Cisco Multi-Site REST API Configuration Guide, Release 3.3(x)

First Published: 2021-04-01

Last Modified: 2021-05-07

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

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED “AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

All printed copies and duplicate soft copies of this document are considered uncontrolled. See the current online version for the latest version.

Cisco has more than 200 offices worldwide. Addresses and phone numbers are listed on the Cisco website at www.cisco.com/go/offices.

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: https://www.cisco.com/c/en/us/about/legal/trademarks.html. Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1721R)

© 2020 Cisco Systems, Inc. All rights reserved.

CONTENTS

CHAPTER 1 **New and Changed Information** 1

New and Changed Information 1

CHAPTER 2 **Using the REST API** 3

REST API Overview 3

REST API User Roles and Authorization 4

REST API Requests 4

GET Requests 4

POST and PUT Requests 5

DELETE Requests 6

PATCH Requests 6

Concurrent Configuration Updates 14

REST API Reference (OpenAPI/Swagger) 16

Accessing OpenAPI Reference 17

Using OpenAPI for Authentication 17

Using OpenAPI to Manage Users 19

Using In-Browser DevTools to View REST API Calls 20

Using In-Browser DevTools to Work with REST API Calls 21

CHAPTER 3 **REST API Data Structures** 25

REST API Data Structures Overview 25

Schema 25

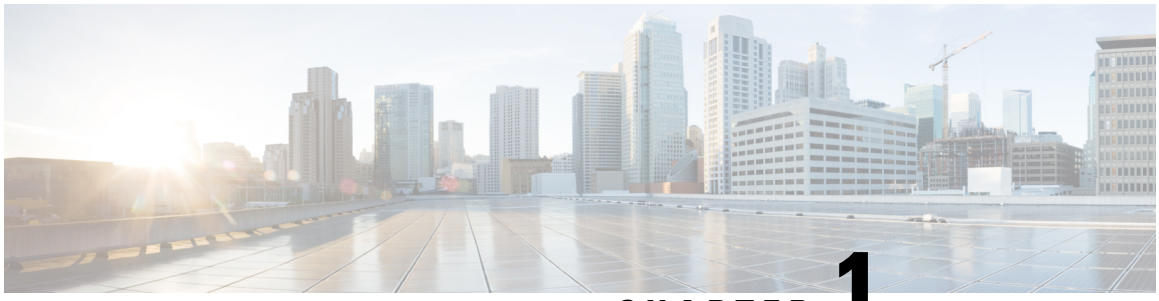
Template 26

anp 26

epg 27

bd 27

contract 28
filter 28
site 29
Reference Fields 29
Local References 30
References from Another Schema 30



CHAPTER 1

New and Changed Information

This chapter contains the following sections:

- New and Changed Information, on page 1

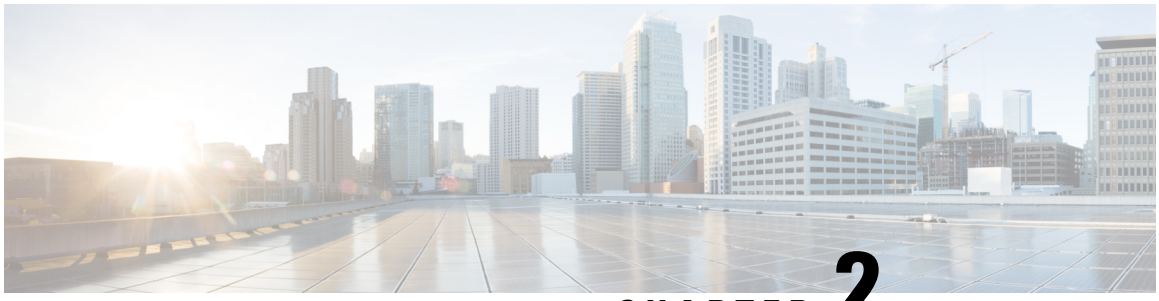
New and Changed Information

The following table provides an overview of the significant changes to the organization and features in this guide from the release the guide was first published to the current release. The table does not provide an exhaustive list of all changes made to the guide.

Table 1: Latest Updates

Release	New Feature or Update	Where Documented
3.3(1)	First release of this document.	--

New and Changed Information



CHAPTER 2

Using the REST API

This chapter contains the following sections:

- REST API Overview, on page 3
- REST API User Roles and Authorization, on page 4
- REST API Requests, on page 4
- Concurrent Configuration Updates, on page 14
- REST API Reference (OpenAPI/Swagger), on page 16
- Using In-Browser DevTools to View REST API Calls, on page 20
- Using In-Browser DevTools to Work with REST API Calls, on page 21

REST API Overview

Multi-Site REST API is a set of programming interfaces that uses Representational State Transfer (REST) architecture. The API contains resources represented by Uniform Resource Identifiers (URIs), which allow to unambiguously identify each resource. Each URI contains a protocol used to exchange the messages and the resource location string. For example, the `https://<mso-ip>/api/v1/schemas` URI specifies that the HTTPS protocol is to be used and the schemas resource path relative to the Multi-Site Orchestrator address.

URIs can refer to a single object or a collection of objects. For example, `http://<mso-ip>/api/v1/schemas` represents all the schemas that exist in the fabric, whereas `http://<mso-ip>/api/v1/schemas/{id}` specifies a schema with a specific ID.

When you want to retrieve information or make changes to the fabric, you use API calls to exchange messages between your client and an URI. The messages must be in JavaScript Object Notation (JSON) format, but you can use any programming language to generate and send them.

The following standard REST methods are supported by the Multi-Site REST API:

- `GET`
- `POST`
- `PUT`
- `DELETE`
- `PATCH`

The PUT and PATCH methods are idempotent, which means that there is no additional effect if they are called more than once with the same input parameters. The GET method is nullipotent, meaning that it can be called zero or more times without making any changes, in other words it is a read-only operation.

REST API User Roles and Authorization

The Multi-Site Orchestrator API supports multiple users, each with their own user-specific authorization and set of privileges based on their role. A user can be associated with specific roles for access based on their function and REST endpoints can be restricted based on the user's role. The admin user has unrestricted access. For more information on creating and manager users and their roles, see the *Multi-Site Configuration Guide*.

REST API Requests

The Multi-Site REST API supports a number of standard API calls, which allow you to retrieve information about or make changes to your fabric. A typical REST API operation consists of three elements:

- **Request URL**: The address of the resource to which you make the API call.
- **Request message**: The JSON-formatted payload that contains the new data you want to add or update. For read-only operation (GET) the request message is empty.
- **Request response**: The JSON-formatted response that contains the requested information.

The following sections provide an overview of each call as well as an example JSON payload.

GET Requests

The GET request is a read-only operation that allows you to retrieve information about one or more objects in the fabric. The following example uses a GET request to obtain information about Multi-Site Orchestrator users, such as their names, contact information, status, and privileges.

Request URL:

```
https://<mso-ip>/api/v1/users
```

Request payload:

```
EMPTY
```

Request response:

```
{
    "users": [{
            "id": "5b6380972d0000f85ddea55e",
            "username": "User01",
            "password": "******",
            "firstName": "fName01",
            "lastName": "lName01",
            "emailAddress": "User01@cisco.com",
            "phoneNumber": "098-765-4321",
            "accountStatus": "active",
            "needsPasswordUpdate": true,
            "roles": [{
                    "roleId": "0000ffff0000000000000031"
                }, {
```

```
                    "roleId": "0000ffff0000000000000033"
                }, {
                    "roleId": "0000ffff0000000000000035"
                }
            ],
            "domainId": "0000ffff0000000000000090"
        }, {
            "id": "5bb7aabc2c0000f34c7b89f7",
            "username": "User02",
            "password": "******",
            "firstName": "fName02",
            "lastName": "lName02",
            "emailAddress": "User02@cisco.com",
            "phoneNumber": "123-456-7890",
            "accountStatus": "active",
            "needsPasswordUpdate": true,
            "roles": [{
                    "roleId": "0000ffff0000000000000031"
                }, {
                    "roleId": "0000ffff0000000000000032"
                }
            ],
            "domainId": "0000ffff0000000000000090"
        }
    ]
}
```

POST and PUT Requests

The `POST` and `PUT` requests are write operations that allow you to create a new or update an existing object. Keep in mind, that if you are updating an existing object, you must provide the object in its entirety. Any previously existing fields that are missing from the `POST` payload, will be replaced by an empty string or `null`.

A `PUT` request is idempotent, which is the main difference between the `PUT` and `POST` requests.

The following example uses a `POST` request to create a new user. The request response contains the newly created object.

Request URL:

```
https://<mso-ip>/api/v1/users
```

Request payload:

```
{
    "id": "",
    "username": "<username>",
    "password": "<user-pass>",
    "confirmPassword": "<user-pass>",
    "firstName": "<user-first-name>",
    "lastName": "<user-last-name>",
    "emailAddress": "<user-email>",
    "phoneNumber": "<user-phone>",
    "accountStatus": "active",
    "needsPasswordUpdate": true,
    "roles": [{
            "roleId": "0000ffff0000000000000031"
        }
    ]
}
```

Request response:

```
{
    "id": "5c40b8832b0000744a77ec1a",
    "username": "<username>",
    "password": "******",
    "firstName": "<user-first-name>",
    "lastName": "<user-last-name>",
    "emailAddress": "<user-email>",
    "phoneNumber": "<user-phone>",
    "accountStatus": "active",
    "needsPasswordUpdate": true,
    "roles": [{
            "roleId": "0000ffff0000000000000031"
        }
    ],
    "domainId": "0000ffff0000000000000090"
}
```

DELETE Requests

The `DELETE` request is a write operation that allows you to delete an existing object. A `DELETE` request does not require a payload and does not return a response.

The following example uses a `DELETE` request to delete the user we created in the `POST` example.

Request URL:

```
https://<mso-ip>/api/v1/users/5c40b8832b0000744a77ec1a
```

Request payload:

```
EMPTY
```

Request response:

```
EMPTY
```

PATCH Requests

The `PATCH` request is a write operation that allow you to update an existing object. The main difference between `PATCH` and `POST` or `PUT` requests is that you can provide only the fields that contain new data rather than the entire object being updated. A PATCH request is neither safe nor idempotent, because a PATCH operation cannot ensure the entire resource has been updated. Also, unlike other API requests, a PATCH request contains instructions on how to modify a resource, rather than a version of the resource itself.

Note The current release of Multi-Site Orchestrator supports `PATCH` requests only for some objects. Check the Swagger API reference to see if an object supports PATCH requests.

When creating PATCH requests, the payload must contain the following:

- `op`: the operation to be performed by the request. Currently supported operations are `add`, `remove`, or `replace`.
- `path`: the path to the resource that you are updating. The value contains the URI of the resource and the position inside that resource where the information will be added.

  For `add` operations, the position value can be any positive integer or a dash (`-`) to specify the end of the schema. For example, `/templates/<template-name>/vrfs/-` would indicate adding a new VRF at the

end of the current list, whereas `/templates/<template-name>/vrfs/2` would add the VRF at the second position.

For `replace` operations, the position value can also be a the name of the object to replace in addition to the index. For example, `/templates/<template-name>/anps/AP1` would replace the application profile named `AP1`

- `value`: the new or updated value or object. For example, `{"vrfname" : "vrf1"}` would specify a new VRF with the name **vrf1**.

  You do not need to provide the `value` field for `remove` operations.

The following two examples illustrate how to uses the `PATCH` requests to add and remove a VRF in a template in an existing schema. The request response contains the entire object that was updated.

Guidelines and Limitations

When using PATCH API, the following guidelines apply:

- You cannot use the PATCH API to change a template name, because multiple references in the fabric must be updated. Use PUT request instead.

  For site local schema objects, refer to the Site ID and Template name combination as `<site-id>-<template-name>`, for example `/sites/5b7d29c2a7fa00a7fae9bbf3-SampleTemplate/epgs`

- You can reference objects using index or name.

  Prior to Release 3.3(1), many objects could only be referenced by their index in the schema.

  For example, if you had multiple subnets and you wanted to update the third one, you would have to use the following code with index `2`:

```
[
    {
        "op": "replace",
        "path": "/templates/Template1/externalEpgs/epgName/subnets/2/ip",
        "value": "2.2.2.2/24"
    }
]
```

  Starting with Release 3.3(1), any objects which had to be referenced by an index can now also be referenced using their name or unique identifier. For some objects, which don't have a name or ID, you will need to continue to use an index.

Note You must continue using index for `domainAssociation` objects, as the `dn` fields can contain multiple slash (`/`) characters and cannot be easily identified within the full path string.

References by index remain valid for compatibility with existing scripts

  Similar to the previous examples, you can now update any subnet by simply providing its IP address and mask:

```
[
    {
        "op": "replace",
        "path": "/templates/Template1/externalEpgs/epgName/subnets/1.1.1.1/24/ip",
        "value": "2.2.2.2/24"
```

```
        }
    ]
```

Add an Object Using PATCH Request

The following example uses a `PATCH` request to add a VRF to a template in an existing schema. The request response contains the entire object that was updated.

Original schema:

```
{
  "id": "5c4b55db1a00003422f2215e",
  "displayName": "SampleSchema",
  "templates": [
    {
      "name": "Template1",
      "displayName": "Template1",
      "tenantId": "0000ffff0000000000000010",
      "anps": [],
      "vrfs": [],
      "bds": [],
      "contracts": [],
      "filters": [],
      "externalEpgs": [],
      "serviceGraphs": [],
      "intersiteL3outs": []
    }
  ]
}
```

Request URL:

```
PATCH https://<mso-ip>/api/v1/schemas/5c4b55db1a00003422f2215e
```

Request payload:

```
[{
    "op": "add",
    "path": "/templates/Template1/vrfs/-",
    "value": {
        "displayName" : "vrf1",
        "name" : "vrf1" }
}]
```

Request response:

```
{
  "id": "5c4b55db1a00003422f2215e",
  "displayName": "SampleSchema",
  "templates": [
    {
      "name": "Template1",
      "displayName": "T1",
      "tenantId": "0000ffff0000000000000010",
      "anps": [],
      "vrfs": [
        {
          "name": "vrf1",
          "displayName": "vrf1",
          "vrfRef": "/schemas/5c4b55db1a00003422f2215e/templates/Template1/vrfs/vrf1",
          "vzAnyProviderContracts": [],
          "vzAnyConsumerContracts": []
        }
      ],
      "bds": [],
```

```
      "contracts": [],
      "filters": [],
      "externalEpgs": [],
      "serviceGraphs": [],
      "intersiteL3outs": []
    }
  ]
}
```

Remove an Object Using PATCH Request (VRF Example)

The following example uses a `PATCH` request to remove a VRF from a template in an existing schema. The request response contains the entire object that was updated.

Original schema:

```
{
  "id": "5c4b55db1a00003422f2215e",
  "displayName": "SampleSchema",
  "templates": [
    {
      "name": "Template1",
      "displayName": "T1",
      "tenantId": "0000ffff0000000000000010",
      "anps": [],
      "vrfs": [
        {
          "name": "vrf1",
          "displayName": "vrf1",
          "vrfRef": "/schemas/5c4b55db1a00003422f2215e/templates/Template1/vrfs/vrf1",
          "vzAnyProviderContracts": [],
          "vzAnyConsumerContracts": []
        }
      ],
      "bds": [],
      "contracts": [],
      "filters": [],
      "externalEpgs": [],
      "serviceGraphs": [],
      "intersiteL3outs": []
    }
  ]
}
```

Request URL:

```
PATCH https://<mso-ip>/api/v1/schemas/5c4b55db1a00003422f2215e
```

Request payload:

```
[{
    "op": "remove",
    "path": "/templates/Template1/vrfs/vrf1"
}]
```

Request response:

```
{
  "id": "5c4b55db1a00003422f2215e",
  "displayName": "SampleSchema",
  "templates": [
    {
      "name": "Template1",
      "displayName": "T1",
      "tenantId": "0000ffff0000000000000010",
```

```
      "anps": [],
      "vrfs": [],
      "bds": [],
      "contracts": [],
      "filters": [],
      "externalEpgs": [],
      "serviceGraphs": [],
      "intersiteL3outs": []
    }
  ]
}
```

Remove an Object Using PATCH Request (staticPort Example)

An additional example of using `PATCH` request to remove a static port.

Original schema:

```
{
    "id": "601acfed38000070a4ee9ec0",
    "displayName": "Schema1",
    "description": "",
    "templates": [
        {
            "name": "Template1",
            "displayName": "Template 1",
            "tenantId": "0000ffff0000000000000010",
            "anps": [
                {
                    "name": "AP1",
                    "displayName": "AP 1",
                    "anpRef":
"/schemas/601acfed38000070a4ee9ec0/templates/Template1/anps/AP1",
                    "epgs": [
                        {
                            "name": "EPG1",
                            "displayName": "EPG 1",
                            "epgRef":
"/schemas/601acfed38000070a4ee9ec0/templates/Template1/anps/AP1/epgs/EPG1",
                            "contractRelationships": [],
                            "subnets": [],
                            "uSegEpg": false,
                            "uSegAttrs": [],
                            "intraEpg": "unenforced",
                            "prio": "unspecified",
                            "proxyArp": false,
                            "preferredGroup": false,
                            "bdRef":
"/schemas/601acfed38000070a4ee9ec0/templates/Template1/bds/BD1",
                            "vrfRef": "",
                            "selectors": [],
                            "epgType": "application"
                        }
                    ]
                }
            ],
            "vrfs": [
                {
                    "name": "VRF1",
                    "displayName": "VRF 1",
                    "vrfRef":
"/schemas/601acfed38000070a4ee9ec0/templates/Template1/vrfs/VRF1",
                    "l3MCast": false,
                    "preferredGroup": false,
```

```
                        "vzAnyEnabled": false,
                        "vzAnyProviderContracts": [],
                        "vzAnyConsumerContracts": [],
                        "rpConfigs": [],
                        "pcEnfPref": "enforced",
                        "ipDataPlaneLearning": "enabled"
                    }
                ],
                "bds": [
                    {
                        "name": "BD1",
                        "displayName": "BD 1",
                     "bdRef": "/schemas/601acfed38000070a4ee9ec0/templates/Template1/bds/BD1",

                        "l2UnknownUnicast": "proxy",
                        "intersiteBumTrafficAllow": true,
                        "optimizeWanBandwidth": true,
                        "l2Stretch": true,
                        "subnets": [],
                        "vrfRef":
"/schemas/601acfed38000070a4ee9ec0/templates/Template1/vrfs/VRF1",
                        "unkMcastAct": "flood",
                        "v6unkMcastAct": "flood",
                        "arpFlood": true,
                        "multiDstPktAct": "bd-flood"
                    }
                ],
                "contracts": [],
                "filters": [],
                "externalEpgs": [],
                "serviceGraphs": [],
                "intersiteL3outs": [],
                "templateType": "stretched-template",
                "templateSubType": []
            }
        ],
        "_updateVersion": 1,
        "sites": [
            {
                "siteId": "5efceb4a3600002738221157",
                "templateName": "Template1",
                "anps": [
                    {
                        "anpRef":
"/schemas/601acfed38000070a4ee9ec0/templates/Template1/anps/AP1",
                        "epgs": [
                            {
                                "epgRef":
"/schemas/601acfed38000070a4ee9ec0/templates/Template1/anps/AP1/epgs/EPG1",
                                "domainAssociations": [],
                                "staticPorts": [
                                    {
                                        "type": "port",
                                        "path": "topology/pod-1/paths-101/pathep-[eth1/1]",
                                        "portEncapVlan": 1,
                                        "deploymentImmediacy": "lazy",
                                        "mode": "regular"
                                    },
                                    {
                                        "type": "port",
                                        "path": "topology/pod-1/paths-102/pathep-[eth1/2]",
                                        "portEncapVlan": 2,
                                        "deploymentImmediacy": "lazy",
                                        "mode": "regular"
```

```
                                }
                            ],
                            "staticLeafs": [],
                            "uSegAttrs": [],
                            "subnets": [],
                            "selectors": []
                        }
                    ]
                }
            ],
            "vrfs": [],
            "bds": [],
            "contracts": [],
            "externalEpgs": [],
            "serviceGraphs": [],
            "intersiteL3outs": []
        }
    ]
}
```

Request URL:

```
PATCH https://<mso-ip>/api/v1/schemas/601acfed38000070a4ee9ec0
```

Request payload:

```
[{
    "op": "remove",
    "path": "/sites/0/anps/0/epgs/0/staticPorts/1"
}]
```

Request response:

```
{
    "id": "601acfed38000070a4ee9ec0",
    "displayName": "Schema1",
    "description": "",
    "templates": [
        {
            "name": "Template1",
            "displayName": "Template 1",
            "tenantId": "0000ffff0000000000000010",
            "anps": [
                {
                    "name": "AP1",
                    "displayName": "AP 1",
                    "anpRef":
"/schemas/601acfed38000070a4ee9ec0/templates/Template1/anps/AP1",
                    "epgs": [
                        {
                            "name": "EPG1",
                            "displayName": "EPG 1",
                            "epgRef":
"/schemas/601acfed38000070a4ee9ec0/templates/Template1/anps/AP1/epgs/EPG1",
                            "contractRelationships": [],
                            "subnets": [],
                            "uSegEpg": false,
                            "uSegAttrs": [],
                            "intraEpg": "unenforced",
                            "prio": "unspecified",
                            "proxyArp": false,
                            "preferredGroup": false,
                            "bdRef":
"/schemas/601acfed38000070a4ee9ec0/templates/Template1/bds/BD1",
                            "vrfRef": "",
                            "selectors": [],
```

```
                            "epgType": "application"
                        }
                    ]
                }
            ],
            "vrfs": [
                {
                    "name": "VRF1",
                    "displayName": "VRF 1",
                    "vrfRef":
"/schemas/601acfed38000070a4ee9ec0/templates/Template1/vrfs/VRF1",
                    "l3MCast": false,
                    "preferredGroup": false,
                    "vzAnyEnabled": false,
                    "vzAnyProviderContracts": [],
                    "vzAnyConsumerContracts": [],
                    "rpConfigs": [],
                    "pcEnfPref": "enforced",
                    "ipDataPlaneLearning": "enabled"
                }
            ],
            "bds": [
                {
                    "name": "BD1",
                    "displayName": "BD 1",
                     "bdRef": "/schemas/601acfed38000070a4ee9ec0/templates/Template1/bds/BD1",

                    "l2UnknownUnicast": "proxy",
                    "intersiteBumTrafficAllow": true,
                    "optimizeWanBandwidth": true,
                    "l2Stretch": true,
                    "subnets": [],
                    "vrfRef":
"/schemas/601acfed38000070a4ee9ec0/templates/Template1/vrfs/VRF1",
                    "unkMcastAct": "flood",
                    "v6unkMcastAct": "flood",
                    "arpFlood": true,
                    "multiDstPktAct": "bd-flood"
                }
            ],
            "contracts": [],
            "filters": [],
            "externalEpgs": [],
            "serviceGraphs": [],
            "intersiteL3outs": [],
            "templateType": "stretched-template",
            "templateSubType": []
        }
    ],
    "_updateVersion": 1,
    "sites": [
        {
            "siteId": "5efceb4a3600002738221157",
            "templateName": "Template1",
            "anps": [
                {
                    "anpRef":
"/schemas/601acfed38000070a4ee9ec0/templates/Template1/anps/AP1",
                    "epgs": [
                        {
                            "epgRef":
"/schemas/601acfed38000070a4ee9ec0/templates/Template1/anps/AP1/epgs/EPG1",
                            "domainAssociations": [],
                            "staticPorts": [
```

```
                                    {
                                        "type": "port",
                                        "path": "topology/pod-1/paths-101/pathep-[eth1/1]",
                                        "portEncapVlan": 1,
                                        "deploymentImmediacy": "lazy",
                                        "mode": "regular"
                                    }
                                ],
                                "staticLeafs": [],
                                "uSegAttrs": [],
                                "subnets": [],
                                "selectors": []
                            }
                        ]
                    }
                ],
                "vrfs": [],
                "bds": [],
                "contracts": [],
                "externalEpgs": [],
                "serviceGraphs": [],
                "intersiteL3outs": []
            }
        ]
}
```

Concurrent Configuration Updates

The Multi-Site Orchestrator GUI will ensure that any concurrent updates on the same site or schema object cannot unintentionally overwrite each other. If you attempt to make changes to a site or schema that was updated by another user since you opened it, the GUI will reject any subsequent changes you try to make and present a warning requesting you to refresh the object before making additional changes:

However, the default REST API functionality was left unchanged in order to preserve backward compatibility with existing applications. In other words, while the UI is always enabled for this protection, you must explicitly enable it for your API calls for MSO to keep track of configuration changes.

Note When enabling this feature, note the following:

- This release supports detection of conflicting configuration changes for Site and Schema objects only.
- Only `PUT` and `PATCH` API calls support the version check feature.
- If you do not explicitly enable the version check parameter in your API calls, MSO will not track any updates internally. And as a result, any configuration updates can be potentially overwritten by both subsequent API calls or GUI users.

To enable the configuration version check, you can pass the `enableVersionCheck=true` parameter to the API call by appending it to the end of the API endpoint you are using, for example:

`https://<mso-ip-address>/mso/api/v1/schemas/<schema-id>?enableVersionCheck=true`

Example

We will use a simple example of updating the display name of a template in a schema to show how to use the version check attribute with `PUT` or `PATCH` calls.

First, you would `GET` the schema you want to modify, which will return the current latest version of the schema in the call's response:

```
{
    "id": "601acfed38000070a4ee9ec0",
    "displayName": "Schema1",
    "description": "",
    "templates": [
        {
            "name": "Template1",
            "displayName": "current name",
            [...]
        }
    ],
    "_updateVersion": 12,
    "sites": [...]
}
```

Then you can modify the schema in one of two ways appending `enableVersionCheck=true` to the request URL:

Note You must ensure that the value of the `"_updateVersion"` field in the payload is the same as the value you got in the original schema.

- Using the `PUT` API with the entire updated schema as payload:

```
PUT /v1/schemas/601acfed38000070a4ee9ec0?enableVersionCheck=true
```

```
{
    "id": "601acfed38000070a4ee9ec0",
    "displayName": "Schema1",
    "description": "",
    "templates": [
        {
            "name": "Template1",
            "displayName": "new name",
            [...]
        }
    ],
    "_updateVersion": 12,
    "sites": [...]
}
```

- Using any of the `PATCH` API operations to make a specific change to one of the objects in the schema:

```
PATCH /v1/schemas/601acfed38000070a4ee9ec0?enableVersionCheck=true
```

```
[
    {
        "op": "replace",
        "path": "/templates/Template1/displayName",
        "value": "new name",
        "_updateVersion": 12
    }
]
```

When the request is made, the API will increment the current schema version by 1 (from `12` to `13`) and attempt to create the new version of the schema. If the new version does not yet exist, the operation will succeed and the schema will be updated; if another API call (with `enableVersionCheck` enabled) or the UI have modified the schema in the meantime, the operation fails and the API call will return the following response:

```
{
    "code": 400,
    "message": "Update failed, object version in the DB has changed, refresh your client
and retry"
}
```

REST API Reference (OpenAPI/Swagger)

Multi-Site Orchestrator uses OpenAPI (also known as Swagger) to provide a complete API reference for developers linked directly from the Orchestrator's GUI. OpenAPI allows you to visualize and interact with the API's resources. The direct connection from the reference document to the live Orchestrator API provides an easy way to write and test simple request directly from the OpenAPI reference UI.

- **APIC Information API**: Allows you to query the Cisco APIC sites directly for specific information, such as Pods, Tenants, VMM domains, L3Outs, and so on. This API is part of the `site` API.
- **Audit API**: Allows you to access Multi-Site Orchestrator's audit logs and query for information, such as records and users, as well as download all or specific records.
- **Backup API**: Allows you to create, update, delete, or restore a backup of Multi-Site Orchestrator configuration, as well as schedule a backup.
- **Deployment API**: Allows you to deploy the Schemas and Templates directly to sites using `SiteId` and `TemplateName`.
- **Fabric Connectivity API**: Allows you to query for fabric connectivity, connectivity status, multipod information, and so on. This API is part of the `site` API.
- **Infrastructure Logs API**: Allows you to query for infrastructure and platform logs. This API is part of the `platform` API.
- **Platform API**: Allows you to query for information about platform nodes and labels.
- **Policy Report API**: Allows you to query for policy deployment reports. This API is part of the `platform` API.
- **Schema API**: Allows you to create, update, patch, or delete schemas, as well as query for information associated with a schema.
- **Site API**: Allows you to create, update, patch, or delete sites. This API is part of the `site` API.
- **Tenant API**: Allows you to create, update, patch, or delete tenants, as well as query for information associated with a tenant. This API is part of the `schema` API.
- **User API**: Allows you to create, update, or delete users, update user roles, and configure authentication providers, such as LDAP, TACACs, or RADIUS.

Accessing OpenAPI Reference

This section describes how to access the OpenAPI (Swagger) reference documentation from your Multi-Site Orchestrator.

Step 1 Log in to your Multi-Site Orchestrator.

Step 2 In the top right corner of the main window, click the **Options** icon.

Step 3 Choose **View Swagger Docs**.

Step 4 In the **Swagger APIs** window that opens, select the API that you want to view

For more information on each of the listed APIs, see REST API Reference (OpenAPI/Swagger), on page 16.

What to do next

The OpenAPI reference provides a simple, visual representation of the REST API. It also allows you to create and test simple API requests directly on your fabric. The following two sections provide examples that will help you to get started working with Multi-Site Orchestrator API and OpenAPI reference.

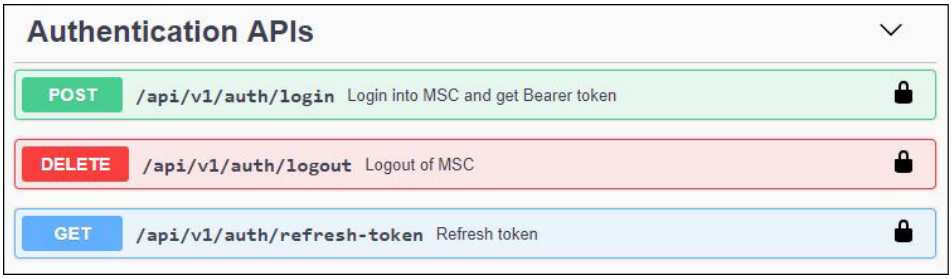
Using OpenAPI for Authentication

The following example shows how to use the OpenAPI GUI to provide your login credentials to log in to the Orchestrator and receive an authentication token for use in any subsequent REST API requests.

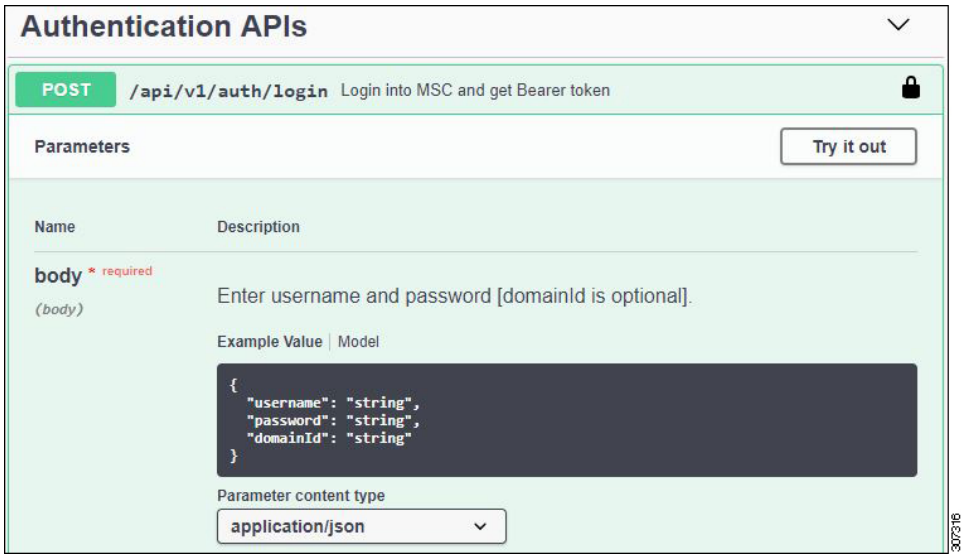
Before you begin

Step 1 Navigate to the User API reference as described in Accessing OpenAPI Reference, on page 17, or you can open your browser to `<mso-ip>/docs/userdocs#/` after logging into the Orchestrator.

Step 2 Scroll down to **Authentication APIs** section.

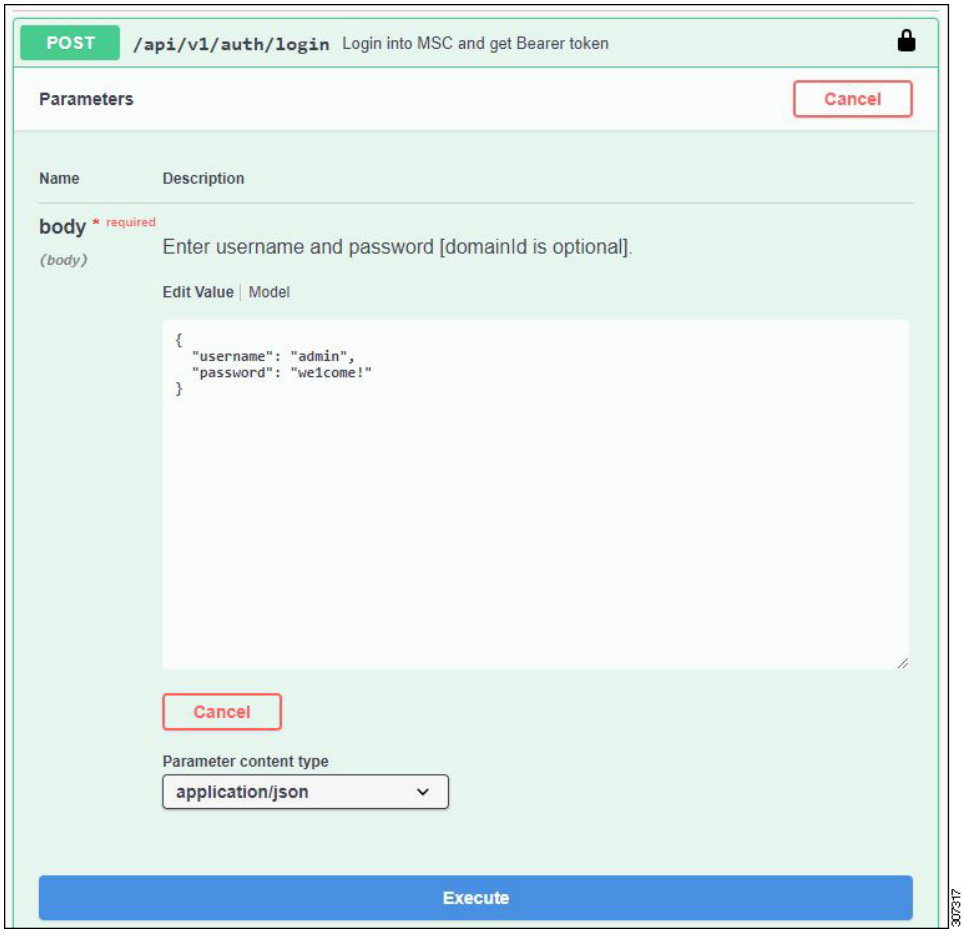


Step 3 Click the `POST /api/v1/auth/login` row to expand the section.



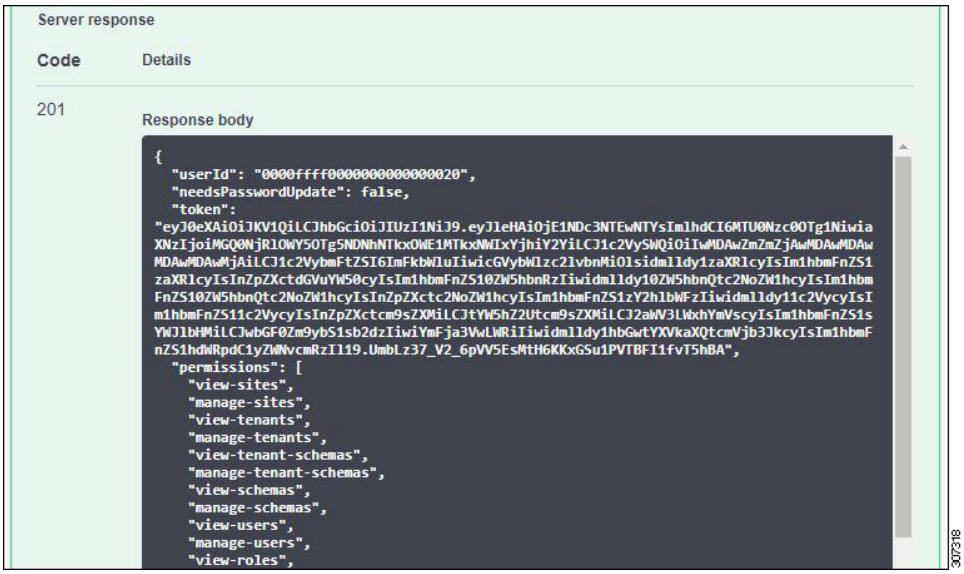
Step 4 Click the **Try it out** button.

Step 5 Edit the `POST` request message



Step 6 Click **Execute** to send the request.

Step 7 If the request was successful, you will be able to see the server response and get the authentication token you can use in future requests.



Using OpenAPI to Manage Users

The following example shows how to use the OpenAPI GUI to send and receive sample REST requests to create, modify, and then delete a user.

Before you begin

You must obtain an authentication token as described in Using OpenAPI for Authentication, on page 17.

Step 1 Navigate to the User API reference as described in Accessing OpenAPI Reference, on page 17, or you can open your browser to `<mso-ip>/docs/userdocs#/` after logging into the Orchestrator.

Step 2 Scroll down to **User APIs** section.

Step 3 Click the `POST /api/v1/users` row and then **Try it out**.

Step 4 Edit the `POST` request message and then click **Execute**.

In the following sample post, provide the ID of the user you have created in previous steps.

```
{
  "id": "string",
  "username": "jsmith",
  "password": "welc0me!",
  "firstName": "John",
  "lastName": "Smith",
  "emailAddress": "jsmith@company.com",
  "phoneNumber": "123-456-7890",
  "accountStatus": {},
  "needsPasswordUpdate": 0,
  "roles": [
    {
```

```
      "roleId": "0000ffff0000000000000031"
    }
  ],
  "domainId": "0000ffff0000000000000090",
  "remote": false,
  "active": false
}
```

Step 5 If the request is successful, you will receive a response XML message containing the ID of the user that you created.

You can verify that the new user was created by logging into your Multi-Site Orchestrator GUI and checking the **Users** page.

Step 6 Now click on `PATCH /api/v1/users/{id}` and then **Try it out**.

In the following sample post, provide the ID of the user you created in previous steps as well as any details you wish to update.

```
{
  "id": "<user-id>",
  "emailAddress": "johnsmith@company.com",
}
```

Step 7 If the request is successful, you will receive a response XML message containing the updated user details.

You can verify that the user was updated by logging into your Multi-Site Orchestrator GUI and checking the **Users** page.

Step 8 Now click on `DELETE /api/v1/users/{id}` and then **Try it out**.

Step 9 Provide the ID of the user you created in previous steps and click Exectue.

You can verify that the user was delete by logging into your Multi-Site Orchestrator GUI and checking the **Users** page.

Using In-Browser DevTools to View REST API Calls

Multi-SiteOrchestrator is fully RESTful, as such each action performed in the GUI triggers one or more REST API calls to retrieve the fabric information to be displayed or to make changes to the fabric. You can view these API calls using the built-in developer tools in your browser of choice. Inspecting API calls associated with one or more UI actions may be useful for becoming familiar with the specific call syntax and contents.

This section describes how to access developer tools in your browser.

Step 1 Open the built-in browser developer tools.

a) If you are using Google Chrome, right click in the main browser window and choose **Inspect**.
b) If you are using Mozilla Firefox, right click in the main browser window and choose **Inspect Element**.
c) If you are using Microsoft Edge, right click in the main browser window and choose **Inspect Element**.
d) If you are using Apple Safari, you must first enable the Develop menu.

   From the top menu bar, choose **Safari** > **Preferences**. Then select the **Advanced** tab and check the **Show Develop menu in menu bar** checkbox.

   Finally, open the developer tools pane by choosing **Develop** > **Show Web Inspector** from the top menu bar.

A developer tools panel will open in the main browser window alongside the current page.

Step 2 In the developer tools panel that opens, select the **Network** tab.

Step 3 Navigate to your Multi-Site Orchestrator and log in.

Step 4 Perform an action in the Multi-Site GUI to trigger one or more API calls.

You will be able to see the calls as they are being made by the browser to the Multi-Site Orchestrator.

Note You may want to clear the list before performing an action by clicking **Clear** button in the **Network** tab's menu bar.

Step 5 (Optional) Filter the list of API calls to view the relevant ones.

You can filter the list to display only the REST API calls by clicking the **XHR** button in the **Network** tab's menu bar.

Step 6 (Optional) Sort the list of API calls to group similar ones.

You can click on the column title to sort the network calls based on the property.

Step 7 Select one of the listed API calls to inspect its content.

When you select one of the calls, you can view its details in the pane that opens:

- **Headers** tab shows the general, request, and response headers for the selected API call. You can view the call's request method, such as `GET` or `POST`, the remote address, authorization token, and the cookies.
- **Preview** tab shows the XML text of the call sent to the server.
- **Response** tab shows the XML text of the response received from the server.
- **Cookies** tab shows the cookie information.

What to do next

For in-depth information on each browser's developer tools and their capabilities, see their respective documentation:

- Google Chrome: https://developers.google.com/web/tools/chrome-devtools/#network
- Mozilla Firefox: https://developer.mozilla.org/en-US/docs/Tools/Network_Monitor
- Microsoft Edge: https://docs.microsoft.com/en-us/microsoft-edge/devtools-guide/network
- Apple Safari: https://support.apple.com/guide/safari-developer/network-tab-dev1f3525e58/mac

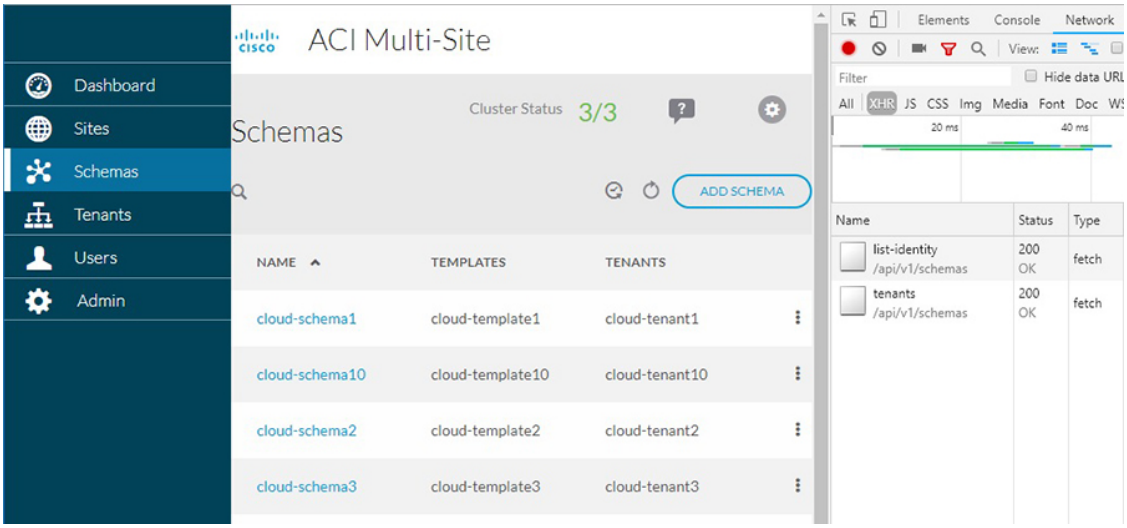
Using In-Browser DevTools to Work with REST API Calls

This section illustrates some simple examples of using browser tools to view, modify, and re-send REST API calls. The examples presented here use Google Chrome, however the steps are similar for other browsers. Accessing each browser's developer tools is described in Using In-Browser DevTools to View REST API Calls, on page 20.

When you perform an action in the Multi-Site Orchestrator GUI with DevTools open, you can see a list of the API calls made by the GUI to the Orchestrator. From there on, you can select each individual call to inspect its request and response contents.

For example, if you click on the **Schemas** view in the left-hand navigation sidebar of the GUI, two calls are made to `/api/v1/schemas` – one to get a list of schemas and another to get a list of tenants for each schema:

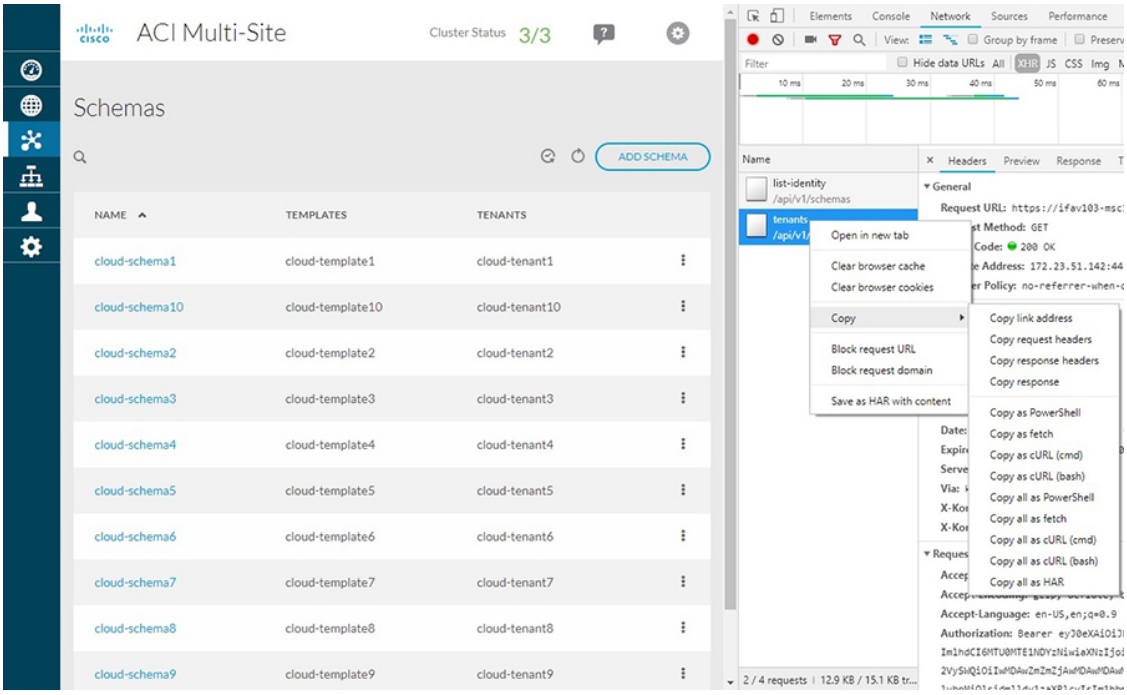
Figure 1: Schemas API Calls



If you then select one of the calls, for example `tenants`, you can view the call details, such as the request method, request URL, authentication token, and so on. The response tab provides the exact XML string received in response to the call.

If you right-click on any of the listed API calls, you can choose to copy the call in any of the available formats:

Figure 2: Copy API Call



For example, you can choose to copy the request as cURL, which you can then execute in 3rd party scripts:

```
curl "https://172.31.187.59/api/v1/schemas/tenants" -H "Authorization: Bearer
 -H "DNT: 1" -H "Accept-Encoding: gzip, deflate, br" -H "Accept-Language: en-US,en;q=0.9"
-H "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/69.0.3497.100 Safari/537.36" -H "Content-Type: application/json" -H "Accept:
 application/json" -H "Referer: https://172.31.187.59/schemas" -H "Connection: keep-alive"
 --compressed --insecure
```

Another example would be the `users` API which is called when you select the **Users** view. The GUI retrieves the list of users using the following corresponding cURL request:

```
curl "https://172.31.187.59/api/v1/users" -H "Authorization: Bearer
 -H "DNT: 1" -H "Accept-Encoding: gzip, deflate, br" -H "Accept-Language: en-US,en;q=0.9"
-H "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/69.0.3497.100 Safari/537.36" -H "Content-Type: application/json" -H "Accept:
 application/json" -H "Referer: https://172.31.187.59/users" -H "Connection: keep-alive"
--compressed --insecure
```

And the request returns a JSON-formatted XML message similar to the following:

```
{
    "users": [{
            "id": "59f17fe7120000da00bf0d11",
            "username": "max-usermgr",
            "password": "******",
            "firstName": "firstName",
            "lastName": "lastName",
            "emailAddress": "email@domain.com",
            "phoneNumber": "7474747477",
            "accountStatus": "active",
            "needsPasswordUpdate": false,
            "roles": [{
                    "roleId": "0000ffff0000000000000035"
                }
            ],
            "domainId": "0000ffff0000000000000090"
        }, {
            "id": "59f17f9e0e00001701000326",
            "username": "max-schemamgr",
            "password": "******",
            "firstName": "firstName",
            "lastName": "lastName",
            "emailAddress": "email@domain.com",
            "phoneNumber": "34343442234",
            "accountStatus": "active",
            "needsPasswordUpdate": false,
            "roles": [{
                    "roleId": "0000ffff0000000000000033"
                }
            ],
            "domainId": "0000ffff0000000000000090"
        }, {
            "id": "59f17f6b120000eb00bf0d10",
            "username": "max-sitemgr",
            "password": "******",
            "firstName": "firstName",
            "lastName": "lastName",
            "emailAddress": "email@domain.com",
            "phoneNumber": "3838833838",
            "accountStatus": "active",
            "needsPasswordUpdate": false,
            "roles": [{
                    "roleId": "0000ffff0000000000000032"
                }
```

```
            ],
            "domainId": "0000ffff0000000000000090"
        }
    ]
}
```

Modifying Examples and Making Changes to the Fabric

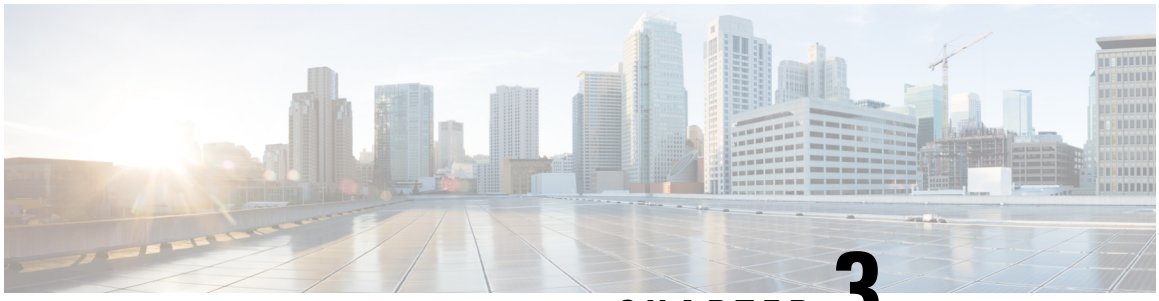
In addition to being able to retrieve information about your Multi-Site environment using the REST API, you can use it to make changes.

For instance, if instead of viewing the users, you create one in the GUI, you can then copy the call as an example, modify it, and then use the modified call in your 3rd party script to create or update user accounts. Below is a sample cURL command to create a user along with the XML POST contents:

```
curl "https://172.31.187.59/api/v1/users" -H "Origin: https://172.31.187.59" -H
"Accept-Encoding: gzip, deflate, br" -H "Accept-Language: en-US,en;q=0.9" -H "Authorization:
 Bearer
[illegible]
 -H "Content-Type: application/json" -H "Accept: application/json" -H "Referer:
https://172.31.187.59/users/create" -H "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64;
x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36" -H
"Connection: keep-alive" -H "DNT: 1" --data-binary
[illegible]
 --compressed --insecure
```

And this is the contents of the `POST` call to create a user:

```
{
    "id": "5bb7be240f00008206d3f137",
    "username": "newUserName",
    "password": "******",
    "firstName": "firstName",
    "lastName": "lastName",
    "emailAddress": "email@domain.com",
    "phoneNumber": "1234567890",
    "accountStatus": "active",
    "needsPasswordUpdate": true,
    "roles": [{
            "roleId": "0000ffff0000000000000031"
        }, {
            "roleId": "0000ffff0000000000000032"
        }
    ],
    "domainId": "0000ffff0000000000000090"
}
```



CHAPTER 3

REST API Data Structures

This chapter contains the following sections:

- REST API Data Structures Overview, on page 25
- Schema, on page 25
- Reference Fields, on page 29

REST API Data Structures Overview

This section describes the objects used by the Cisco Multi-Site REST API and their hierarchy. The objects are represented using JavaScript Object Notation (JSON) format with the top level containing Schemas. The data structure is consistent across Multi-Site with Schemas containing templates and tempaltes containing lower level objects such as application network profiles (anps), bridge domains (bds), virtual routing and forwarding (vrfs) instances and so on. These entities also contain other smaller structures.

A few high-level entities or concepts are described in the following sections. In the examples presented, when a field name is pluralized (such as `vrfs` or `bds`), the field contains an array of that type of objects associated with the parent structure. For example, the `vrfs` array contains all the VRF instances associated with the template.

For a complete object and method reference, see the OpenAPI reference as described in REST API Reference (OpenAPI/Swagger), on page 16.

Schema

A `schema` object represents a schema. Each `schema` object contains the number templates and sites associated with it. When a site is chosen, it automatically gets added to the schema and mapped to a template in that schema.

Each `schema`object contain the following fields:

- `id`: the ID of the schema object.
- `displayName`: the name of the schema as displayed by the GUI.
- `templates`: templates associated with the schema.
- `sites`: sites associated with the schema.

The following snippet provides a high-level example of a schema object:

```
{
    "id": "583c7c48250100250106198 5",
    "displayName": "Schema 1",
    "templates": [ ],
    "sites": [ ]
}
```

Template

A template is the detailed structure that contains policies that you want to push to Cisco APIC. Templates exist in the context of a schema and not on their own. You can define the policy of a new template as desired. If there are multiple templates, you can choose a template from which to inherit the policy.

Template objects contain the following fields:

- `name`: the name of the template object.
- `displayName`: the name as displayed by the GUI.
- `tenantId`: the ID of the tenant with which the template is associated.
- `anps`: application network profiles associated with the template.
- `vrfs`: virtual route forwarding instances associated with the template.
- `bds`: bridge domains associated with the template.
- `contracts`: contracts associated with the template.
- `filters`: filters associated with the schema.

The following snippet provides a high-level example of a template object:

```
{
    "name": "Template1",
    "displayName": "Template 1",
    "tenantId": "5b90695f1e00005d3b46efa2",
    "anps": [ ],
    "vrfs": [ ],
    "bds": [ ],
    "contracts": [ ],
    "filters": [ ]
}
```

anp

An `anp` object represents an application profile.

Each `anp` object contain the following fields:

- `name`: the name of the application profile.
- `displayName`: the name of the application profile as displayed by the GUI.
- `anpRef`: application profile reference.
- `epgs`: EPGs associated with the application profile.

The following is a sample `anp` object:

```
{
    "name": "anp1",
    "displayName": "Anp 1",
    "anpRef": "/schemas/583c7c482501002501061985/templates/template1/anps/anp1",
    "epgs": [ ]
}
```

epg

An `epg` object represents an End-Point Group (EPG). Each EPG can contain multiple contract relationships, bridge domains (BDs), subnets, and micro-segmentation attributes.

The following snippet provides a sample `epg` object:

```
{
    "name": "epg1",
    "displayName": "EPG 1",
    "epgRef": "/schemas/583c7c482501002501061985/templates/template1/anps/anp1/epgs/epg1",
    "contractRelationships": [ ],
    "subnets": [ ],
    "uSegEpg": true,
    "uSegAttrs": [ ],
    "bdRef": "/schemas/583c7c482501002501061985/templates/template1/bds/bd1"
}
```

bd

An `bd` object represents Bridge Domain (BD).

The following snippet provides a sample `bd` object:

```
{
    "name": "bd1",
    "displayName": "BD 1",
    "bdRef": "/schemas/583c7c482501002501061985/templates/template1/bds/bd1",
    "l3UnknownMulticastFlooding": "opt-flood",
    "intersiteBumTrafficAllow": true,
    "multiDestinationFlooding": "encap-flood",
    "l2UnknownUnicast": "flood",
    "l2Stretch": true,
    "subnets": [...],
    "vrfRef": "/schemas/583c7c482501002501061985/templates/template1/vrfs/vrf1"
}
```

The following values are supported for `l3UnknownMulticastFlooding`:

- `flood`
- `opt-flood`

The following values are supported for `multiDestinationFlooding`:

- `bd-flood`
- `drop`
- `encap-flood`

The following values are supported for `l2UnknownUnicast`:

- `flood`
- `proxy`

contract

A `contract` object represents a contract between two EPGs.

The following snippet provides a sample `contract` object:

```
{
    "name": "contract1",
    "displayName": "Contract 1",
    "filterRelationships": [{
            "filterRef": "/templates/template1/filters/filter1",
            "directives": [ "log" ]
        }
    ],
    "scope": "global"
}
```

The following `scope` values are supported:

- `global`
- `tenant`
- `context`
- `application-profile`

filter

An `filter` object represents a filter for a contract.

The following snippet provides a sample `filter` object:

```
{
    "name": "filter1",
    "displayName": "Filter 1",
    "description": "",
    "entries": [{
            "name": "filterEntry11",
            "displayName": "Filter Entry 11",
            "description": "",
            "etherType": "ip",
            "ipProtocol": "icmp"
        }, {
            "name": "filterEntry12",
            "displayName": "Filter Entry 12",
            "description": "",
            "etherType": "ip",
            "ipProtocol": "udp",
            "matchOnlyFragments": false,
            "sourceFrom": "dns",
            "sourceTo": "http",
            "destinationFrom": "dns",
            "destinationTo": "80"
        }
    ]
}
```

The following filter criteria are supported:

```
"etherType": "arp|fcoe|ip|mac_security|mpls_ucast|trill|unspecified",
"arpFlag": "request|reply|unspecified",
"ipProtocol": "eigrp|egp|icmp|icmpv6|igmp|igp|l2tp|ospfigp|pim|tcp|udp|unspecified",
"matchOnlyFragments": false,
```

```
"stateful": false,
"sourceFrom": "dns|ftp-data|http|https|pop3|rtsp|smtp|unspecified|[0..65535]",
"sourceTo": "dns|ftp-data|http|https|pop3|rtsp|smtp|unspecified|[0..65535]",
"destinationFrom": "dns|ftp-data|http|https|pop3|rtsp|smtp|unspecified|[0..65535]",
"destinationTo": "dns|ftp-data|http|https|pop3|rtsp|smtp|unspecified|[0..65535]",
"tcpSessionRules": [
    "acknowledgement|established|finish|synchronize|reset|unspecified",
    "acknowledgement|established|finish|synchronize|reset|unspecified"
]
```

site

Each `site` object represents a site associated with a template. The policies from specific templates are pushed to sites based on the list of sites associated with the template.

The `siteId` field contains the ID of the site as defined in Multi-Site Orchestrator.

Each `site`object contain the following fields:

- `siteId`: the ID of the site.
- `templateName`: template associated with the site.
- `bds`: bridge domains associated with the site.

The following is an example of a site API snippet:

```
{
    "siteId": "58202f7066e6e10001c41236",
    "templateName": "template1",
    "bds": [ ]
}
```

Reference Fields

Reference fields are generated for each and every policy to provide relationships between the policies. These fields are typically used in situations when you need to refer to other entities from within a particular entity, for example for policies within a template. A reference is generated on the first save, however, if you define a policy within the Schema and you want to reference it right away, you can specify a reference without having to wait for it to be generated.

Sample References

As an example, the `anpRef` and `vrfRef` fields are automatically generated by the system to identify application profiles (`anpRef`) and virtual routing and forwarding instance (`vrfRef`) respectively.

The following snippet provides an example of the `anpRef` field in an application profile object:

```
{
    "name": "anp1",
    "displayName": "AP 1",
    "anpRef": "/schemas/583c7c482501002501061985/templates/template1/anps/anp1",
    "epgs": [ ]
}
```

The following snippet provides an example of the `vrfRef` field in a VRF instance object:

```
{
    "name": "vrf1",
```

```
    "displayName": "VRF 1",
    "vrfRef": "/schemas/583c7c482501002501061985/templates/template1/vrfs/vrf1"
}
```

Local References

In some cases, the reference that is supplied as input to the post operation, does not contain the Schema ID (as system generated references contain). A local reference means that an entity within this Schema is referenced by the path of that entity. In such a case, the ID is not used as a reference, and the reference does not contain Schema ID.

For example, you create a contract, and you want to reference the contract within the EPG immediately. The system will not yet have generated a reference for the contract, so the generated 'ref' is not available. However, you can still ensure the relationship by referencing the contract to the path of an entity that already exists in the system.

Once it persists, the system makes a local reference an absolute reference. The system prepends the Schema ID even though it is a local reference that is available in the same Schema, and thereby fully populates the reference.

The following is an example of a local contract reference API snippet:

```
{
   "name": "epg1",
   "displayName": "EPG 2",
   "contractRelationships":
  {
     "contractRef": "contractRef": "/templates/template1/contracts/contract1",
     "relationshipType":  "consumer"
    }
 },
```

References from Another Schema

It is possible to associate an entity (for example, a provider contract) from another Schema. You can specify the contract reference by prepending it with an ID contained in an absolute reference from another Schema. This references another policy in another Schema.

The following is an API snippet example of a reference from another Schema:

```
{
   "name": "epg1",
   "displayName": "EPG 2",
   "contractRelationships":
  {
     "contractRef":
"/schemas/590ca1811f000062006eef23/templates/template1/contracts/contract2",
     "relationshipType":  "provider"
    }
 },
```