Cisco Unified Communications Domain Manager, Release 11.5(1) API Reference Guide

First Published: 2017-02-14

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

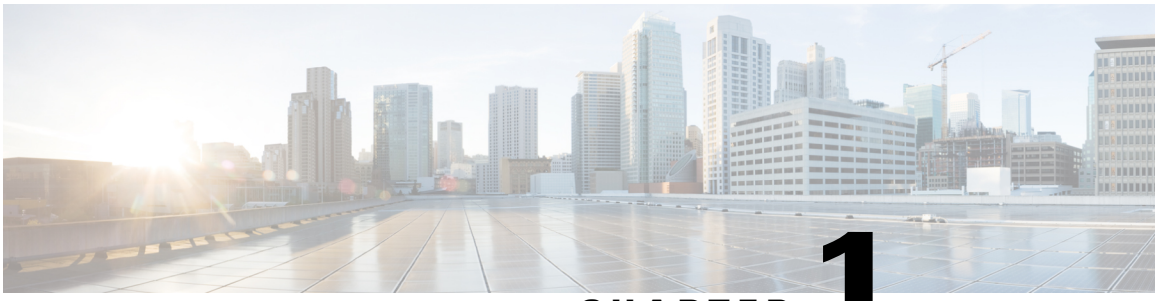
CONTENTS

Change History

Change History

Section	Description	Date
API Parameters, on page 7	Updated	January 2017
Transaction Choices, on page 37	New topic	January 2017
Transaction Filters, on page 38	New topic	January 2017
API Request Headers, on page 12	New topic	January 2017
Non-interactive Login, on page 14	New topic	January 2017
	Initial Version	January 2017

Change History



CHAPTER 1

Introduction

Introduction

API Introduction

Cisco Unified Communications Domain Manager introduces a completely new approach to Unified Communications management that emphasizes the API as the common point of entry to the system. All Cisco Unified Communications Domain Manager functionality is available through the API, including the Cisco Unified Communications Domain Manager GUI itself, which uses the same API. Because the GUI uses the same API that is available to service providers externally, developers can use trace tools (such as Chrome developer tools) to learn details on the API schema, how the API operates, how to create, update, and delete objects, etc. Any operation performed by the Cisco Unified Communications Domain Manager GUI can be traced and replicated by an external client.

The Cisco Unified Communications Domain Manager API is a secure, normalized, and integrated REST-based API. Secure in the sense that it supports HTTPS digest authentication. Normalized in the sense that all APIs follow the same overall schema, semantics, and operations; only the detailed attribute schemas and authorized operations differ with each object depending on the type of object and the user credentials used to access the API. Integrated in the sense that the Cisco Unified Communications Domain Manager platform allows access to the following HCS components with this normalized API:

- Cisco Unified Communications Domain Manager orchestrated workflows
- Cisco Hosted Collaboration Mediation Fulfillment database
- Cisco Unified Communications Manager cluster databases
- Cisco Unity Connection databases
- LDAP user data

The API uses a well defined JSON schema with consistent meta-data across all device types integrated by Cisco Unified Communications Domain Manager , regardless of the devices native API format. For example, the native Cisco Unified Communications Domain Manager API is SOAP based XML but this is converted to object-oriented REST-based JSON, consistent with all other APIs in the Cisco Unified Communications Domain Manager system.

There are many APIs in the HCS system, including the native APIs for the devices integrated into Cisco Unified Communications Domain Manager (Cisco Hosted Collaboration Mediation Fulfillment, Cisco Unified Communications Domain Manager, Cisco Unity Connection), APIs for other devices not integrated in Cisco Unified Communications Domain Manager . These APIs are still available and can still be used for operations that aren't supported by Cisco Unified Communications Domain Manager orchestrated workflows. For example, Cisco Hosted Collaboration Mediation Fulfillment supports Prime Collaboration Assurance (PCA) configuration via API, but Cisco Unified Communications Domain Manager 10.1 does not currently support configuring PCA directly. Therefore, you can use the native Cisco Hosted Collaboration Mediation Fulfillment API for configuring PCA on Cisco Hosted Collaboration Mediation Fulfillment directly.

API System Concepts

In order to understand the API, an understanding of two basic concepts is required:

- Models
- Hierarchy

The term "model" is used to describe the types of JSON objects fulfilling purposes such as defining data structures, containing data, defining GUI forms, mapping data from devices or other models. The system employs the following types of models:

- Data Models
- Device Models
- Domain Models
- Relations
- Views

Data in the system is represented using Data and Device models.

Device models are generated from the application API of entities that are provisioned on devices.

Domain models, relations and views wrap the Data or Device models by means of references to them.

Data models can be created and are stored in the database. Data models contain a JSON schema/metadata for the entities exposed by the underlying database. The schemas for the data models are stored in the database and represent the structure that instances of the data model conforms to.

Device models interface with devices and services on the system. For example:

- Unified CM device models interface with the Call Manager's AXL SOAP API.
- CUC device models interface with Unity Connection's RESTful API.

The ability to rapidly develop and deploy new device interfaces provides an extensible mechanism to add support for additional provisioning tasks or additional southbound integration into other business systems.

Domain models act as "containers" of other data-, device- and domain models along with provisioning workflows to represent the management of a created feature.

Relations do not store data on the system. Instead, they relate groups of resource types such as device models, data models or other domain models.

Views provide a mechanism to define an arbitrary schema which can be used to define a user input screen.

Hierarchy

A system hierarchy node is present at first startup of the system. Each entity that is attached to the hierarchy has an address represented by a PKID, that is defined as a standard URI. Hierarchies can be created under the system hierarchy node, because the hierarchy is exposed as a RESTful API. API calls are made with reference to the hierarchy.

For more information, refer to the Hierarchy section in the *Cisco Unified Communications Domain Manager, Release 11.5(1) Planning and Install Guide*.

Basic REST

The system uses a REST (Representational State Transfer) API. For details on this type of API, see for example:

- http://en.wikipedia.org/wiki/Representational_state_transfer

API Traversal

The system represent the reference of an entity in the system as http://en.wikipedia.org/wiki/HATEOAS (HATEOS). Each reference position is represented by an object pair PKID and href.

A client integrates with Cisco Unified Communications Domain Manager 10.x/11.5(x) entirely through hypermedia dynamically provided by the application and does not need any prior knowledge of how to interact with the system other than a generic understanding of hypermedia. This means that no http://en.wikipedia.org/wiki/Web_Application_Description_Language is provided. This also means that the client and the application can be decoupled in a way that allows the application to evolve independently.

A client enters the the application through a simple fixed URL. All future actions the client may take are discovered within resource representations returned from the server

The URL tree information is obtained in the form of a list response from the application endpoint:

GET /api/?format=json

This response emulates the HierarchyNode list response and utilizes the parent and children in the meta references section of the response as discussed in Meta Data References.

Request and Response Patterns

The request and response patterns between service requester and Cisco Unified Communications Domain Manager 10.x/11.5(x) is summarized below.

For synchronous operations:

1. Service Requestor sends an accessor (e.g. Get, List) request with request parameters.
2. Either:
   a. responds synchronously with a Get/List response.
   b. responds synchronously with a fault response.

For asynchronous operations:

1. Service Requestor sends a mutator (e.g. Add, Modify, Delete) request with parameters.
2. The Add/Update/Delete transaction is scheduled in the transaction queue with a transactionID.
3. Responds synchronously with either:
   a. An Add/Update/Delete response and a transactionID.
   b. A fault response.
4. The external system either:
   a. Polls the system to retrieve the status of the transaction as needed, or
   b. Specifies a callback URL (with an optional username and password if the interface is secured (recommended)) and waits for a asynchronous transaction status callback (recommended).

      When the transaction completes, Cisco Unified Communications Domain Manager 10.x/11.5(x) sends an async transaction status callback message to the callback URL specified in the request.

Anatomy of an API Request

General Structure of the API

The Cisco Unified Communications Domain Manager 10.x/11.5(x) API accesses system resources or tools.

- **Resources**

The general structure of an API URL for accessing a system resource (an endpoint) is:

Method https://servername/api/Version/Resource/Action/?Parameters

Where:

Method

```
[GET|POST|DELETE|PUT|PATCH]
```

Servername

The installation server determines the base URL, e.g. `https://servername`. In a cluster environment, this is the address of the web proxy node. Refer to the Install Guide for cluster deployment information.

Version

```
(int:major[.int:minor])/
```

Resource

```
(str:modeltype/str:modelname)[/pkid]
```

Action

```
[add|create|list|update|remove|configuration_template|field_display_policy|
 export_bulkload_template|bulk_update|clone|move|export|execute|translation|
 migration|help|+tag|+tag_version|+[non-CRUD operation|CustomWF|
 other custom actions (see API Reference Guides)]]
```

Parameters

```
[(str:api parameter)[&(str:api parameter)...]]
```

The HTTP methods and parameters are described in relevant sections. The different resources supported in the system are described in the API Reference Guides.

- **Tools**

For tools, the general structure of the URL structure is for example:

[GET|POST] /api/tool/(str:tool_name)/

Format

The system API supports the following format HTTP headers when handling and responding to requests.

Field Name	Description	Value
Content-Type	The format type of the body of the request (used with POST and PUT requests)	application/json
Content-Type	The format type of the body of the request (used with PATCH requests)	application/json-patch+json
Accept	Content-Types that are acceptable in response	application/json

Authentication

The system controls access to its service through HTTP basic authentication. The technique is defined in section 11.1 of RFC1945 which is simple to implement, uses standard HTTP headers.

The HTTP Basic Access Authentication requires authorization credentials in the form of a user name and password before granting access to resources in the system. The username and password are passed as Base64 encoded text in the header of API requests.

The HTTP header format for authentication is defined in the table below.

Field Name	Description	Value
Authorization	Basic authentication is supported.	Basic [Base64 encoded credentials]

For example:

The Base64 encoded credentials for user name of joe and a password of bloggs.

For example, from a command line (note the removal of the new line in the `echo` command):

```
$ echo -n "joe:bloggs" | base64
am9lOmJsb2dncw==
```

the header is:

```
Authorization: Basic am9lOmJsb2dncw==
```

For example, using **curl**:

```
curl -k -H "Authorization: Basic am9lOmJsb2dncw==" 'https://hostname/api/data/MyModel/'
```

It is required that all requests be conducted over a secure session, such as HTTPS or SSL.

A Cisco Unified Communications Domain Manager 10.x/11.5(x) self-signed certificate needs to be installed into a local trust store of the client application.

Authorization

The access profile of a user determines whether he or she can perform a given operation on a model. The user can also only access items below the position they are defined in the hierarchy.

HTTP Methods

The API supports the following HTTP methods:

GET

- Used to query a resource or a list of resource.

POST

- Used to create a new resource.
- The data is submitted as a JSON object.
- The return value is the PKID of the resource.

PUT

- Used to update the data of a resource.
- The resource URL includes the resource PKID.
- The data to be updated is submitted as a JSON object.

PATCH

- Used to update the data of a resource.
- PATCH request body in JSON Patch format
- Content-Type is "application/json-patch+json"
- JSON Patch: http://tools.ietf.org/html/rfc6902

DELETE

- Used to delete a resource.

- The resource URL includes the resource PKID.
- The DELETE method can also be used to delete multiple resources on one request as a "bulk delete".

PUT Versus PATCH

For PUT methods the resource data is replaced with the data specified in the request. All fields of the resource are replaced with the fields in the request.

This means that:

- Fields not present in the request that are present in the resource are dropped from the resource.
- Fields present in the request that are not present in the resource are appended to the resource.
- The data of fields present in the request is used to update fields that already exist in the resource.

PATCH methods operate in two modes depending on the content type:

- `Content type: application/json`
- The values of data fields present in the request is used to update the corresponding resource fields. This means that:
  - Fields present in the request but not in the resource are appended to the resource.
  - The value of each field that is already present in the resource is updated from the request data.
  - Field values that are set to null in the request are dropped from the resource.
  - Fields that are present in the resource but not in the request are left untouched.
- `Content type: application/json-patch+json`
  - Existing resource data is patched according to RFC6902.

Modifying data fields:

- To drop the field from a data model, specify null as the parameter value (i.e. `{"field": null}`).
- To blank out a string value set the parameter value to an empty string (i.e. `{"field":""}`).

API Parameters

The hierarchy parameter is required for each API request and can be specified as any of the following:

- the primary key id (PKID) of the hierarchy node in the form of a Universally Unique ID(UUID), for example `1c055772c0deab00da595101`
- in dot notation, for example `ProviderName.CustomerName.LocationName`

To obtain the PKID of a hierarchy node, refer to the `path` element in the metadata of `data/HierarchyNode` resource.

Note For the purposes of simplifying the documentation, the hierarchy API parameter `&hierarchy=[hierarchy]` is not included in all examples in this document. Specifying the hierarchy is however required in all API requests where the instance PKID is not referenced. In the examples, `[hierarchy]` is substituted with the caller's hierarchy id.

The system API supports the following request parameters for data format when handling requests.

Key	Description	Value
format	The format type of the body of the request	json

A request of the following format returns HTML:

```
GET /api/(str:model_type)/(str:model_name)/help/
```

A parameter `&format=json` is not displayed in all examples, but is required for all requests unless a different format is specifically stated.

The Configuration Template can be specified in the POST request parameters for a resource as follows:

```
POST /api/(str:model_type)/(str:model_name)/&template_name=[CFG name]
```

Key	Description	Value
template	Apply the Configuration Template with PKID [CFG pkid] to the payload of the POST request.	[CFG pkid]
template_name	Apply the Configuration Template with name [CFG name] to the payload of the POST request.	[CFG name]

Field Display Policy can be specified in the GET request parameters for a resource as follows:

```
GET /api/(str:model_type)/(str:model_name)/add/
```

Key	Description	Value
policy	Return a model form schema where the Field Display Policy with PKID [FDP pkid] is applied to it. Use `policy` with the parameters `schema` and `format=json`.	[FDP pkid]
policy_name	Return a model form schema where the Field Display Policy with name [FDP name] is applied to it. Use `policy` with the parameters `schema` and `format=json`.	[FDP name]

The API can return cached data from the system or data from devices, using the following format:

```
GET /api/(str:model_type)/(str:model_name)/[pkid]/
```

Key	Description	Value	Default
cached	System will respond with resource information where the data was obtained from cache. (Functionally only applicable to device models and domain models containing device models)	true, false	true

To identify a single resource, the API call contains the single resource (PKID) using the following format:

```
GET /api/(str:model_type)/(str:model_name)/(pkid)/
```

To obtain the schema or schema rules of a resource, use the following parameters to an API request:

```
GET /api/(str:model_type)/(str:model_name)/?
  hierarchy=[hierarchy]&schema=true&schema_rules=true&schema_version=[version_no]
```

Key	Description	Value
schema	Return the schema of the resource. Use with the parameter `format=json`	true, false
schema_rules	Return the GUI Rules and Field Display Policies of the resource if available. Use with the parameters `format=json` and `schema` to see `schema_rules` in the response.	true, false

The system API supports the following API request parameters for when specifying the format of and structure of the resources to list.

Key	Description	Value	Default
skip	The list resource offset		0
limit	The maximum number of resources returned		50
count	Specify if the number of resources should be counted. If false, the `pagination` object in the response shows the `total` as 0, so no total is calculated and the API performance is improved		
order_by	The summary attribute field to sort on		First summary attribute
direction	The direction of the summary attribute field sort (asc:ascending, desc: descending)	asc, desc	asc
summary	Only summary data is returned in the data object	true, false	true

Key	Description	Value	Default
traversal	The direction of the resource lookup of resources tied to the hierarchy tree from the hierarchy node provided as parameter	up, down, local	down

Models that have the list action defined in their schema can also be filtered by using a number of URL filter parameters in parameter sets of four key-value pairs.

Filters also apply to the api/tool/Transaction/ endpoint, which has additional filter functionality to filter by transaction ID. Refer to the topic on Filter Transactions.

These parameters can be added in addition to the parameters available to list resources as in the topic on API Parameters.

A single filter query can contains one or more sets of the following four parameters:

Key	Description	Value	Default
filter_field	The model attribute name to filter.	The name of the attribute in the list of `summary_attrs` in the model schema.	0
filter_condition	The matching operator for the `filter_field`. If `equals` is used in a condition, then other filter sets are ignored.	One of the conditions below, applied to a filter_text string value. • `startswith` • `endswith` • `contains` • `notcontain` • `equals` • `notequal`	`contains`
filter_text	A text string applied to the `filter_field` by a `filter_condition`.	Plain text	
ignore_case	Additional specifier applied to the case of the `filter_text`.	Either `true` or `false`.	`true`

Example showing a single filter set:

```
GET /api/(str:model_type)/(str:model_name)/?
 hierarchy=[hierarchy]
 &filter_field=[attribute_name]
 &filter_condition=startswith
 &filter_text=John
 &ignore_case=false
```

If more than one filter set is used, all similar keys are grouped, so that the key position indicates the filter set. For example:

```
GET /api/(str:model_type)/(str:model_name)/?
 hierarchy=[hierarchy]
 &filter_field=[attribute_name]
 &filter_field=[attribute_name2]
 &filter_condition=startswith
 &filter_condition=endswith
 &filter_text=John
 &filter_text=an
 &ignore_case=false
 &ignore_case=false
```

The two filter sets in this example, are:

- `&filter_field=[attribute_name]`
- `&filter_condition=startswith`
- `&filter_text=John`
- `&ignore_case=false`

and

- `&filter_field=[attribute_name2]`
- `&filter_condition=endswith`
- `&filter_text=th`
- `&ignore_case=false`

It is possible to submit mutator type operations with API parameters to complete synchronously, in which case the synchronous response to the transaction either includes the status of the transaction or a fault response. This is not recommended as long-running transactions or a busy system may exceed the HTTP timeout.

This is only available for models where the actions in the meta data contains `support_async`.

Key	Description	Value	Default
nowait	Controls the API synchronous or asynchronous behavior for requests resulting in transactions. Please refer to the `support_async` property in the model schema under **meta** -> **actions**, for an indication of support per action.	true, false	false

To specify a specific API version of a resource, use the following parameter to an API request:

```
GET /api/(str:model_type)/(str:model_name)/?
hierarchy=[hierarchy]&api_version=<version_number>
```

Key	Description	Value
api_version	Return the the resource with `api_version`. Use with the parameter `format=json`	supported version no.

Where the parameters below are added to the GET call:

```
schema=true&
format=json&
schema_rules=true
```

then the JSON schema `meta` property will contain `schema_version` in accordance with `api_version=<version_number>`.

For Unified CM device models, the `schema_version` will match the version of the Unified CM that corresponds with the `api_version`, for example:

```
GET /api/device/cucm/(str:model_name)/?
hierarchy=[hierarchy]&
schema=true&
format=json&
schema_rules=true&
api_version=10.1.2
```

Returns:

```
"meta": {
"tags": [],
"cached": true,
"title": "",
"business_key": {},
"schema_version": 10.0,
```

The following table shows the current mapping for `/device/cucm` models:

api_version	schema_version
10.1.2	10.0
10.6.1	10.5
10.6.2	10.5
10.6.3	10.5

If no `api_version` is specified in the GET call, then the default `schema_version` is determined by the latest entry in the mapping table.

Note For more details on API versioning, refer to the topic on API Backwards Compatibility.

API Request Headers

API Headers are available for:

- Pagination of choices and macro results in an API call.

  The headers are `X-range` and `Range`, with the starting value as 0. These can be used instead of the `skip` and `limit` API parameters.

For example, the following examples return the same results:

```
GET /api/tool/Macro/?method=evaluate
&hierarchy=[hierarchy]
&input={{fn.lines}}
&skip=0
&limit=6
GET /api/tool/Macro/?method=evaluate
&hierarchy=[hierarchy]
&input={{fn.lines}}
Request headers:
X-Range: items=0-5
Range: items=0-5
```

If the request is items=0-199 (for 200 items) and there are more results, the response will show:

```
Content-Range:items 0-199/999999999
```

In this example, if a subsequent request is for the next 200 items (200-399), the response will also show the total number of items (298) returned by the macro:

```
Content-Range:items 200-399/298
```

- Backward compatibility. The X-Version header is available to take an API version as value. For example:

```
GET /api/data/Countries/?hierarchy=[hierarchy]
&schema=true
&format=json
Request headers
X-Version: 10.1.2
```

Refer to the topics on API backwards compatibility.

Login and Authorization Tokens

The API includes, as part of responses, a `X-CSRFToken` response header that is set to the CSRF token, for example to `KEMzraBRygy2ZJ7fLuvbfKhAEIPK9D4s`. API clients should source the CSRF token from this header.

The API also includes, as a part of responses, a `csrftoken` cookie containing the CSRF token. This cookie is marked `httponly` and as such is not readable by browser-based client scripts. API clients should not try to source the CSRF token from this cookie.

The `X-CSRFToken` response header and `csrftoken` cookie values are identical.

When performing requests that require CSRF token validation, API clients should follow the general procedure:

1. Prior to performing the principal request, perform a request to the API and retrieve a CSRF token from the resulting response's `X-CSRFToken` response header. The CSRF token remains constant for the duration of a session, so clients could perform this request once per session (post authentication), storing the CSRF token and using it for subsequent requests.

   Clients should also retrieve the `csrftoken` cookie from the response.

2. For the primary request, include a `X-CSRFToken` request header containing the CSRF token as sourced from the response header, as well as the unchanged `csrftoken` cookie.

For example:

```
GET http://localhost:8000/login/

Raw response headers:
Cache-Control: max-age=0
Connection: keep-alive
Content-Encoding: gzip
Content-Language: en-us
Content-Type: text/html; charset=utf-8
```

```
Date: Mon, 20 Apr 2015 09:18:47 GMT
Expires: Mon, 20 Apr 2015 09:18:47 GMT
Last-Modified: Mon, 20 Apr 2015 09:18:47 GMT
Server: nginx/1.4.6 (Ubuntu)
Set-Cookie: csrftoken=KEMzraBRygy2ZJ7fLuvbfKhAEIPK9D4s; httponly; Path=/
sessionid=5d1ccc96cbd7e7f290020aaedd64c1b3; httponly; Path=/
sso_login_url=; Path=/
Transfer-Encoding: chunked
Vary: Accept-Encoding, Cookie, Accept-Language, X-CSRFToken
X-CSRFToken: KEMzraBRygy2ZJ7fLuvbfKhAEIPK9D4s
```

1. Source the CSRF token from response's `X-CSRFToken` header.
2. Retain the CSRF cookie from response's `csrftoken` cookie.
3. Now perform the primary POST /login/ request to login, including the CSRF token as a `X-CSRFToken` request header as well as the unchanged `csrftoken` cookie:

```
POST http://localhost:8000/login/

Raw request headers:
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:37.0) Gecko/20100101 Firefox/37.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:8000/login/
Cookie: sessionid=5d1ccc96cbd7e7f290020aaedd64c1b3;
csrftoken=KEMzraBRygy2ZJ7fLuvbfKhAEIPK9D4s; sso_Connection: keep-alive
X-CSRFToken: KEMzraBRygy2ZJ7fLuvbfKhAEIPK9D4s
```

With for example payload as parameters:

```
&username=joe
&password=bloggs
&next=%2F
```

Non-interactive Login

The following request and endpoint is available on the API:

```
POST <hostname>/noninteractivelogin/
```

The payload:

- Content-Type: application/json
- JSON containing user credentials, for example:

  co

  ```
  {
  "username":"joebloggs@email.com",
  "password":"mysecret"
  }
  ```

The purpose of the endpoint, payload and response is for an API client to be able to better report the reason for failed logins without having to parse the login form.

- If the request is successful:
  - the HTTP response is 200
  - the JSON body is for example:

    ```
    {
    "last_successful_login_time": "2016-04-25T09:50:34Z",
    "num_of_failed_login_attempts": 2
    }
    ```

- the X-CSRFToken value

Upon the first successful login, the `last_successful_login_time` is empty. Upon a subsequet successful login, the `last_successful_login_time` is the login time prior to current session. The `num_of_failed_login_attempts` value is reset to 0 after a successful login.

- If the request fails:
  - the HTTP response is 403
  - the JSON body is:

```
{
"error_message": "Please enter a valid username and password.",
"error_code": 27009
}
```

  - the X-CSRFToken value

Anatomy of an API Response

API Response Overview

Below are the typical elements of an API response:

- meta - Meta data.
- data - Actual data contained in the model as name: value pairs.
- schema - Schema describing the structure of the data of the resource, in particular the data types of the names in the name: value pairs in the data.
- resources - An object grouping a list of single resource's meta and data objects in an API list response
- pagination - an object containing pagination data in an API list response

Not all the elements above exist in each response. These differ depending on request parameters and whether response contains a list of resource or a single resource.

Single Resource Response

A single resource response outline is as follows:

```
{
  "meta": {
    ...
  },
  "data": {
    ...
  },
  "schema": {
    ...
  }
}
```

Resource List Response

The response object outline is as follows:

```
{
    "pagination": {
        ...
    },
    "meta": {
        ...
    },
    "resources": [{
        "meta": {
            ...
        },
        "data": {
            ...
        }
    },
    {
        "meta": {
            ...
        },
        "data": {
            ...
        }
    }]
}
```

POST/PUT/DELETE/PATCH Response

Support for synchronous and asynchronous request resulting in transactions, is controlled by the `nowait` parameter in the request URL. The support for asynchronous request handling is indicated in the API schema structure **actions** with the `support_async` property.

The outline of the default synchronous transaction response of mutator transactions when the API parameter `nowait` is set to be false, is as follows:

```
{
  "pkid": "51f7e09bd0278d4b28e981da",
  "model_type": "data/CallManager",
  "meta": {
    "parent_id": {
      "pkid": "51f7d06ad0278d4b34e98134",
      "uri": "/api/data/HierarchyNode/51f7d06ad0278d4b34e98134"
    },
    "summary_attrs": [
      {
        "name": "description",
        "title": "Description"
      },
      {
        "name": "host",
        "title": "Host Name"
      },
      {
        "name": "port",
        "title": "Port"
      }
    ],
    "uri": "/api/data/CallManager/51f7e09bd0278d4b28e981da"
  },
  "success": true
}
```

The outline of the synchronous response to asynchronous mutator transactions when the API parameter `nowait` is set to be true, is as follows:

```
{
  "href": "/api/tool/Transaction/cfe8a8fd-98e6-4290-b0c3-2dfa2224b808",
  "success": true,
  "transaction_id": "cfe8a8fd-98e6-4290-b0c3-2dfa2224b808"
}
```

To retrieve (for example by polling) the transaction status of any mutator transactions, use the *transaction_id* in the synchronous response to the asynchronous mutator transaction as follows:

```
GET /api/tool/Transaction/cfe8a8fd-98e6-4290-b0c3-2dfa2224b808
```

The response contains the status ad replay action URL, for example:

```
{
 "meta": {
        "model_type": "tool/Transaction",
        "summary_attrs":
        {
         "name": "name",
         "title": "Name"
        },
        "references": {}
        "actions":
          {
           "replay": {
             "class": "execute",
             "href": "/api/tool/Transaction/cfe8a8fd-98e6-4290-b0c3-2dfa2224b808/
               replay?format=json",
             "method": "GET",
             "title": "Replay"
            }
          }
        }
 "data": {
       "status": "Completed",
       "username": "sysadmin",
       "resource": {
        "hierarchy: "sys",
        "after_transaction": "/api/data/GeneralHelp/5268c7d3a616540a766b91f5/?
          cached=5268f2eba616540a736b926c Entity",
        "current_state": "/api/data/GeneralHelp/5268c7d3a616540a766b91f5/ Entity",
        "before_transaction": "/api/data/GeneralHelp/5268c7d3a616540a766b91f5/
          ?cached=5268c7d3a616540a766b91f7 Entity",
        "pkid": "5268c7d3a616540a766b91f5",
        "model_type": "data/GeneralHelp",
       }
  [...]
```

This mechanism can be used to retrieve the transaction status of any transaction or its sub-transaction, using the PKID of the (sub) transaction.

Asynchronous Mutator Transaction Status Callback

When using the API parameter `nowait=true`, the service requester can submit optional request meta data - containing a callback URL - with any mutator request by appending the `request_meta` tag to the normal payload of the request.

In order to receive asynchronous transaction status notifications, the requesting system needs to publish an HTTP service to service requests made by the callback URL. An example of a simple http service is provided in a separate section.

The callback operation supports an optional username and password that Cisco Unified Communications Domain Manager 10.x/11.5(x) uses to perform HTTP basic authentication on requests made to the callback

service. The optional elements `external_id` and `external_reference` are explained in the section on correlation identifiers.

```
{
   <Actual request data goes here>,
   "request_meta": {
        "external_id": "3x4mpl3-3xtern4l-FF",
        "external_reference": "Example External Reference-FF",
        "callback_url": "http://my.callbackservice:8080",
        "callback_username": "username",
        "callback_password": "password"
    }
}
```

The following details should be noted here:

- The schema of system resources or system tools do not include reference to the request meta data in the schema definition of each resource in the system.
- The `<Actual request data goes here>` request data needed to for example add a `country_name` instance for data/Countries would be similar to: `"country_name": "South Africa"`.
- The request data for deleting two countries for example would be

  ```
  "hrefs":[
     "/api/data/Countries/534fdf190dd19012066433ce",
     "/api/data/Countries/534fda1d0dd1901206643397"
  ]
  ```

- For the callback service to function, the callback service needs to be accessible from the fulfillment server.

Upon completion of the asynchronous mutator transaction posted with a callback URL, the application POSTs an HTTP request (asynchronous transaction status callback) to the callback service specified by the callback URL. The callback service needs to respond with an HTTP 200 ACK *before* internal processing of the callback. The callback includes the transaction ID sent to the requesting system as part of the synchronous response. To correlate the asynchronous transaction status callbacks with the original request, the requesting system would need to record the `transaction_id` returned in the synchronous response.

The HTTP headers and the payload of the asynchronous transaction status callback includes the following information:

HTTP headers:

```
{
    'accept-encoding': 'identity',
    'authorization': 'BasicdXNlcm5hbWU6cGFzc3dvcmQ=',
    'content-length': '275',
    'content-type': 'application/json',
    'host': 'localhost: 8080'
}
```

Payload:

```
{
    "external_id": "3x4mpl3-3xtern4l-FF",
    "external_reference": "ExampleExternalReference",
    "status": "Success",
    "transaction": {
        "href":
          "http:
//my.fulfillmentserver/api/tool/Transaction/e6ac7c1e-c63a-11e3-9af5-08002791605b/",
        "id": "e6ac7c1e-c63a-11e3-9af5-08002791605b"
    }
}
```

The following details should be noted here:

- Correlation identifiers (see correlation identifiers) are included in the payload if they are present.
- The status of the transaction is as in the transaction log: Fail or Success.

The transaction status is not affected by the response of the HTTP service published by the requesting system. The transaction log information includes the callback request and the response returned by the callback service published by the external system.

For transactions with multiple sub-transactions, a single transaction status callback request is made upon the completion of the parent transaction. Transaction status callbacks are not supported for the parent transactions tool/BulkLoad and tool/DataImport.

In the event that the transaction status callback is not received by the external system due to for example a network outage, the external system can poll to retrieve the transaction status. For example:

```
GET /api/tool/Transaction/e6ac7c1e-c63a-11e3-9af5-08002791605b
```

Example of an Asynchronous Mutator Transaction with nowait=true

Request:

```
POST http://172.29.232.238/api/data/Countries/?hierarchy=1c0ffee2c0deab00da595101&nowait=true
Payload of the request:

{'country_name': 'Callback Created Example Country Name',
 'request_meta': {'callback_password': 'password',
                  'callback_url': 'http://localhost:9365',
                  'callback_username': 'username',
                  'external_id': '3x4mpl3-3xt3rn4l-7d',
                  'external_reference': 'External Ref'}}
```

Synchronous response:

```
{
href: "/api/tool/Transaction/e6ac7c1e-c63a-11e3-9af5-08002791605b"
success: true
transaction_id: "e6ac7c1e-c63a-11e3-9af5-08002791605b"
}
HTTP 202 ACCEPTED
```

Asynchronous transaction status callback (console output of the simple http service provided in the separate example section):

```
POST - 2014-04-17 16:16:43.737509


Headers:

{'accept-encoding': 'identity',
 'authorization': 'Basic dXNlcm5hbWU6cGFzc3dvcmQ=',
 'content-length': '275',
 'content-type': 'application/json',
 'host': 'localhost:8080'}


Raw Callback Body:

'{"status": "Fail", "transaction":
{"href":
   "http://django.testserver/api/tool/Transaction/34866060-fd47-11e3-88dd-080027880ca6/",
 "id": "34866060-fd47-11e3-88dd-080027880ca6"},
 "resource": {"hierarchy": "1c0ffee2c0deab00da595101",
              "model_type": "data/Countries",
              "pkid": "53ac3d41c9527062809c0021"},
              "external_reference": "External Ref",
              "external_id": "3x4mpl3-3xt3rn4l-7d"}'

Pretty Callback Body:
```

```
{u'external_id': u'3x4mpl3-3xt3rn4l-7d',
 u'external_reference': u'External Ref',
 u'resource': {u'hierarchy': u'1c0ffee2c0deab00da595101',
               u'model_type': u'data/Countries',
               u'pkid': u'53ac3d41c9527062809c0021'},
 u'status': u'Fail',
 u'transaction': {u'href':
   u'http://django.testserver/api/tool/Transaction/34866060-fd47-11e3-88dd-080027880ca6/',
                  u'id': u'34866060-fd47-11e3-88dd-080027880ca6'}}

localhost - - [17/Apr/2014 16:16:43] "POST / HTTP/1.1" 200 –
```

Correlation Identifiers

In order to allow an external system use its own identifiers to cross-reference transactions in the system, the Cisco Unified Communications Domain Manager 10.x/11.5(x) API supports two external identifiers for all transactions. This allows the external system to:

1. Tie together multiple transactions in the system (using for example an order number)
2. Track individual requests in the system using the external IDs.

External identifiers are not supported for the parent transactions tool/BulkLoad and tool/DataImport.

The transaction log includes these two IDs and the transaction log, as in the figure below called *An example transaction log showing IDs*.

You can obtain the details of the parent transaction with a given ID by using the following API call:

```
GET http://my.fulfillmentserver/api/v0/tool/Transaction/?hierarchy=1c0ffee2c0deab00da595101&

     filter_condition=contains&
     format=json&
     filter_text=3x4mpl3-3xtern4l-FF&
     filter_field=external.id
```

You can obtain the details of transactions tied together using an external reference number using the following API call:

```
GET http://my.fulfillmentserver/api/v0/tool/Transaction/?hierarchy=1c0ffee2c0deab00da595101&

     filter_condition=contains&
     format=json&
```

```
filter_text=Example%20External%20Reference-FF&
filter_field=external.reference
```

Figure 1: An example transaction log showing IDs - snippet A.

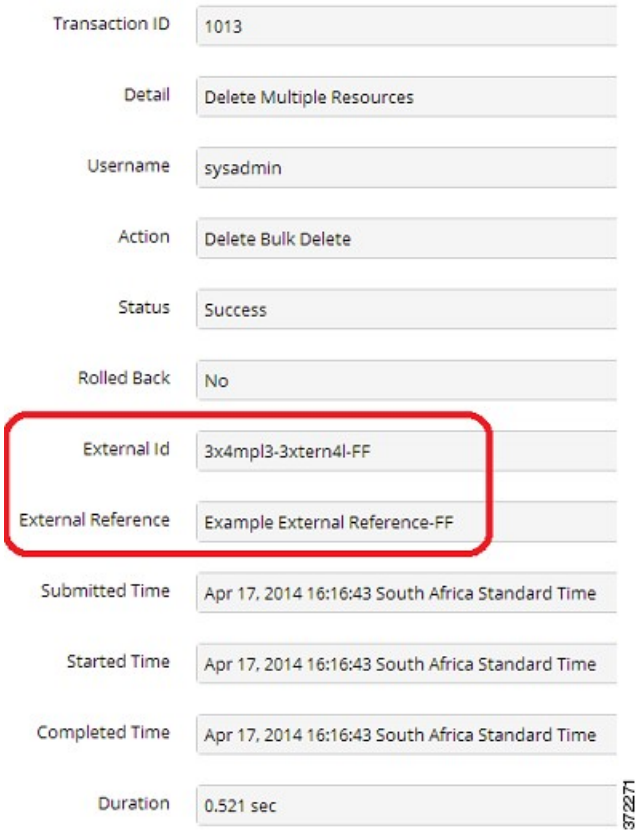
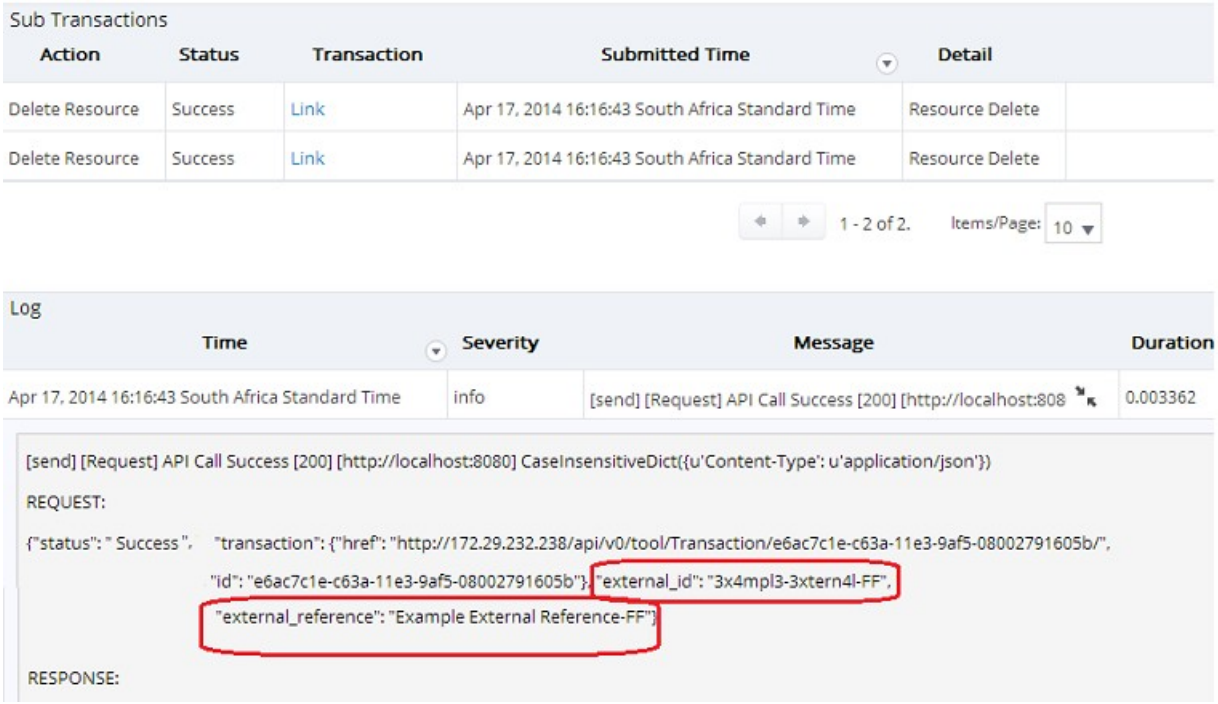


Figure 2: An example transaction log showing IDs - snippet B.



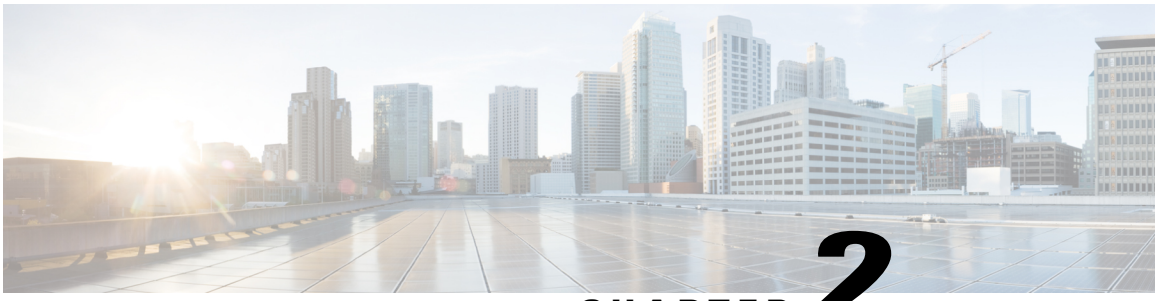
Example Of A Simple HTTP Server

The following code is an example of a simple HTTP server that can be used to test basic async transaction status callback operations. The code is not intended for actual use.

Note The HTTP 200 ACK is sent asynchronously *before* internal processing of the callback.

```
#!/usr/bin/env python
from datetime import datetime
import SimpleHTTPServer
import SocketServer
import logging
import cgi
import json
from pprint import pprint
PORT = 8080

class ServerHandler(SimpleHTTPServer.SimpleHTTPRequestHandler):

    def do_GET(self):
        SimpleHTTPServer.SimpleHTTPRequestHandler.do_GET(self)

    def do_POST(self):
        self.send_response(200)
        self.wfile.write("ACK")

        # Insert internal processing here.
        # Below is an example of internal processing that simply prints out the
        # callback request.
        print "\nPOST - {}".format(datetime.now())
        print "Headers:"
        pprint(dict(self.headers))
        print "\nRaw Body:"
```

```
        body = self.rfile.read(int(self.headers['Content-Length'])).decode('utf-8')
        pprint(body)
        print "\nPretty Body:"
        pprint(json.loads(body))


Handler = ServerHandler

httpd = SocketServer.TCPServer(("", PORT), Handler)

print "Serving at port", PORT
httpd.serve_forever()
```



CHAPTER 2

Using the API

Developer Guidelines

The following practices are recommended to all developers. The aim is to reduce the number and extent of any updates that may be necessary.

1. Third-party API clients **must** specify the API Version when integrating with Cisco Unified Communications Domain Manager 10.x/11.5(x). This is required if the third-party clients want to make use of backwards compatibility, since it ensures that they continue to receive consistent schemas.
2. The order of elements within the interface schema and messages may change, within the constraints of the interface specification. Developers should avoid unnecessary dependence on the order of elements to interpret information exchanged with Cisco Unified Communications Domain Manager 10.x/11.5(x).
3. New interface methods, operations, actions, requests, responses, headers, parameters, attributes, other elements, or new values of existing elements, may be introduced into the Cisco Unified Communications Domain Manager 10.x/11.5(x) interfaces. Developers should disregard or provide generic treatments where necessary for any unknown elements or unknown values of known elements encountered.
4. Notifications, operations, methods, actions, requests, responses, headers, parameters, attributes, and other elements from previous versions ofCisco Unified Communications Domain Manager 10.x/11.5(x) interfaces, will remain, and will maintain their previous meaning and behavior to the extent possible and consistent with the need to correct defects.
5. Applications should not be dependent on interface behavior resulting from defects (behavior not consistent with published interface specifications), since the behavior can change when defects are fixed.
6. The use of deprecated methods, operations, actions, handlers, requests, responses, headers, parameters, attributes, or other elements should be removed from applications as soon as possible to avoid issues when those deprecated items are removed fromCisco Unified Communications Domain Manager 10.x/11.5(x) or its interfaces.

7 Application Developers should be aware that not all new features and new supported devices (for example, phones) will be forward compatible. New features and devices may require application modifications to be compatible or to make use of the new features or devices.

Workflow Tasks

Procedure

Step 1 Log in with the username "hcsadmin@sys.hcs ", using the password that was set during the installation.

Step 2 Get the Provider Name & Provider PKID using the **data/HierarchyNode** API in the url, where "hierarchy" = "sys.hcs".

Step 3 **Note** For all POST/PUT/DELETE operations to be asynchronous transactions, set the query param "nowait=true" in the URI.

To create a provider Admin use the **relation/HcsAdminUserREL** API in the url, with the hierarchy value that you receive in the get call of Step-1 (PKID or the dot notation).

Step 4 **Note** Creating a Reseller is not mandatory, and it depends on the structure of provisioning. A Reseller must be created if the tree structure of the provisioning is: Provider -> Reseller -> Customer -> Site.

To create a Reseller use the **relation/HcsResellerREL** API, with the Provider hierarchy of the API of Step-1.

Step 5 To create a Reseller Admin use the **relation/HcsAdminUserREL** API in the url, with the hierarchy value of the Reseller (PKID or the dot notation).

Step 6 To create a Shared Network Device (Cisco Unified Communications Manager or Unity), it needs to be done either at either the Provider Hierarchy or the Reseller Hierarchy Level. Use the following APIs for each of the devices listed below:

Device	API
Cisco Unified Communications Manager	relation/HcsCallManagerREL
Unity	relation/HcsUnityConnectionREL
Presence	relation/HcsPresenceREL
WebEx	relation/HcsWebExREL

Step 7 **Note** The Customer is directly under Provider if the deployment structure is **Provider -> Customer -> Site** or under the Reseller if the deployment structure is **Provider ->Reseller ->Customer -> Site**.

To create a Customer use the **relation/HcsCustomerREL** API, with the hierarchy of provider/reseller that can be retrieved using the respective API.

Step 8 To create a Customer Admin use the **relation/HcsAdminUserREL** API in the url, with the hierarchy value of the Customer(PKID or the dot notation).

Step 9 **Note** If the Customer is using "shared_uc_apps" then, you cannot add dedicated devices for that customer.

Adding a Dedicated Network Device for a Customer(Cisco Unified Communications Manager or Unity) needs to be done at the Customer hierarchy Level. Use the following APIs for each of the devices listed below:

Device	API
Cisco Unified Communications Manager	relation/HcsCallManagerREL

Device	API
Unity	relation/HcsUnityConnectionREL
Presence	relation/HcsPresenceREL
WebEx	relation/HcsWebExREL

Step 10 Once the devices are configured a Network Device List (NDL) needs to be configured for the Customer. To create an NDL use the **relation/HcsNetworkDeviceListREL** API. At least one device is required to add an NDL.

Step 11 **Note** It is not mandatory to have a Network Device List (NDL) to create a Site. However, an NDL is needed to add a Subscriber or Phone or Lines to a Site. Sites created without an NDL can later be able associated to one.

To create a site use the **relation/HcsSiteREL** API.

Step 12 **Note** Using the **relation/HcsAdminUserREL** API will only be local Cisco Unified Communications Domain Manager admin. To add an Admin who is also a Subscriber, use the **relation/HcsUserREL** API, which can later be moved to any Cisco Unified Communications Manager that is associated with the Site.

To create a Site Admin use the **relation/HcsAdminUserREL** API in the url, with the hierarchy value of the Site (PKID or the dot notation).

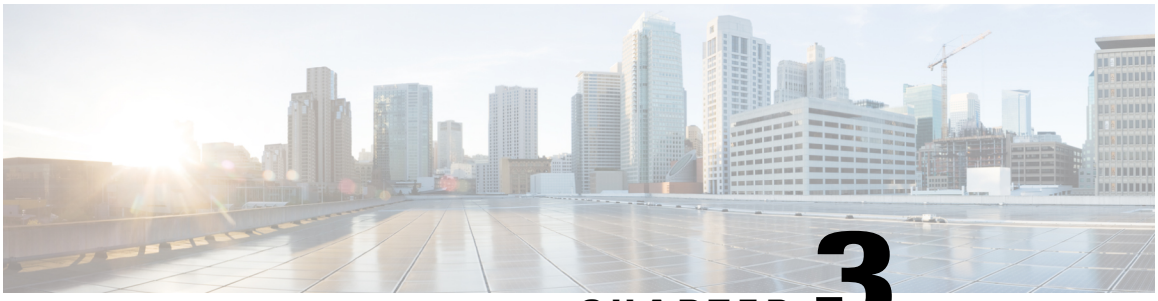
Step 13 Complete the following activities at the Site level:

a) To create a Subscriber use the **relation/Subscriber** API.
b) To create a Line use the **view/HcsDNMgmtVIEW** API.
c) To create a Phone use the **relation/SubscriberPhone** API.
d) To create a Voicemail use the **relation/Voicemail** API.

Developer Tools

The Developer tools that are available in Mozilla Firefox and Google Chrome allow all the actions exposed by each API to be displayed as they are being called in the GUI. This gives us the opportunity to view the request and response actions as they occur, and provide the details of what each API provides and its relationship to the GUI.

With Developer tools enabled, the network tab of the Developer tools show the information that is contained in each request and response as you navigate and use the GUI. This allows service providers a direct view as to what data each API requires.



CHAPTER 3

Tools API

Introduction to Tool APIs

This section describes API calls that are not related to a specific model. The full URL would include the host name: *http://[hostname]*.

The calls described here all contain */tool/* in the URL.

Variables are enclosed in square brackets, e.g.:

- [hierarchy] is the hierarchy UUID
- [filename] refers to a file

Other parameters are described with the relevant API call.

Search and Search Result Export

For an API call that carries out a search, a POST payload in JSON format is added.

Task	Call	URL	Parameters	Payload
Search	POST	/api/tool/Search/	format=json hierarchy=[hierarchy]	{"query":"[query]"}

The value of [query] follows Search syntax, for example:

```
{"query":"data/Countries with country_name contains King"}
```

The Request payload can also be a GET parameter, for example:

Task	Call	URL	Parameters	Response
Search	GET	/api/tool/Search/	format=json hierarchy=[hierarchy] query=[url_query]	JSON format of the search result.

The value of [url_query] is URL encoded string, for example:

```
data/Countries%20with%20country_name%20contains%20King
```

Furthermore, the meta property of the schema in the response to /api/tool/Search/ contain action details for the export of search results. This includes the URL for the data export POST request:

```
/api/export/export_data/?url=/api/tool/Search/
```

as well as the URL:

```
/api/view/ExportData/add
```

which has a schema that lists the data export data type choices that are used as a parameter to the POST call.

Bulk Load API

Two API calls are required.

Task	Call	URL	Parameters	Response

Submit file	POST	`/api/uploadfiles/`
		This URL will be moved to tool/UploadFile in future.
			`hierarchy=[hierarchy]`
			`Content-Type: multipart/form-data`
			
			`name='uploadedfile'`
			`filename=<filename>`
			`the file to upload`
				`{"uploadedfiles": [{"id": "<file_id>", "name": "<filename>"}]}`

The response is HTTP 202

Task	Call	URL	Parameters	Payload

Bulk Load	POST	`/api/tool/ BulkLoad/`	`method= bulkload_spreadsheet hierarchy=[hierarchy]`	Examples:
				`{'bulkload_file': '<filename>', 'execute_immediately': true}`
				or:
				`{'bulkload_file': '<filename>', 'execute_immediately': false 'execute_date': '2013-06-20', 'execute_time': '12:00:00', 'execute_timezone': '0'}`

The following `curl` commands illustrate the two steps:

Step 1

```
curl -H 'Authorization: Basic <auth_key>'
     -F uploadedfile="@<file>.xlsx"
     'http://<hostname>/api/uploadfiles/'
```

Step 2

```
curl -H 'Authorization: Basic <auth_key>'
     -H 'Content-Type: application/json'
     -H 'accept: application/json'
     --data-binary '{"bulkload_file":"DEMO.xlsx","execute_immediately":true}'
     'http://<host>/api/tool/BulkLoad/?hierarchy=[hierarchy]&
       method=bulkload_spreadsheet&
       nowait=true&
       format=json'
```

The response to this call is for example as in the following table.

Response
`{"href": "/api/tool/Transaction/0b340a6f-b658-48bb-ac8c-7562adc5572d", "success": true, "transaction_id": "0b340a6f-b658-48bb-ac8c-7562adc5572d"}`

- If the Bulk Load is to be scheduled, the payload of the second task includes schedule details:
  - *execute_immediately* is set to false
  - *execute_date* is added in the format YYYY-MM-DD
  - *execute_time* is added in the format HH:MM:SS
  - *execute_timezone* is added in the format of a numeric value in minutes relative to UTC. For example, UTC is 0, UTC+2:00 is 120, UTC-1:00 is -60, and so on.
- An entry is also generated in the schedule; that is, an instance is added to the data/Schedule module.

- If the second task payload has *'execute_immediately': true*, a POST is generated to /api/data/Bulkload/. The payload includes the uploaded filename and a generated name and time stamp as well as a description, for example:

```
{'filename': '<file>.xlsx', 'description': 'Generated by Bulk Loader
Administration Tools', 'name': 'AnyUser.xlsx -- 2013-05-21
16:47:11.801664 (UTC)'}
```

To inspect the detailed progress and status of the transaction, use the API call from the response above:

`GET /api/tool/Transaction/[pkid]`

with parameters:

- `hierarchy=[hierarchy]`
- `format=json`

The response to this GET call is a JSON object that provides details of the transaction, as for example in the truncated snippet:

```
...
   "href": "/api/tool/Transaction/[pkid]
    "log_id": "53a8053ea616540708141f44",
    "message": "data_Countries_bulkloadsheet.xlsx is a valid
    "severity": "info",
    "time": "2014-06-23T10:45:18.029000",
    "transaction_id": "[pkid]"
  }
],
"pkid": "[pkid]",
"resource": {},
"rolled_back": "No",
"started_time": "2014-06-23T10:45:17.813000",
"status": "Success",
"sub_transactions": [
  {
    "action": "Execute Resource",
    "detail": "Execute : data_Countries_bulkloadsheet.xlsx -- ...
    "status": "Success",
    "submitted_time": "2014-06-23T10:45:19.567000",
    "transaction": "/api/tool/Transaction/[pkid1]    ...
  },
  {
    "action": "Create Schedule",
    "detail": "Name:data_Countries_bulkloadsheet.xlsx -- 2014- ...
    "status": "Success",
    "submitted_time": "2014-06-23T10:45:18.912000",
    "transaction": "/api/tool/Transaction/[pkid2]    ...
  },
  {
    "action": "Create Bulk Load",
    "detail": "Name:data_Countries_bulkloadsheet.xlsx -- 2014-06 ...
    "status": "Success",
    "submitted_time": "2014-06-23T10:45:18.419000",
    "transaction": "/api/tool/Transaction/[pkid3]    ...
  }
],
"submitted_time": "2014-06-23T10:45:17.794000",
```

The same transaction displays on the GUI.

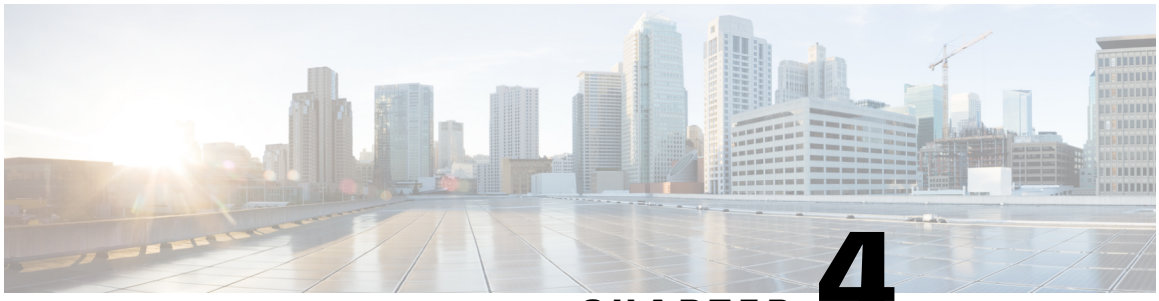
For long transactions, to retrieve a summary of the status of the transaction, the transaction can be polled, using `poll` in the URL, using the same parameters:

`GET /api/tool/Transaction/poll/?transactions=[pkid]`

In this case, there is a shortened response, for example:

```
{"[pkid]":
 {"status": "Processing",
```

```
  "href": "/api/tool/Transaction/0b340a6f-b658-48bb-ac8c-7562adc5572d",
  "description": null}
}
```



CHAPTER 4

Transactions

List Transaction

To list transactions on the system use the following operation

```
GET https://<server_address>/api/tool/Transaction/
?hierarchy=[hierarchy]
&format=json
&summary=true
```

The following query parameter illustrates how a second page of 50 transactions in the transaction log is requested.

```
skip=50
&limit=50
&hierarchy=[hierarchy]
&format=json
&summary=true
&direction=desc
```

```
&order_by=submitted_time
```

For further information on the query parameters refer to API Parameters above.

The synchronous response contains:

- pagination information
- meta data specifying the summary attributes of the transaction log view
- resources containing a list of the transactions in the transaction log

Get Instance Transactions

The status of a specific transaction can be retrieved by using a GET call to /tool/Transaction for a specific transaction PKID. The PKID, also known as the transaction ID, or transaction_id is available in the synchronous response to an asynchronous mutator transaction.

For example, if the transaction_id in the response is [pkid], then the transaction can be polled with:

```
GET https://<server_address>/api/tool/Transaction/[pkid]
```

Poll Transactions

It is recommended to use asynchronous transaction call back mechanism described in “Asynchronous Mutator Transaction Status Callback”. If this can not be used, a consumer of the Cisco Unified Communications Domain Manager 10.x/11.5(x) API can also use this polling mechanism to poll the status of individual transactions using the poll action of the transaction tool. A user interface that allows a user to monitor the progress of a given transaction can use the following method to retrieve the status of a given transaction:

```
GET /api/tool/Transaction/[pkid]/poll/?format=json
```

The response contains essential status of the transaction, for example:

```
{
[pkid]: {
status: "Success",
href: "/api/tool/Transaction/[pkid]",
description: "Name:RDP-auser1857 Description:RD for auser1857"
}
}
```

Replay Transactions

Transactions that have failed can, under certain circumstances, be replayed. This means that the transaction is re-submitted with the original request parameters. This is done by specifying the PKID of the transaction

```
GET https://<server_address>/api/tool/Transaction/[pkid]/replay/
```

The transaction current operation replays the transaction and the result returns the list view of the transaction log.

Edit and Reply Transactions

Transactions can be edited and then replayed under certain circumstances. This means that the transaction is re-submitted with the updated request parameters. This is done by specifying the PKID of the transaction:

```
GET https://<server_address>/api/tool/Transaction/[pkid]/edit-replay/
```

The transaction current operation edit and then replays the transaction and the result returns the list view of the transaction log.

Sub Transactions

The sub-transactions of a transaction with PKID can be retrieved by submitting the following URI

```
GET https://<server_address>/api/tool/Transaction/[pkid]/sub_transaction/
```

Log Transactions

The log messages of a transaction with PKID can be retrieved by submitting the following URI

```
GET https://<server_address>/api/tool/Transaction/[pkid]/log/
```

Transaction Choices

Use the available URL endpoint and parameter to list the transaction actions as they may be shown in the transaction log.

The API call to get the list of transaction actions uses the parameter and value: field=action, for example:

```
GET api/tool/Transaction/choices/?
    field=action&
    hierarchy=[hierarchy]&
    format=json
```

The output shows the list of transaction actions:

```
[
    {
    "value": "Auto Migrate Base Customer Dat",
    "title": "Auto Migrate Base Customer Dat"
    },
    {
    "value": "Auto Migrate Base Provider",
    "title": "Auto Migrate Base Provider"
    },
    {
    "value": "Auto Migrate Base Site Dat",
    "title": "Auto Migrate Base Site Dat"
    },
    {
    "value": "Auto Migrate Dial Plan",
    "title": "Auto Migrate Dial Plan"
    },
    {
    "value": "Auto Migrate Feature Subscriber Phone Cft",
    "title": "Auto Migrate Feature Subscriber Phone Cft"
    },
    {
    "value": "Auto Migrate Hotdial Data",
```

```
    "title": "Auto Migrate Hotdial Data"
    },
    {
    "value": "Auto Migrate Init Ippbx",
    "title": "Auto Migrate Init Ippbx"
    },
    {
    "value": "Auto Migrate Internal Number Inventory",
    "title": "Auto Migrate Internal Number Inventory"
},
...
```

Transaction Filters

In addition to the filter parameters that can be applied to transactions as indicated in the topic on AP Parameters, transactions in particular can be filtered:

- by the following values for the URL parameter `filter_field`:
  - Transaction ID: `id`
  - Start or end submitted time: `submitted_time`
  - The transaction message: `message`
- by also listing sub transactions using the URL parameter and value `subtransactions=true`. By default, sub transactions are not listed, in other words, the value is `false`.
- To carry out a filter on sub-transactions of a parent transaction, the `/sub-transactions/` endpoint is added to the GET request:

  `/api/tool/Transaction/[parent-pkid]/sub-transactions/`
- To carry out a filter on transaction logs of a parent transaction, the `/logs/` endpoint is added to the GET request:

  `/api/tool/Transaction/[parent-pkid]/log/`

  The transaction filters do not apply to logs.

The parameters can have the `filter_condition` values:

- `eq` (equals)
- `ne` (not equals)
- `gt` (greater than)
- `gte` (greater than or equals)
- `lt` (less than)
- `lte` (less than or equals)

The date-time is a `filter_text` value for `filter_field=submitted_time`.

The format follows RFC3339: `[https://tools.ietf.org/html/rfc3339]` and is *YYYY-MMDDTHH:MM:SS.fZ*, where:

- “T” is the time separator and the character should be added.

- "Z" indicates UTC time and the character should be added.
- "f" represents the decimal fraction of a second and the character should not be added. The specification of the decimal fraction is optional.

For example:

June 29 2016 14 hours 41 minutes 0.01 seconds UTC, is:
```
2016-06-29T14:41:00.01Z
```

To filter for transactions after this date-time, the API call is:
```
GET api/tool/Transaction/?
    hierarchy=[hierarchy]
    &format=json
    &filter_field=submitted_time
    &filter_text=2016-06-29T14:41:00.01Z
    &filter_condition=gt
```

To filter between transaction IDs or times, two parameter sets are needed.

For example:

- To filter transaction IDs between 12000 and 13000:
```
GET api/tool/Transaction/?
    hierarchy=[hierarchy]
    &format=json
    &filter_field=id
    &filter_text=12000
    &filter_condition=gt
    &filter_field=id
    &filter_text=13000
    &filter_condition=lt
```
- To filter transactions between June 29 2016 14 hours 41 minutes UTC and June 29 2016 15 hours 41 minutes UTC (no fraction of a second in the example):
```
GET api/tool/Transaction/?
    hierarchy=[hierarchy]
    &format=json
    &filter_field=submitted_time
    &filter_text=2016-06-29T14:41:00Z
    &filter_condition=gt
    &filter_field=submitted_time
    &filter_text=2016-06-29T15:41:00Z
    &filter_condition=lt
```

If the upper or lower bound in the filter are not available, the transactions with values between the filte values and the bound are returned.

When the URL parameter `subtransactions=true` is used, the `data` object in the JSON API response shows:

- a parent transaction has: `parent: null`
- a sub transaction has: `parent: <pkid>`, where `<pkid>` is the value of the parent attribute `pkid`.

The example snippets below show the values of `parent`:
```
data: {
    username: "system",
    status: "Success",
    description: "",
    parent: null,
    pkid: "01a559c5-e77f-40e7-8403-683d7204d1e1",
    friendly_status: "Success",
    detail: "HcsLdapSyncSchedule--1",
    action: "Execute Schedule",
    href: "/api/tool/Transaction/01a559c5-e77f-40e7-8403-683d7204d1e1/",
    txn_seq_id: "17693",
```

```
    data_type_: "tool/Transaction",
    message: "",
    submitted_time: "2016-07-14T12:13:41.758000Z"
}
data: {
    username: "system",
    status: "Success",
    description: "",
    parent: "f4daa234-590d-4002-a3b0-8c329c583d1d",
    pkid: "019f44a3-df6e-4e4f-86f3-a09a6b91e482",
    friendly_status: "Success",
    detail: "10.120.2.221",
    action: "Import Ldap",
    href: "/api/tool/Transaction/019f44a3-df6e-4e4f-86f3-a09a6b91e482/",
    txn_seq_id: "17695",
    data_type_: "tool/Transaction",
    message: "models completed.",
    submitted_time: "2016-07-14T12:13:43.075000Z"
}
```

- In the case of a transaction error, the `message` attribute value will show the corresponding error message. If a custom message was defined in a provisioning workflow, and the response is the result of the workflow, the value will be the custom message.

  To filter transaction messages, the parameter `filter_field=message` is used, with at least one of the following additional filter criteria:

  - a date range of maximum 7 days using `submitted_time`
  - an additional `filter_field`, with one of the conditions:
    - `filter_condition=contains`
    - `filter_condition=startswith`

  Also required is the case sensitive parameter:

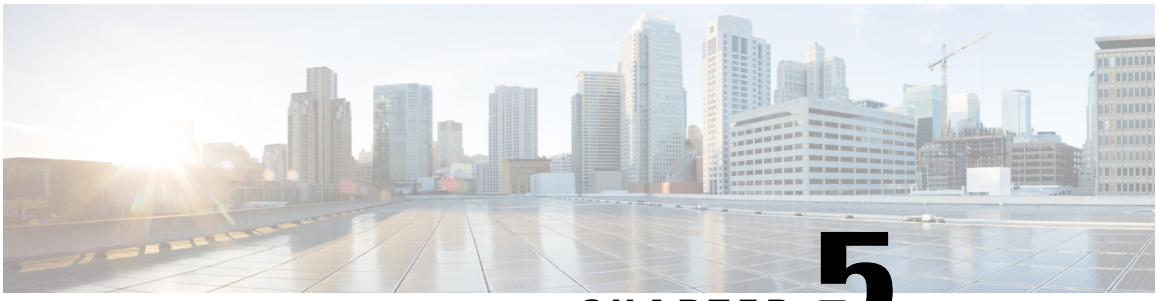
  - `ignore_case`

Note Note that a filter is by default case insensitive. If the case is explicitly set, then it must be added to each filter parameter group to ensure proper parameter grouping.

The additional criteria do not apply to sub-transaction message filters, because the `[parent-pkid]` in the URL serves as an additional filter.

The example below is a message filter that contains “Invalid business key”, by date range:

```
GET api/tool/Transaction/?
    hierarchy=[hierarchy]
    &format=json
    &filter_field=submitted_time
    &filter_text=2016-06-25T14:41:00Z
    &filter_condition=gt
    &ignore_case=false
    &filter_field=submitted_time
    &filter_text=2016-06-29T15:41:00Z
    &filter_condition=lt
    &ignore_case=false
    &filter_field=message
    &filter_text=Invalid%20business%20key
    &filter_condition=contains
    &ignore_case=false
```



CHAPTER 5

Examples

Sample API Activities

Overview

The examples displayed herein illustrate using API request to perform varying actions, including creating, retrieving, updating, and deleting single instances, creating multiple instances, exporting instance json, and viewing transaction details for a requested operation. Activities are performed in these examples using the curl command line tool. Each example shows the command and the console output.

Conventions

- API calls are illustrated:
  - From server https://localhost.
  - Referencing a relation: relation/LineRelation.
- User authorization is for two administrator users:
  - One user has additional permissions to import and bulk load.
  - The users can be distinguished by the authorization header and hierarchy parameter in the URL.
- Field Display Policies and Configuration Templates in URL parameters can differ according to the MenuLayout associated with the user role; for example, the url parameter below represents a Field Display Policy applied to relation/LineRelation from a Site administrator user menu.

  ```
  &policy_name=LineMenuFDPSite
  ```
- Where a response to an API call shows an instance of /api/tool/Transaction/, the transaction instance details can be inspected with a GET call to this instance.
- Some console output and payload files are truncated (indicated with ellipses or text: "snippets").
- Line breaks have been added to console output in the examples for better formatting.

Add an Instance

Description

In this example, we create a single instance of relation/LineRelation.

Example Details

Task

POST an instance of relation/LineRelation.

User

Site Administrator

Json Input Body (snippet of the file post-payload.json)

Represents the model instance details

```
{
"data": {
   "partyEntranceTone": "Default",
   "cfaCssPolicy": "Use System Default",
   "autoAnswer": "Auto Answer Off",
   "callForwardNotRegisteredInt": {
      "callingSearchSpaceName": "Intl24HrsEnh-locus1"
   },
   "routePartitionName": "Site-locus1",
   "callForwardOnFailure": {
      "callingSearchSpaceName": "Intl24HrsEnh-locus1"
   },
   "shareLineAppearanceCssName": "Intl24HrsEnh-locus1",
   "callForwardBusy": {
      "callingSearchSpaceName": "Intl24HrsEnh-locus1"
   },
   "pattern": "90217",
   "patternPrecedence": "Default",
   "callForwardNoAnswer": {
      "callingSearchSpaceName": "Intl24HrsEnh-locus1"
   },
   "callForwardNoCoverage": {
      "callingSearchSpaceName": "Intl24HrsEnh-locus1"
   },
   "callForwardNotRegistered": {
      "callingSearchSpaceName": "Intl24HrsEnh-locus1"
   },
   "usage": "Device",
   "alertingName": "techsupport",
   "enterpriseAltNum": {
   "isUrgent": false,
   "addLocalRoutePartition": false,
   "advertiseGloballyIls": true
...
}
```

Example

Request

```
$ curl -v -H 'Authorization: Basic YWRtaW5AbG9jdXMxLmNvbTpwYXNzd29yZA==' -H
'Content-Type:application/json'
```

```
 --data-binary @post-payload.json -X POST 'http://localhost/api/relation/LineRelation/
?hierarchy=55b9dc81a6165413b9d16ab6&policy_name=LineMenuFDPSite&template_name=line-cft&nowait=true&format=json'
+ About to connect() to localhost port 80 (#0)
+ Trying 127.0.0.1... connected
> POST
/api/relation/LineRelation/?hierarchy=55b9dc81a6165413b9d16ab6&policy_name=LineMenuFDPSite&template_name=
line-cft&nowait=true&format=json HTTP/1.1
> Authorization: Basic YWRtaW5AbG9jdXMxLmNvbTpwYXNzd29yZA==
> User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu)
              libcurl/7.22.0
              OpenSSL/1.0.1
              zlib/1.2.3.4
              libidn/1.23
              librtmp/2.3
> Host: localhost
> Accept: */*
> Content-Type:application/json
> Content-Length: 1941
> Expect: 100-continue
```

Table 1: Request Items to Note

POST /api/relation/LineRelation	Requested API operation.
hierarchy=55b9dc81a6165413b9d16ab6	Hierarchy at which the POST is performed.
template_name=line-cft	ConfigurationTemplate name.
policy_name=LineMenuFDPSite	FieldDisplayPolicy name.
nowait=true	Perform the operation asynchronously.
format=json	Request a json response body.

Response

```
< HTTP/1.1 100 Continue
< HTTP/1.1 202 ACCEPTED
< Server: nginx/1.1.19
< Date: Thu, 30 Jul 2015 13:10:46 GMT
< Content-Type: application/json
< Transfer-Encoding: chunked
< Connection: keep-alive
< Vary: Accept, Cookie, Accept-Language, X-CSRFToken
< Content-Language: en-us
< Allow: GET, POST, DELETE, HEAD, OPTIONS
< X-CSRFToken: d2q7nV4aWDWFpuazsnRvJVMcj9qX5Ksg
< Set-Cookie: csrftoken=d2q7nV4aWDWFpuazsnRvJVMcj9qX5Ksg;httponly; Path=/
< Set-Cookie: sessionid=hahbo0wy7sa8u8rfaiz2tcqxvkvwshp8;httponly;Path=/
<
+ Connection #0 to host localhost left intact
+ Closing connection #0
{"href": "/api/tool/Transaction/aff36c0b-ff6a-431b-be58-d2f636edb7cd/",
 "success": true,
 "transaction_id": "aff36c0b-ff6a-431b-be58-d2f636edb7cd"}
```

Table 2: Response Items to Note

HTTP/1.1 202 ACCEPTED	The request was accepted and is being processed. If this were a successful synchronous request, the HTTP response would have been 201 CREATED.

"transaction_id": "aff36c0b-ff6a-431b-be58-d2f636edb7cd"	The operation's transaction id.
"success" : true	The request was successfully posted.

Retreive an Instance

Description

In this example, we retrieve instances of relation/LineRelation.

Example Details

Task

GET instances of relation/LineRelation.

User

Site Administrator

Json Input Body

No body is necessary when retrieving an instance.

Example

Request

```
$ curl -v -H 'Authorization: Basic YWRtaW5AbG9jdXMxLmNvbTpwYXNzd29yZA==='
-X GET
'http://localhost/api/relation/LineRelation/?hierarchy=55b9dc81a6165413b9d16ab6&policy_name
=LineMenuFDPSite&template_name=line-cft&format=json'
* About to connect() to localhost port 80 (#0)
+ Trying 127.0.0.1... connected
> GET
/api/relation/LineRelation/?hierarchy=55b9dc81a6165413b9d16ab6&policy_name=LineMenuFDPSite&template_name
=line-cft&format=json HTTP/1.1
> Authorization: Basic YWRtaW5AbG9jdXMxLmNvbTpwYXNzd29yZA==
> User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu)
              libcurl/7.22.0
              OpenSSL/1.0.1
              zlib/1.2.3.4
              libidn/1.23
              librtmp/2.3
> Host: localhost
> Accept: */*
```

Table 3: Request Items to Note

GET /api/relation/LineRelation/	Requested API operation. If the PKID were included here, then only that single instance would be retrieved
hierarchy=55b9dc81a6165413b9d16ab6	Hierarchy at which the operation is performed

template_name=line-cft	ConfigurationTemplate name
policy_name=LineMenuFDPSite	FieldDisplayPolicy name
format=json	Request a json response body

Response

```
< HTTP/1.1 200 OK
< Server: nginx/1.1.19
< Date: Thu, 30 Jul 2015 13:31:00 GMT
< Content-Type: application/json
< Transfer-Encoding: chunked
< Connection: keep-alive
< Vary: Accept, Cookie, Accept-Language, X-CSRFToken
< Content-Language: en-us
< Allow: GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS
< X-CSRFToken: a6fFDYZyk9ET8K8xTq9HITFrRi8TROrV
< Set-Cookie: csrftoken=a6fFDYZyk9ET8K8xTq9HITFrRi8TROrV;httponly;Path=/
< Set-Cookie: sessionid=9i0w39d1d32mdx6fs2skl564y8pmhmu9;httponly;Path=/
<
+ Connection #0 to host localhost left intact
+ Closing connection #0
{
...
  "data": {
    "pattern": "90124",
    "patternPrecedence": "Default",
    "callForwardNoAnswer": {
      "destination": null,
      "forwardToVoiceMail": false,
      "callingSearchSpaceName": null
    },
    "hrInterval": null,
    "callForwardNoCoverage": {
      "destination": null,
      "forwardToVoiceMail": false,
      "callingSearchSpaceName": null
    },
    "callForwardNotRegistered": {
      "destination": null,
      "forwardToVoiceMail": false,
      "callingSearchSpaceName": null
    },
    "usage": "Device",
    "summary_device": "10.120.2.216, 8443, prov1.cust1",
    "hrDuration": null,
    "parkMonForwardNoRetrieveVmEnabled": false,
    "alertingName": "Helpdesk",
    "description": "DN created without device from QAS.",
    "directoryURIs": null,
    "aarVoiceMailEnabled": false,
    "hierarchy_path": "sys.prov1.cust1.locus1",
    "parkMonForwardNoRetrieveIntCssName": null,
    "parkMonForwardNoRetrieveDn": null,
    "allowCtiControlFlag": true,
    "defaultActivatedDeviceName": null,
    "parkMonReversionTimer": null,
    "releaseClause": "No Error",
    "e164AltNum": {
      "numMask": null,
      "addLocalRoutePartition": false,
      "advertiseGloballyIls": false,
      "routePartition": null,
      "isUrgent": false
    },
    "callForwardAll": {
```

```
...
}
```

Table 4: Response Items to Note

HTTP/1.1 200 OK	The request was accepted and processed.

Update an Instance

Description

In this example, we update a single instance of relation/LineRelation.

Example Details

Task

PUT an instance of relation/LineRelation.

User

Site Administrator

Json Input Body

Represents the requested model instance details. Note the updated alertingName value of "Helpdesk".

```
{
"data": {
   "partyEntranceTone": "Default",
   "cfaCssPolicy": "Use System Default",
   "autoAnswer": "Auto Answer Off",
   "callForwardNotRegisteredInt": {
      "callingSearchSpaceName": "Intl24HrsEnh-locus1"
   },
   "routePartitionName": "Site-locus1",
   "callForwardOnFailure": {
      "callingSearchSpaceName": "Intl24HrsEnh-locus1"
   },
   "shareLineAppearanceCssName": "Intl24HrsEnh-locus1",
   "callForwardBusy": {
      "callingSearchSpaceName": "Intl24HrsEnh-locus1"
   },
   "pattern": "90217",
   "patternPrecedence": "Default",
   "callForwardNoAnswer": {
      "callingSearchSpaceName": "Intl24HrsEnh-locus1"
   },
   "callForwardNoCoverage": {
      "callingSearchSpaceName": "Intl24HrsEnh-locus1"
   },
   "callForwardNotRegistered": {
      "callingSearchSpaceName": "Intl24HrsEnh-locus1"
   },
   "usage": "Device",
   "alertingName": "techsupport",
   "enterpriseAltNum": {
      "isUrgent": false,
      "addLocalRoutePartition": false,
      "advertiseGloballyIls": true
```

```
...
}
```

Example

Request

```
$ curl -v -H 'Authorization: Basic YWRtaW5AbG9jdXMxLmNvbTpwYXNzd29yZA==' -H
'Content-Type:application/json' --data-binary @put-payload.json -X PUT
'http://localhost/api/relation/LineRelation/55b9fe59a6165413b9d17628/?hierarchy=55b9dc81a6165413b9d16ab6&policy_name=LineMenuFDPSite&template_name=line-cft&nowait=true&format=json'
+ About to connect() to localhost port 80 (#0)
+ Trying 127.0.0.1... connected
> PUT
/api/relation/LineRelation/55b9fe59a6165413b9d17628/?hierarchy=55b9dc81a6165413b9d16ab6&policy_name=LineMenuFDPSite&template_name=line-cft&nowait=true&format=json
 HTTP/1.1
> Authorization: Basic YWRtaW5AbG9jdXMxLmNvbTpwYXNzd29yZA==
> User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu)
              libcurl/7.22.0
              OpenSSL/1.0.1
              zlib/1.2.3.4
              libidn/1.23
              librtmp/2.3
> Host: localhost
> Accept: */*
> Content-Type:application/json
> Content-Length: 1926
> Expect: 100-continue
```

Table 5: Request Items to Note

PUT /api/relation/LineRelation/55b9fe59a6165413b9d17628	Requested API operation
55b9fe59a6165413b9d17628	PKID of the model instance to update (from the POST example)
hierarchy=55b9dc81a6165413b9d16ab6	Hierarchy at which the operation is performed
template_name=line-cft	ConfigurationTemplate name
policy_name=LineMenuFDPSite	FieldDisplayPolicy name
nowait=true	Perform the operation asynchronously
format=json	Request a json response body

Response

```
< HTTP/1.1 100 Continue
< HTTP/1.1 202 ACCEPTED
< Server: nginx/1.1.19
< Date: Thu, 30 Jul 2015 13:00:33 GMT
< Content-Type: application/json
< Transfer-Encoding: chunked
< Connection: keep-alive
< Vary: Accept, Cookie, Accept-Language, X-CSRFToken
< Content-Language: en-us
< Allow: GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS
< X-CSRFToken: GgxBBhTjkB2IUib2lHgIVzeohhmK2arc
< Set-Cookie: csrftoken=GgxBBhTjkB2IUib2lHgIVzeohhmK2arc;httponly;Path=/
```

```
< Set-Cookie: sessionid=8skxwiqojuyz5xl37cdcflbr5ct5ncrk;httponly;Path=/
<
+ Connection #0 to host localhost left intact
+ Closing connection #0
{"href": "/api/tool/Transaction/0bebcaa2-df37-420f-bd15-3a00ea056092/",
 "success": true,
 "transaction_id": "0bebcaa2-df37-420f-bd15-3a00ea056092"}
```

Table 6: Response Items to Note

HTTP/1.1 202 ACCEPTED	The request was accepted and is being processed. If this were a successful synchronous request, the HTTP response would have been 204 NO CONTENT
"transaction_id": "0bebcaa2-df37-420f-bd15-3a00ea056092"	The operation's transaction id
"success" : true	The request was successfully posted

Delete an Instance

Description

In this example, we delete a single instance of relation/LineRelation.

Example Details

Task

DELETE an instance of relation/LineRelation.

User

Site Administrator

Json Input Body

No body is necessary when deleting a single instance. If the intent is to delete multiple instances, a json body may be included which containing the hrefs of the target instances.

Example

Request

```
$ curl -v -H 'Authorization: Basic YWRtaW5AbG9jdXMxLmNvbTpwYXNzd29yZA==' -X DELETE
'http://localhost/api/relation/LineRelation/55ba2482a6165413b9d19fb8/?hierarchy=55b9dc81a6165413b9d16ab6&policy_name
=LineMenuFDPSite&template_name=line-cft&nowait=true&format=json'
* About to connect() to localhost port 80 (#0)
+ Trying 127.0.0.1... connected
> DELETE
/api/relation/LineRelation/55ba2482a6165413b9d19fb8/?hierarchy=55b9dc81a6165413b9d16ab6&policy_name
=LineMenuFDPSite&template_name=line-cft&nowait=true&format=json HTTP/1.1
> Authorization: Basic YWRtaW5AbG9jdXMxLmNvbTpwYXNzd29yZA==
> User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu)
```

```
              libcurl/7.22.0
              OpenSSL/1.0.1
              zlib/1.2.3.4
              libidn/1.23
              librtmp/2.3
> Host: localhost
> Accept: */*
```

Table 7: Request Items to Note

DELETE /api/relation/LineRelation/55b9fe59a6165413b9d17628	Requested API operation
55b9fe59a6165413b9d17628	PKID of the model instance to update (from the POST example)
hierarchy=55b9dc81a6165413b9d16ab6	Hierarchy at which the operation is performed
template_name=line-cft	ConfigurationTemplate name
policy_name=LineMenuFDPSite	FieldDisplayPolicy name
nowait=true	Perform the operation asynchronously
format=json	Request a json response body

Response

```
< HTTP/1.1 202 ACCEPTED
< Server: nginx/1.1.19
< Date: Thu, 30 Jul 2015 13:21:00 GMT
< Content-Type: application/json
< Transfer-Encoding: chunked
< Connection: keep-alive
< Vary: Accept, Cookie, Accept-Language, X-CSRFToken
< Content-Language: en-us
< Allow: GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS
< X-CSRFToken: a6fFDYZyk9ET8K8xTq9HITFrRi8TROrV
< Set-Cookie: csrftoken=a6fFDYZyk9ET8K8xTq9HITFrRi8TROrV;httponly;Path=/
< Set-Cookie: sessionid=9i0w39d1d32mdx6fs2skl564y8pmhmu9;httponly;Path=/
<
+ Connection #0 to host localhost left intact
+ Closing connection #0
{"href": "/api/tool/Transaction/01de8720-d627-4e53-8e1b-e1ad66edb7bd/",
 "success": true,
 "transaction_id": "01de8720-d627-4e53-8e1b-e1ad66edb7bd"}
```

Table 8: Response Items to Note

HTTP/1.1 202 ACCEPTED	The request was accepted and is being processed. If this were a successful synchronous request, the HTTP response would have been 204 NO CONTENT
"transaction_id": "01de8720-d627-4e53-8e1b-e1ad66edb7bd"	The operation's transaction id
"success" : true	The request was successfully posted

Bulk Load an Instance

Description

In this example, we create a several instances of relation/LineRelation with one request through the bulk load mechanism. The bulk load mechanism consists of first uploading the prospective instances in a spreadsheet and then executing the bulkload operation on the uploaded file.

Example Details for Task One

Task

Upload the bulkload file (spreadsheet).

User

Provider Administrator.

Input file (snippet of the file LineRelation.xlsx)

Contains the details of the requested instances.

Example Details for Task One

Task

Upload the bulkload file (spreadsheet).

User

Provider Administrator.

Input file (snippet of the file LineRelation.xlsx)

Contains the details of the requested instances.

# Hierarchy Node	# Device	# CFT Template	# Directory Number	# Alerting Name
sys.prov1.cust1.locus1	10.120.2.216, 8443, prov1.cust1	line-cft	90218	techsupport
sys.prov1.cust1.locus1	10.120.2.216, 8443, prov1.cust1	line-cft	90219	techsupport

# Hierarchy Node	# Device	# CFT Template	# Directory Number	# Alerting Name
sys.prov1.cust1.locus1	10.120.2.216, 8443, prov1.cust1	line-cft	90220	techsupport
sys.prov1.cust1.locus1	10.120.2.216, 8443, prov1.cust1	line-cft	90221	techsupport
sys.prov1.cust1.locus1	10.120.2.216, 8443, prov1.cust1	line-cft	90222	techsupport

Example Task One

Request

```
$ curl -v -H 'Authorization: Basic YWRtaW5AcHJvdjEuY29tOnBhc3N3b3Jk' -F
uploadedfile='@LineRelation.xlsx'
'http://localhost/api/uploadfiles/?hierarchy=55b9daeca6165413b9d166de'
* About to connect() to localhost port 80 (#0)
+ Trying 127.0.0.1... connected
> POST /api/uploadfiles/?hierarchy=55b9daeca6165413b9d166de HTTP/1.1
> Authorization: Basic YWRtaW5AcHJvdjEuY29tOnBhc3N3b3Jk
> User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu)
              libcurl/7.22.0
              OpenSSL/1.0.1
              zlib/1.2.3.4
              libidn/1.23
              librtmp/2.3
> Host: localhost
> Accept: */*
> Content-Length: 10455
> Expect: 100-continue
> Content-Type: multipart/form-data;boundary=----------------------------5a0f36378f19
```

Table 9: Request Items to Note

POST /api/uploadfiles	Requested API operation
hierarchy=55b9daeca6165413b9d166de	Hierarchy at which the POST is performed

Response

```
< HTTP/1.1 100 Continue
< HTTP/1.1 200 OK
< Server: nginx/1.1.19
< Date: Thu, 30 Jul 2015 15:09:25 GMT
< Content-Type: text/html; charset=utf-8
< Transfer-Encoding: chunked
< Connection: keep-alive
< Vary: Accept-Encoding
< Vary: Accept, Cookie, Accept-Language, X-CSRFToken
< Content-Language: en-us
< Allow: POST, OPTIONS
< X-CSRFToken: C4ceiFEWSbjif1O4Jzhr1gZV9ytd9f2F
< Set-Cookie: csrftoken=C4ceiFEWSbjif1O4Jzhr1gZV9ytd9f2F;httponly;Path=/
< Set-Cookie: sessionid=07z03pbatb1qelahcc01ygufgzsr6i35;httponly;Path=/
<
+ Connection #0 to host localhost left intact
+ Closing connection #0
```

```
{"uploadedfiles": [
  {"name": "LineRelation.xlsx",
   "id": "55ba3e25a616541bb906b209"
  }
 ]
}
```

Table 10: Response Items to Note

HTTP/1.1 200 OK	The request was accepted the file was uploaded.

Example Details for Task Two

Task

Execute the bulkload operation on the uploaded spreadsheet.

User

Provider Administrator.

Request Input (JSON body)

Indicates the target bulk load spreadsheet

```
{"bulkload_file" : "LineRelation.xlsx","execute_immediately":true}
```

Example Task Two

Request

```
$ curl -v -H 'Authorization: Basic YWRtaW5AcHJvdjEuY29tOnBhc3N3b3Jk' -H 'Content-Type:
application/json' -H 'accept: application/json' --data-binary
'{"bulkload_file":"LineRelation.xlsx","execute_immediately":true}' -X POST
'http://localhost/api/tool/BulkLoad/?hierarchy=55b9daeca6165413b9d166de&method=bulkload_spreadsheet&nowait=true&format=json'
+ About to connect() to localhost port 80 (#0)
+ Trying 127.0.0.1... connected
> POST
/api/tool/BulkLoad/?hierarchy=55b9daeca6165413b9d166de&method=bulkload_spreadsheet&nowait=true&format=json
 HTTP/1.1
> Authorization: Basic YWRtaW5AcHJvdjEuY29tOnBhc3N3b3Jk
> User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu)
              libcurl/7.22.0
              OpenSSL/1.0.1
              zlib/1.2.3.4
              libidn/1.23
              librtmp/2.3
> Host: localhost
> Content-Type: application/json
> accept: application/json
> Content-Length: 64
>
+ upload completely sent off: 64out of 64 bytes
```

Table 11: Request Items to Note

POST /api/tool/BulkLoad	Requested API operation
hierarchy=55b9daeca6165413b9d166de	Hierarchy at which the POST is performed

method=bulkload_spreadsheet	Bulk load via spreadsheet
nowait=true	Execute the operation asynchronously

Response

```
< HTTP/1.1 202 ACCEPTED
< Server: nginx/1.1.19
< Date: Thu, 30 Jul 2015 14:51:22 GMT
< Content-Type: application/json
< Transfer-Encoding: chunked
< Connection: keep-alive
< Vary: Accept, Cookie, Accept-Language, X-CSRFToken
< Content-Language: en-us
< Allow: GET, POST, HEAD, OPTIONS
< X-CSRFToken: iFh5q8FUBxoXyyiLcELHoO8W5IDFbAiP
< Set-Cookie: csrftoken=iFh5q8FUBxoXyyiLcELHoO8W5IDFbAiP;httponly;Path=/
< Set-Cookie: sessionid=3ayny2y73i43u6sj9bdyoawhhtr8wbm8;httponly;Path=/
<
+ Connection #0 to host localhost left intact
+ Closing connection #0
{"href": "/api/tool/Transaction/16e1e599-494a-4898-944a-0528915d2f42/",
 "success": true,
 "transaction_id": "16e1e599-494a-4898-944a-0528915d2f42"}
```

Table 12: Response Items to Note

HTTP/1.1 202 ACCEPTED	The request was accepted and the operation is processing.
"transaction_id" : "16e1e599-494a-4898-944a-0528915d2f42"	The operation's transaction id

Export an Instance

Description

In this example, we export a single instance of relation/LineRelation to a formatted .xlsx spreadsheet file.

Example Details

Task

GET an exported instance of a relation/LineRelation.

User

Provider Administrator

Json Input Body

No input body is necessary to export an instance.

Example

Request

```
$ curl -v -H 'Authorization: Basic YWRtaW5AcHJvdjEuY29tOnBhc3N3b3Jk' -o
55ba3e55a6165413b9d1a18d.xlsx

'http://localhost/api/relation/LineRelation/55ba3e55a6165413b9d1a18d/export/?hierarchy=55b9daeca6165413b9d166de&export
_format=xlsx&template_name=line-cft&policy_name=LineMenuFDPProv&schema=true&schema_rules=true'
+ About to connect() to localhost port 80 (#0)
+ Trying 127.0.0.1...
> GET
/api/relation/LineRelation/55ba3e55a6165413b9d1a18d/export/?hierarchy=55b9daeca6165413b9d166de&export
_format=xlsx&template_name=line-cft&policy_name=LineMenuFDPProv&schema=true&schema_rules=true
 HTTP/1.1
> Authorization: Basic YWRtaW5AcHJvdjEuY29tOnBhc3N3b3Jk
> User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu)
              libcurl/7.22.0
              OpenSSL/1.0.1
              zlib/1.2.3.4
              libidn/1.23
              librtmp/2.3
> Host: localhost
> Accept: */*
>
```

Table 13: Request Items to Note

GET /api/relation/LineRelation/55ba3e55a6165413b9d1a18d/export	Requested API operation
55ba3e55a6165413b9d1a18d	PKID of the instance to export
hierarchy=55b9daeca6165413b9d166de	Hierarchy at which the operation is performed
template_name=line-cft	ConfigurationTemplate name
policy_name=LineMenuFDPProv	FieldDisplayPolicy name
export_format=xlsx	Requested format of the exported instance
schema=true	Include model schema details
schema_rules=true	Include model schema rules

Response

```
< HTTP/1.1 200 OK
< Server: nginx/1.1.19
< Date: Thu, 30 Jul 2015 15:45:05 GMT
< Content-Type:application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
< Transfer-Encoding: chunked
< Connection: keep-alive
< X-CSRFToken: tey9Z6fdlDtwEMYczJ2UmSleIolfG4ys
< Content-Disposition:
attachment;filename=relation_LineRelation_exportedsheet_formatted_2015-07-30_17-45-03.xlsx
< Content-Language: en-us
< Vary: Accept, Cookie, Accept-Language, X-CSRFToken
< Allow: GET, HEAD, OPTIONS
< Set-Cookie: fileDownloadToken=downloaded; Path=/
```

```
< Set-Cookie: csrftoken=tey9Z6fdlDtwEMYczJ2UmSleIolfG4ys;httponly;Path=/
< Set-Cookie: sessionid=aioz1ykt36ht47fzthjpljektbg1z1yr;httponly;Path=/
<
 [data not shown]
+ Connection #0 to host localhost left intact
+ Closing connection #0
```

Table 14: Response Items to Note

HTTP/1.1 200 OK	The request was successful
Content-Type:application/vnd.openxmlformats-officedocument.spreadsheetml.sheet	Format of the response body
Content-Disposition: attachment;filename=relation_LineRelation_export.sheet_format_2015-07-30_17-45-03.xlsx	Export body filename

Retrieving a Transaction

Description

In this example, we retrieve details from a posted transaction.

Example Details

Task

GET a tool/Transaction instance.

User

Site Administrator

Json Input Body

No body is necessary when retrieving a transaction instance.

Example

Request

```
curl -v -H 'Authorization: Basic YWRtaW5AbG9jdXMxLmNvbTpwYXNzd29yZA==='
'http://localhost/api/tool/Transaction/01de8720-d627-4e53-8e1b-e1ad66edb7bd/?hierarchy
=55b9dc81a6165413b9d16ab6&nowait=true&format=json'
+ About to connect() to localhost port 80 (#0)
+ Trying 127.0.0.1... connected
> GET
/api/tool/Transaction/01de8720-d627-4e53-8e1b-e1ad66edb7bd/?hierarchy=55b9dc81a6165413b9d16ab6&format=json
 HTTP/1.1
> User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu)
              libcurl/7.22.0
              OpenSSL/1.0.1
              zlib/1.2.3.4
              libidn/1.23
              librtmp/2.3
> Host: localhost
```

```
> Accept: */*
> Authorization: Basic YWRtaW5AbG9jdXMxLmNvbTpwYXNzd29yZA==
```

Table 15: Request Items to Note

GET /api/tool/Transaction/01de8720-d627-4e53-8e1b-e1ad66edb7bd	Requested API operation. Note that 01de8720-d627-4e53-8e1b-e1ad66edb7bd represents the transaction from the Delete example
hierarchy=55b9dc81a6165413b9d16ab6	Hierarchy at which the operation is performed
format=json	Request a json response body

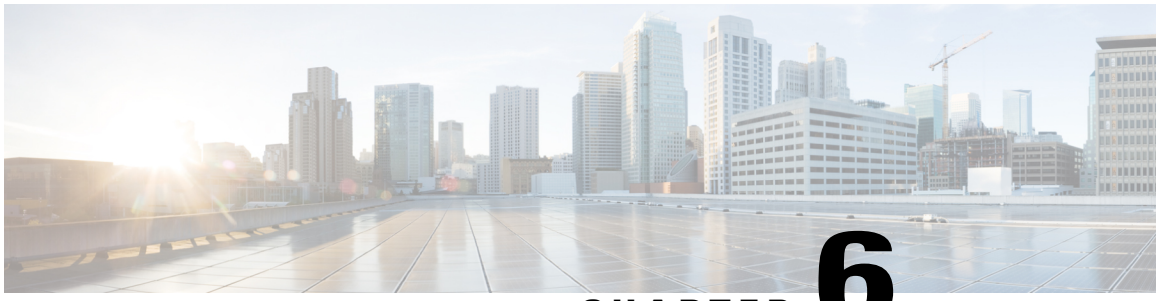
Response

```
< HTTP/1.1 200 OK
< Server: nginx/1.1.19
< Date: Fri, 31 Jul 2015 11:44:27 GMT
< Content-Type: application/json
< Transfer-Encoding: chunked
< Connection: keep-alive
< Vary: Accept, Cookie, Accept-Language, X-CSRFToken
< Content-Language: en-us
< Allow: GET, POST, HEAD, OPTIONS
< X-CSRFToken: pcWhI6fzSbevYskrNVcP34JDZOWH6Nti
< Set-Cookie: csrftoken=pcWhI6fzSbevYskrNVcP34JDZOWH6Nti;httponly;Path=/
< Set-Cookie: sessionid=nyoefznzm1qy9t51qq6v2x0vgkmbvbij;httponly;Path=/
<
{
...
"data": {
"username": "admin",
"status": "Success",
"rolled_back": "No",
"resource": {
"hierarchy": "sys.prov1.cust1.locus1",
"model_type": "relation/LineRelation",
"current_state": "/api/relation/LineRelation/55ba2482a6165413b9d19fb8/ Entity",
"pkid": "55ba2482a6165413b9d19fb8"
},
"log": [
{
"severity": "info",
"format": "text",
"log_id": "55ba24bea6165413b9d19fcd",
"href":
"/api/tool/Transaction/01de8720-d627-4e53-8e1b-e1ad66edb7bd/log/?log_id=55ba24bea6165413b9d19fcd",
"time": "2015-07-30T13:21:02.637000",
"message": "Step 2 - End",
"transaction_id": "01de8720-d627-4e53-8e1b-e1ad66edb7bd"
},
{
"severity": "info",
"format": "text",
"log_id": "55ba24bea6165413b9d19fcc",
"href":
"/api/tool/Transaction/01de8720-d627-4e53-8e1b-e1ad66edb7bd/log/?log_id=55ba24bea6165413b9d19fcc",
"time": "2015-07-30T13:21:02.637000",
"message": "Step 2 - Condition unmet, skipping step. \n[\n
..(SNIPPED)
"transaction_id": "01de8720-d627-4e53-8e1b-e1ad66edb7bd"
},
{
"severity": "info",
"format": "text",
```

```
"log_id": "55ba24bea6165413b9d19fcb",
"href":
"/api/tool/Transaction/01de8720-d627-4e53-8e1b-e1ad66edb7bd/log/?log_id=55ba24bea6165413b9d19fcb",
"time": "2015-07-30T13:21:02.609000",
"message": "Step 2 - Start update data/InternalNumberInventory\nat hierarchy level
..(SNIPPED)
"transaction_id": "01de8720-d627-4e53-8e1b-e1ad66edb7bd"
},
{
"severity": "info",
"format": "text",
"log_id": "55ba24bea6165413b9d19fca",
"href":
"/api/tool/Transaction/01de8720-d627-4e53-8e1b-e1ad66edb7bd/log/?log_id=55ba24bea6165413b9d19fca",
"time": "2015-07-30T13:21:02.605000",
"message": "Step 1 - End",
"transaction_id": "01de8720-d627-4e53-8e1b-e1ad66edb7bd"
},
{
"severity": "info",
"format": "text",
"log_id": "55ba24bda6165413b9d19fc5",
"href":
"/api/tool/Transaction/01de8720-d627-4e53-8e1b-e1ad66edb7bd/log/?log_id=55ba24bda6165413b9d19fc5",
"time": "2015-07-30T13:21:01.280000",
"message": "Step 1 - Start remove device/cucm/Line\nat hierarchy level
sys.prov1.cust1.locus1",
"transaction_id": "01de8720-d627-4e53-8e1b-e1ad66edb7bd"
},
...
}
```


Table 16: Response Items to Note

HTTP/1.1 200 OK	The request was accepted and processed.



CHAPTER 6

API Reference

Using the API Reference

For each resource, a study of the general Reference for Actions in conjunction with the lists of actions for a resource provides the reference for the resource. An examination of the Field Reference for a resource provides payload details.

In addition, a number of conventions are followed some general guidelines should be noted.

- The full URL includes the host name: `https://[hostname]`; for example, `https://172.29.232.62`
- Variables are enclosed in square brackets.
- [hierarchy] is the hierarchy which can be specified as:
  - UUID (Universally Unique Identifier); for example, `1c012432c0deab00da595101` or
  - In dot notation; for example, `ProviderName.CustomerName.LocationName`

  For a list of available hierarchy UUIDs and their dot notations, refer to the data in the response of the call:

```
GET /api/data/HierarchyNode/?format=json
```

- [PKID] is the ID of the resource instance. Refer to the List action reference for the resource.
- [filename] refers to a file.
- where a custom action (with "+" in the URL) is available, the POST method is used to execute the Provisioning workflow with the name following the "+". For more information, consult the custom workflow section of the API Guide.

Relations, Domain models, and Views may have parameters where the choices are constructed from unexposed models (and that may not be available in the API Reference). You can obtain these choices by using the URL specified in the `target` attribute of the schema of parameter.

To illustrate, below is an extract of the schema for a model called relation/SystemUser that contains a parameter SSOUser, which links a user in the system to a user in an SSO identity provider server. This is done by mapping the SSO user name (sso_username) of the user in the SSO server (sso_idp) to a user in the system (data/User).

The schema of relation/SystemUser shows that the choices that are available from SSO Identity provider servers are stored in the model data/SsoIdentityProvider. The list of SSO identity providers could be obtained by using the URL in the `target` attribute of the schema.

```
GET /api/relation/SystemUser/?hierarchy=[hierarchy]
  &format=json
  &schema=1
```

The following is an extract of the schema of relation/SystemUser:

```
...

"SSOUser": {
            "items": {
                 "type": "object",
                 "properties": {
                      "sso_username": {
                           "required": true,
                           "type": "string",
                           "description": "The name identifier that is used for
                             an SSO authenticated user.",
                           "title": "SSO Username"
                      },
                      "sso_idp": {
                           "target": "/api/data/SsoIdentityProvider/choices/
                              ?hierarchy=[hierarchy]
                              &field=entity_id
                              &format=json
                              &auth_token=[auth_token]",
                           "format": "uri",
                           "required": true,
                           "choices": [],
                           "target_attr": "entity_id",
                           "target_model_type": "data/SsoIdentityProvider",
                           "title": "SSO Identity Provider",
                           "type": "string",
                           "description": "The entity id of the SSO Identity Provider."
                      },

 ...
```

API Schema

The schema for a resource is obtained in the request parameter: *?format=json&schema=1*. This way of requesting the schema is only available when requesting an Add form or when viewing a resource.

A specific url is also available for obtaining the schema of a resource:

GET /api/{model_type}/{model_name}/schema/?format=json&hierarchy=pkid
All the schemas are in JSON format.

For more information on the specific API schemas, refer to the *Cisco Unified Communications Domain Manager, API Schema Reference Guide*

http://www.cisco.com/c/dam/en/us/td/docs/voice_ip_comm/hcs/11_5/CUCDM_11_5_1/API_Schema_Guide/Schema11_5_1-11_5_1_98000-1000-Drop83_0-HCM_Standard.docx

Available APIs

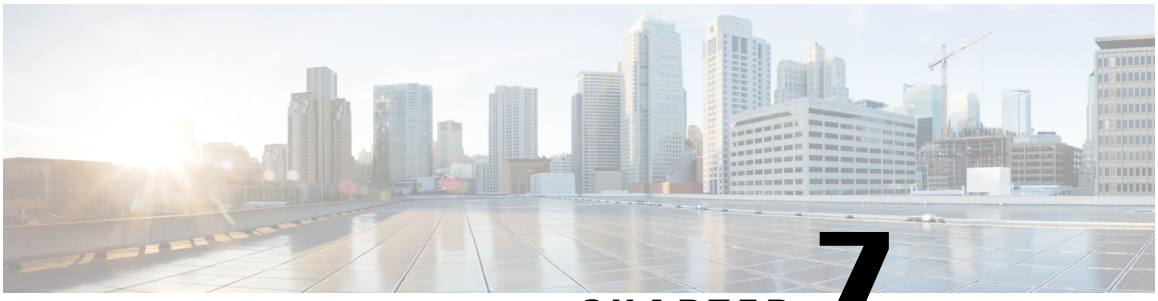
The APIs which are available to the service provider are determined by the role of the user corresponding to the credentials provided in the API. The Cisco Unified Communications Domain Manager 10.x/11.5(x) system has a number of roles that are available for different types of administrators (provider, reseller, customer, site). See the *Cisco Unified Communications Domain Manager, GUI Access by Role* for the complete list of available APIs.

http://www.cisco.com/c/dam/en/us/td/docs/voice_ip_comm/hcs/11_5/CUCDM_11_5_1/API_Schema_Guide/EdgeList11_5_1-11_5_1_98000-1000-Drop83_0-HCM_Standard.xlsx

Note For the most accurate information, gather the schema from the system.

Notifications

Cisco Unified Communications Domain Manager APIs support sending out notifications when instances of certain models are added, deleted or changed. For more details on which models are supported for notifications, and how to configure these notifications please refer to the *Cisco Unified Communications Domain Manager, Release 11.5(1) Maintain and Operate Guide*



CHAPTER 7

API Backwards Compatibility

API Backwards Compatibility

Backwards Compatibility Overview

The Cisco Unified Communications Domain Manager API is versioned and while the latest API and models are in use, the system is backwards compatible with earlier API versions for several models and operations on these. This means that API requests follow the schema and data as specified by the API version.

From 10.6.2 onward, the default web server security protocol has been set to TLSv1.2 and that TSLv1 is no longer supported. SSLv3 can be enabled from the command line with the command **web sslv3 <on/off>**

It is therefore easier to use a network device or custom code written for an earlier Cisco Unified Communications Domain Manager version. An HTTP Request parameter or Header change in developer code may be required for API calls.

Third-party clients that are written to use a particular API version continues to work against newer servers as long as the Cisco Unified Communications Domain Manager server continues to support the API version used by the client. Cisco Unified Communications Domain Manager supports APIs from the current release and two previous releases (N-2).

For device models, an internal version mapping table for API versions and device versions is also maintained. This table creates a fixed mapping of device versions to API version. The mapping is created on a principle of the latest supported device version at the time of the API version release. For example, the version 10.0 UC Application schema is mapped to API version 10.1.2.

No transforms are carried out on Relation models, because transforms are carried out on their component Data- and Device models.

Tools such as Bulk Load and Search, are backwards compatible from an operations and URL structure point of view. However, the data processed or generated by the tools do not support Backwards Compatibility. Data

must be adapted to conform to the latest schema when presenting it to the API. Data obtained from the Tool interfaces are interpreted according to the latest schema.

Note For Update operations: if a model schema was changed so that a new field is added to a schema within a list of objects and data exists in the field for a current instance on the system, then the data in the new field cannot be maintained when updating the instance through a backwards compatible request. Request data replaces all the data within a list of objects. There is currently no workaround for this issue.

API Version

The API Version is represented in major.minor.revision format, for example: 10.1.2, 10.6.1, 10.6.2, or 10.6.3.

The API Version can be seen in the meta section of the resource as follows:

```
"meta": {
"tags": [],
"pkid": "",
"schema_version": "0.1",
"hierarchy": "sys",
"version_tag": "0.2",
"api_version": "10.6.1",
"model_type": "data/DataModel"
},
```

Supported Models and Methods

Supported models are:

- Data Models
- Device Models
- Relations
- Views

The supported HTTP methods on models from the API are:

- GET
- POST
- PUT

Versions Supported

Support for backwards compatibility was introduced in 10.6(1) release.

- The 10.6(2) release is backwards compatible with the 10.6(1) release.
- The 10.6(1) release is backwards compatible with the 10.1(2) release.
- The 10.6(1) release is **not** compatible with the 10.1(1) release.
- The 10.1(2) release is **not** compatible with the 10.1(1) release.

Specifying the API Version

Third-party API clients **must** specify the api_version when integrating with Cisco Unified Communications Domain Manager 10.x/11.5(x). This is required if the third-party client wants to use backwards compatibility, since it ensures that the client continues to receive consistent schemas. Specifying the API version can be done in one of two ways.

In the Query Parameter

```
GET http://localhost/api/data/Countries/?hierarchy=[hierarchy]
&schema=true&format=json&api_version=10.6.1
```

The Query Parameter approach is the recommended method for a client to specify the API Version.

In the Request Header

```
GET http://localhost/api/data/Countries/?hierarchy=[hierarchy]
&schema=true&format=json

Request headers
X-Version: 10.6.1
```

Omitting the API Version

If the API Version is omitted, then the following behavior is expected:

- If the URL contains /v0/, the 10.1(2) API schemas are used.
- If the URL does not contain /v0/, the most recent version API schemas are used.

Backwards Compatibility Exceptions

10.6(3) Exceptions

Prior to 10.6(3), the SyncTo hierarchy of the user in the Provisioning Status was set to the hierarchy where the User is created. Whenever the user is moved up or down the SyncTo was updated to the hierarchy where it is moved. To move subscribers between Sites, the SyncTo hierarchy behavior has been changed in a backwards incompatible way. The SyncTo Hierarchy focuses on updating the syncTo in ProvisionalStatusDAT when the user changes from a “Manual” user to a “Subscriber” user pushed to a Cisco Unified Communications Manager. In 10.6(3) the strategy is changed as follows:

- Manually created Users (User Management not Subscriber Management) have the SyncTo hierarchy set to the hierarchy the user is created.
- Manually created Users (User Management) that are pushed to Cisco Unified Communications Manager (becoming a Subscriber) from Customer or Site has their SyncTo hierarchy updated to that of the Cisco Unified Communications Manager.

Note If the user SyncTo hierarchy is above the Cisco Unified Communications Manager hierarchy when it was created, then the SyncTo does not change and remains at the upper hierarchy.

- Manually created Subscribers (Subscriber Management) have there SyncTo hierarchies set to to that of the Cisco Unified Communications Manager.

- LDAP users are not affected (do not have their SyncTo hierarchy changed).
- If a user is created on an intermediate node or site directly (either manually or through QuickAddSubscriber) and then pushed to Cisco Unified Communications Manager, the user is scoped only to the hierarchy where he or she was created.

10.6(1) Exceptions

The following changes in Cisco Unified Communications Domain Manager may impact API backwards compatibility with the previous version:

- Customer Management – Customer name change is now allowed.
- Unicode – Customer names may not have spaces.
- Localization – Language codes are now four characters.

10.1(2) Exceptions

The 10.1(2) release is not backwards compatible with the 10.1(1) release.

API Version Differences

There are some differences in customer facing models in this release with respect to the previous releases. These differences are captured in the following document.

http://www.cisco.com/c/dam/en/us/td/docs/voice_ip_comm/hcs/11_5/CUCDM_11_5_1/API_Schema_Guide/CUCDM_API_Schema_Diffs_From11_5_1-11_5_0_98000-351-Drop82_0-HCM_Standard.pdf

API Backward Compatibility and Import

The Cisco Unified Communications Domain Manager 10.x/11.5(x) system maintains a Data Model version data store containing all versions that have been imported onto the system.

While there is always a current version of a Data Model in use on the system, a check is carried out during the import of data:

- If the current version is newer than the definition of the imported data, then the imported definition data is flagged internally as `automigration: false` to prevent resources from auto-migrating from a newer version to an older version.
- Importing an older version will not replace the latest definition as the default schema. The older version will only be added to the version store.

The snippet example below shows the `automigration` attribute:

```
{
   "meta": {},
   "resources": [
      {
         "data": {
            "name": "test_mig_dm"
             ...
         },
         "meta": {
            "hierarchy": "sys",
```

```
            "model_type": "data/DataModel",
            "schema_version": "0.1",
            "version_tag": "0.3",
            "automigration": false
        }
    }
]
}
```

This model definition version store makes it possible for version definition imports to be sequence independent, allowing a freshly installed system to construct the version history for backwards compatibility.

HIL API Backward Compatibility

HIL currently supports API versions for v10_1_2 and v10_6_1. You can use the following commands to get the current API version of the target system and to set the API version of the target system:

- `show hcs hil target apiversion` - To get the current API version of target system
- `set hcs hil target apiversion` - To set the API version of target system

Note If there are any active HIL sessions, CLI will prompt the number of active sessions and will check for confirmation before changing the API version