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.

Sample Input XML

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

<platform></platform>

<networkServerType>NFS</networkServerType>

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

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

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

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

<nwPathUserName></nwPathUserName>

<nwPathPassword></nwPathPassword>

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

</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.

See Also

Updating Firmware Network Image, on page 3

Deleting Firmware Image Profile, on page 8

Updating Firmware Network Image

Objective

Update a firmware image in a network location.

Prerequisites

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

REST URL

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

Components

The parameters of the NETWORK_IMAGE_UPDATE API are:

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

Sample Input XML

```
<cuicOperationRequest>
<operationType>NETWORK_IMAGE_UPDATE</operationType>
<payload>
<![CDATA[
<NetworkImage>
<imageId></imageId>

<platform></platform>

<networkServerType>NFS</networkServerType>

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

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

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

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

<nwPathUserName></nwPathUserName>

<nwPathPassword></nwPathPassword>

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

</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.

See Also

Creating a Firmware Network Image, on page 1

Deleting Firmware Image Profile, on page 8

Finding Firmware Image

Objective

Find a firmware image on cisco.com.

Prerequisites

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

REST URL

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

Components

The parameters of the LOCAL_IMAGE_FIND API are:

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

Sample Input XML

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

<username></username>

<password></password>

<enableProxy>false</enableProxy>

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

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

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

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

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

</LocalImage>

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

Implementation

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

See Also

Creating a Firmware Local Image, on page 6

Creating a Firmware Local Image

Objective

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

Prerequisites

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

REST URL

```
/cloupia/api-v2/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 username—ISO share login user name.
- String password—ISO share login password.
- String availableImage—The available .iso image.
- boolean enableProxy—Optional. Enable proxy configuration.
- String host—The host name for the proxy configuration.
- String port—Port for the proxy configuration.
- boolean enableProxyAuth—Optional. Enable proxy authentication.
- String proxyAuthUserName—Proxy username for the proxy authentication.
- String proxyAuthPassword—Password for the proxy username.
- boolean acceptLicense—Accept license agreement.
- boolean downloadNow—download the .iso image immediately after adding a profile.

Sample Input XML

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

<platform></platform>

<username></username>

<password></password>

<availableImage></availableImage>

<enableProxy>false</enableProxy>

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

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

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

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

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

<acceptLicense>false</acceptLicense>

<downloadNow>false</downloadNow>

</LocalImage>

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

Implementation

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

See Also

Finding Firmware Image, on page 4

Downloading Firmware Local Image

Objective

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

Prerequisites

The firmware image profile must be already configured.

REST URL

```
/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 6

Deleting Firmware Image Profile, on page 8

Deleting Firmware Image Profile

Objective

Delete one or more existing firmware image profiles.

Prerequisites

None

REST URL

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

Components

The parameters of the FIRMWARE_IMAGE_DELETE API are:

- String profileId—The unique ID of the profile.

Sample Input XML

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

</DeleteFirmwareImage>

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

Implementation

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

See Also

Creating a Firmware Local Image, on page 6

Creating a Firmware Network Image, on page 1

Updating Firmware Network Image, on page 3

Running Firmware Upgrade

Objective

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

Prerequisites

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

REST URL

```
/cloupia/api-v2/RunFirmwareUpgrade
```

Components

The parameters of the RUN_FIRMWARE_UPGRADE API are:

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

Sample Input XML

```
<cuicOperationRequest>
<operationType>RUN_FIRMWARE_UPGRADE</operationType>
<payload>
<![CDATA[
<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 12

Reading Firmware Upgrade Status by IP Address, on page 13

Reading Firmware Image by a Profile Name

Objective

Get Firmware Image By Profile Name

Prerequisites

None

REST URL

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

Implementation

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

See Also

Reading Firmware Image by Platform, on page 11

Reading Firmware Image by Type, on page 11

Reading Firmware Image by Type

Objective

Get firmware image by type.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCFirmwareImageByType/{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 11

Reading Firmware Image by a Profile Name, on page 10

Reading Firmware Image by Platform

Objective

Get firmware image by platform.

Prerequisites

None

REST URL

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

Implementation

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

See Also

Reading Firmware Image by a Profile Name, on page 10

Reading Firmware Image by Type, on page 11

Reading Download Status by Profile Name

Objective

Image download status by profile name.

Prerequisites

None

REST URL

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

Implementation

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

See Also

Downloading Firmware Local Image, on page 7

Reading Firmware Upgrade Status by Profile Name

Objective

Firmware upgrade status by profile name.

Prerequisites

None

REST URL

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

Implementation

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

See Also

Running Firmware Upgrade, on page 9

Reading Firmware Upgrade Status by IP Address, on page 13

Reading Firmware Upgrade Status by IP Address

Objective

Firmware upgrade status by server IP address.

Prerequisites

None

REST URL

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

Implementation

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

See Also

Running Firmware Upgrade, on page 9

Reading Firmware Upgrade Status by Profile Name, on page 12

Managing Platform Tasks

Overview

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

Creating an Email Alert Rule

Objective

Create an email alert rule for notification of faults.

Prerequisites

None

REST URL

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

Components

The parameters of the EMAIL_ALERT_RULE_CREATE API are:

- String name—The name for the email alert.
- String alertLevel—The alert level.
- String serverGroups—Optional. The server groups 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>
</cuicOperationRequest>
```

Implementation

Rule name cannot be modified.

See Also

Reading an Email Alert Rule

Creating an Email Alert Rule

Deleting Email Alert Rules

Deleting Email Alert Rules

Objective

Delete one or more existing Email Alert Rules.

Prerequisites

None

REST URL

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

Components

String emailAlertRule—The email alert rule.

Sample Input XML

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

</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 20
Updating a Schedule, on page 21
Deleting Schedules, on page 22
Enabling Schedules, on page 23
Disabling Schedules, on page 24

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 19
Updating a Schedule, on page 21
Deleting Schedules, on page 22
Enabling Schedules, on page 23
Disabling Schedules, on page 24
Reading Schedules by Type, on page 25

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 19

Reading Schedules, on page 20

Deleting Schedules, on page 22

Enabling Schedules, on page 23

Disabling Schedules, on page 24

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 scheduleName—Name of the schedule task.

Sample Input XML

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

Implementation

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

See Also

Creating Schedules, on page 19
Reading Schedules, on page 20
Updating a Schedule, on page 21
Enabling Schedules, on page 23
Disabling Schedules, on page 24

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 scheduleName—Name of the schedule task.

Sample Input XML

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

Implementation

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

See Also

Creating Schedules, on page 19
Reading Schedules, on page 20
Updating a Schedule, on page 21
Deleting Schedules, on page 22
Disabling Schedules, on page 24

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 scheduleName—Name of the schedule task.

Sample Input XML

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

Implementation

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

See Also

Creating Schedules, on page 19
Reading Schedules, on page 20
Updating a Schedule, on page 21
Deleting Schedules, on page 22
Enabling Schedules, on page 23

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 19
Updating a Schedule, on page 21
Deleting Schedules, on page 22
Enabling Schedules, on page 23
Disabling Schedules, on page 24

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 26

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 25

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 27

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 26

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 28

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 27

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 29

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 28

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 policyNames—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>
<policyNames></policyNames>

<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 31

Applying Policy on Servers, on page 32

Deleting Policies, on page 33

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 policyNames—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>
<policyNames></policyNames>

<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 29

Applying Policy on Servers, on page 32

Deleting Policies, on page 33

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 policyNames—The name of the policy to apply.
- String servers—The servers 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>
<policyNames></policyNames>
<policyLevel>SERVERGROUP</policyLevel> <!-- Set this value only when policyLevel not
equals to
SERVER  -->
<serverGroups></serverGroups> <!-- Set this value only when policyLevel not equals to
 SERVERGROUP-->
<servers></servers>
<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 33

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 32

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 35

Reading FlexFlash Policy, on page 41

Reading LDAP Policy, on page 46

Reading Legacy Boot Order Policy, on page 47

Reading Network Security Policy, on page 44

Reading NTP Policy, on page 38

Reading Precision Boot Order Policy, on page 36

Reading RAID Policy, on page 43

Reading Serial Over LAN Policy, on page 48

Reading SNMP Policy, on page 40

Reading SSH Policy, on page 39

Reading User Policy, on page 42

Reading VIC Adapter Policy, on page 37

Reading Virtual KVM Policy, on page 49

Reading vMedia Policy, on page 45

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 41

Reading IPMI Over LAN Policy, on page 34

Reading LDAP Policy, on page 46

Reading Legacy Boot Order Policy, on page 47

Reading Network Security Policy, on page 44

Reading NTP Policy, on page 38

Reading Precision Boot Order Policy, on page 36

Reading RAID Policy, on page 43

Reading Serial Over LAN Policy, on page 48

Reading SNMP Policy, on page 40

Reading SSH Policy, on page 39

Reading User Policy, on page 42

Reading VIC Adapter Policy, on page 37

Reading Virtual KVM Policy, on page 49

Reading vMedia Policy, on page 45

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 35

Reading FlexFlash Policy, on page 41

Reading IPMI Over LAN Policy, on page 34

Reading LDAP Policy, on page 46

Reading Legacy Boot Order Policy, on page 47

Reading Network Security Policy, on page 44

Reading NTP Policy, on page 38

Reading RAID Policy, on page 43

Reading Serial Over LAN Policy, on page 48

Reading SNMP Policy, on page 40

Reading SSH Policy, on page 39

Reading User Policy, on page 42

Reading VIC Adapter Policy, on page 37

Reading Virtual KVM Policy, on page 49

Reading vMedia Policy, on page 45

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 35

Reading FlexFlash Policy, on page 41

Reading IPMI Over LAN Policy, on page 34

Reading LDAP Policy, on page 46

Reading Legacy Boot Order Policy, on page 47

Reading Network Security Policy, on page 44

Reading NTP Policy, on page 38

Reading Precision Boot Order Policy, on page 36

Reading RAID Policy, on page 43

Reading Serial Over LAN Policy, on page 48

Reading SNMP Policy, on page 40

Reading SSH Policy, on page 39

Reading User Policy, on page 42

Reading Virtual KVM Policy, on page 49

Reading vMedia Policy, on page 45

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 35

Reading FlexFlash Policy, on page 41

Reading IPMI Over LAN Policy, on page 34

Reading LDAP Policy, on page 46

Reading Legacy Boot Order Policy, on page 47

Reading Network Security Policy, on page 44

Reading Precision Boot Order Policy, on page 36

Reading RAID Policy, on page 43

Reading Serial Over LAN Policy, on page 48

Reading SNMP Policy, on page 40

Reading SSH Policy, on page 39

Reading User Policy, on page 42

Reading VIC Adapter Policy, on page 37

Reading Virtual KVM Policy, on page 49

Reading vMedia Policy, on page 45

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 35

Reading FlexFlash Policy, on page 41

Reading IPMI Over LAN Policy, on page 34

Reading LDAP Policy, on page 46

Reading Legacy Boot Order Policy, on page 47

Reading Network Security Policy, on page 44

Reading NTP Policy, on page 38

Reading Precision Boot Order Policy, on page 36

Reading RAID Policy, on page 43

Reading Serial Over LAN Policy, on page 48

Reading SNMP Policy, on page 40

Reading User Policy, on page 42

Reading VIC Adapter Policy, on page 37

Reading Virtual KVM Policy, on page 49

Reading vMedia Policy, on page 45

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 35

Reading FlexFlash Policy, on page 41

Reading IPMI Over LAN Policy, on page 34

Reading LDAP Policy, on page 46

Reading Legacy Boot Order Policy, on page 47

Reading Network Security Policy, on page 44

Reading NTP Policy, on page 38

Reading Precision Boot Order Policy, on page 36

Reading RAID Policy, on page 43

Reading Serial Over LAN Policy, on page 48

Reading SSH Policy, on page 39

Reading User Policy, on page 42

Reading VIC Adapter Policy, on page 37

Reading Virtual KVM Policy, on page 49

Reading vMedia Policy, on page 45

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 35

Reading IPMI Over LAN Policy, on page 34

Reading LDAP Policy, on page 46

Reading Legacy Boot Order Policy, on page 47

Reading Network Security Policy, on page 44

Reading NTP Policy, on page 38

Reading Precision Boot Order Policy, on page 36

Reading RAID Policy, on page 43

Reading Serial Over LAN Policy, on page 48

Reading SNMP Policy, on page 40

Reading SSH Policy, on page 39

Reading User Policy, on page 42

Reading VIC Adapter Policy, on page 37

Reading Virtual KVM Policy, on page 49

Reading vMedia Policy, on page 45

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 35

Reading FlexFlash Policy, on page 41

Reading IPMI Over LAN Policy, on page 34

Reading LDAP Policy, on page 46

Reading Legacy Boot Order Policy, on page 47

Reading Network Security Policy, on page 44

Reading NTP Policy, on page 38

Reading Precision Boot Order Policy, on page 36

Reading RAID Policy, on page 43

Reading Serial Over LAN Policy, on page 48

Reading SNMP Policy, on page 40

Reading SSH Policy, on page 39

Reading VIC Adapter Policy, on page 37

Reading Virtual KVM Policy, on page 49

Reading vMedia Policy, on page 45

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 35

Reading FlexFlash Policy, on page 41

Reading IPMI Over LAN Policy, on page 34

Reading LDAP Policy, on page 46

Reading Legacy Boot Order Policy, on page 47

Reading Network Security Policy, on page 44

Reading NTP Policy, on page 38

Reading Precision Boot Order Policy, on page 36

Reading Serial Over LAN Policy, on page 48

Reading SNMP Policy, on page 40

Reading SSH Policy, on page 39

Reading User Policy, on page 42

Reading VIC Adapter Policy, on page 37

Reading Virtual KVM Policy, on page 49

Reading vMedia Policy, on page 45

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 35

Reading FlexFlash Policy, on page 41

Reading IPMI Over LAN Policy, on page 34

Reading LDAP Policy, on page 46

Reading Legacy Boot Order Policy, on page 47

Reading NTP Policy, on page 38

Reading Precision Boot Order Policy, on page 36

Reading RAID Policy, on page 43

Reading Serial Over LAN Policy, on page 48

Reading SNMP Policy, on page 40

Reading SSH Policy, on page 39

Reading User Policy, on page 42

Reading VIC Adapter Policy, on page 37

Reading Virtual KVM Policy, on page 49

Reading vMedia Policy, on page 45

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 35

Reading FlexFlash Policy, on page 41

Reading IPMI Over LAN Policy, on page 34

Reading LDAP Policy, on page 46

Reading Legacy Boot Order Policy, on page 47

Reading Network Security Policy, on page 44

Reading NTP Policy, on page 38

Reading Precision Boot Order Policy, on page 36

Reading RAID Policy, on page 43

Reading Serial Over LAN Policy, on page 48

Reading SNMP Policy, on page 40

Reading SSH Policy, on page 39

Reading User Policy, on page 42

Reading VIC Adapter Policy, on page 37

Reading Virtual KVM Policy, on page 49

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 35

Reading FlexFlash Policy, on page 41

Reading IPMI Over LAN Policy, on page 34

Reading Legacy Boot Order Policy, on page 47

Reading Network Security Policy, on page 44

Reading NTP Policy, on page 38

Reading Precision Boot Order Policy, on page 36

Reading RAID Policy, on page 43

Reading Serial Over LAN Policy, on page 48

Reading SNMP Policy, on page 40

Reading SSH Policy, on page 39

Reading User Policy, on page 42

Reading VIC Adapter Policy, on page 37

Reading Virtual KVM Policy, on page 49

Reading vMedia Policy, on page 45

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 35

Reading FlexFlash Policy, on page 41

Reading IPMI Over LAN Policy, on page 34

Reading LDAP Policy, on page 46

Reading Network Security Policy, on page 44

Reading NTP Policy, on page 38

Reading Precision Boot Order Policy, on page 36

Reading RAID Policy, on page 43

Reading Serial Over LAN Policy, on page 48

Reading SNMP Policy, on page 40

Reading SSH Policy, on page 39

Reading User Policy, on page 42

Reading VIC Adapter Policy, on page 37

Reading Virtual KVM Policy, on page 49

Reading vMedia Policy, on page 45

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 35

Reading FlexFlash Policy, on page 41

Reading IPMI Over LAN Policy, on page 34

Reading LDAP Policy, on page 46

Reading Legacy Boot Order Policy, on page 47

Reading Network Security Policy, on page 44

Reading NTP Policy, on page 38

Reading Precision Boot Order Policy, on page 36

Reading RAID Policy, on page 43

Reading SNMP Policy, on page 40

Reading SSH Policy, on page 39

Reading User Policy, on page 42

Reading VIC Adapter Policy, on page 37

Reading Virtual KVM Policy, on page 49

Reading vMedia Policy, on page 45

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 35

Reading FlexFlash Policy, on page 41

Reading IPMI Over LAN Policy, on page 34

Reading LDAP Policy, on page 46

Reading Legacy Boot Order Policy, on page 47

Reading Network Security Policy, on page 44

Reading NTP Policy, on page 38

Reading Precision Boot Order Policy, on page 36

Reading RAID Policy, on page 43

Reading Serial Over LAN Policy, on page 48

Reading SNMP Policy, on page 40

Reading SSH Policy, on page 39

Reading User Policy, on page 42

Reading VIC Adapter Policy, on page 37

Reading vMedia Policy, on page 45

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 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_CREATE</operationType>
<payload>
<![CDATA[<CIMCHardwareProfile><profileNames></profileNames>
<Cisco UCS C3260></Cisco UCS C3260>
<policies></policies></CIMCHardwareProfile>
<TargetPlatform></TargetPlatform>]]>
</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 51

Updating Hardware Profile, on page 52

Deleting Hardware Profile, on page 53

Applying Hardware Profile, on page 54

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 50

Updating Hardware Profile, on page 52

Deleting Hardware Profile, on page 53

Applying Hardware Profile, on page 54

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 50

Reading Hardware Profile, on page 51

Deleting Hardware Profile, on page 53

Applying Hardware Profile, on page 54

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 50

Reading Hardware Profile, on page 51

Updating Hardware Profile, on page 52

Applying Hardware Profile, on page 54

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 50

Reading Hardware Profile, on page 51

Updating Hardware Profile, on page 52

Deleting Hardware Profile, on page 53

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 57
Updating a Rack Group, on page 58
Deleting a Rack Group, on page 59

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 56
Updating a Rack Group, on page 58
Deleting a Rack Group, on page 59

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 56

Reading All Rack Groups, on page 57

Deleting a Rack Group, on page 59

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 56

Reading All Rack Groups, on page 57

Updating a Rack Group, on page 58

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 CREATE_RACK_ACCOUNT 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>CREATE_RACK_ACCOUNT</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 61

Deleting a Rack Account, on page 63

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 UPDATE_RACK_ACCOUNT 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>UPDATE_RACK_ACCOUNT</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

Deleting a Rack Account, on page 63

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 DELETE_RACK_ACCOUNT API are:

- String devices—The account to delete.

Sample Input XML

```
<cuicOperationRequest><operationType>DELETE_RACK_ACCOUNT</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 60

Updating a Rack Account, on page 61

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 65

Deleting Server Diagnostics Report, on page 67

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 65

Deleting Server Diagnostics Report, on page 67

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 serverIP—The diagnostics report to delete.

Sample Input XML

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

Implementation

The serverIP argument must be a valid IP address.

See Also

Reading Server Diagnostics Status by Server IP, on page 66

Running Server Diagnostics, on page 65

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 69

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 68

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 71

Reading Technical Support Logs by Server IP, on page 72

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 70

Reading Technical Support Logs by Server IP, on page 72

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 70

Clearing Technical Support Logs, on page 71

Creating a Discovery Profile

Objective

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

Prerequisites

None

REST URL

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

Components

The parameters of the DISCOVERY_PROFILE_CREATE API are:

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

Sample Input XML

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

<option>IP</option>

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

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

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

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

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

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

<credentialPolicy>false</credentialPolicy>

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

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

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

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

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

</CIMCDeviceDiscoveryConfig>

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

Implementation

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

See Also

Updating a Discovery Profile, on page 75

Deleting a Discovery Profile, on page 78

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 72

Updating a Discovery Profile, on page 75

Deleting a Discovery Profile, on page 78

Updating a Discovery Profile

Objective

Update an existing discovery profile.

Prerequisites

None

REST URL

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

Components

The parameters of the DISCOVERY_PROFILE_UPDATE API are:

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

Sample Input XML

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

<option>IP</option>

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

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

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

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

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

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

<credentialPolicy>false</credentialPolicy>

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

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

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

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

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

</ModifyCIMCDeviceDiscoveryProfile>

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

Implementation

Profile Name cannot be modified.

See Also

Creating a Discovery Profile, on page 72

Deleting a Discovery Profile, on page 78

Deleting a Discovery Profile

Objective

Delete one or more existing discovery profiles.

Prerequisites

None

REST URL

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

Components

The parameters of the DISCOVERY_PROFILE_DELETE API are:

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

Sample Input XML

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

</DeleteCIMCDeviceDiscoveryProfile>

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

Implementation

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

See Also

Creating a Discovery Profile, on page 72

Updating a Discovery Profile, on page 75

Reading a Discovery Profile, on page 75

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[
<CIMCAutoDiscoveryConfig>
<profileNames></profileNames>

<enableSchedule>false</enableSchedule>

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

</CIMCAutoDiscoveryConfig>

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

Implementation

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

See Also

Importing Discovered Devices, on page 80

Reading Discovered Devices

Objective

Get discovered device details.

Prerequisites

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

REST URL

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

Implementation

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

Importing Discovered Devices

Objective

Import one or more discovered devices.

Prerequisites

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

REST URL

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

Components

The parameters of the IMPORT_SERVER API are:

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

Sample Input XML

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

<userPrefix></userPrefix>

<description></description>

<contact></contact>

<location></location>

<rackGroup>Default Group</rackGroup>

</ImportRackServersConfig>

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

Implementation

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

See Also

Running Server Discovery, on page 78

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 82
Power On Server, on page 84
Power Off Server, on page 83
Shutdown Server, on page 85
Set Label on Server, on page 86
Toggle Locator LED on Server, on page 87

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 81

Power On Server, on page 84

Power Off Server, on page 83

Shutdown Server, on page 85

Set Label on Server, on page 86

Toggle Locator LED on Server, on page 87

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 81

Power Cycle Server, on page 82

Power On Server, on page 84

Shutdown Server, on page 85

Set Label on Server, on page 86

Toggle Locator LED on Server, on page 87

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 81

Power Cycle Server, on page 82

Power Off Server, on page 83

Shutdown Server, on page 85

Set Label on Server, on page 86

Toggle Locator LED on Server, on page 87

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 82

Power On Server, on page 84

Power Off Server, on page 83

Hard Reset Server, on page 81

Set Label on Server, on page 86

Toggle Locator LED on Server, on page 87

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 82

Power On Server, on page 84

Power Off Server, on page 83

Shutdown Server, on page 85

Hard Reset Server, on page 81

Toggle Locator LED on Server, on page 87

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 82

Power On Server, on page 84

Power Off Server, on page 83

Shutdown Server, on page 85

Set Label on Server, on page 86

Hard Reset Server, on page 81

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 96
Reading Servers by Rack Group, on page 98
Reading Servers by Serial Number, on page 97
Reading Servers by Server IP, on page 97
Reading Servers by Tag Value, on page 89
Reading Servers by UUID, on page 96
Reading Servers by Product ID, on page 95

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 88
Reading Servers by Account Name, on page 96
Reading Servers by Rack Group, on page 98
Reading Servers by Serial Number, on page 97
Reading Servers by Server IP, on page 97
Reading Servers by UUID, on page 96
Reading Servers by Product ID, on page 95

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 91
Reading Server Faults by Fault Code, on page 92
Reading Server Faults by IP Address, on page 90
Reading Server Faults by Severity, on page 91

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 89
Reading Server Faults by Fault Code, on page 92
Reading Server Faults by Account Name, on page 91
Reading Server Faults by Severity, on page 91

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 89
Reading Server Faults by Fault Code, on page 92
Reading Server Faults by IP Address, on page 90
Reading Server Faults by Severity, on page 91

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 89
Reading Server Faults by Fault Code, on page 92
Reading Server Faults by IP Address, on page 90
Reading Server Faults by Account Name, on page 91

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 89
Reading Server Faults by Account Name, on page 91
Reading Server Faults by IP Address, on page 90
Reading Server Faults by Severity, on page 91

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 94
Reading Server Faults History by IP Address, on page 93
Reading Server Faults History by Severity, on page 94
Reading Server Faults History by Account Name, on page 93

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 94
Reading Server Faults History by DN, on page 92
Reading Server Faults History by Severity, on page 94
Reading Server Faults History by Account Name, on page 93

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 94
Reading Server Faults History by DN, on page 92
Reading Server Faults History by Severity, on page 94
Reading Server Faults History by IP Address, on page 93

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 94
Reading Server Faults History by DN, on page 92
Reading Server Faults History by Account Name, on page 93
Reading Server Faults History by IP Address, on page 93

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 94
Reading Server Faults History by DN, on page 92
Reading Server Faults History by Account Name, on page 93
Reading Server Faults History by IP Address, on page 93

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 88
Reading Servers by Account Name, on page 96
Reading Servers by Rack Group, on page 98
Reading Servers by Serial Number, on page 97
Reading Servers by Server IP, on page 97
Reading Servers by UUID, on page 96
Reading Servers by Tag Value, on page 89

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 88
Reading Servers by Tag Value, on page 89
Reading Servers by Rack Group, on page 98
Reading Servers by Serial Number, on page 97
Reading Servers by Server IP, on page 97
Reading Servers by UUID, on page 96
Reading Servers by Product ID, on page 95

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 88
Reading Servers by Tag Value, on page 89
Reading Servers by Account Name, on page 96
Reading Servers by Rack Group, on page 98
Reading Servers by Serial Number, on page 97
Reading Servers by Server IP, on page 97
Reading Servers by Product ID, on page 95

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 88
Reading Servers by Account Name, on page 96
Reading Servers by Rack Group, on page 98
Reading Servers by Serial Number, on page 97
Reading Servers by Server IP, on page 97
Reading Servers by UUID, on page 96
Reading Servers by Product ID, on page 95

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 88
Reading Servers by Tag Value, on page 89
Reading Servers by Account Name, on page 96
Reading Servers by Rack Group, on page 98
Reading Servers by Server IP, on page 97
Reading Servers by Product ID, on page 95
Reading Servers by UUID, on page 96

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 88
Reading Servers by Tag Value, on page 89
Reading Servers by Account Name, on page 96
Reading Servers by Server IP, on page 97
Reading Servers by Serial Number, on page 97
Reading Servers by Product ID, on page 95
Reading Servers by UUID, on page 96

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 99

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 99

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 100

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 100

Reading Server Utilization History by Account Name

Objective

Get Server Utilization History By Account Name.

Prerequisites

None

REST URL

```
/cloupia/api-v2/CIMCServerUtilizationHistoryByAccountName/{CIMCServerUtilizationHistoryByAccountNa
```

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 101

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 101

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 101

Reading Server Utilization History by Server IP, on page 101

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 103

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 102

Managing Users and Groups

Overview

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

Creating a User Group

Objective

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

Prerequisites

None

REST URL

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

Components

The parameters of the CREATE API are:

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

Sample Input XML

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

<groupDescription></groupDescription>

<parentGroup></parentGroup>

<groupCode></groupCode>

<groupContact></groupContact>

<firstName></firstName>

<lastName></lastName>

<phone></phone>

<address></address>

<groupSharePolicyId>0</groupSharePolicyId>

<allowPrivateUsers>false</allowPrivateUsers>

</AddGroupConfig>
```

Implementation

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

See Also

Updating a User Group , on page 105
Deleting a User Group, on page 106
Enabling All Users in a Group, on page 107
Disabling All Users in a Group, on page 108

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 103

Deleting a User Group, on page 106

Enabling All Users in a Group, on page 107

Disabling All Users in a Group, on page 108

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 103

Updating a User Group , on page 105

Enabling All Users in a Group, on page 107

Disabling All Users in a Group, on page 108

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 103

Updating a User Group , on page 105

Deleting a User Group, on page 106

Disabling All Users in a Group, on page 108

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 103

Deleting a User Group, on page 106

Updating a User Group , on page 105

Enabling All Users in a Group, on page 107

Creating a User

Objective

This task allows the user to create a new user.

Prerequisites

None

REST URL

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

Components

The parameters of the CREATE API are:

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

Sample Input XML

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


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

<mspOrganization></mspOrganization>

<loginName></loginName>


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


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

<userContactEmail></userContactEmail>

<firstName></firstName>

<lastName></lastName>

<phone></phone>

<address></address>

</AddUserConfig>

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

Implementation

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

See Also

Reading a User, on page 111
Updating a User , on page 112
Deleting a User, on page 113
Enabling a User, on page 114
Disabling a User, on page 115
Updating a User Expiry Date, on page 116
Updating a User Password, on page 117

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 109
Updating a User , on page 112
Deleting a User, on page 113
Enabling a User, on page 114
Disabling a User, on page 115
Updating a User Expiry Date, on page 116
Updating a User Password, on page 117

Updating a User

Objective

This task allows to update an existing user.

Prerequisites

None

REST URL

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

Components

The parameters of the UPDATE USER API are:

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

Sample Input XML

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

<userType>GroupAdmin</userType>

<userGroup>1</userGroup>

<mspOrganization></mspOrganization>

<userContactEmail></userContactEmail>

<firstName></firstName>

<lastName></lastName>

<phone></phone>

<address></address>

</ModifyUserConfig>

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

Implementation

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

See Also

Creating a User, on page 109

Reading a User, on page 111

Deleting a User, on page 113

Enabling a User, on page 114

Disabling a User, on page 115

Updating a User Expiry Date, on page 116

Updating a User Password, on page 117

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 109

Reading a User, on page 111

Updating a User , on page 112

Enabling a User, on page 114

Disabling a User, on page 115

Updating a User Expiry Date, on page 116

Updating a User Password, on page 117

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 109

Reading a User, on page 111

Updating a User , on page 112

Deleting a User, on page 113

Disabling a User, on page 115

Updating a User Expiry Date, on page 116

Updating a User Password, on page 117

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 109

Reading a User, on page 111

Updating a User , on page 112

Deleting a User, on page 113

Enabling a User, on page 114

Updating a User Expiry Date, on page 116

Updating a User Password, on page 117

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 109
Reading a User, on page 111
Updating a User , on page 112
Deleting a User, on page 113
Enabling a User, on page 114
Disabling a User, on page 115
Updating a User Password, on page 117

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 109

Reading a User, on page 111

Updating a User , on page 112

Deleting a User, on page 113

Enabling a User, on page 114

Disabling a User, on page 115

Updating a User Expiry Date, on page 116