REST API Data Structures

This chapter contains the following sections:

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).

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": "583c7c482501002501061985",
    "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"
    }
 },
```