Using the REST Interface

Before You Begin

Authentication and Authorization

The Cisco Hosted Collaboration Mediation Fulfillment REST APIs use HTTP Basic Authentication. The only account that is authorized to use the Cisco Hosted Collaboration Mediation Fulfillment REST APIs is the Cisco HCS admin. You should use the Cisco HCS Admin username and password to authenticate any API calls.

Transport Layer Security

You should always use the HTTPS protocol on port 8443 to communicate with the Cisco Hosted Collaboration Mediation Fulfillment REST APIs. You will need to enable SSL support in your application. Depending on the specific technology used by your application, you may need to manually install the Cisco HCS certificate into a local trust store for your application.

Session Reuse

It is highly recommended that client reuse HTTP sessions when communicating with Cisco Hosted Collaboration Mediation Fulfillment. This will cut down authentication times, and increase throughput of the Cisco Hosted Collaboration Mediation Fulfillment REST API interface.

Create, Read, Update and Delete (CRUD)

Reading a Resource

There are many different options to read a resource using the REST service.

Reading a Single Resource

Every REST resource can be read by sending a GET to the following URL:

```
GET
    https://your-hcs-box.example.com:8443/<SERVICE>/rest/[<VERSION>/]
entity/<YOUR-RESOURCE>/<IDENTIFIER>
```

- "SERVICE" refers to either the Shared Data Repository (sdr) or Fulfillment web service (ff).
- "VERSION" is optional. If it is not specified, the interface defaults to the latest version supported for the respective service.
- "YOUR-RESOURCE" refers to the resource you wish to obtain.
- "IDENTIFIER" is the string (which is currently always a numeric value) that corresponds to the id of the resource you're trying to read.

If there is a matching resource with the provided identifier, then a response message with an HTTP Status Code of 200 (OK) will be returned. The body of the 200 OK will contain the resource in JSON format. If there is no matching resource, then a response with an HTTP status code of 204 (No content) will be returned.

Example

To obtain a Customer resource from SDR webservice with an identifier of "472", you would use the following url:

```
REQUEST
  GET https://your-hcs-box.example.com:8443/sdr/rest/entity/Customer/472

RESPONSE
  HTTP 200 OK
  {
    "type":"Customer",
    "id":"472",
    ...
  }
```

Reading a List of Resources

The interface supports reading two kinds of lists of resources. First, you can list all instances of a particular resource. For example, you can get a list of all Customers. Second, all the resources related to a given resource can be read. For instance, you can get the Blades related to a given Chassis. Both options are covered in detail below.

Reading a list of all instances of a resource

A list of all resources of the same type can be obtained by sending a GET to the following URL:

```
GET
    https://your-hcs-box.example.com:8443/<SERVICE>/rest/[<VERSION>/]
entity/<YOUR-RESOURCE>/paged
```

This returns a paged list of resources with a default page start of 1 and a default page size of 10. You can modify the size of the page you get back by specifying the _pageStart and _pageSize query parameters.

To get a count of the resources you can go to the following URL:

```
GET
    https://your-hcs-box.example.com:8443/<SERVICE>/rest/[<VERSION>/]
entity/<YOUR-RESOURCE>/count
```

To get a JSON schema of the paged response you can go to the following URL:

```
GET
    https://your-hcs-box.example.com:8443/<SERVICE>/rest/[<VERSION>/]
entity/<YOUR-RESOURCE>/paged/schemameta
```

Example

To get a list of Customers, you would use this URL:

```
REQUEST
  GET https://your-hcs-box.example.com:8443/sdr/rest/entity/Customer/paged


RESPONSE
  HTTP 200 OK
  {
    "customers" : [
        {
            "type":"Customer",
            "id":"472,
            ...
        },
        {
           "type":"Customer",
           "id":"705",
           ...
        },
        ...
    ]
    "next" : {
        "href" :
```

```
"/sdr/rest/v10_0/entity/Customer/paged?_pageStart=11&_pageSize=10",
        "rel" : "Customer"
    },
    "previous" : {
        "href" :
"/sdr/rest/v10_0/entity/Customer/paged?_pageStart=1&_pageSize=10",
        "rel" : "Customer"
    },
    "pageStart" : 1,
    "pageSize" : 10
  }
```

Here is an example getting the count of Customers:

```
REQUEST
  GET https://your-hcs-box.example.com:8443/sdr/rest/entity/Customer/count


RESPONSE
  HTTP 200 OK
  5
```

Here is an example showing usage of the _pageSize and _pageStart query parameters to get the second Customer.

```
REQUEST
  GET
https://your-hcs-box.example.com:8443/sdr/rest/entity/Customer/paged?_pageSize=1&_pageStart=2


RESPONSE
  HTTP 200 OK
  {
    "customers" : [
        {
           "type":"Customer",
           "id":"705",
           ...
        }
    ]
    "next" : {
        "href" :
"/sdr/rest/v10_0/entity/Customer/paged?_pageStart=3&_pageSize=1",
        "rel" : "Customer"
    },
    "previous" : {
        "href" :
"/sdr/rest/v10_0/entity/Customer/paged?_pageStart=1&_pageSize=1",
        "rel" : "Customer"
    },
    "pageStart" : 2,
    "pageSize" : 1
  }
```

Reading a List of Related resources

The REST interface also provides a way to obtain a list of all a type of resource which is related to the current resource. The URL for that usually takes this format:

```
GET

https://your-hcs-box.example.com:8443/<SERVICE>/rest/[<VERSION>/]entity/<CURRENT-RESOURCE>
/<CURRENT-RESOURCE-IDENTIFIER>/<LABEL>/<TARGET-RESOURCE>/paged
```

This returns a paged list of resources with a default page start of 1 and a default page size of 10. You can modify the size of the page you get back by specifying the _pageStart and _pageSize query parameters.

To get a count of all resources related to the current resource you can go to the following URL:

```
GET

https://your-hcs-box.example.com:8443/<SERVICE>/rest/[<VERSION>/]entity/<CURRENT-RESOURCE>
/<CURRENT-RESOURCE-IDENTIFIER>/<LABEL>/<TARGET-RESOURCE>/count
```

To get a schema of the paged response you can go to the following URL for the target resource:

```
GET

https://your-hcs-box.example.com:8443/<SERVICE>/rest/[<VERSION>/]entity/<TARGET-RESOURCE>
/paged/schema
```

The relative portion of the URL is also displayed under each link in the "href" field under the current resource.

The paged response should always be a 200 OK message, and should always be returned in the form of a list. If there are no target resources related to the current resource, then an empty array is returned.

Example

To read a list of Blades (target resource) from a Chassis (current resource) with an identifer of "111", this URL will be used:

```
REQUEST
  GET
https://your-hcs-box.example.com:8443/sdr/rest/entity/Chassis/111/Blade/paged


RESPONSE
  HTTP 200 OK
  {
    "blades" : [
        {
            "type":"Blade",
            "id":"223,
            ...
        },
        {
            "type":"Blade",
            "id":"224",
            ...
        },
```

```
        ...
    ]
    "next" : {
        "href" :
"/sdr/rest/v10_0/entity/Chassis/111/Blade/paged?_pageStart=11&_pageSize=10",

        "rel" : "Blade"
    },
    "previous" : {
        "href" :
"/sdr/rest/v10_0/entity/Chassis/111/Blade/paged?_pageStart=1&_pageSize=10",

        "rel" : "Blade"
    },
    "pageStart" : 1,
    "pageSize" : 10
  }
```

HTML View

All GET requests have the option to receive an "HTML" response. An "HTML" repsonse is merely a more human readable response than the standard JSON response and can more easily be viewed in a web browser. Additionally, the "HTML" response also converts all of the "hrefs" of links to executable hyperlinks. Thus, when viewed through a web browser, the html view can be seen as a limited read-only user interface.

In order to convert a standard GET response to the HTML format, merely append the "html" to the end of the path (but before any query parameters).

Example

To convert a GET Customers request expecting a JSON response to a request expecting an "HTML" response, the URL would be modified from:

```
GET https://your-hcs-box.example.com:8443/sdr/rest/entity/Customer/paged
```

To

```
GET
https://your-hcs-box.example.com:8443/sdr/rest/entity/Customer/paged/html
```

Reading a Base Class Resource

When a Base Class Resource is read, by default, the interface will return the representation of just the base class. If you are interested in receiving the representation of the derived class, you can send in the query parameter "_derived" with a value of true. This will cause the system to return a representation of the derived class. The "_derived" query parameter also works while reading lists.

Example

To read the base class OrgUnit you would do the following:

```
GET https://your-hcs-box.example.com:8443/sdr/rest/entity/OrgUnit/1
```

To read the derived class, you would do the following:

```
GET

https://your-hcs-box.example.com:8443/sdr/rest/entity/OrgUnit/1?_derived=true
```

Creating a Resource

To create a resource you simply need to send a HTTP POST request to the base URL of that resource, with the resource in the JSON format in the body. The base URL of a resource is:

```
POST
https://your-hcs-box.example.com:8443/sdr/rest/[<VERSION>/]entity/<YOUR-RESOURCE>
```

If the create is successful, the response will be an HTTP 201 and the body will contain a URL with a relative reference to the created resource.

Creating a Resource using the Base Resource URL

You can create a resource by sending a POST to the base URL of the resource, with the resource in JSON format in the body, and any required relationships explicitly included as hard links.

If you decide to use this method to create resources, you must explicitly list all mandatory relationship references as hard links within the JSON body.

Example

For instance, to create a Customer, which has a required relationship with ServiceProvider

```
REQUEST
  POST https://your-hcs-box.example.com:8443/sdr/rest/entity/Customer
  {
    "shortName": "wall-e",
    "serviceProvider" : {
      "href" : "/sdr/rest/v10_0/entity/ServiceProvider/1"
    }
  }
RESPONSE
  HTTP 201 Created
  /sdr/rest/v10_0/entity/Customer/1
```

Note In the above example body, the ServiceProvider reference is specified in the body, since it is a mandatory relationship.

Creating a Resource using the Associated URL

You can create a resource by sending a POST to an associated URL of the resource, with the resource in JSON format in the body. The associated URL is a URL to the current resource via the required resource:

```
POST https://your-hcs-box.example.com:8443/sdr/rest/[<VERSION>/]
entity/<ASSOCIATED-RESOURCE>/<ASSOCIATED-RESOURCE-KID>/<YOUR-RESOURCE>
```

For instance, a ServiceProvider is the required relationship for creating a Customer, so the associated URL for a Customer is:

```
POST https://your-hcs-box.example.com:8443/sdr/rest/entity/
ServiceProvider/<ServiceProvider-KID>/Customer
```

Example

To create a Customer you could send the following request:

```
REQUEST
POST
https://your-hcs-box.example.com:8443/sdr/rest/entity/ServiceProvider/1/Customer
{
"shortName":"wall-e"
}
RESPONSE
HTTP 201 Created
/sdr/rest/v10_0/entity/Customer/1
```

Updating a Resource

You can update a resource by sending a PUT to the base URL of the resource, with the updated resource in JSON format as the body. This is what the base URL of a resource is:

```
PUT

https://your-hcs-box.example.com:8443/<SERVICE>/rest/[<VERSION>/]entity/
<YOUR-RESOURCE>/<IDENTIFIER>
```

On a successful update, the client will receive an HTTP 201 response and the body will contain a URL with a relative reference to the updated resource.

There is no way to update resources using the associated URL. All required relationships should be included in the JSON body as hard links.

Example

For instance, to update a Customer, the following request would be needed.

```
REQUEST
  PUT https://your-hcs-box.example.com:8443/sdr/rest/entity/Customer/1
  {
    "shortName":"wall-e 2",
    "serviceProvider":{
      "href":"/sdr/rest/v10_0/entity/ServiceProvider/1"
    }
  }
```

```
RESPONSE
  HTTP 201
  /sdr/rest/v10_0/entity/Customer/1
```

Deleting a Resource

You can delete a resource by sending a DELETE to the base URL of the resource with no body. This is what the base URL of a delete request looks like:

```
DELETE

https://your-hcs-box.example.com:8443/<SERVICE>/rest/[<VERSION>/]entity/<YOUR-RESOURCE>
/<IDENTIFIER>
```

On a successful delete, the client will receive an HTTP status code of 204 (No content). If a user attempts to delete a resource that is not present in the database, an HTTP status code of 400 (Bad Request) will be returned.

There is no way to delete resources using the associated URL.

Example

To delete a Customer with the Identifier of "777", the following request would be needed:

```
REQUEST
  DELETE https://your-hcs-box.example.com:8443/sdr/rest/entity/Customer/777


RESPONSE
  204 No Content
```

Associating Resources

There are 3 different ways to associate a resource. These will be discussed in the sections below:

Associating Two Resources Using Direct URL

You can associate two resources directly using the URL. The format for this request takes the form of:

```
https://your-hcs-box.example.com:8443/<SERVICE>/rest/[<VERSION>/]entity/<RESOURCE-1>/
<RESOURCE-1-IDENTIFIER> /[<LABEL>/]<RESOURCE-2>/<RESOURCE-2-IDENTIFIER>
```

The Resource order is actually interchangeable, as long as the resource immediately precedes its corresponding identifier.

On successful association, the client will receive an HTTP 204 No Content response. If the association fails for some reason, an HTTP 400 Bad Request response will be returned.

Example

The example below shows how to associate a CUCMCluster with the identifier of 111 to the Customer with an identifier of 777.

```
REQUEST
  PUT https://your-hcs-box.example.com:8443/sdr/rest/entity/Customer/
777/CUCMCluster/111

RESPONSE
  204 No Content
```

Or

```
REQUEST
  PUT https://your-hcs-box.example.com:8443/sdr/rest/entity/CUCMCluster/
111/Customer/777

RESPONSE
  204 No Content
```

Associating Two Resources on a Create

Mandatory relationships are always established on a create. This can be done by creating a resource with the associated URL or creating a resource with the base URL and specifying the parent in JSON format in the body.

Optional relationships can also be established on creates as well under both the associated URL and base URL. These are always specified in JSON format in the body.

On a successful create, the client will receive an HTTP 201 Created response and the body will contain a URL with a relative reference to the created resource. A failed association of two resources within a create will result in an HTTP 400 Bad Request response.

Example

The below example shows how to create a Customer and associate it with a pre-existing DataCenter with the ID of 555:

```
REQUEST
  POST https:/your-hcs-box.example.com:8443/sdr/rest/entity/Customer
  {
    "shortName":"wall-e 2",
    "serviceProvider":{
      "href":"/sdr/rest/v10_0/entity/ServiceProvider/1"
    }
    "dataCenter":{
      "href":"/sdr/rest/v10_0/entity/DataCenter/555"
    }
  }

RESPONSE
  HTTP 201 Created
  /sdr/rest/v10_0/entity/Customer/1
```

Associating Two Resources on an Update

Optional relationships can be either associated or disassociated on an update request. If an existing relationship is not specified on an update, then the target resource will be disassociated from the updated resource. For this reason, all existing relationships must be specified in the body of an update request, unless the disassociation of the two resources is desired.

If there is already an existing relationship between the base resource and target resource, and you wish to remove that relationship and establish a new relationship, then the target href can be changed to point to the new resource. It is not necessary to do a separate disassociate and associate, as this can all be handled in a single update request.

Mandatory relationships should always be specified in the body of an update request. These relationships cannot be removed, but if the mandatory relationships are not identified in the schema as defining, they can be modified.

On a successful update, the client will receive an HTTP 201 response and the body will contain a URL with a relative reference to the updated resource. A failed association of two resources within an update request will result in an HTTP 400 Bad Request response.

Example

The below example shows how to update a Customer and associate it with a pre-existing DataCenter with the ID of 555:

```
REQUEST
  PUT https://your-hcs-box.example.com:8443/sdr/rest/entity/Customer/1
  {
    "shortName":"wall-e 2",
    "serviceProvider":{
      "href":"/sdr/rest/v10_0/entity/ServiceProvider/1"
    }
    "dataCenter":{
      "href":"/sdr/rest/v10_0/entity/DataCenter/555"
    }
  }

RESPONSE
  HTTP 201
  /sdr/rest/v10_0/entity/Customer/1
```

Associating Lists of Resources

In most instances, only one target resource can be associated to the current resource for a given relationship. However, there are some relationships where it is allowed for more than one association to be established. When establishing these associations on a PUT or POST request, the hard links must be listed within an array. The order of the hard links does not matter, but to preserve the relationships on PUT requests, the complete existing array of hard links must always be passed in as an argument. If it is desired for any additional resources to be associated, then these new resources should be appended to the current hard link array.

For these relationships, the behavior for association via direct URL is consistent with the behavior for relationships where only one target resource can be associated at any given time.

Example

The below example shows how to associate 2 existing Clusters to a Customer on a create.

```
REQUEST
  POST https://your-hcs-box.example.com:8443/sdr/rest/entity/Customer
  {
    "shortName":"wall-e 2",
    "serviceProvider":{
      "href":"/sdr/rest/v10_0/entity/ServiceProvider/1"
    }
    "clusters":[
      {
        "href":"/sdr/rest/v10_0/entity/Cluster/1"
      },
      {
        "href":"/sdr/rest/v10_0/entity/Cluster/2"
      }
    ]
  }

RESPONSE
  HTTP 201 Created
  /sdr/rest/v10_0/entity/Customer/1
```

Disassociating Resources

There are 2 different ways to disassociate a resource. These will be discussed in the sections below.

Disassociating Two Resources Using Direct URL

You can disassociate two resources directly using the URL. The format for this request takes the form of:

```
https://your-hcs-box.example.com:8443/<SERVICE>/rest/[<VERSION>/]entity/<RESOURCE-1>/
<RESOURCE-1-IDENTIFIER> /[<LABEL>/]<RESOURCE-2>/<RESOURCE-2-IDENTIFIER>
```

The Resource order is actually interchangeable, as long as the resource immediately precedes its corresponding identifier.

On a successful disassociation, the client will receive an HTTP 204 No Content response. If the disassociation fails for some reason, an HTTP 400 Bad Request response is returned.

Example

The below examples below shows how to disassociate a pre-existing relationship between a Customer with an ID of 777 and a CUCMCluster with an ID of 111:

```
REQUEST
  DELETE
https://your-hcs-box.example.com:8443/sdr/rest/entity/Customer/777/CUCMCluster/111
```

```
RESPONSE
  204 No Content
```

or

```
REQUEST
  DELETE
https://your-hcs-box.example.com:8443/sdr/rest/entity/CUCMCluster/111/Customer/777


RESPONSE
  204 No Content
```

Disassociating Two Resources on an Update

Optional relationships can be either associated or disassociated on an update request. If an existing relationship is not specified on an update, then the target resource will be disassociated from the updated resource. For this reason, all existing relationships must be specified in the body of an update request, unless the disassociation of the two resources is desired.

Mandatory relationships should always be specified in the body of an update request. These relationships cannot be disassociated, only modified.

A disassociation can be specified by either setting the reference to null or by leaving out the reference entirely.

On a successful update, the client will receive an HTTP 201 response and the body will contain a URL with a relative reference to the updated resource. If the update request fails due to a failed association, an HTTP 400 Bad Request response will be returned.

Example

The below examples show how to update a Customer and disassociate it from the pre-existing relationship with DataCenter with the ID of 555:

```
REQUEST
  PUT https://your-hcs-box.example.com:8443/sdr/rest/entity/Customer/1
  {
    "shortName":"wall-e 2",
    "serviceProvider":{
      "href":"/sdr/rest/v10_0/entity/ServiceProvider/1"
    }
    "dataCenter": null
  }

RESPONSE
  HTTP 201
  /sdr/rest/v10_0/entity/Customer/1
```

or

```
REQUEST
  PUT https://your-hcs-box.example.com:8443/sdr/rest/entity/Customer/1
  {
    "shortName":"wall-e 2",
    "serviceProvider":{
```

```
      "href":"/sdr/rest/v10_0/entity/ServiceProvider/1"
    }
 }

RESPONSE
  HTTP 201
  /sdr/rest/v10_0/entity/Customer/1
```

Disassociating List of Resources

In most instances, only one target resource can be associated (and thus disassociated) to the current resource for a given relationship. However, there are some relationships where it is allowed for more than one association to be established. When establishing these associations on a PUT or POST request, the hard links must be listed within an array. The order of the hard links does not matter, but to preserve the relationships on PUT requests, the complete existing array of hard links must always be passed in as an argument. If it is desired for any existing target resource to be disassociated from the current resource, then the target resource should be left out of the hard link array. If it is desired to disassociate all of the target resources from the current resources, then the hard link array can be set to null or an empty array.

For these relationships, the behavior for disassociation via direct URL is consistent with the behavior for relationships where only one target resource can be disassociated at any given time.

Example

Assuming that the Customer has a pre-existing relationship between two Clusters with an ID of "1" and an ID of "2". the example below shows how to disassociate 1 existing Cluster with an ID of "1" from a Customer while preserving the relationship between the Customer and the Cluster with the ID of "2" on an update.

```
REQUEST
  PUT https://your-hcs-box.example.com:8443/sdr/rest/entity/Customer/1
  {
    "shortName":"wall-e 2",
    "serviceProvider":{
      "href":"/sdr/rest/v10_0/entity/ServiceProvider/1"
    }
    "clusters":[
      {
        "href":"/sdr/rest/v10_0/entity/Cluster/2"
      }
    ]
  }

RESPONSE
  HTTP 201 Created
  /sdr/rest/v10_0/entity/Customer/1
```

Versioning and Mediation

The REST URLs do not require a version to be specified. If you choose not to specify a version you will be redirected to the latest version. It is highly recommended that you use a version so that you are explicitly indicating which version you are dealing with.

For the 10.6(1) release, the SDR REST Service now supports a "v10_6" URL. This URL provides the exact same functionality as the "v10_0" URL, so messages will experience identical behavior whether they are sent using the "v10_0" or "v10_6" URL. The Fulfillment service only supports the "v10_0" URL for this release."

The interface also supports receiving the version in the accept header of the request. If you choose to leave the version out of the URL and specify it in the Accept header, that will also work. Here is an example of the version in the Accept header :

```
Accept: application/json;version=v10_0
```

Schema

All of the necessary information about a particular resource, including field information and its relationship(s) to other resources, can be obtained from the JSON schema.

Reading the Schema of a Resource

The schema is published in two ways. The schema is published in the XML XSD format as part of the WADL and also in a JSON format accessible through the a custom URL. It is recommended that the JSON schema is used since this schema contains specialized information for the Cisco Hosted Collaboration Mediation Fulfillment REST Services and because requests and response utilize JSON format.

The schema is published in XML format for the REST service here:

```
https://your-hcs-box.example.com:8443/<SERVICE>/rest/
application.wadl/xsd0.xsd
```

The schema is published in JSON format for the REST service here:

```
https://your-hcs-box.example.com:8443/<SERVICE>/rest/
[<VERSION>/]entity/
<YOUR-RESOURCE>/schemameta
```

Note For this document, unless explicitly specified, any reference made to the schema should be assumed to be the JSON schema.

JSON Schema Organization

There are two main types of information conveyed in each schema resource: fields and links.

Fields

A field will usually have the type of a string, integer, enum, or boolean, and occasionally will take the format of a list of strings. Every field member in the JSON schema has a "type" name-value pair. This "type" name-value pair lists the expected format of the field.

Below is an example of some fields from the CUCM entity. The JSON schema member "description" has a "type" name-value pair with value of string. This indicates that the field "description" is of type string. Similarly the field "isConferenceServer" is of type boolean.

```
"description" : {
      "type" : [ "string", "null" ]
    },
    "isConferenceServer" : {
      "type" : [ "boolean", "null" ]
    },
```

Enum Fields

If a field is of type string and the JSON schema member that defines the field contains an "enum" name-value pair, it indicates that the field is an enum. The value of the "enum" name-value pair will be an array of all the possible enum values for that field.

```
"typeCucmCupSharingMode" : {
        "type" : [ "string", "null" ],
        "enum" : [ "DEDICATED", "SHARED" ]
    },
```

Nullable vs Non Nullable Fields

In the JSON schema, if the value of the "type" name-value pair is an array with a "null" element, then that indicates that "null" is an accepted value for that field.

```
...
    "shortName" : {
      "type" : "string",
      "required" : true
    },
    "extendedName" : {
      "type" : [ "string", "null" ]
    },
    ...
```

Readonly Fields

If the JSON schema member that defines the field contains a "readonly" name-value pair with a value of true, that indicates that the field is readonly and can not be set via the REST interface. The default value for the "readonly" name-value pair is false. If the "readonly" name-value pair is missing it indicates that the field is not readonly.

```
"cdmUpdateTimestamp" : {
      "type" : [ "string", "null" ],
      "readonly" : true
    },
```

Required Fields

If the JSON schema member that defines the field contains a "required" name-value pair with a value of true, that indicates that the field is required and can not be set to null via the REST interface. The default value for

the "required" name-value pair is false. If the "required" name-value pair is missing it indicates that the field is not required.

```
"distinguishedName" : {
      "type" : "string",
      "required" : true
    },
```

Password Fields

If the JSON schema member that defines the field contains a “is_password" name-value pair with a value of true ,that indicates that the field is a password. The default value for the “is_password" name-value pair is false. If the “is_password" name-value pair is missing it indicates that the field is not a password.

In the 10.0(1) release, passwords were indicated with a field entitled “password”. This has been changed to “is_password” for 10.1(2), so this may require some changes if updating from 10.0(1).

```
"password_CommunityString" : {
         "type" : [ "string", "null" ],
         “is_password" : true
},
```

Default Values of Fields

If the JSON schema member that defines the field contains a "default" name-value pair with some value, that indicates that the field has a default value. This means that the field cannot be set to null from the HCM_F SDR REST interface. The "default" name-value pair is not a boolean which indicates whether a field has a default value, but is the actual representation of the default value itself. The "default" value can be a boolean, integer or enum. If the "default" name-value pair is missing, it indicates that the field does not have a default value.

```
"isAutoSyncEnabled" : {
      "type" : [ "boolean", "null" ],
      "default" : false
    },
    "syncInterval" : {
      "type" : [ "integer", "null" ],
      "default" : 15
    },
```

Searchable fields

If the JSON schema member that defines the field contains a "searchable" name-value pair with a value, that indicates that you can get a list of the entity in question where that field is matched. If the value of the "searchable" name-value pair contains the name of the entity in question that indicates you can match on that field within the entity. This is called a global scope search. If the value of the "searchable" name-value pair contains the name of the entity in question's parent that indicates you can match on that field within the entity for a specific parent. This is called a parental scope search.

The default value for the "searchable" name-value pair is none. If the "searchable" name-value pair is missing it indicates that the field is not searchable.

```
"uUID" : {
      "type" : [ "string", "null" ],
      "searchable" : "Blade"
    },
```

OrderBy Fields

If the JSON schema member that defines the field contains a "orderby" name-value pair with a value of true, that indicates that you can get a list of the entity in question ordered by that field. You can do this by using the _orderBy query parameter in your REST query.

The default value for the "orderby" name-value pair is false. If the "orderby" name-value pair is missing it indicates that you can not get a list of the entity in question ordered by that field.

```
"initiatedTimestamp" : {
      "type" : [ "string", "null" ],
      "orderby" : true
    },
```

Embedded Resources

In some instances, it is possible for a resource to contain multiple levels of embedded resources. In the schema, these resources show up as an array of objects, and exhibit the same behavior as a standalone resource. However, an embedded resource cannot exist independent of its target resource.

```
"networkAddresses" : {
      "type" : [ "array", "null" ],
      "items" : {
        "type" : "object",
        "properties" : {
          "description" : {
            "type" : [ "string", "null" ]
          },
          "domain" : {
            "type" : [ "string", "null" ]
          },
          "iPAddrV4" : {
            "type" : [ "string", "null" ]
          },
          "iPAddrV6" : {
            "type" : [ "string", "null" ]
          },
          "hostShortNameOnly" : {
            "type" : [ "string", "null" ]
          },
          "typeAddressSpace" : {
            "type" : "string",
            "required" : true,
            "enum" : [ "APPLICATION_SPACE", "SERVICE_PROVIDER_SPACE",
"CUSTOMER_SPACE" ]
          },
          "isSRVAddress" : {
            "type" : "boolean",
            "required" : true
          },
          "isDeletable" : {
```

```
            "type" : [ "boolean", "null" ],
            "readonly" : true
          },
          "isModifiable" : {
            "type" : [ "boolean", "null" ],
            "readonly" : true
          },
          "cdmUpdateTimestamp" : {
            "type" : [ "string", "null" ],
            "readonly" : true
          },
          "syncTimestamp" : {
            "type" : [ "string", "null" ],
            "readonly" : true
          },
          "id" : {
            "type" : [ "string", "null" ]
          }
        }
      }
    },
```

Links

A link is a reference to one or more other resources. In the schema it is identified as any member that has "rel", "href" and "relation" name-value pairs.

There are two kinds of links: Hard Links and Soft Links.

Soft Links

Soft Links are readonly, you can not update the relationship between two resources using a soft link. A soft link is identified by the presence of "readonly" name-value pair with a value of true.

Hard Links

Any link that does not have a "readonly" name-value pair with a value of true is considered to be a hard link. Hard Links are writable, and will update the relationship between two resources. Hard links can be further characterized as required, defining or optional.

Required Hard Links

Any links that have a "required" name-value pair with a value of true are called Required Hard Links. If a link is a required hard link then is MUST be specified on creates and updates.

Defining Hard Links

Any links that have a "defining" name-value pair with a value of true are called Defining Hard Links. All defining hard links are also required. If a link is a defining hard link, then it must be specified on creates and can not be changed on updates.

A link will either reference the current resource or the relationship between the current resource and a target resource.

Business Keys

There exists a need to identify referenced resources in a human readable fashion for displaying in lists. For this purpose it is important to be able to identify which fields of a resource make up the human readable unique identifier for that resource. We call this human readable unique identifier the business key. We identify the business key fields in the schema in a businessKeyFields name-value pair. The value of the businessKeyFields name-value pair is an array of strings. Each string represents a field within the resource.

The businessKeyFields name-value pair is represented in the schema in the "self" soft link of each resource. It is also present in all hard links. Here is an example of the businessKeyFields:

```
"businessKeyFields": ["name","id"],
```

This feature is particularly important for auto generated user interfaces .

Business Keys for base classes

A base class does not have its own set of fields that create a business key. Instead, it just lists a "basekey" as its business key. Base classes will just wrap up all the business key fields of the derived class into a single string called the "basekey". Note that users can never create a base class, they can only create a derived class. Here is an example of the business key of a base class (OrgUnit):

```
"businessKeyFields": ["baseKey","id"]
```

Business Keys for hierarchically nested nodes

When dealing with hierarchies, a nested child node may only be unique within a parent node. It would not be possible to get a human readable unique identifier of the child node using just the fields within the child node. To get a human readable unique identifier for that child node, aka business key, we would need to include the human readable unique identifier of the parent node as well. When we reference a parents field in the businessKeyFields, they are in the format "bk" + <Parent Resource Name> + "_" + <Parent Field Name>. Here is an example of a business key for a nested child node (Blade) which references fields from multiple levels of hierarchies:

```
"businessKeyFields":
    ["bkUCSManager_name","bkChassis_chassisID","slotID","id"]
```

Link Sub Schema

Each link has a properties subschema, and both hard links and soft links will always have the rel, href, and relation fields. Hard links will also contain two additional fields: businessKey and businessKeyFields.

rel

This is always a read-only field. It always returns the constant value specified in the relation field of the link for the schema. It is useful to determine the type of relationship between the current resource and the target resource. The rel will always take the form "relationship_type/target_resource".

The relationship type indicates the relationship between the current resource and the target resource from the perspective of the current resource. The relationship type currently can be one of three values:

Value	Description
zeroToOne	The base resource can have a relationship with zero or one of the target resource. (Optional)
exactlyOne	The base resource must always have a relationship with one (and only one) of the target resource. (Mandatory)
zeroToMany	The base resource can have a relationship with zero or many of the target resources. (Optional)

Note The "target_resource" should always directly correspond to an existing resource within the same service.

href

The href is always present in a link. It is readonly for soft links. This field must be set appropriately to identify the target resource. This is the URL you would need to access the resource specified by the link.

businessKey

This is only present for hard links. This is the actual list of fields that make up the business key fields. When associating or disassociating relationships, either the businessKey or the href can be used to determine the target entity. It is recommended that you use the href.

businessKeyFields

This is only present for hard links. This is a schema only attribute. It identifies the fields which make up a human readable unique identifier for a resource. This is useful when displaying a list of resources in a UI drop down or a UI table.

relation

This is the schema only attribute. It returns the same value that is returned for the "rel" attribute.