Image From—The location from where the image must be downloaded from.
- String Server—The IP address of the server.
- String File Path Name—The file path
- String File Type—The file type.
- String File Name—The name of the file.
- String User Name—The user name.
- String Password—The password
- Boolean Map After Download—Map the .iso image after download

- Boolean Delete All Images—Deletes all images on the server.
- Boolean Run Upgrade After Download—Run upgrade immediately after downloading the image.

Sample Input XML

```
<cuicOperationRequest>
<operationType>MODIFY_HOST_IMAGE_PROFILE</operationType>
<payload>
<![CDATA[
<HostImageNetworkUpdateImage>
<profileName>sample</profileName>
<platform>EN120S M2</platform>
<option>FTP Server</option>
<server>10.10.10.10</server>
<pathFileName>/var/sample_path</pathFileName>
<fileType>ISO</fileType>
<fileName>huu.iso</fileName>
<!-- Set this value only when option not equals to any of {HTTP Server,HTTPS Server,}
 -->
<username>admin</username>
<!-- Set this value only when option not equals to any of {HTTP Server,HTTPS Server,}
 -->
<password>YWRtaW4=</password>
<!-- Set this value only when fileType not equals to any of {CIMC,BIOS,}  -->
<mapAfterDownload>true</mapAfterDownload>
<deleteAllImages>true</deleteAllImages>
<upgradeNow>true</upgradeNow>
</HostImageNetworkUpdateImage>
]]>
</payload>
</cuicOperationRequest>
```

Implementation

Profile is a mandatory field and it must be unique. Platform, Download Image From, Server IP Address, File Path and File Name are also mandatory fields.

See Also

Creating a Host Image Profile, on page 16

Deleting a Host Image Mapping Profile, on page 19

Running a Host Image Upgrade

Objective

Run a Host Image Upgrade on one or more servers using an already configured Host Image Profile.

Prerequisites

The Host Image Profile must be already configured and must contain a valid Host Image.

REST URL

```
/cloupia/api-v2/RunHostImageUpgrade
```

Components

The parameters of the RunHostImageUpgrade API are:

- String Profile Name—The unique name of the profile.

- String Platform—The platform that manages the server.
- String Image Version—The image version.
- String File Type—The file type.
- String Image Path—The path to the image.
- String Servers—The servers on which the firmware must be upgraded.
- Boolean Enable Schedule—The option to schedule the firmware upgrade to a later time.
- String Schedule Name—The name of the schedule.

Sample Input XML

```
<cuicOperationRequest>
<operationType>RUN_HOST_IMAGE_UPGRADE</operationType>
<payload>
<![CDATA[
<RunHostImageUpgrade>
<profileName>sample_profile</profileName>
<platform>EN120S M2</platform>
<imageVersion>CIMC_3.2.4.bin</imageVersion>
<fileType>CIMC</fileType>
<imagePath>10.105.219.218/opt/infra/uploads/external/downloads/dir1529291857206/CIMC_3.2.4.bin</imagePath>
<servers>10.65.183.87;10.65.183.87</servers>
<enableSchedule>false</enableSchedule>
<!-- Set this value only when enableSchedule not equals to false  -->
<associatedScheduleName></associatedScheduleName>
</RunHostImageUpgrade>
]]>
</payload>
</cuicOperationRequest>
```

Implementation

Profile Name is mandatory, must be a valid existing profile. The serverIdKey must consist of a comma-separated list of Ids. Each Id is of the format: {AccountName};{ServerIPAddress}. In case of schedule option, a valid schedule name must be provided.

See Also

Creating a Host Image Profile, on page 16

Applying a Host Image Profile, on page 18

Modifying a Host Image Mapping Profile, on page 22

Downloading Firmware Image to an SD Card

Objective

Download an ISO image to Micro SD cards or FlexFlash cards. You can also choose to initiate the upgrade immediately after the image is downloaded.

Prerequisites

Rack accounts are created in the system.

Local and network image profiles are created in the system.

On Cisco UCS M4 servers, ensure that the FlexFlash controller is configured in the Util mode and not the mirror mode. If the controller is configured in the mirror mode, you cannot download the ISO file to the SD card. Use the FlexFlash policy to configure the controller in the Util mode.

REST URL

```
/cloupia/api-v2/CIMCSDImageDownloadConfig
```

Components

The parameters of the DOWNLOAD_IMAGE_SD API are:

- downloadFrom—Download image from either local or network location. (String, mandatory)
- localProfile—Select profile. Set this value only when downloadFrom parameter is not set to Network. (String, mandatory)
- networkProfile—Select profile. Set this value only when downloadFrom parameter is not set to Local. (String, mandatory)
- runUpgradeNow—Run upgrade after download. (boolean, optional)
- servers—Comma-separated list of server IDs. Each ID is of the format: {AccountName};{ServerIPAddress}.Servers (String, mandatory)

Sample Input XML

```
<cuicOperationRequest>
<operationType>DOWNLOAD_IMAGE_SD</operationType>
<payload>
<![CDATA[
<CIMCSDImageDownloadConfig>
<downloadFrom>LOCAL</downloadFrom>

<!-- Set this value only when downloadFrom not equals to NETWORK  -->
<localProfile>cco_c220_M4</localProfile>

<!-- Set this value only when downloadFrom not equals to LOCAL  -->
<networkProfile></networkProfile>

<runUpgradeNow>false</runUpgradeNow>

<servers>10.10.10.10;10.11.111.111</servers>

</CIMCSDImageDownloadConfig>

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

Implementation

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

See Also

Running Firmware Upgrade from SD Card, on page 26

Running Firmware Upgrade from SD Card

Objective

Run a firmware upgrade on one or more servers using ISO images downloaded on Micro SD cards or FlexFlash cards.

Prerequisites

The firmware image is downloaded.

REST URL

```
/cloupia/api-v2/CIMCSDRunFirmwareUpgrade
```

Components

The parameters of the RUN_UPGRADE_SD API are:

- Servers—Servers on which the firmware must be upgraded. (String, mandatory)
- enableSchedule—To schedule the firmware upgrade at a later point in time. (boolean, mandatory)
- String associatedScheduleName—Name of the associate schedule.

Sample Input XML

```
<cuicOperationRequest>
<operationType>RUN_UPGRADE_SD</operationType>
<payload>
<![CDATA[
<CIMCSDRunFirmwareUpgrade>
<servers>10.10.10.10;10.11.11.11</servers>

<enableSchedule>false</enableSchedule>

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

</CIMCSDRunFirmwareUpgrade>

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

Implementation

- The serverIdKey must consist of a comma-separated list of IDs. Each ID is of the format: {AccountName};{ServerIPAddress}.
- If you choose the schedule option, then you must provide a valid schedule name.

See Also

Downloading Firmware Image to an SD Card, on page 24

Reading Download Status by Server IP, on page 27

Reading Download Status by Account Name , on page 27

Reading Download Status by Server IP

Objective

Get status on the download and upgrade process of an ISO image to Micro SD cards or FlexFlash cards for specific servers using the IP address of the servers.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCSDImageDownloadStatusByServerIP/{CIMCSDImageDownloadStatusByServerIPId}
```

Implementation

The CIMCSDImageDownloadStatusByServerIPId argument must be a valid IP address of a server. If no argument is specified, status of all image download/upgrade operations is returned.

The dots in the IP address must be substituted with an underscore.

See Also

Reading Download Status by Account Name , on page 27

Reading Download Status by Account Name

Objective

Get status on the download and upgrade process of an ISO image to Micro SD cards or FlexFlash cards for specific servers using the account name of the servers.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCSDImageDownloadStatusByAccountName/{CIMCSDImageDownloadStatusByAccountNameId}
```

Implementation

The CIMCSDImageDownloadStatusByAccountNameId argument must be a valid account name of a server. If no argument is specified, status for all image download and upgrade operations is returned.

See Also

Reading Download Status by Server IP, on page 27

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 for which email alerts are sent.
- String emailAddress—The email addresses 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[
<CIMCEmailAlertRuleConfig>
<name></name>

<alertLevel>SYSTEM</alertLevel>

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

<servers></servers>

<emailAddress></emailAddress>

<severity></severity>

<enabled>false</enabled>

</ModifyEmailAlertRuleConfig>

]]>
</payload>
</CIMCEmailAlertRuleConfig>
```

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 emailAlertRules—The email alert rule.

Sample Input XML

```
<cuicOperationRequest>
<operationType>EMAIL_ALERT_RULE_DELETE</operationType>
<payload>
<![CDATA[
<EmailAlertRuleConfig>
<emailAlertRules></emailAlertRules>

</EmailAlertRuleConfig>

]]>
</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

Enabling an Email Alert Rule

Objective

This task allows the user to enable one or more existing Email Alert Rules.

Prerequisites

None

REST URL

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

Components

The parameters of the EMAIL_ALERT_RULE_ENABLE API are:

- String emailAlertRuleNames—The name for the email alert.

Sample Input XML

```
<cuicOperationRequest>
<operationType>EMAIL_ALERT_RULE_ENABLE</operationType>
<payload>
<![CDATA[
<CIMCEmailAlertRuleConfig>
<emailAlertRuleNames></emailAlertRuleNames>

</CIMCEmailAlertRuleConfig>

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

Implementation

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

See Also

Disabling Email Alert Rules

Disabling an Email Alert Rule

Objective

This task allows the user to disable one or more existing Email Alert Rules.

Prerequisites

None

REST URL

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

Components

The parameters of the EMAIL_ALERT_RULE_DISABLE API are:

- String emailAlertRuleNames—The names for the email alert.

Sample Input XML

```
<cuicOperationRequest>
<operationType>EMAIL_ALERT_RULE_DISABLE</operationType>
<payload>
<![CDATA[
<CIMCEmailAlertRuleConfig>
<emailAlertRuleNames></emailAlertRuleNames>

</CIMCEmailAlertRuleConfig>

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

Implementation

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

See Also

Enabling Email Alert Rule

Creating Schedules

Objective

This task allows the user to create a new schedule.

Prerequisites

None

REST URL

```
/cloupia/api-v2/ImcsManageScheduleConfig
```

Components

The parameters of the SCHEDULE_CREATE API are:

- String scheduleName—Name of the schedule task.
- Boolean enableSchedule—Enable the tasks associated with the schedule.

- String scheduleType—A one time or recurring schedule frequency.
- Long scheduleTime—Optional. A schedule time.
- String currentSystemTime—Optional. The system time.
- String daysSchedule—Optional. Number of days to set the schedule time.
- String hoursSchedule—Optional. Number of hours to set the schedule time.
- String minutesSchedule—Optional. Number of minutes to set the schedule time.

Sample Input XML

```
<cuicOperationRequest>
<operationType>SCHEDULE_CREATE</operationType>
<payload>
<![CDATA[
<ImcsManageScheduleConfig>
<scheduleName></scheduleName>
<enableSchedule>true</enableSchedule>
<scheduleType>One Time</scheduleType>
<!-- Set this value only when scheduleType not equals to Recurring  -->
<!-- Accepts value from the list: date_time-->
<scheduleTime>1462353000000</scheduleTime>
<!-- Set this value only when scheduleType not equals to Recurring  -->
<currentSystemTime></currentSystemTime>
<!-- Set this value only when scheduleType equals to Recurring  -->
<daysSchedule>0</daysSchedule>
<!-- Set this value only when scheduleType equals to Recurring  -->
<hoursSchedule>0</hoursSchedule>
<!-- Set this value only when scheduleType equals to Recurring  -->
<minutesSchedule>5</minutesSchedule></ImcsManageScheduleConfig>]]>
</payload>
</cuicOperationRequest>
```

Implementation

Schedule Name is mandatory and must be unique. In case of a One-Time schedule, the date or time must be a future date or time. In case of a Recurring schedule, both hours and minutes cannot be set to zero.

See Also

Reading Schedules, on page 33

Updating a Schedule, on page 34

Deleting Schedules, on page 35

Enabling Schedules, on page 36

Disabling Schedules, on page 37

Reading Schedules

Objective

This task allows the user to query the details of one or more existing schedules.

Prerequisites

None

REST URL

```
/cloupia/api-v2/ImcsManageScheduleConfig/{ImcsManageScheduleConfigId}
```

Implementation

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

See Also

Creating Schedules, on page 32

Updating a Schedule, on page 34

Deleting Schedules, on page 35

Enabling Schedules, on page 36

Disabling Schedules, on page 37

Reading Schedules by Type, on page 37

Updating a Schedule

Objective

This task allows the user to update an existing schedule.

Prerequisites

None

REST URL

```
/cloupia/api-v2/ImcsManageScheduleConfig
```

Components

The parameters of the SCHEDULE_UPDATE API are:

- String scheduleName—Name of the schedule task.
- Boolean enableSchedule—Enable the tasks associated with the schedule.
- String scheduleType—A one time or recurring schedule frequency.
- Long scheduleTime—Optional. A schedule time.
- String currentSystemTime—Optional. The system time.
- String daysSchedule—Optional. Number of days to set the schedule time.
- String hoursSchedule—Optional. Number of hours to set the schedule time.
- String minutesSchedule—Optional. Number of hours to set the schedule time.

Sample Input XML

```
<cuicOperationRequest><operationType>SCHEDULE_UPDATE</operationType>
<payload>
<![CDATA[<ImcsManageScheduleConfig>
<scheduleName></scheduleName>
<enableSchedule>true</enableSchedule>
<scheduleType>One Time</scheduleType>
<!-- Set this value only when scheduleType not equals to Recurring  -->
<!-- Accepts value from the list: date_time-->
<scheduleTime>1462354500000</scheduleTime>
<!-- Set this value only when scheduleType equals to Recurring  -->
<daysSchedule>0</daysSchedule>
<!-- Set this value only when scheduleType equals to Recurring  -->
<hoursSchedule>0</hoursSchedule>
<!-- Set this value only when scheduleType equals to Recurring  -->
<minutesSchedule>5</minutesSchedule>
</ImcsManageScheduleConfig>]]></payload></cuicOperationRequest>
```

Implementation

Schedule Name is mandatory and must refer to an existing schedule and cannot be changed. In case of a One-Time schedule, the date and time must be a future date and time. In case of a Recurring schedule, both hours and minutes cannot be set to zero.

See Also

Creating Schedules, on page 32

Reading Schedules, on page 33

Deleting Schedules, on page 35

Enabling Schedules, on page 36

Disabling Schedules, on page 37

Deleting Schedules

Objective

This task allows the user to delete one or more existing schedules.

Prerequisites

None

REST URL

```
/cloupia/api-v2/ImcsManageScheduleConfig
```

Components

The parameters of the SCHEDULE_DELETE API are:

- String scheduleNames—Name of the schedule task.

Sample Input XML

```
<cuicOperationRequest>
<operationType>SCHEDULE_DELETE</operationType>
<payload>
<![CDATA[<ImcsManageSchedulesConfig>
<scheduleNames></scheduleNames></ImcsManageSchedulesConfig>]]>
</payload>
</cuicOperationRequest>
```

Implementation

Schedule Names must be a comma-separated string of one or more existing schedules.

See Also

Creating Schedules, on page 32

Reading Schedules, on page 33

Updating a Schedule, on page 34

Enabling Schedules, on page 36

Disabling Schedules, on page 37

Enabling Schedules

Objective

This task allows the user to enable one or more existing schedules.

Prerequisites

None

REST URL

`/cloupia/api-v2/EnableSchedules`

Components

The parameters of the SCHEDULE_ENABLE API are:

- String scheduleNames—Names of the schedule task.

Sample Input XML

```
<cuicOperationRequest>
<operationType>SCHEDULE_ENABLE</operationType>
<payload>
<![CDATA[<ImcsManageSchedulesConfig>
<scheduleNames></scheduleNames></ImcsManageSchedulesConfig>]]>
</payload>
</cuicOperationRequest>
```

Implementation

Schedule Names must be a comma-separated string of one or more existing schedules.

See Also

Creating Schedules, on page 32

Reading Schedules, on page 33

Updating a Schedule, on page 34

Deleting Schedules, on page 35

Disabling Schedules, on page 37

Disabling Schedules

Objective

This task allows the user to disable one or more existing schedules.

Prerequisites

None

REST URL

```
/cloupia/api-v2/DisableSchedules
```

Components

The parameters of the SCHEDULE_DISABLE API are:

- String scheduleNames—Names of the schedule task.

Sample Input XML

```
<cuicOperationRequest>
<operationType>SCHEDULE_DISABLE</operationType>
<payload>
<![CDATA[<ImcsManageSchedulesConfig>
<scheduleNames></scheduleNames>
</ImcsManageSchedulesConfig>]]>
</payload>
</cuicOperationRequest>
```

Implementation

Schedule Names must be a comma-separated string of one or more existing schedules.

See Also

Creating Schedules, on page 32

Reading Schedules, on page 33

Updating a Schedule, on page 34

Deleting Schedules, on page 35

Enabling Schedules, on page 36

Reading Schedules by Type

Objective

This task allows the user to query the details of one or more existing schedules. The **Id** argument must be one of the two Schedule Types - **One Time** or **Recurring**. If no argument is specified, all schedules configured in the system will be returned.

Prerequisites

None

REST URL

```
/cloupia/api-v2/ScheduleByType/{ScheduleByTypeId}
```

Implementation

The **Id** argument must be one of the two **Schedule Types** - **One Time** or **Recurring**. If no argument is specified, all schedules configured in the system will be returned.

See Also

Creating Schedules, on page 32

Updating a Schedule, on page 34

Deleting Schedules, on page 35

Enabling Schedules, on page 36

Disabling Schedules, on page 37

Reading Scheduled Discovery Tasks by Schedule Name

Objective

This task allows the user to query the details of scheduled discovery tasks for a given schedule. The **Id** argument must be a valid schedule name. If no argument is specified, all scheduled discovery tasks configured in the system will be returned.

Prerequisites

None

REST URL

```
/cloupia/api-v2/DiscoveryScheduleTasksBySchedule/{DiscoveryScheduleTasksByScheduleId}
```

Implementation

The **Id** argument must be a valid schedule name. If no argument is specified, all scheduled discovery tasks configured in the system will be returned.

See Also

Reading Scheduled Discovery Tasks by Profile Name , on page 38

Reading Scheduled Discovery Tasks by Profile Name

Objective

This task allows the user to query the details of scheduled discovery tasks for a given profile.

Prerequisites

None

REST URL

```
/cloupia/api-v2/DiscoveryScheduleTasksByProfile/{DiscoveryScheduleTasksByProfileId}
```

Implementation

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

See Also

Reading Scheduled Discovery Tasks by Schedule Name, on page 38

Reading Scheduled Firmware Upgrade Tasks by Schedule Name

Objective

This task allows the user to query the details of scheduled firmware upgrade tasks for a given schedule.

Prerequisites

None

REST URL

```
/cloupia/api-v2/FirmwareScheduleTasksBySchedule/{FirmwareScheduleTasksByScheduleId}
```

Implementation

The **Id** argument must be a valid Schedule name. If no argument is specified, all scheduled firmware upgrade tasks configured in the system will be returned.

See Also

Reading Scheduled Firmware Upgrade Tasks by Profile Name , on page 39

Reading Scheduled Firmware Upgrade Tasks by Profile Name

Objective

This task allows the user to query the details of scheduled firmware upgrade tasks for a given profile.

Prerequisites

None

REST URL

```
/cloupia/api-v2/FirmwareScheduleTasksByProfile/{FirmwareScheduleTasksByProfileId}
```

Implementation

The **Id** argument must be a valid profile name. If no argument is specified, all scheduled firmware upgrade tasks configured in the system will be returned.

See Also

Reading Scheduled Firmware Upgrade Tasks by Schedule Name , on page 39

Reading Scheduled Policy Tasks by Schedule Name

Objective

This task allows the user to query the details of scheduled policy tasks for a given schedule.

Prerequisites

None

REST URL

```
/cloupia/api-v2/PolicyScheduleTasksByScheduleName/{PolicyScheduleTasksByScheduleNameId}
```

Implementation

The **Id** argument must be a valid schedule name. If no argument is specified, all scheduled policy tasks configured in the system will be returned.

See Also

Reading Scheduled Policy Tasks by Policy Name , on page 40

Reading Scheduled Policy Tasks by Policy Name

Objective

This task allows the user to query the details of scheduled policy tasks for a given policy.

Prerequisites

None

REST URL

```
/cloupia/api-v2/PolicyScheduleTasksByPolicyName/{PolicyScheduleTasksByPolicyNameId}
```

Implementation

The **Id** argument must be a valid policy name. If no argument is specified, all scheduled policy tasks configured in the system will be returned.

See Also

Reading Scheduled Policy Tasks by Schedule Name , on page 39

Reading Scheduled Profile Tasks by Schedule Name

Objective

This task allows the user to query the details of scheduled profile tasks for a given schedule.

Prerequisites

None

REST URL

```
/cloupia/api-v2/ProfileScheduleTasksByScheduleName/{ProfileScheduleTasksByScheduleNameId}
```

Implementation

The **Id** argument must be a valid schedule name. If no argument is specified, all scheduled profile tasks configured in the system will be returned.

See Also

Reading Scheduled Profile Tasks by Profile Name , on page 41

Reading Scheduled Profile Tasks by Profile Name

Objective

This task allows the user to query the details of scheduled profile tasks for a given profile.

Prerequisites

None

REST URL

```
/cloupia/api-v2/ScheduledTasksByProfileName/{ScheduledTasksByProfileNameId}
```

Implementation

The **Id** argument must be a valid profile name. If no argument is specified, all scheduled policy tasks configured in the system will be returned.

See Also

Reading Scheduled Profile Tasks by Schedule Name , on page 40

Managing Policy and Profile Tasks

Overview

The examples in this category consist of various policy and profile management tasks on Cisco IMC Supervisor. These include creating, reading, updating, and deleting policies and profiles.

Creating Hardware Policy

Objective

This task allows the user to create a hardware policy.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCHardwarePolicy
```

Components

The parameters of the HARDWARE_POLICY_CREATE API are:

- String policyName—The name of the policy.
- String policyType—The hardware policy type.
- String modular—The Cisco UCS C3260 modular dense storage rack server.
- String policyDefinition—The policy definition.

Sample Input XML

```
<cuicOperationRequest>
<operationType>HARDWARE_POLICY_CREATE</operationType>
<payload>
<![CDATA[
<CIMCHardwarePolicy>
<policyName></policyName>

<policyType>BIOS Policy</policyType>

<modular>false</modular>

<policyDefinition></policyDefinition>

</CIMCHardwarePolicy>

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

Implementation

The hardware policy name must be unique, containing valid policy type and definition. Enable 'Cisco UCS C3260' for modular, dense storage rack server with dual server nodes. The policy definition can either be obtained from the management guide or can be obtained by exporting policy from an already created one on the appliance.

See Also

Updating Hardware Policy, on page 43

Applying Policy on Servers, on page 44

Deleting Policies, on page 45

Creating and Updating Policies through REST API

Before you begin

A policy must be available in the Cisco IMC Supervisor appliance.

Step 1 From the menu bar, choose **Policies** > **Manage Policies and Profiles**.

Step 2 Choose the **Hardware Policies** tab.

Step 3 Select an existing policy and click **Export**.

Step 4 In the Export dialog box, copy the **XML Encoded Format**.

Step 5 Click Close.

Step 6 From the menu bar, choose **Policies** > **API and Orchestration**.

Step 7 In the left pane, select **Policy and Profile Tasks**.

Step 8 Double-click **HARDWARE_POLICY_CREATE** or **HARDWARE_POLICY_UPDATE** operation.

Step 9 Enter **Policy Name** and select the **Policy Type** to create a policy or modify the existing policy details.

Step 10 Check the **Cisco UCS C3260** check box if you need to create a Cisco UCS C3260 Rack Server policy. For more information about the various rack mount server policies and chassis policies see, Managing Cisco UCS C3260 Dense Storage Rack Server in the Cisco IMC Supervisor Rack-Mount Servers Management Guide.

Step 11 Paste the copied **XML Encoded Format** in the **Policy Definition** box.

Step 12 Click **Generate XML**.
The **Sample XML** box is filled with the XML code.

Step 13 Click **Execute REST API**.
The policy is now created.

Step 14 Click **Close**.

Updating Hardware Policy

Objective

This task allows the user to update existing hardware policy.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCHardwarePolicy
```

Components

The parameters of the HARDWARE_POLICY_UPDATE API are:

- String policyName—The name of the policy.
- String policyType—The hardware policy type.
- String modular—The Cisco UCS C3260 modular dense storage rack server.
- String policyDefinition—The policy definition.

Sample Input XML

```
<cuicOperationRequest>
<operationType>HARDWARE_POLICY_UPDATE</operationType>
<payload>
<![CDATA[
<CIMCHardwarePolicy>
<policyName></policyName>

<policyType>BIOS Policy</policyType>

<modular>false</modular>

<policyDefinition></policyDefinition>

</CIMCHardwarePolicy>

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

Implementation

The hardware profile name must be an existing one, containing comma separated list of valid policies.

See Also

Creating Hardware Policy, on page 41

Applying Policy on Servers, on page 44

Deleting Policies, on page 45

Applying Policy on Servers

Objective

This task allows the user to apply hardware policies on one more servers.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCHardwarePolicy
```

Components

The parameters of the HARDWARE_POLICY_APPLY API are:

- String policyName—The name of the policy to apply.
- String servers—The servers to which you want to apply the policy.
- String chassis—The C3260 server to which you want to apply the policy.
- boolean enableSchedule—Enable a schedule.
- String associatedScheduleName—The associated schedule name.

Sample Input XML

```
<cuicOperationRequest>
<operationType>HARDWARE_POLICY_APPLY</operationType>
<payload>
<![CDATA[<CIMCHardwarePolicy>
<policyName></policyName>
<servers></servers>
<chassis></chassis>
<enableSchedule>false</enableSchedule> <!-- Set this value only when enableSchedule not
 equals to
false  -->
<associatedScheduleName></associatedScheduleName>
</CIMCHardwarePolicy>]]>
</payload>
</cuicOperationRequest>
```

Implementation

Selected policy must be a valid one. The servers argument must consist of a comma-separated list of Id's. Each Id is in the format: {AccountName};{ServerIPAddress}. The chassis argument must consist of a comma-separated list of Id's. Each Id is in the format: {AccountName};{ChassisAddress}.

See Also

Deleting Policies, on page 45

Deleting Policies

Objective

This task allows the user to delete one or more existing policies.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCHardwarePolicy
```

Components

The parameters of the HARDWARE_POLICY_DELETE API are:

- String policyNames—The name of the policy to delete.

Sample Input XML

```
<cuicOperationRequest>
<operationType>HARDWARE_POLICY_DELETE</operationType>
<payload>
<![CDATA[<CIMCHardwarePolicy>
<policyNames></policyNames>
</CIMCHardwarePolicy>]]>
</payload>
</cuicOperationRequest>
```

Implementation

Comma separated list of policies, all of which must be valid existing policies.

See Also

Applying Policy on Servers, on page 44

Reading Disk Group Policy

Objective

This task allows the user to query the details of Disk Group Policy.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCDiskGroupPolicyConfig/{CIMCDiskGroupPolicyConfigId}
```

Implementation

The Id argument must be a valid policy name. If no argument is specified, all Disk Group policies created in the system will be returned.

See Also

Reading FlexFlash Policy, on page 46

Reading IPMI Over LAN Policy, on page 47

Reading LDAP Policy, on page 47

Reading Legacy Boot Order Policy, on page 48

Reading Network Security Policy, on page 49

Reading NTP Policy, on page 50

Reading Precision Boot Order Policy, on page 52

Reading RAID Policy, on page 53

Reading Serial Over LAN Policy, on page 53

Reading SNMP Policy, on page 54

Reading SSH Policy, on page 55

Reading User Policy, on page 56

Reading VIC Adapter Policy, on page 58

Reading Virtual KVM Policy, on page 57

Reading vMedia Policy, on page 56

Reading FlexFlash Policy

Objective

This task allows the user to query the details of FlexFlash Policy.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCFFlashPolicyConfig/{CIMCFFlashPolicyConfigId}
```

Implementation

The Id argument must be a valid policy name. If no argument is specified, all FlexFlash policies created in the system will be returned.

See Also

Reading Disk Group Policy, on page 45

Reading IPMI Over LAN Policy, on page 47

Reading LDAP Policy, on page 47

Reading Legacy Boot Order Policy, on page 48

Reading Network Security Policy, on page 49

Reading NTP Policy, on page 50

Reading Precision Boot Order Policy, on page 52

Reading RAID Policy, on page 53

Reading Serial Over LAN Policy, on page 53

Reading SNMP Policy, on page 54

Reading SSH Policy, on page 55

Reading User Policy, on page 56

Reading VIC Adapter Policy, on page 58

Reading Virtual KVM Policy, on page 57

Reading vMedia Policy, on page 56

Reading IPMI Over LAN Policy

Objective

This task allows the user to query the details of IPMI Over LAN Policy.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCIpmiPolicyConfig/{CIMCIpmiPolicyConfigId}
```

Implementation

The Id argument must be a valid policy name. If no argument is specified, all IPMI Over LAN policies created in the system will be returned.

See Also

Reading Disk Group Policy, on page 45

Reading FlexFlash Policy, on page 46

Reading LDAP Policy, on page 47

Reading Legacy Boot Order Policy, on page 48

Reading Network Security Policy, on page 49

Reading NTP Policy, on page 50

Reading Precision Boot Order Policy, on page 52

Reading RAID Policy, on page 53

Reading Serial Over LAN Policy, on page 53

Reading SNMP Policy, on page 54

Reading SSH Policy, on page 55

Reading User Policy, on page 56

Reading VIC Adapter Policy, on page 58

Reading Virtual KVM Policy, on page 57

Reading vMedia Policy, on page 56

Reading LDAP Policy

Objective

This task allows the user to query the details of LDAP Policy.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCLdapConfig/{CIMCLdapConfigId}
```

Implementation

The Id argument must be a valid policy name. If no argument is specified, all LDAP policies created in the system will be returned.

See Also

Reading Disk Group Policy, on page 45

Reading FlexFlash Policy, on page 46

Reading IPMI Over LAN Policy, on page 47

Reading Legacy Boot Order Policy, on page 48

Reading Network Security Policy, on page 49

Reading NTP Policy, on page 50

Reading Precision Boot Order Policy, on page 52

Reading RAID Policy, on page 53

Reading Serial Over LAN Policy, on page 53

Reading SNMP Policy, on page 54

Reading SSH Policy, on page 55

Reading User Policy, on page 56

Reading VIC Adapter Policy, on page 58

Reading Virtual KVM Policy, on page 57

Reading vMedia Policy, on page 56

Reading Legacy Boot Order Policy

Objective

This task allows the user to query the details of Legacy Boot Order Policy.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCBootOrderLegacyConfig/{CIMCBootOrderLegacyConfigId}
```

Implementation

The Id argument must be a valid policy name. If no argument is specified, all Legacy Boot Order policies created in the system will be returned.

See Also

Reading Disk Group Policy, on page 45

Reading FlexFlash Policy, on page 46

Reading IPMI Over LAN Policy, on page 47

Reading LDAP Policy, on page 47

Reading Network Security Policy, on page 49

Reading NTP Policy, on page 50

Reading Precision Boot Order Policy, on page 52

Reading RAID Policy, on page 53

Reading Serial Over LAN Policy, on page 53

Reading SNMP Policy, on page 54

Reading SSH Policy, on page 55

Reading User Policy, on page 56

Reading VIC Adapter Policy, on page 58

Reading Virtual KVM Policy, on page 57

Reading vMedia Policy, on page 56

Reading Network Security Policy

Objective

This task allows the user to query the details of Network Security Policy.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCNetworkSecurityPolicyConfig/{CIMCNetworkSecurityPolicyConfigId}
```

Implementation

The Id argument must be a valid policy name. If no argument is specified, all Network Security policies created in the system will be returned.

See Also

Reading Disk Group Policy, on page 45

Reading FlexFlash Policy, on page 46

Reading IPMI Over LAN Policy, on page 47

Reading LDAP Policy, on page 47

Reading Legacy Boot Order Policy, on page 48

Reading NTP Policy, on page 50

Reading Precision Boot Order Policy, on page 52

Reading RAID Policy, on page 53

Reading Serial Over LAN Policy, on page 53

Reading SNMP Policy, on page 54

Reading SSH Policy, on page 55

Reading User Policy, on page 56

Reading VIC Adapter Policy, on page 58

Reading Virtual KVM Policy, on page 57

Reading vMedia Policy, on page 56

Reading NTP Policy

Objective

This task allows the user to query the details of NTP Policy.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCNtpPolicyConfig/{CIMCNtpPolicyConfigId}
```

Implementation

The Id argument must be a valid policy name. If no argument is specified, all NTP policies created in the system will be returned.

See Also

Reading Disk Group Policy, on page 45

Reading FlexFlash Policy, on page 46

Reading IPMI Over LAN Policy, on page 47

Reading LDAP Policy, on page 47

Reading Legacy Boot Order Policy, on page 48

Reading Network Security Policy, on page 49

Reading Precision Boot Order Policy, on page 52

Reading RAID Policy, on page 53

Reading Serial Over LAN Policy, on page 53

Reading SNMP Policy, on page 54

Reading SSH Policy, on page 55

Reading User Policy, on page 56

Reading VIC Adapter Policy, on page 58

Reading Virtual KVM Policy, on page 57

Reading vMedia Policy, on page 56

Reading Password Expiration Policy

Objective

This task allows the user to query the details of Password Expiration Policy.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCPasswordExpirationPolicyConfig/{CIMCPasswordExpirationPolicyConfigId}
```

Implementation

The Id argument must be a valid policy name. If no argument is specified, all Password Expiration policies created in the system will be returned.

See Also

Reading Disk Group Policy, on page 45

Reading FlexFlash Policy, on page 46

Reading IPMI Over LAN Policy, on page 47

Reading LDAP Policy, on page 47

Reading Legacy Boot Order Policy, on page 48

Reading Network Security Policy, on page 49

Reading NTP Policy, on page 50

Reading Precision Boot Order Policy, on page 52

Reading RAID Policy, on page 53

Reading Serial Over LAN Policy, on page 53

Reading SNMP Policy, on page 54

Reading SSH Policy, on page 55

Reading VIC Adapter Policy, on page 58

Reading Virtual KVM Policy, on page 57

Reading vMedia Policy, on page 56

Reading Power Restore Policy

Objective

This task allows the user to query the details of the power restore policy.

Prerequisites

None.

REST URL

```
/cloupia/api-v2/CIMCPowerRestorePolicyConfig/{CIMCPowerRestorePolicyConfigId}
```

Components

The parameters of the CIMCPowerRestorePolicyConfig API are:

- String Policy Name—The unique name of the policy.
- String Value
- String Note

Implementation

The Id argument must be a valid policy name. If no argument is specified, all Power Restore policies created in the system will be returned.

Reading Precision Boot Order Policy

Objective

This task allows the user to query the details of Boot Order Precision Policy.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCBootOrderPrecisionConfig/{CIMCBootOrderPrecisionConfigId}
```

Implementation

The Id argument must be a valid policy name. If no argument is specified, all Precision Boot Order policies created in the system will be returned.

See Also

Reading Disk Group Policy, on page 45

Reading FlexFlash Policy, on page 46

Reading IPMI Over LAN Policy, on page 47

Reading LDAP Policy, on page 47

Reading Legacy Boot Order Policy, on page 48

Reading Network Security Policy, on page 49

Reading NTP Policy, on page 50

Reading RAID Policy, on page 53

Reading Serial Over LAN Policy, on page 53

Reading SNMP Policy, on page 54

Reading SSH Policy, on page 55

Reading User Policy, on page 56

Reading VIC Adapter Policy, on page 58

Reading Virtual KVM Policy, on page 57

Reading vMedia Policy, on page 56

Reading RAID Policy

Objective

This task allows the user to query the details of RAID Policy.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCRaidPolicyConfig/{CIMCRaidPolicyConfigId}
```

Implementation

The Id argument must be a valid policy name. If no argument is specified, all RAID policies created in the system will be returned.

See Also

Reading Disk Group Policy, on page 45

Reading FlexFlash Policy, on page 46

Reading IPMI Over LAN Policy, on page 47

Reading LDAP Policy, on page 47

Reading Legacy Boot Order Policy, on page 48

Reading Network Security Policy, on page 49

Reading NTP Policy, on page 50

Reading Precision Boot Order Policy, on page 52

Reading Serial Over LAN Policy, on page 53

Reading SNMP Policy, on page 54

Reading SSH Policy, on page 55

Reading User Policy, on page 56

Reading VIC Adapter Policy, on page 58

Reading Virtual KVM Policy, on page 57

Reading vMedia Policy, on page 56

Reading Serial Over LAN Policy

Objective

This task allows the user to query the details of Serial Over LAN Policy.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCSoLPolicyConfig/{CIMCSoLPolicyConfigId}
```

Implementation

The Id argument must be a valid policy name. If no argument is specified, all Serial Over LAN policies created in the system will be returned.

See Also

Reading Disk Group Policy, on page 45

Reading FlexFlash Policy, on page 46

Reading IPMI Over LAN Policy, on page 47

Reading LDAP Policy, on page 47

Reading Legacy Boot Order Policy, on page 48

Reading Network Security Policy, on page 49

Reading NTP Policy, on page 50

Reading Precision Boot Order Policy, on page 52

Reading RAID Policy, on page 53

Reading SNMP Policy, on page 54

Reading SSH Policy, on page 55

Reading User Policy, on page 56

Reading VIC Adapter Policy, on page 58

Reading Virtual KVM Policy, on page 57

Reading vMedia Policy, on page 56

Reading SNMP Policy

Objective

This task allows the user to query the details of SNMP Policy.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCSNMPPolicyConfig/{CIMCSNMPPolicyConfigId}
```

Implementation

The Id argument must be a valid policy name. If no argument is specified, all SNMP policies created in the system will be returned.

See Also

Reading Disk Group Policy, on page 45

Reading FlexFlash Policy, on page 46

Reading IPMI Over LAN Policy, on page 47

Reading LDAP Policy, on page 47

Reading Legacy Boot Order Policy, on page 48

Reading Network Security Policy, on page 49

Reading NTP Policy, on page 50

Reading Precision Boot Order Policy, on page 52

Reading RAID Policy, on page 53

Reading Serial Over LAN Policy, on page 53

Reading SSH Policy, on page 55

Reading User Policy, on page 56

Reading VIC Adapter Policy, on page 58

Reading Virtual KVM Policy, on page 57

Reading vMedia Policy, on page 56

Reading SSH Policy

Objective

This task allows the user to query the details of SSH Policy.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCSshPolicyConfig/{CIMCSshPolicyConfigId}
```

Implementation

The Id argument must be a valid policy name. If no argument is specified, all SSH policies created in the system will be returned.

See Also

Reading Disk Group Policy, on page 45

Reading FlexFlash Policy, on page 46

Reading IPMI Over LAN Policy, on page 47

Reading LDAP Policy, on page 47

Reading Legacy Boot Order Policy, on page 48

Reading Network Security Policy, on page 49

Reading NTP Policy, on page 50

Reading Precision Boot Order Policy, on page 52

Reading RAID Policy, on page 53

Reading Serial Over LAN Policy, on page 53

Reading SNMP Policy, on page 54

Reading User Policy, on page 56

Reading VIC Adapter Policy, on page 58

Reading Virtual KVM Policy, on page 57

Reading vMedia Policy, on page 56

Reading User Policy

Objective

This task allows the user to query the details of User Policy.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCUserPolicyConfig/{CIMCUsersConfigTableId}
```

Implementation

The Id argument must be a valid policy name. If no argument is specified, all User policies created in the system will be returned.

See Also

Reading Disk Group Policy, on page 45

Reading FlexFlash Policy, on page 46

Reading IPMI Over LAN Policy, on page 47

Reading LDAP Policy, on page 47

Reading Legacy Boot Order Policy, on page 48

Reading Network Security Policy, on page 49

Reading NTP Policy, on page 50

Reading Precision Boot Order Policy, on page 52

Reading RAID Policy, on page 53

Reading Serial Over LAN Policy, on page 53

Reading SNMP Policy, on page 54

Reading SSH Policy, on page 55

Reading VIC Adapter Policy, on page 58

Reading Virtual KVM Policy, on page 57

Reading vMedia Policy, on page 56

Reading vMedia Policy

Objective

This task allows the user to query the details of vMedia Policy.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCVMediaPolicyConfig/{CIMCVMediaPolicyConfigId}
```

Implementation

The Id argument must be a valid policy name. If no argument is specified, all vMedia policies created in the system will be returned.

See Also

Reading Disk Group Policy, on page 45

Reading FlexFlash Policy, on page 46

Reading IPMI Over LAN Policy, on page 47

Reading LDAP Policy, on page 47

Reading Legacy Boot Order Policy, on page 48

Reading Network Security Policy, on page 49

Reading NTP Policy, on page 50

Reading Precision Boot Order Policy, on page 52

Reading RAID Policy, on page 53

Reading Serial Over LAN Policy, on page 53

Reading SNMP Policy, on page 54

Reading SSH Policy, on page 55

Reading User Policy, on page 56

Reading VIC Adapter Policy, on page 58

Reading Virtual KVM Policy, on page 57

Reading Virtual KVM Policy

Objective

This task allows the user to query the details of vKVM Policy.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCvKVMPolicyConfig/{CIMCvKVMPolicyConfigId}
```

Implementation

The Id argument must be a valid policy name. If no argument is specified, all vKVM policies created in the system will be returned.

See Also

Reading Disk Group Policy, on page 45

Reading FlexFlash Policy, on page 46

Reading IPMI Over LAN Policy, on page 47

Reading LDAP Policy, on page 47

Reading Legacy Boot Order Policy, on page 48

Reading Network Security Policy, on page 49

Reading NTP Policy, on page 50

Reading Precision Boot Order Policy, on page 52

Reading RAID Policy, on page 53

Reading Serial Over LAN Policy, on page 53

Reading SNMP Policy, on page 54

Reading SSH Policy, on page 55

Reading User Policy, on page 56

Reading VIC Adapter Policy, on page 58

Reading vMedia Policy, on page 56

Reading VIC Adapter Policy

Objective

This task allows the user to query the details of VIC Policy.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCVicPolicy/{CIMCVicPolicyId}
```

Implementation

The Id argument must be a valid policy name. If no argument is specified, all VIC policies created in the system will be returned.

See Also

Reading Disk Group Policy, on page 45

Reading FlexFlash Policy, on page 46

Reading IPMI Over LAN Policy, on page 47

Reading LDAP Policy, on page 47

Reading Legacy Boot Order Policy, on page 48

Reading Network Security Policy, on page 49

Reading NTP Policy, on page 50

Reading Precision Boot Order Policy, on page 52

Reading RAID Policy, on page 53

Reading Serial Over LAN Policy, on page 53

Reading SNMP Policy, on page 54

Reading SSH Policy, on page 55

Reading User Policy, on page 56

Reading Virtual KVM Policy, on page 57

Reading vMedia Policy, on page 56

Creating Hardware Profile

Objective

This task allows the user to create a hardware profile.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCHardwareProfile
```

Components

The parameters of the HARDWARE_PROFILE_CREATE API are:

- String profileName—The name of the profile.
- String policyIds—(Optional) The hardware policies created on the system.
- boolean modular—(Optional) Cisco UCS C3260 dense storage rack server.
- String nonmodularPolicies—If server is not a Cisco UCS C3260 dense storage rack server.
- String modular Policies—If server policy is for a Cisco UCS C3260 dense storage rack server.
- String targetPlatforms—The target platform of a server.

Sample Input XML

```
<cuicOperationRequest>
<operationType>HARDWARE_PROFILE_CREATE</operationType>
<payload>
<![CDATA[<CIMCHardwareProfile>
<profileName></profileName>

<modular>false</modular>

   <!-- Set this value only when modular not equals to true  -->
<nonmodularPolicies></nonmodularPolicies>

   <!-- Set this value only when modular not equals to false  -->
<modularPolicies></modularPolicies>

   <!-- Set this value only when modular not equals to false  -->
<targetPlatforms></targetPlatforms>
</CIMCHardwareProfile>

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

Implementation

The hardware profile name must be unique, containing comma separated list of valid policies. Enable 'Cisco UCS C3260' for dense storage rack server with dual server nodes. The policies must already exist in the appliance. The list of policies are specific to the selected server platform. The target platforms must be comma separated list of servers/chassis in the same sequence in which policies are specified.

See Also

Reading Hardware Profile, on page 60

Updating Hardware Profile, on page 60

Deleting Hardware Profile, on page 63

Applying Hardware Profile, on page 64

Reading Hardware Profile

Objective

This task allows the user to query the details of Hardware Profiles.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCHardwareProfile/{CIMCHardwareProfileId}
```

Implementation

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

See Also

Creating Hardware Profile, on page 59

Updating Hardware Profile, on page 60

Deleting Hardware Profile, on page 63

Applying Hardware Profile, on page 64

Updating Hardware Profile

Objective

This task allows the user to update existing hardware profile.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCHardwareProfile
```

Components

The parameters of the HARDWARE_PROFILE_UPDATE API are:

- String profileNames—The name of the profile.
- String policyIds—(Optional) The hardware policies created on the system.
- boolean modular—(Optional) Cisco UCS C3260 dense storage rack server.
- String nonmodularPolicies—If server is not a Cisco UCS C3260 dense storage rack server.
- String modular Policies—If server policy is for a Cisco UCS C3260 dense storage rack server.
- String targetPlatforms—The target platform of a server.

Sample Input XML

```
<cuicOperationRequest>
<operationType>HARDWARE_PROFILE_UPDATE</operationType>
<payload>
<![CDATA[
<CIMCHardwareProfile>
<profileName></profileName>

<modular>false</modular>

   <!-- Set this value only when modular not equals to true  -->
<nonmodularPolicies></nonmodularPolicies>

   <!-- Set this value only when modular not equals to false  -->
<modularPolicies></modularPolicies>

   <!-- Set this value only when modular not equals to false  -->
<targetPlatforms></targetPlatforms>

</CIMCHardwareProfile>

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

Implementation

The hardware profile name must be an existing one, containing comma separated list of valid policies. Enable 'Cisco UCS C3260' for dense storage rack server with dual server nodes. The list of policies specified here will completely override any previous list of associated policies that was specified when this profile was created. The target platforms must be comma separated list of servers/chassis in the same sequence in which policies are specified.

See Also

Creating Hardware Profile, on page 59

Reading Hardware Profile, on page 60

Deleting Hardware Profile, on page 63

Applying Hardware Profile, on page 64

Deriving a Hardware Profile

Objective

This task allows the user to derive a hardware profile.

Prerequisites

None.

REST URL

```
/cloupia/api-v2/CIMCDeriveHardwareProfile
```

Components

The parameters of the CIMCDeriveHardwareProfile API are:

- String Profile Name—The unique name of the profile.
- String Policy ID—The IDs of the profile.
- Boolean Modular—For Cisco UCS S3260.
- Boolean Manual—The server details entered manually.
- String Choose Server—The server list.
- String Server IP—The IP addresses of the server.
- String Chassis—The chassis details. Applicable only when modular option is enabled.
- Boolean Credential Policy—The option to use a credential policy.
- String Credential Policy—The credential policy to be used.
- String User Name—The user name.
- String Password—The password.
- String Protocol—The protocol to be used.
- Port—The port to be used.
- String Policy—The policy types.

Sample Input XML

```
<cuicOperationRequest>
<operationType>HARDWARE_PROFILE_DERIVE</operationType>
<payload>
<![CDATA[
<CIMCDeriveHardwareProfile>
<profileName>sample</profileName>
<modular>true</modular>
<manual>false</manual>
<!-- Set this value only when manual not equals to true  -->
<chooseServer></chooseServer>
<!-- Set this value only when manual not equals to false  -->
<server></server>
<!-- Set this value only when manual not equals to true  -->
<chooseChassis></chooseChassis>
<!-- Set this value only when manual not equals to false  -->
<credentialPolicy>false</credentialPolicy>
<!-- Set this value only when manual not equals to false  -->
<policy></policy>
<!-- Set this value only when manual not equals to false  -->
<username></username>
<!-- Set this value only when manual not equals to false  -->
<password></password>
<!-- Set this value only when manual not equals to false  -->
<protocol>https</protocol>
<!-- Set this value only when manual not equals to false  -->
<port>443</port>
<policyTypes>BIOS Policy</policyTypes>
</CIMCDeriveHardwareProfile>
]]>
</payload>
</cuicOperationRequest>
```

Implementation

The hardware profile name must be unique, containing comma separated list of valid profiles. Enable Modular for modular, dense storage rack server with dual server nodes. Enter Server Details Manually - enable to manually input the server details. Choose Server - Choose the server from which the configurations are to be retrieved. Choose Chassis - Choose the chassis from which the configurations are to be retrieved. Choose Policies - Choose the policies to be created from the server.

Deleting Hardware Profile

Objective

This task allows the user to delete hardware profiles.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCHardwareProfile
```

Components

The parameters of the HARDWARE_PROFILE_DELETE API are:

- String profileNames—The name of the profile.

Sample Input XML

```
<cuicOperationRequest>
<operationType>HARDWARE_PROFILE_DELETE</operationType>
<payload>
<![CDATA[<CIMCHardwareProfile>
<profileNames></profileNames></CIMCHardwareProfile>]]>
</payload>
</cuicOperationRequest>
```

Implementation

The hardware profiles name(s) must be existing ones.

See Also

Creating Hardware Profile, on page 59

Reading Hardware Profile, on page 60

Updating Hardware Profile, on page 60

Applying Hardware Profile, on page 64

Applying Hardware Profile

Objective

This task allows the user to apply hardware profile.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCHardwareProfile
```

Components

The parameters of the HARDWARE_PROFILE_APPLY API are:

- String profileNames—The name of the profile to apply.
- String servers—The servers to which you want to apply the profile.
- String Chassis—The chassis groups to which you want to apply the profile.
- boolean enableSchedule—Enable a schedule.
- String associatedScheduleName—The associated schedule name.

Sample Input XML

```
<cuicOperationRequest>
<operationType>HARDWARE_PROFILE_APPLY</operationType>
<payload>
<![CDATA[
<CIMCHardwareProfile>
<profileName></profileName>

<servers></servers>

<chassis></chassis>

<enableSchedule>false</enableSchedule>

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

</CIMCHardwareProfile>

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

Implementation

The servers argument must consist of a comma-separated list of Id's. Each Id is in the format: {AccountName};{ServerIPAddress}. The ServerIPAddress must be a non CISCO C3260 UCS server. The chassis argument must consist of a comma-separated list of Id's. Each Id is in the format: {AccountName};{ChassisAddress}.

See Also

Creating Hardware Profile, on page 59

Reading Hardware Profile, on page 60

Updating Hardware Profile, on page 60

Deleting Hardware Profile, on page 63

Reading Hardware Policy Apply Status

Objective

This task allows the user to query the apply status details of hardware policies.

Prerequisites

None.

REST URL

```
/cloupia/api-v2/CIMCPolicyApplyStatusByPolicyName/{CIMCPolicyApplyStatusByPolicyNameId}
```

Components

The parameters of the CIMCPolicyApplyStatusByPolicyName API are:

- String Policy Name—The unique name of the profile.
- String Policy Type—The type of policy.
- String Server Address—The server address.
- String Host Name—The host name of the server.
- String Account Name—The name of the account.
- String Last Message—The last message on the server.
- Boolean Is Successful—The indication if the apply status is successful or not.
- String Last Policy Update—The indication of the last policy update on the servers.

Implementation

The ID argument must be a valid policy name. If no argument is specified, apply status of all policies created in the system will be returned.

Reading Hardware Profile Apply Status

Objective

This task allows the user to query the apply status details of Hardware Profiles.

Prerequisites

None.

REST URL

```
/cloupia/api-v2/CIMCProfileApplyStatusByProfileName/{CIMCProfileApplyStatusByProfileNameId}
```

Components

The parameters of the CIMCProfileApplyStatusByProfileName API are:

- String Policy Name—The unique name of the profile.
- String Policy Type—The type of policy.
- String Server Address—The server address.
- String Host Name—The host name of the server.
- String Account Name—The name of the account.
- String Last Message—The last message on the server.
- Boolean Is Successful—The indication if the apply status is successful or not.
- String Last Policy Update—The indication of the last policy update on the servers.

Implementation

The ID argument must be a valid policy name. If no argument is specified, apply status of all policies created in the system will be returned.

Viewing Hardware Profiles Associated with a Server

Objective

This task allows the user to query the list of hardware profiles that are associated with a specific server.

Prerequisites

None.

REST URL

```
/cloupia/api-v2/AssociatedHardwareProfilesByServer
```

Components

The parameters of the AssociatedHardwareProfilesByServer API are:

- String Account Name—The name of the account.

Sample Input XML

```
<cuicOperationRequest>
<operationType>HARDWARE_PROFILES</operationType>
<payload>
<![CDATA[
<AssociatedHardwareProfilesByServer>
<servers>CIMC192;<ip_address of server></servers>

</AssociatedHardwareProfilesByServer>

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

Implementation

The ID argument must be a valid rack server account name.

Viewing Servers Associated with a Hardware Profile

Objective

This task allows the user to query the list of servers that are associated with a specific hardware profile.

Prerequisites

None.

REST URL

```
/cloupia/api-v2/AssociatedServersByPolicyName
```

Components

The parameters of the AssociatedServersByPolicyName API are:

- Boolean Modular—Cisco UCS S3260 server
- String Non-modular Hardware Policy—The name of the hardware policy that is for non-modular servers.
- String Modular Hardware Policy—The name of the hardware policy that is for a modular server.

Sample Input XML

```
<cuicOperationRequest>
<operationType>SERVER_PROFILES</operationType>
<payload>
<![CDATA[
<AssociatedServersByPolicyName>
<modular>false</modular>

   <!-- Set this value only when modular not equals to true  -->
<nonmodularPolicies>CIMC52(BIOS Policy)</nonmodularPolicies>

   <!-- Set this value only when modular not equals to false  -->
<modularPolicies></modularPolicies>

</AssociatedServersByPolicyName>

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

Implementation

Specifying a hardware policy is mandatory.

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 68

Updating a Rack Group, on page 69

Deleting a Rack Group, on page 70

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></description>
<groupName>colusa</groupName></CIMCRackGroup><CIMCRackGroup><actionId>0</actionId>
<configEntryId>0</configEntryId><defaultGroup>false</defaultGroup><description></description>
<groupName>eseries</groupName></CIMCRackGroup><CIMCRackGroup><actionId>0</actionId>
<configEntryId>0</configEntryId><defaultGroup>false</defaultGroup>
<description>Test Rack Group 1</description>
<groupName>TestGroup</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 68

Updating a Rack Group, on page 69

Deleting a Rack Group, on page 70

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[
<CIMCRackGroup>
<groupName></groupName>

<description></description>

</CIMCRackGroup>

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

Implementation

Group name is mandatory and must be unique.

See Also

Creating a Rack Group, on page 68

Reading All Rack Groups, on page 68

Deleting a Rack Group, on page 70

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[
<CIMCRackGroup>
<groupNames></groupNames>

<deleteRackAccountsInGroup>false</deleteRackAccountsInGroup>

</CIMCRackGroup>

]]>
</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 68

Reading All Rack Groups, on page 68

Updating a Rack Group, on page 69

Creating a Rack Account

Objective

This task allows user to create a rack account.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCInfraAccount
```

Components

The parameters of the RACK_ACCOUNT_CREATE API are:

- String accountName—The account name.
- String server—Optional. The server name.
- String description—Optional. The description of the account.
- Boolean credentialPolicy—Optional. Create a credential policy.
- String policy—The policy name.
- String username—The server login name.
- String password—The server login password.
- String protocol—Optional. Port for the configuration.
- String port—The port number.
- Boolean acceptCertificate—Optional. The option to accept certificate.
- String rackGroup—The name of the rack group.
- String contact—Optional. The contact number.
- String location—Optional. The location address.

Sample Input XML

```
<cuicOperationRequest>
<operationType>RACK_ACCOUNT_CREATE</operationType>
<payload>
<![CDATA[<CIMCInfraAccount>
<accountName></accountName>
<server></server>
<description></description>
<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>
<rackGroup>apitest-ren</rackGroup>
<contact></contact>
<location></location>
</CIMCInfraAccount>]]>
</payload>
</cuicOperationRequest>
```

Implementation

Account name is mandatory and must be unique. ServerIP is mandatory. Username/Password are mandatory.

See Also

Updating a Rack Account, on page 72

Deleting a Rack Account, on page 73

Updating a Rack Account

Objective

This task allows the user to update an existing rack account.

Prerequisites

None

REST URL

`/cloupia/api-v2/CIMCInfraAccount`

Components

The parameters of the RACK_ACCOUNT_UPDATE API are:

- String accountName—The account name.
- String server—Optional. The server name.
- String description—Optional. The description of the account.
- Boolean credentialPolicy—Optional. Create a credential policy.
- String policy—The policy name.
- String username—The server login name.
- String password—The server login password.
- String protocol—Optional. Port for the configuration.

- String port—The port number.
- Boolean acceptCertificate—Optional. The option to accept certificate.
- String rackGroup—The name of the rack group.
- String contact—Optional. The contact number.
- String location—Optional. The location address.

Sample Input XML

```
<cuicOperationRequest><operationType>RACK_ACCOUNT_UPDATE</operationType><payload>
<![CDATA[<CIMCInfraAccount><accountName></accountName><server></server>
<description></description>
<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><rackGroup>apitest-ren</rackGroup><contact></contact><location></location>
</CIMCInfraAccount>]]>
</payload>
</cuicOperationRequest>
```

Implementation

ServerIP cannot be changed.

See Also

Creating a Rack Account, on page 71

Deleting a Rack Account, on page 73

Deleting a Rack Account

Objective

This task allows user to delete one or more existing rack accounts.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCInfraAccount
```

Components

The parameters of the RACK_ACCOUNT_DELETE API are:

- String devices—The account to delete.

Sample Input XML

```
<cuicOperationRequest><operationType>RACK_ACCOUNT_DELETE</operationType>
<payload>
```

```
<![CDATA[<CIMCInfraAccount>
<devices></devices></CIMCInfraAccount>]]>
</payload>
</cuicOperationRequest>
```

Implementation

Comma separated list of account names, all of which must be valid existing rack accounts.

See Also

Creating a Rack Account, on page 71

Updating a Rack Account, on page 72

Running Server Inventory

Objective

This task allows user to run inventory on one or more servers.

Prerequisites

None

REST URL

```
/cloupia/api-v2/RunInventory
```

Components

The parameters of the RUN_INVENTORY API are:

- String inventoryLevel—Optional. The inventory on rack account or rack group.
- String serverGroups—The rack groups.
- String servers—Optional. The rack server.

Sample Input XML

```
<cuicOperationRequest><operationType>RUN_INVENTORY</operationType>
<payload>
<![CDATA[
<RunInventory>
<inventoryLevel>RACK GROUP</inventoryLevel>
<!-- Set this value only when inventoryLevel not equals to RACK ACCOUNT  -->
<serverGroups></serverGroups>
<!-- Set this value only when inventoryLevel not equals to RACK GROUP  -->
<servers></servers></RunInventory>]]>
</payload>
</cuicOperationRequest>
```

Implementation

Comma separated list of account names, all of which must be valid existing rack accounts or comma separated list of rack groups, all of which must be valid existing rack groups.

Testing Server Connection

Objective

This task allows user to test connection to one or more servers.

Prerequisites

None

REST URL

```
/cloupia/api-v2/TestConnection
```

Components

The parameters of the TEST_CONNECTION API are:

- String devices—The rack account to test connection.

Sample Input XML

```
<cuicOperationRequest><operationType>TEST_CONNECTION</operationType>
<payload>
<![CDATA[<TestConnection><devices></devices></TestConnection>]]>
</payload>
</cuicOperationRequest>
```

Implementation

Account name is mandatory.

Assigning Rack Groups to Servers

Objective

This task allows user to assign rack group to one or more servers.

Prerequisites

None

REST URL

```
/cloupia/api-v2/AssignRackGroup
```

Components

The parameters of the ASSIGN_RACK_GROUP API are:

- String servers—The rack account to assign to a rack group.
- String serverGroup —The rack server group.

Sample Input XML

```
<cuicOperationRequest>
<operationType>ASSIGN_RACK_GROUP</operationType>
<payload><![CDATA[<AssignRackGroup><servers></servers>
<serverGroup></serverGroup></AssignRackGroup>]]>
</payload>
</cuicOperationRequest>
```

Implementation

Comma separated list of account names, all of which must be valid existing rack accounts. Rack group is mandatory.

Running Server Diagnostics

Objective

This task allows user to run diagnostics on one or more servers.

Prerequisites

SCU image location and SCP User password are configured.

REST URL

```
/cloupia/api-v2/RunServerDiagnostics
```

Components

The parameters of the RUN_SERVER_DIAGNOSTICS API are:

- String selectProfile—The server profile.
- String diagLevel—The server or rack group to run diagnostics.
- String serverGroups—The rack server group.
- String servers—The rack server.

Sample Input XML

```
<cuicOperationRequest>
<operationType>RUN_SERVER_DIAGNOSTICS</operationType>
<payload>
<![CDATA[
<CIMCDiagnosticsRunConfig>
<selectProfile></selectProfile>

<servers></servers>

</CIMCDiagnosticsRunConfig>

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

Implementation

The servers argument must consist of a comma-separated list of IDs. Each ID format is: {AccountName};{ServerIPAddress}. The **serverGroups** argument must consist of comma separated list of rack groups, all of which must be valid existing rack groups.

See Also

Running Server Diagnostics, on page 76

Deleting Server Diagnostics Report, on page 77

Reading Server Diagnostics Status by Server IP

Objective

This task allows the user to query the status of diagnostics being run on a server based on Server IP.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCDiagnosticsStatusByServerIP/{CIMCDiagnosticsStatusByServerIPId}
```

Implementation

The **CIMCDiagnosticsStatusByServerIPId** argument must be a valid IP address. If no argument is specified, an empty set of results will be returned. The dots in the IP address must be substituted with an underscore.

See Also

Running Server Diagnostics, on page 76

Deleting Server Diagnostics Report, on page 77

Deleting Server Diagnostics Report

Objective

This task allows the user to delete diagnostics report of one or more servers based on Server IP.

Prerequisites

None

REST URL

```
/cloupia/api-v2/DeleteServerDiagnosticsReport
```

Components

The parameters of the DELETE_DIAGNOSTICS_REPORT API are:

- String serverIPs—The diagnostics report to delete.

Sample Input XML

```
<cuicOperationRequest>
<operationType>DELETE_DIAGNOSTICS_REPORT</operationType>
<payload>
<![CDATA[<CIMCDeleteDiagnosticsReportConfig>
<serverIPs></serverIPs></CIMCDeleteDiagnosticsReportConfig>]]>
</payload>
</cuicOperationRequest>
```

Implementation

The serverIP argument must be a valid IP address.

See Also

Reading Server Diagnostics Status by Server IP, on page 76

Running Server Diagnostics, on page 76

Adding Compute Tags

Objective

This task allows the user to add compute tag(s) to a rack server or chassis.

Prerequisites

None

REST URL

```
/cloupia/api-v2/ComputeTags
```

Components

The parameters of the COMPUTE_TAGS_DELETE API are:

- String (optional) physicalComputeType—The compute type.
- String rackServer—The rack server.
- String chassis—The chassis.
- String tags—The tag name.

Sample Input XML

```
<cuicOperationRequest>
<operationType>COMPUTE_TAGS_ADD</operationType>
<payload>
<![CDATA[
<ComputeTags>
<physicalComputeType>Rack Servers</physicalComputeType>

   <!-- Set this value only when physicalComputeType equals to Rack Servers  -->
<rackServer></rackServer>

   <!-- Set this value only when physicalComputeType equals to Chassis  -->
<chassis></chassis>

<tags></tags>

</ComputeTags>

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

Implementation

Rack Server or Chassis is mandatory. Tag Names are mandatory. Tag names are key value pairs separated with ';'. Example:- <TagName1>:<TagValue1>;<TagName2>:<TagValue2>

See Also

Deleting Compute Tags, on page 78

Deleting Compute Tags

Objective

This task allows the user to delete compute tag(s) from a rack server or chassis.

Prerequisites

None

REST URL

```
/cloupia/api-v2/ComputeTags
```

Components

The parameters of the COMPUTE_TAGS_DELETE API are:

- String (optional) physicalComputeType—The compute type.
- String rackServer—The rack server.
- String chassis—The chassis.
- String tags—The tag name.

Sample Input XML

```
<cuicOperationRequest>
<operationType>COMPUTE_TAGS_DELETE</operationType>
<payload>
<![CDATA[
<ComputeTags>
<physicalComputeType>Rack Servers</physicalComputeType>

   <!-- Set this value only when physicalComputeType equals to Rack Servers  -->
<rackServer></rackServer>

   <!-- Set this value only when physicalComputeType equals to Chassis  -->
<chassis></chassis>

<tags></tags>

</ComputeTags>

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

Implementation

Comma separated list of tag names, all of which must be valid existing server tags.

See Also

Adding Compute Tags, on page 78

Creating a Technical Support Log

Objective

This task allows the user to create tech support for a rack servers.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CreateTechSupport
```

Components

The parameters of the CREATE_TECH_SUPPORT API are:

- String rackServers—The rack servers.
- String destination—List of the Destination Types and the Options.
- String option—The option to select network transfer type.
- String server—The IP address or account name of the server on which the support data file should be stored.
- String pathFileName—The path and filename that must be used when exporting the file to the remote server.
- String username—The username the system should use to log in to the remote server.
- String password—The password for the remote server username.

Sample Input XML

```
<cuicOperationRequest>
<operationType>CREATE_TECH_SUPPORT</operationType>
<payload>
<![CDATA[
<CreateTechSupport>
<rackServers></rackServers>
<destination>REMOTE</destination>
<!-- Set this value only when destination not equals to LOCAL  -->
<option>SCP</option>
<!-- Set this value only when destination not equals to LOCAL  -->
<server></server>
<!-- Set this value only when destination not equals to LOCAL  -->
<pathFileName></pathFileName>
<!-- Set this value only when option not equals to TFTP  -->
<username></username>
<!-- Set this value only when option not equals to TFTP  -->
<password></password>
</CreateTechSupport>]]>
</payload>
</cuicOperationRequest>
```

Implementation

Rack servers are mandatory. Destination type is mandatory. If destination type is 'LOCAL' then no other fields are required. If destination type is 'REMOTE' then the fields 'ServerIP/Host name' and 'Path and File name' needs to be entered. The fields 'username' and 'password' are not required if 'Network Type' is 'TFTP'.

See Also

Clearing Technical Support Logs, on page 80

Reading Technical Support Logs by Server IP, on page 81

Clearing Technical Support Logs

Objective

This task allows the user to clear entry for one or more existing technical support logs.

Prerequisites

None

REST URL

```
/cloupia/api-v2/ClearTechSupport
```

Components

The parameters of the CLEAR_TECH_SUPPORT API are:

- String techsupportFileName—The name of the technical support log file.

Sample Input XML

```
<cuicOperationRequest>
<operationType>CLEAR_TECH_SUPPORT</operationType>
<payload>
<![CDATA[
<ClearTechSupport><techSupportFileName></techSupportFileName></ClearTechSupport>]]>
</payload>
</cuicOperationRequest>
```

Implementation

Comma separated list of technical support names, all of which must be valid existing tech support log names.

See Also

Creating a Technical Support Log, on page 79

Reading Technical Support Logs by Server IP, on page 81

Reading Technical Support Logs by Server IP

Objective

This task allows the user to query the technical support log details based on the IP address of a rack server. The **CIMCTechLogSupportStatusByServerIPId** argument must be a valid IP address of a server being managed by Cisco IMC Supervisor.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCTechLogSupportStatusByServerIP/{CIMCTechLogSupportStatusByServerIPId}
```

Implementation

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

See Also

Creating a Technical Support Log, on page 79

Clearing Technical Support Logs, on page 80

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.
- String description—Description of the account.
- String contact—The contact number.
- String location—The location address.
- String rackGroup—The name of the rack group.

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>

<!-- Set this value only when option not equals to CSV  -->
<description></description>

   <!-- Set this value only when option not equals to CSV  -->
<contact></contact>

   <!-- Set this value only when option not equals to CSV  -->
<location></location>

   <!-- Set this value only when option not equals to CSV  -->
<rackGroup>Default Group</rackGroup>

</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 84

Deleting a Discovery Profile, on page 86

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 82

Updating a Discovery Profile, on page 84

Deleting a Discovery Profile, on page 86

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.
- String description—Description of the account.
- String contact—The contact number.
- String location—The location address.
- String rackGroup—The name of the rack group.

Sample Input XML

```
<cuicOperationRequest>
<operationType>DISCOVERY_PROFILE_UPDATE</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>
<!-- Set this value only when option not equals to CSV  -->
<description></description>

   <!-- Set this value only when option not equals to CSV  -->
<contact></contact>

   <!-- Set this value only when option not equals to CSV  -->
<location></location>

   <!-- Set this value only when option not equals to CSV  -->
<rackGroup>Default Group</rackGroup>

</CIMCDeviceDiscoveryConfig>

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

Implementation

Profile Name cannot be modified.

See Also

Creating a Discovery Profile, on page 82

Deleting a Discovery Profile, on page 86

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 profileNames—Optional. The name of the profile.

Sample Input XML

```
<cuicOperationRequest>
<operationType>DISCOVERY_PROFILE_DELETE</operationType>
<payload>
<![CDATA[
<CIMCDeviceDiscoveryConfig>
<profileNames></profileNames>

</CIMCDeviceDiscoveryConfig>

]]>
</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 82

Updating a Discovery Profile, on page 84

Reading a Discovery Profile, on page 84

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

```
/cloupia/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[
<RunServerDiscovery>
<profileNames></profileNames>

<enableSchedule>false</enableSchedule>

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

</RunServerDiscovery>

]]>
</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 88

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/ImportRackServers
```

Components

The parameters of the IMPORT_SERVER API are:

- String devices—The discovered devices.
- String userPrefix—Optional. The prefix for the user.

Sample Input XML

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

<userPrefix></userPrefix>

</ImportRackServers>

]]>
</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 87

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 90

Power On Server, on page 91

Power Off Server, on page 90

Shutdown Server, on page 92

Set Label on Server, on page 93

Toggle Locator LED on Server, on page 94

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 89

Power On Server, on page 91

Power Off Server, on page 90

Shutdown Server, on page 92

Set Label on Server, on page 93

Toggle Locator LED on Server, on page 94

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 89

Power Cycle Server, on page 90

Power On Server, on page 91

Shutdown Server, on page 92

Set Label on Server, on page 93

Toggle Locator LED on Server, on page 94

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 89

Power Cycle Server, on page 90

Power Off Server, on page 90

Shutdown Server, on page 92

Set Label on Server, on page 93

Toggle Locator LED on Server, on page 94

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 90

Power On Server, on page 91

Power Off Server, on page 90

Hard Reset Server, on page 89

Set Label on Server, on page 93

Toggle Locator LED on Server, on page 94

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 90

Power On Server, on page 91

Power Off Server, on page 90

Shutdown Server, on page 92

Hard Reset Server, on page 89

Toggle Locator LED on Server, on page 94

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 90

Power On Server, on page 91

Power Off Server, on page 90

Shutdown Server, on page 92

Set Label on Server, on page 93

Hard Reset Server, on page 89

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 101

Reading Servers by Rack Group, on page 103

Reading Servers by Serial Number, on page 103

Reading Servers by Server IP, on page 102

Reading Servers by Tag Value, on page 95

Reading Servers by UUID, on page 102

Reading Servers by Product ID, on page 100

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 95

Reading Servers by Account Name, on page 101

Reading Servers by Rack Group, on page 103

Reading Servers by Serial Number, on page 103

Reading Servers by Server IP, on page 102

Reading Servers by UUID, on page 102

Reading Servers by Product ID, on page 100

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 97

Reading Server Faults by Fault Code, on page 98

Reading Server Faults by IP Address, on page 96

Reading Server Faults by Severity, on page 97

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 96

Reading Server Faults by Fault Code, on page 98

Reading Server Faults by Account Name, on page 97

Reading Server Faults by Severity, on page 97

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 96

Reading Server Faults by Fault Code, on page 98

Reading Server Faults by IP Address, on page 96

Reading Server Faults by Severity, on page 97

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 96

Reading Server Faults by Fault Code, on page 98

Reading Server Faults by IP Address, on page 96

Reading Server Faults by Account Name, on page 97

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 96

Reading Server Faults by Account Name, on page 97

Reading Server Faults by IP Address, on page 96

Reading Server Faults by Severity, on page 97

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 100

Reading Server Faults History by IP Address, on page 99

Reading Server Faults History by Severity, on page 100

Reading Server Faults History by Account Name, on page 99

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 100

Reading Server Faults History by DN, on page 98

Reading Server Faults History by Severity, on page 100

Reading Server Faults History by Account Name, on page 99

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 100

Reading Server Faults History by DN, on page 98

Reading Server Faults History by Severity, on page 100

Reading Server Faults History by IP Address, on page 99

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 100

Reading Server Faults History by DN, on page 98

Reading Server Faults History by Account Name, on page 99

Reading Server Faults History by IP Address, on page 99

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 100

Reading Server Faults History by DN, on page 98

Reading Server Faults History by Account Name, on page 99

Reading Server Faults History by IP Address, on page 99

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 95

Reading Servers by Account Name, on page 101

Reading Servers by Rack Group, on page 103

Reading Servers by Serial Number, on page 103

Reading Servers by Server IP, on page 102

Reading Servers by UUID, on page 102

Reading Servers by Tag Value, on page 95

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 95

Reading Servers by Tag Value, on page 95

Reading Servers by Rack Group, on page 103

Reading Servers by Serial Number, on page 103

Reading Servers by Server IP, on page 102

Reading Servers by UUID, on page 102

Reading Servers by Product ID, on page 100

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 95

Reading Servers by Tag Value, on page 95

Reading Servers by Account Name, on page 101

Reading Servers by Rack Group, on page 103

Reading Servers by Serial Number, on page 103

Reading Servers by Server IP, on page 102

Reading Servers by Product ID, on page 100

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 95

Reading Servers by Account Name, on page 101

Reading Servers by Rack Group, on page 103

Reading Servers by Serial Number, on page 103

Reading Servers by Server IP, on page 102

Reading Servers by UUID, on page 102

Reading Servers by Product ID, on page 100

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 95

Reading Servers by Tag Value, on page 95

Reading Servers by Account Name, on page 101

Reading Servers by Rack Group, on page 103

Reading Servers by Server IP, on page 102

Reading Servers by Product ID, on page 100

Reading Servers by UUID, on page 102

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 95

Reading Servers by Tag Value, on page 95

Reading Servers by Account Name, on page 101

Reading Servers by Server IP, on page 102

Reading Servers by Serial Number, on page 103

Reading Servers by Product ID, on page 100

Reading Servers by UUID, on page 102

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 104

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 104

Reading Server Utilization by Account Name

Objective

Get Server Utilization By Account Name

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCServerUtilizationByAccountName/{CIMCServerUtilizationByAccountNameId}
```

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 105

Reading Server Utilization by Server IP

Objective

Get Server Utilization By IP Address.

Prerequisites

None

REST URL

```
/cloupia/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 104

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 106

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 105

Reading Server Utilization History by Days

Objective

This task allows the user to query the server utilization history based on the last N days. The **CIMCServerUtilizationHistoryByDaysId** argument must be a number between 1 and 180.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCServerUtilizationHistoryByDays/{CIMCServerUtilizationHistoryByDaysId}
```

Implementation

The **CIMCServerUtilizationHistoryByDaysId** argument must be a number between 1 and 180.

See Also

Reading Server Utilization History by Account Name, on page 105

Reading Server Utilization History by Server IP, on page 106

Reading Server Utilization History by Days for a Server using Account Name

Objective

This task allows the user to query the server utilization history based on the last N days for a specific server, based on account name. The **CIMCServerUtilizationHistoryByDaysId** argument must be a number between 1 and 180. The **AccountNameId** argument must be a valid account name of a server being managed by Cisco IMC Supervisor.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCServerUtilizationHistoryByDays/{CIMCServerUtilizationHistoryByDaysId}
/AccountName/{AccountNameId}
```

Implementation

The **CIMCServerUtilizationHistoryByDaysId** argument must be a number between 1 and 180. The **AccountNameId** argument must be a valid account name of a server being managed by Cisco IMC Supervisor.

See Also

Reading Server Utilization History by Days for a Server using Server IP, on page 107

Reading Server Utilization History by Days for a Server using Server IP

Objective

This task allows the user to query the server utilization history based on the last N days for a specific server, based on server IP. The **CIMCServerUtilizationHistoryByDaysId** argument must be a number between 1 and 180. The **ServerIPId** argument must be a valid IP address of a server being managed by Cisco IMC Supervisor.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCServerUtilizationHistoryByDays/{CIMCServerUtilizationHistoryByDaysId}
/ServerIP/{ServerIPId}
```

Implementation

The **CIMCServerUtilizationHistoryByDaysId** argument must be a number between 1 and 180. The **ServerIPId** 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 Days for a Server using Account Name, on page 106

Mapping Host Image

Objective

This task allows the user to apply a host image profile on the E-Series server configured in the system which will download the image you entered in the selected servers.

Prerequisites

One or more E-series server must be configured as Rack Accounts.

REST URL

```
/cloupia/api-v2/HostImageMap
```

Components

The parameters of the MAP_HOST_IMAGE API are:

- String ServerIdKey—The server key.
- String imageName—The name of the image that you want to map.

Sample Input XML

```
<cuicOperationRequest>
<operationType>MAP_HOST_IMAGE</operationType>
<payload>
<![CDATA[
<HostImageMap>
<serverIdKey></serverIdKey>

<imageName></imageName>

</HostImageMap>

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

Implementation

ServerIdKey format: {AccountName};{ServerIPAddress}.

Unmapping Host Image

Objective

This task allows the user to unmap an image on the E-Series server configured in the system.

Prerequisites

One or more E _series server must be configured as Rack Accounts.

REST URL

```
/cloupia/api-v2/UnmapHostImageMap
```

Components

The parameters of the UNMAP_HOST_IMAGE API are:

- String ServerIdKey—The server key

Sample Input XML

```
<cuicOperationRequest>
<operationType>UNMAP_HOST_IMAGE</operationType>
<payload>
<![CDATA[
<UnmapHostImageMap>
<serverIdKey></serverIdKey>

</UnmapHostImageMap>

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

Implementation

ServerIdKey format: {AccountName};{ServerIPAddress}.

Deleting Host Image

Objective

This task allows you to delete an image on the E-Series Server configured in the system.

Prerequisites

One or more E _series server must be configured as Rack Accounts.

REST URL

```
/cloupia/api-v2/DeleteHostImageMap
```

Components

The parameters of the DELETE_HOST_IMAGE API are:

- String ServerIdKey—The server key
- String imageNames—The image name that you want to delete.

Sample Input XML

```
<cuicOperationRequest>
<operationType>DELETE_HOST_IMAGE</operationType>
<payload>
<![CDATA[
<DeleteHostImageMap>
<serverIdKey></serverIdKey>

<imageNames></imageNames>

</DeleteHostImageMap>

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

Implementation

ServerIdKey format: {AccountName};{ServerIPAddress} is a mandatory field. imageNames is a mandatory field and can be comma (,) separated value.

Creating an HCL Profile

Objective

This task allows the user to create a Hardware Compatibility List (HCL) profile on selected rack server(s) configured in the system.

Prerequisites

One or more Servers must be configured as Rack Accounts.

REST URL

```
/cloupia/api-v2/CIMCManageHCLProfileConfig/{CIMCHCLReportByProfileNameId}
```

Components

The parameters of the HCL_PROFILE_CREATE API are:

- String profileName—Name of the profile.

- String server—The HCL server.
- String hclReportData—The HCL report data.

Sample Input XML

```
<cuicOperationRequest>
<operationType>HCL_PROFILE_CREATE</operationType>
<payload>
<![CDATA[
<CIMCManageHCLProfileConfig>
<profileName></profileName>

<server></server>

</CIMCManageHCLProfileConfig>

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

Implementation

The Select Profile argument is mandatory and must be unique. The Server(s) argument must consist of a comma-separated list of Ids. Each Id is of the format: {AccountName};{ServerIPAddress}.

See Also

- Setting HCL OS Tag on Servers or Rack Groups , on page 111
- Modifying an HCL Profile, on page 110
- Deleting HCL Profile , on page 114
- Deleting HCL OS Tag on Servers or Rack Groups, on page 113
- Reading HCL OS Tag by Server IP , on page 115
- Reading HCL OS Versions by Vendor Name, on page 116
- Reading HCL Report by Profile Name, on page 116
- Reading HCL Report by Rack Group, on page 117
- Reading HCL Report by Server IP, on page 117

Modifying an HCL Profile

Objective

This task allows the user to modify a Hardware Compatibility List (HCL) profile on selected rack server(s) configured in the system.

Prerequisites

One or more servers must be configured as Rack Accounts.

REST URL

```
/cloupia/api-v2/CIMCModifyHCLProfileConfig
```

Components

The parameters of the HCL_PROFILE_UPDATE API are:

- String profileName—Name of the profile.
- String server—The server.

Sample Input XML

```
<cuicOperationRequest>
<operationType>HCL_PROFILE_UPDATE</operationType>
<payload>
<![CDATA[
<CIMCModifyHCLProfileConfig>
<profileName></profileName>

<server></server>

</CIMCModifyHCLProfileConfig>

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

Implementation

The Select Profile argument is mandatory and must be existing. The Server(s) argument must consist of a comma-separated list of Ids. Each Id is of the format: {AccountName};{ServerIPAddress}.

See Also

- Creating an HCL Profile, on page 109
- Setting HCL OS Tag on Servers or Rack Groups , on page 111
- Deleting HCL Profile , on page 114
- Deleting HCL OS Tag on Servers or Rack Groups, on page 113
- Reading HCL OS Tag by Server IP , on page 115
- Reading HCL OS Versions by Vendor Name, on page 116
- Reading HCL Report by Profile Name, on page 116
- Reading HCL Report by Rack Group, on page 117
- Reading HCL Report by Server IP, on page 117

Setting HCL OS Tag on Servers or Rack Groups

Objective

This task allows you to perform a Set OS Tag action on rack servers or rack groups configured in the system.

Prerequisites

One or more Servers must be configured as Rack Accounts.

REST URL

```
/cloupia/api-v2/SetHCLOSTag
```

Components

The parameters of the CREATE API are:

- String tagLevel—The tag level.
- String serverGroups—The rack group.
- String servers—The server.
- String os—A valid OS vendor name.
- String osVersion—A valid OS version name.

Sample Input XML

```
<cuicOperationRequest>
<payload>
<![CDATA[
<SetHCLOSTag>
<tagLevel>SERVERGROUP</tagLevel>

   <!-- Set this value only when tagLevel not equals to SERVER  -->
<serverGroups></serverGroups>

   <!-- Set this value only when tagLevel not equals to SERVERGROUP  -->
<servers></servers>

<os></os>

<osVersion></osVersion>

</SetHCLOSTag>

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

Implementation

- Choose argument must either be Server or ServerGroup.
- The Server(s) argument must consist of a comma-separated list of Ids. Each Id is of the format: {AccountName};{ServerIPAddress}.
- The Server Group(s) argument must consist of a comma-separated list of Rack Group names.
- The Operating System argument must be a valid OS vendor name.
- The Operating System Version argument must be a valid OS version name.

See Also

- Creating an HCL Profile, on page 109
- Modifying an HCL Profile, on page 110
- Deleting HCL Profile , on page 114
- Deleting HCL OS Tag on Servers or Rack Groups, on page 113

- Reading HCL OS Tag by Server IP , on page 115
- Reading HCL OS Versions by Vendor Name, on page 116
- Reading HCL Report by Server IP, on page 117
- Reading HCL Report by Profile Name, on page 116
- Reading HCL Report by Rack Group, on page 117

Deleting HCL OS Tag on Servers or Rack Groups

Objective

This task allows the user to perform delete OS Tag action on rack servers or rack groups configured in the system.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCHCLTagByServerIP/{serverIP}
```

Components

The parameters of the HCL_TAG_DELETE API are:

- String tagLevel—The tag level.
- String serverGroups—The rack group.
- String servers—The server.

Sample Input XML

```
<cuicOperationRequest>
<operationType>HCL_TAG_DELETE</operationType>
<payload>
<![CDATA[
<DeleteHCLOSTag>
<tagLevel>SERVERGROUP</tagLevel>

   <!-- Set this value only when tagLevel not equals to SERVER  -->
<serverGroups></serverGroups>

   <!-- Set this value only when tagLevel not equals to SERVERGROUP  -->
<servers></servers>

</DeleteHCLOSTag>

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

Implementation

- Choose argument must either be Server or ServerGroup.
- The Server(s) argument must consist of a comma-separated list of Ids. Each Id is of the format: {AccountName};{ServerIPAddress}.

- The Server Group(s) argument must consist of a comma-separated list of Rack Group names.

See Also

- Creating an HCL Profile, on page 109
- Setting HCL OS Tag on Servers or Rack Groups , on page 111
- Modifying an HCL Profile, on page 110
- Deleting HCL Profile , on page 114
- Reading HCL OS Tag by Server IP , on page 115
- Reading HCL OS Versions by Vendor Name, on page 116
- Reading HCL Report by Profile Name, on page 116
- Reading HCL Report by Rack Group, on page 117
- Reading HCL Report by Server IP, on page 117

Deleting HCL Profile

Objective

This task allows the user to delete a Hardware Compatibility List (HCL) profile configured in the system.

Prerequisites

None

REST URL

```
/cloupia/api-v2/DeleteHCLProfileConfig
```

Components

The parameters of the HCL_PROFILE_DELETE API are:

- String profileName—Name of the profile.

Sample Input XML

```
<cuicOperationRequest>
<operationType>HCL_PROFILE_DELETE</operationType>
<payload>
<![CDATA[
<DeleteHCLProfileConfig>
<profileName></profileName>

</DeleteHCLProfileConfig>

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

Implementation

The Select Profile argument is mandatory and must be existing.

See Also

- Setting HCL OS Tag on Servers or Rack Groups , on page 111
- Creating an HCL Profile, on page 109
- Modifying an HCL Profile, on page 110
- Deleting HCL OS Tag on Servers or Rack Groups, on page 113
- Reading HCL OS Tag by Server IP , on page 115
- Reading HCL OS Versions by Vendor Name, on page 116
- Reading HCL Report by Profile Name, on page 116
- Reading HCL Report by Rack Group, on page 117
- Reading HCL Report by Server IP, on page 117

Reading HCL OS Tag by Server IP

Objective

This task allows the user to retrieve OS Tag based on the IP address of the server.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCHCLTagByServerIP/{serverIP}
```

Implementation

The serverIP 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

- Setting HCL OS Tag on Servers or Rack Groups , on page 111
- Creating an HCL Profile, on page 109
- Modifying an HCL Profile, on page 110
- Deleting HCL Profile , on page 114
- Deleting HCL OS Tag on Servers or Rack Groups, on page 113
- Reading HCL OS Versions by Vendor Name, on page 116
- Reading HCL Report by Profile Name, on page 116
- Reading HCL Report by Rack Group, on page 117
- Reading HCL Report by Server IP, on page 117

Reading HCL OS Versions by Vendor Name

Objective

This task allows the user to retrieve OS Versions based on the Vendor Name provided as input.

Prerequisites

None

REST URL

```
/cloupia/api-v2/HCLOSVersionsByVendorName/{osVendor}
```

Implementation

The serverIP argument must be a valid IP Vendor Name available in Cisco IMC Supervisor.

See Also

- Setting HCL OS Tag on Servers or Rack Groups , on page 111
- Creating an HCL Profile, on page 109
- Modifying an HCL Profile, on page 110
- Deleting HCL Profile , on page 114
- Deleting HCL OS Tag on Servers or Rack Groups, on page 113
- Reading HCL OS Tag by Server IP , on page 115
- Reading HCL Report by Profile Name, on page 116
- Reading HCL Report by Rack Group, on page 117
- Reading HCL Report by Server IP, on page 117

Reading HCL Report by Profile Name

Objective

This task allows the user to retrieve HCL Report based on the Profile Name.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCHCLReportByProfileName/{CIMCHCLReportByProfileNameId}
```

Implementation

The CIMCHCLReportByProfileNameId argument must be a valid Profile Name.

See Also

- Setting HCL OS Tag on Servers or Rack Groups , on page 111
- Creating an HCL Profile, on page 109
- Modifying an HCL Profile, on page 110

- Deleting HCL Profile , on page 114
- Deleting HCL OS Tag on Servers or Rack Groups, on page 113
- Reading HCL OS Tag by Server IP , on page 115
- Reading HCL OS Versions by Vendor Name, on page 116
- Reading HCL Report by Rack Group, on page 117
- Reading HCL Report by Server IP, on page 117

Reading HCL Report by Rack Group

Objective

This task allows the user to retrieve HCL Report based on the rack group name.

Prerequisites

One or more Servers must be configured as Rack Accounts.

REST URL

```
/cloupia/api-v2/CIMCServerHCLReportByRackGroup/{CIMCServerHCLReportByRackGroupId}
```

Implementation

The CIMCServerHCLReportByRackGroupId argument must be a valid Rack Group name managed by Cisco IMC Supervisor. The dots in the IP address need to be substituted with an underscore.

See Also

- Setting HCL OS Tag on Servers or Rack Groups , on page 111
- Creating an HCL Profile, on page 109
- Modifying an HCL Profile, on page 110
- Deleting HCL Profile , on page 114
- Deleting HCL OS Tag on Servers or Rack Groups, on page 113
- Reading HCL OS Tag by Server IP , on page 115
- Reading HCL OS Versions by Vendor Name, on page 116
- Reading HCL Report by Profile Name, on page 116
- Reading HCL Report by Server IP, on page 117

Reading HCL Report by Server IP

Objective

This task allows the user to retrieve HCL Report based on the IP address of the server.

Prerequisites

One or more Servers must be configured as Rack Accounts.

REST URL

```
/cloupia/api-v2/CIMCServerHCLReportByServerIP/{CIMCServerHCLReportByServerIPId}
```

Implementation

The CIMCServerHCLReportByServerIPId 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

- Setting HCL OS Tag on Servers or Rack Groups , on page 111
- Creating an HCL Profile, on page 109
- Modifying an HCL Profile, on page 110
- Deleting HCL Profile , on page 114
- Deleting HCL OS Tag on Servers or Rack Groups, on page 113
- Reading HCL OS Tag by Server IP , on page 115
- Reading HCL OS Versions by Vendor Name, on page 116
- Reading HCL Report by Profile Name, on page 116
- Reading HCL Report by Rack Group, on page 117

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

```
<cuicOperationRequest>
<payload>
<![CDATA[
<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>
]]>
</payload>
</cuicOperationRequest>
```

Implementation

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

See Also

Updating a User Group , on page 120

Deleting a User Group, on page 121

Enabling All Users in a Group, on page 122

Disabling All Users in a Group, on page 123

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 centr 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 118

Deleting a User Group, on page 121

Enabling All Users in a Group, on page 122

Disabling All Users in a Group, on page 123

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 118

Updating a User Group , on page 120

Enabling All Users in a Group, on page 122

Disabling All Users in a Group, on page 123

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 118

Updating a User Group , on page 120

Deleting a User Group, on page 121

Disabling All Users in a Group, on page 123

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 118

Deleting a User Group, on page 121

Updating a User Group , on page 120

Enabling All Users in a Group, on page 122

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>

<!-- Accepts value from the list: locale-->
<locale>en_US</locale>

</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 125

Updating a User , on page 126

Deleting a User, on page 127

Enabling a User, on page 128

Disabling a User, on page 129

Updating a User Expiry Date, on page 130

Updating a User Password, on page 131

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 123

Updating a User , on page 126

Deleting a User, on page 127

Enabling a User, on page 128

Disabling a User, on page 129

Updating a User Expiry Date, on page 130

Updating a User Password, on page 131

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>

<!-- Accepts value from the list: locale-->
<locale>en_US</locale>
</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 123

Reading a User, on page 125

Deleting a User, on page 127

Enabling a User, on page 128

Disabling a User, on page 129

Updating a User Expiry Date, on page 130

Updating a User Password, on page 131

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 123

Reading a User, on page 125

Updating a User , on page 126

Enabling a User, on page 128

Disabling a User, on page 129

Updating a User Expiry Date, on page 130

Updating a User Password, on page 131

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 123

Reading a User, on page 125

Updating a User , on page 126

Deleting a User, on page 127

Disabling a User, on page 129

Updating a User Expiry Date, on page 130

Updating a User Password, on page 131

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 123

Reading a User, on page 125

Updating a User , on page 126

Deleting a User, on page 127

Enabling a User, on page 128

Updating a User Expiry Date, on page 130

Updating a User Password, on page 131

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 123

Reading a User, on page 125

Updating a User , on page 126

Deleting a User, on page 127

Enabling a User, on page 128

Disabling a User, on page 129

Updating a User Password, on page 131

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 123

Reading a User, on page 125

Updating a User , on page 126

Deleting a User, on page 127

Enabling a User, on page 128

Disabling a User, on page 129

Updating a User Expiry Date, on page 130