AVD - Nautobot Modelling Conventions

The following conventions are used to model data in nautobot to be consumed by the *nautobot-sync* and *avdbuilder* ansible roles. Value capitalization matters and must match EXACTLY the documented values except where explicitly stated otherwise.

Document conventions:

- MUST is used to denote a mandatory practice, without which the ansible logic will break.
- SHOULD is used to denote a non-mandatory, but encouraged practice.
- EXACTLY is used to emphasize the importance of subjects like capitalization and naming schemes.

1. Custom Fields

A number of custom fields are used by the ansible roles, which MUST be present in the nautobot application. These are:

- base_vni
  - Type: Integer
  - Name: base_vni
  - Object(s): tenancy > tenant
  - Suggested Values: Increments of 10000 or more.
- bgp_asn
  - Type: Text
  - Name: bgp_asn
  - Object(s): dcim > device, dcim > virtual chassis
  - Suggested Values: Valid 2- or 4-byte ASNs
- device_id
  - Type: Integer
  - Name: device_id
  - Object(s): dcim > device
  - Suggested Values: 1-254
- evpn_role
  - Type: Selection
  - Name: evpn_role
  - Object(s): dcim > device, dcim > virtual chassis
  - Selection Options:
    - client
    - none
    - server
  - Default Value (Optional): client
- ospf_enabled
  - Type: Boolean

    - Name: ospf_enabled
    - Object(s): dcim > interface, ipam > VLAN, ipam > VRF
- raw_eos_cli
    - Type: Text
    - Name: raw_eos_cli
    - Object(s): dcim > interface, + Optionally dcim > interface template
    - Suggested Values: Comma-separated eos-cli commands. These commands will be rendered directly onto the object.
- ip_helpers:
    - Type: Text
    - Name: ip_helpers
    - Objects(s): ipam > VLAN
    - Suggested Values: Comma-separated list of helper IP addresses (for example DHCP Relay servers).
- next_hop:
    - Type: Text
    - Name: next_hop
    - Object(s): ipam > prefix
    - Suggested Values: Gateway IP-address to use as next hop for a static route.
- route_map_in:
    - Type: Text
    - Name: route_map_in
    - Object(s): dcim > device
    - Suggested Values: Route-map name as defined in tenant routing policies.
- route_map_out:
    - Type: Text
    - Name: route_map_out
    - Object(s): dcim > device
    - Suggested Values: Route-map name as defined in tenant routing policies.

2. Device Type:

For each switch in the AVD fabric, a corresponding **Device Type** object MUST be created in Nautobot. The device type is a representation of the hardware device model of the switch.

2.1 Requirements:

2.1.1 Components

2.1.1.1 Ethernet Interfaces

The **Device Type** object MUST have Ethernet interfaces assigned.
The interfaces MUST reflect the hardware model in naming and quantity.

The type of the interfaces SHOULD reflect actual hardware format, for example QSFP28, SFP28, SFP+, etc.

2.1.1.2 Management Interface

The **Device Type** object MUST have an interface named Management1 assigned.
The type of this interfaces SHOULD be 1000-BaseT.

2.1.1.3 SVI Interface

The **Device Type** object MUST have at least 1 interface of the type “virtual” assigned that will be used to associate IP-addresses used by tenant services to the device.
The interface(s) SHOULD be named “SVI” or “IRB”, but this has no real impact on the ansible roles.

3. Device Roles

For each type of device in the AVD fabric, a **Device Role** object MUST be created in nautobot.
The required device types for a l3ls-evpn/vxlan fabric are:

- l3leaf
- spine

The following device role(s) may also be added to nautobot if desired:

- bgp_peer
    - The bgp_peer device role is used by external CE devices to model tenant VRF BGP neighborships.

3.1 Requirements

The **Device Role** object names MUST be exactly as written above (all lowercase).
The **Device Role** objects SHOULD not have the VM Role attribute.

4. Tags

At least one **Tag** object MUST be created which is used to identify “interesting” objects for the ansible roles.

4.1 Required Tags

The only mandatory tag that is in use by the ansible logic MUST have the name “avd” and slug “avd”.

4.2 Optional Tags

The following optional tags can be used with the ansible logic:

- uplink
    - Name: uplink
    - Slug: uplink
- peerlink
    - Name: peerlink
    - Slug: peerlink
- dci-link
    - Name: dci-link
    - Slug: dci_link

- Service Tags: (used to arbitrarily tag services for per-device filtering)
    - Name: any
    - Slug: any

5. Sites

One or more **Site** objects MUST be defined in nautobot. These are used to tie other objects to a specific site.

5.1 Requirements

5.1.1 Fields

- Status: Active
- Tags: avd

6. Virtual Chassis

Virtual Chassis objects are used to render switch MLAG Pair associations and all related MLAG logic. A **Virtual Chassis** object MUST be created for each MLAG switch pair in the AVD fabric.

6.1.1 Fields

The following fields MUST be populated:

- Name
- Custom Fields:
    - BGP ASN (bgp_asn)
    - EVPN Role (evpn_role)

- Tags: avd

6.1.2 Members

The members of the **Virtual Chassis** object MUST contain EXACTLY 2 leaf devices. The position and priority fields are not currently used by the ansible logic.

7. Devices

For each device in the AVD fabric, a **Device** object MUST be created in nautobot.

7.1 Requirements

7.1.1 Fields

The following fields MUST be populated with accurate data:

- Name
- Device Role
- Device Type
- Site
- Status: Active
- (Rack: Optional)
- Primary IPv4 - This is used for management IP assignment.
- Custom Fields:
    - BGP ASN (bgp_asn)
    - Device ID (device_id)
    - EVPN Role (evpn_role)
- Tags: avd

7.1.2 Components

7.1.1.1 Ethernet Interfaces

Device Ethernet Interfaces SHOULD be populated from the Device Type object association and otherwise follow the same requirements.

7.1.1.1.1 Uplink/Peerlink Tagging

7.1.1.2 Management Interface

Device Management Interface SHOULD be populated from the Device Type object association and otherwise follow the same requirements.

7.1.1.3 SVI Interface

Device SVI Interface(s) SHOULD be populated from the Device Type object association and otherwise follow the same requirements.

7.1.1.4 MLAG Interfaces (Optional)

Device MLAG interfaces (dual-homed Port-Channels) MUST be of the type LAG and MUST be named “mlagX” where “X” is an arbitrary number that SHOULD match the lowest numbered physical port which is part of the LAG group.

The MLAG interface SHOULD be created on only one of the switches in the Virtual Chassis or MLAG pair. The Virtual Chassis logic will allow binding Ethernet interfaces from different devices to the same LAG group.

7.1.1.4.1 Examples

An MLAG Port-Channel containing switch1a Ethernet 7 and switch1b ethernet7 SHOULD be created on switch1a and named mlag7.

An MLAG Port-Channel containing switch1a Ethernet53/1 and switch1b Ethernet53/1 SHOULD be created on switch1a and named mlag531.

7.1.1.4.1 Member Ports

MLAG Port-Channel member ports are tied to the LAG group by using the Parent LAG field on the Ethernet Interface.

7.1.1.5 Port-Channel Interfaces (Optional)

Port-Channel interfaces are used to build single-homed port-channels. They MUST be of the type LAG and MUST be named Port-ChannelX where X is an arbitrary number that SHOULD match the lowest numbered physical port which is part of the LAG group.

The Port-Channel numbering convention matches the examples for MLAG interfaces provided above.

7.1.1.5.1 Member Ports

Single-homed Port-Channel member ports are tied to the LAG group by using the Parent LAG field on the Ethernet Interface.

8. Tenants

Tenants objects are used to associate different services and objects with their respective tenant/customer. Each VLAN/Prefix/VRF that is part of a tenant network service MUST be associated with the appropriate **Tenant** object.

8.1 Requirements

8.1.1 Fields

The following fields MUST be populated with accurate information:

- Name
- Custom Fields
    - Base VNI (base_vni)
- Tags: avd

9. VLAN/Prefix Roles

VLAN/Prefix Roles objects are used to differentiate between the functions of different VLANs/Prefixes. The following roles are in use by the ansible logic and MUST be defined:

- access:
    - Name: access
    - Slug: access
- peering:
    - Name: peering
    - Slug: peering
- mlag_ibgp
    - Name: mlag_ibgp
    - Slug: mlag_ibgp
- static_route
    - Name: static_route
    - Slug: static_route

10. VLANs

VLAN objects are used to render VLAN-related logic, they are primarily used by the L2VLAN and SVI network services definitions.

The manner in which **VLAN** objects are used to construct tenant network services will be explained in the Network Services section of this document.

10.1 Requirements

10.1.1 Fields

The following fields MUST be populated with accurate information:

- Name
- Status: Active
- Role

- (Site: Optional)
- (Description: Optional)
- Tenant
- Custom Fields:
    - OSPF Enabled (ospf_enabled)
- Tags: avd

11. VRFs

VRF objects are used to build VRF-related logic, they are used by routing-related network services.

The manner in which **VRF** objects are used to construct tenant network services will be explained in the Network Services section of this document.

11.1 Requirements

11.1.1 Fields

The following fields MUST be populated with accurate information:

- Name
- RD
- Tenant
- Custom Fields:
    - OSPF Enabled (ospf_enabled)
- Tags: avd

12. Prefixes

Prefix objects are used to associate IP subnets to tenant network services.

The manner in which **Prefix** objects are used to construct tenant network services will be explained in the Network Services section of this document.

12.1 Requirements

12.1.1 Fields

The following fields MUST be populated with accurate information:

- Prefix
- Status: Active
- VRF

- Role
- VLAN
- Tenant
- Tags: avd

13. IP Addresses

IP Address objects are used to associate IP addresses to tenant network services.

The manner in which **IP Address** objects are used to construct tenant network services will be explained in the Network Services section of this document.

13.1 Requirements

13.1.1 Fields

The following fields MUST be populated with accurate information:

- Address (the mask portion of the address MUST match the parent **Prefix** object subnet mask)
- Status: Active
- VRF
- Role (the following pre-defined roles are used by the ansible logic)
    - Anycast
    - VIP
    - Secondary
- Tenant
- (Device and Interface: Optional)

14. Network Services

This section of the document is dedicated to describing how to utilize and combine nautobot data models (objects) to build abstracted representations of network services to be consumed by the ansible logic and ultimately produce desired device configurations through AVD.

14.1 L2 VLAN Service

An L2 VLAN service is a pure VXLAN-switched VLAN from the fabric perspective and is constructed in the following way:

1. Define a **VLAN** object, keep in mind duplicate VLAN IDs cannot coexist on the same physical switch. For details, see the VLAN section.

2. Set the Status: Active
3. Set the Tenant field to associate it with a customer.
4. Optionally set the Site field.
    a. Associating a VLAN with a specific Site object will ensure that VLAN is only active at that site. This field SHOULD be left blank if the VLAN is to be stretched across multiple sites, or a special site named “Global” may be used to accomplish the same thing.
5. Tag the VLAN with the “avd” tag.
6. If the L2 VLAN service needs to be granularly deployed according to the AVD filter tag logic, multiple arbitrary tag values may be applied to the VLAN, which will be matched against the l3leaf tag values.

Note: No Prefix association is needed because this service will not be routed in the fabric. If a prefix association is desired for documentation purposes, it MAY be defined but the prefix MUST NOT be associated with an “avd”-tagged VRF, which would render it indistinguishable from an L3 VLAN Service (described below).

14.2 L3 VLAN Service

An L3 VLAN service is a VXLAN-switched VLAN with an associated IRB/SVI on the leaf device and is constructed in the following way:

1. Define a **VLAN** object, keep in mind duplicate VLAN IDs cannot coexist on the same physical switch. For details, see the VLAN section.
2. Define a **VRF** object for the service VRF if it doesn’t already exist. For details, see the VRF section.
3. Define a **Prefix** object for the service. For details, see the Prefix section.
4. Set the Status of the **VLAN** and **Prefix** objects created to: Active
5. Set the Tenant field of all objects created in steps 1-3 to associate them with a customer.
6. (Optionally set the Site field for the **VLAN** object.)
7. Tag the **VLAN**/**VRF**/**Prefix** objects with the “avd” tag.
8. Associate the **Prefix** object with the **VRF** object.
9. Associate the **VLAN** object with the **Prefix** object.

We have now created the basis for a L3 VLAN Service. However, based on which variant of L3 VLAN Service we want to create, we must now follow different workflows.

14.2.1 Variant 1: Anycast Gateway Service

This variation of the L3 VLAN Service is used when an ordinary “server access”-type service with an anycast gateway is desired and no dynamic routing is required.

10. Set the Role field of the **VLAN** and **Prefix** objects to “access”.
11. Create an **IP-address** object inside the Prefix associated with the VLAN service.

12. Set the Status of the object to Active and the Role to “Anycast”. This will be the anycast gateway IP associated with the L3 VLAN service.

Note: DO NOT associate the Anycast Gateway IP to a device.

14.2.2 Variant 2: Routing SVI Service

This variation of the L3 VLAN Service is used when dynamic routing of any kind is required over the SVI.

10. Set the Role field of the **VLAN** and **Prefix** objects to “peering”.
11. Create two or more **IP-address** objects inside the Prefix associated with the VLAN service.
    a. One object for the Virtual Router address.
    b. One or more objects for the device-unique addresses which can be used for dynamic routing adjacency/peering relationships.
12. For the Virtual Router address, set the status to Active and the Role to “VIP”.
13. For the device-unique addresses, set the status to Active, the Role to “Secondary”, and associate the object to a **Device** and **Interface** object. The targeted Interface object SHOULD be the SVI/IRB Virtual-type interface of the target device.
14. If OSPF routing is desired on the SVI, set the “OSPF Enabled” custom field to True on the SVI and its associated VRF.
15. If BGP routing is desired on the SVI, continue to the BGP Routing section of the Network Services documentation.

Note: The specifics of which interface a secondary IP address is assigned to has no impact on the ansible logic, the only thing that matters is the IP<->Device association, which cannot function without an interface.

15. Connected Endpoints

This section of the document is dedicated to describing how to utilize and combine nautobot data models (objects) to build abstracted representations of endpoint connections to be consumed by the ansible logic and ultimately produce desired device configurations through AVD.

For any L2 type connection an IP address MUST NOT be assigned to the interface in nautobot, as this would imply a routed connection. Dot1q parameters MUST be defined.

For any L3 type connection an IP address MUST be assigned to the interface, and dot1q parameters MUST NOT be defined.

15.1 L2 Single-connected Endpoints

For a L2 single-connected endpoint such as a standalone server, choose an interface on any leaf switch. The following parameters are relevant for the interface to be recognised and picked up by the ansible templating logic:

1. Interface name MUST be set.
2. “Enabled” MUST be checked.
3. Interface label may be set.
4. Dot1q mode MUST be set.
    a. If dot1q mode is Access, an access VLAN SHOULD be chosen.
    b. If dot1q mode is Tagged, one or more tagged VLANs SHOULD be chosen, one native VLAN may be chosen if desired. If tagged VLANs are omitted, the resulting trunk port will allow all VLANs (same functionality as Tagged (All)).
    c. If dot1q mode is Tagged (All), a native VLAN may be chosen if desired.
5. The EOS Cli (raw_eos_cli) custom field may be used to provide a comma-separated list of additional EOS cli commands to render on the connection.
6. Optionally the interface may be connected to another device interface with a cable in nautobot.

15.1.1 Yaml Connection Naming

The naming of endpoint connections in the resulting yaml files (rendered AVD data model) will use the following priority order:

1. If the interface is connected to another device interface in nautobot, the connected device and interface will be used for the connection name in the resulting yaml files (rendered AVD data model).
2. If the interface is not connected to another device in nautobot, the interface label will be used to construct the connection name. Be aware that connection names MUST be unique.
3. If none of the above options are used, the connection name will simply be constructed from the leaf name and the leaf interface name in order to have a unique connection name.

15.2 L2 Dual-homed Non-MLAG Connection

This type of connection may be utilized when a dual-homed connection without link aggregation is desired. Typically an ESXi host using VMWare VM load balancing may use this type of connection.

This connection is modeled exactly like an L2 single-connected endpoint described in the previous section, the only difference being that instead of modeling a single leaf/interface, an interface on both members of a leaf pair is chosen and configured according to the steps outlined in section 15.1.

Note: This connection will be virtually indistinguishable from two separate single-connected interfaces. Use of the label field for naming these connections is strongly recommended to identify them as belonging to the same connected endpoint.

15.3 L2 MLAG Connection

For MLAG connections, the following parameters are relevant:

1. A LAG Group type interface MUST be created on one of the leafs in the virtual chassis. The first leaf in the virtual chassis SHOULD be used for this.
    a. The type of the interface MUST be LAG Group.
    b. The name of the interface MUST be mlagX (not case sensitive), where X represents a number. The numbering SHOULD be the lowest numbered interface in the LAG group. Examples of naming conventions can be found in the Components section of this document.
2. 1 or more interfaces from each of the virtual chassis member switches SHOULD be added to the LAG group.
3. Dot1q mode MUST be set on the LAG Group (any dot1q settings on the individual interfaces will not be recognised by the templating logic).
    a. If dot1q mode is Access, an access VLAN SHOULD be chosen.
    b. If dot1q mode is Tagged, one or more tagged VLANs SHOULD be chosen, one native VLAN may be chosen if desired. If tagged VLANs are omitted, the resulting trunk port will allow all VLANs (same functionality as Tagged (All)).
    c. If dot1q mode is Tagged (All), a native VLAN may be chosen if desired.
4. The EOS Cli (raw_eos_cli) custom field may be used to provide a comma-separated list of additional EOS cli commands to render on the connection.

15.3.1 Yaml Connection Naming

The naming of endpoint connections in the resulting yaml files (rendered AVD data model) will use the following priority order:

1. Primarily the interface label will be used to construct the connection name. Be aware that connection names MUST be unique. Since it is not possible to connect a LAG group type interface to another interface in nautobot, use of the label to describe the connection is strongly recommended.
2. If the label is not used, the connection name will simply be constructed from the leaf name and the leaf interface name in order to have a unique connection name.

15.4 L2 non-MLAG LACP Connection

This type of connection is a single-homed port-channel for connecting an endpoint to two or more interfaces on the same leaf switch with a LAG. The following parameters are relevant for this type of connection:

1. A LAG Group type interface MUST be created on a leaf. The first leaf in the virtual chassis SHOULD be used for this.
2. Interfaces from the other leaf in the virtual chassis MUST NOT be added to this LAG group, since this would imply an MLAG connection (please refer to the section above).
    a. The type of the interface MUST be LAG Group.
    b. The name of the interface MUST be port-channelX (not case-sensitive), where X represents a number. The numbering SHOULD be the lowest numbered interface in the LAG group. Examples of naming conventions can be found in the Components section of this document.
3. 1 or more interfaces from the leaf switch MUST be added to the LAG group.
4. Dot1q mode MUST be set on the LAG Group (any dot1q settings on the individual interfaces will not be recognised by the templating logic).
    a. If dot1q mode is Access, an access VLAN SHOULD be chosen.
    b. If dot1q mode is Tagged, one or more tagged VLANs SHOULD be chosen, one native VLAN may be chosen if desired. If tagged VLANs are omitted, the resulting trunk port will allow all VLANs (same functionality as Tagged (All)).
    c. If dot1q mode is Tagged (All), a native VLAN may be chosen if desired.
5. The EOS Cli (raw_eos_cli) custom field may be used to provide a comma-separated list of additional EOS cli commands to render on the connection.

15.4.1 Yaml Connection Naming

The naming of endpoint connections in the resulting yaml files (rendered AVD data model) will use the following priority order:

1. Primarily the interface label will be used to construct the connection name. Be aware that connection names MUST be unique. Since it is not possible to connect a LAG group type interface to another interface/device in nautobot, use of the label to describe the connection is strongly recommended.
2. If the label is not used, the connection name will simply be constructed from the leaf name and the leaf interface name in order to have a unique connection name.

Note: If two identically named/numbered single-homed LACP connections are added to two switches in the same virtual chassis, using the nautobot GUI to choose which ports to add to which LAG group will be tricky since there will appear 2 LAG groups with the same name. In this case it is suggested to number the LAG groups differently for the 2 different switches, for example appending 01 to the LAG group belonging to the first switch in the virtual chassis and 02 to the second. The numbering scheme is completely arbitrary and up the the user, and is

only relevant from a GUI ease-of-use perspective. It will not affect port-channel numbering in AVD (eos_designs).

15.5 L3 Routed Interface Connection

This type of connection is a routed “no switchport” connection, typically used to connect to external routing devices. The following parameters are relevant to model this type of connection:

1. An IP address MUST be associated with the interface.
2. Only Ethernet interfaces will be recognised for L3 connections by the ansible templating logic.
3. The IP address MUST have a VRF assigned which in turn MUST be created according to the steps outlined in the VRFs section of this document.
4. The interface SHOULD have a description.
5. The EOS Cli (raw_eos_cli) custom field may be used to render a comma-separated list of EOS Cli commands on the connection.
6. The OSPF Enabled (ospf_enabled) custom field may be used to enable OSPF routing on the interface.

16 Tenant Routing (PE<->CE)

This section describes the steps necessary to enable dynamic routing between the EVPN Leaf device (PE) and a tenant routing device (CE), as well as defining tenant static routes. The protocol choices for dynamic routing are OSPF or BGP and the activation of these protocols are detailed below.

16.1 OSPF

This is the procedure to activate OSPF routing inside a tenant VRF:

1. On the tenant VRF, set the ospf_enabled custom field to True.
2. Create an L3 routed port or “peering” style SVI as outlined in sections 15.5 or 14.2/ 14.2.2 respectively.
3. Set the ospf_enabled custom field to True on the SVI or L3 Routed Port.
4. If an SVI is used for OSPF routing, remember to create some type of L2 endpoint connection for the CE device to use, as outlined in section(s) 15.1-4.

Note: Redistribution between OSPF and BGP is automatically set up in the tenant VRF in the background by AVD once these steps are completed in order to disseminate routing information originating from the CE device into the EVPN fabric.

Since OSPF adjacencies are discovered and set up dynamically by way of a hello-packet exchange, no representation of the CE device and its associated IP addresses need to be modeled in nautobot, although this is encouraged for documentation purposes.

16.1.1 OSPF Authentication

How to do this is TBD.

16.2 BGP

This is the procedure for setting up a BGP neighborship inside a tenant VRF. Because BGP neighborships are typically user-defined, setting up such a neighborship requires a representation of the CE device, its associated IP address and BGP ASN to be modeled in nautobot.

1. Create a L3 routed port or “peering” style SVI for the BGP peering relationship to use, as outlined in sections 15.5 or 14.2/ 14.2.2 respectively.
2. Create a representation of the CE device in nautobot.
    a. The device MUST have the bgp_peer device role.
    b. The device MUST have a bgp_asn associated with it.
    c. The device MUST have an IP address associated with one of its interfaces (physical, virtual, etc).
    d. The IP address MUST be associated with the relevant tenant VRF.
    e. The device IP address MUST be associated with the same IP Prefix object as the IP addresses of the relevant SVI/L3 interface which is used for peering.
    f. The device SHOULD have a site defined. In case the device is a Firewall cluster stretched over multiple sites or similar, the site may be omitted, but this will also require the SVI VLAN to be stretched over multiple sites and requires Secondary type IP addresses to be assigned to nodes in each location where peering is necessary.
    g. The route_map_in and route_map_out custom fields may optionally be populated with the names of route-maps for applying route policy.

16.2.1 Authentication

TBD

16.3 Static Routing

Static routes in tenant VRFs are modeled in the following way:

1. Create a **Prefix** object in nautobot, with the role static_route.
2. Associate the prefix with a tenant **VRF** object.
3. Associate the prefix with a **Tenant** object.
4. Populate the next_hop custom field with a gateway IP address to use.

5. Populate the description field if a description for the static route is desired.

Note: AVD will set up redistribution of static routes into BGP automatically when a static route is configured in a tenant VRF.

16.4 Defining Routing Policy

To define tenant routing policy, a custom configuration context is used and applied to one or more tenant object(s). The context MUST be tagged with the “avd” and “routing_policy” tags and contain valid JSON formatted data.

Since route-maps, prefix lists and other policy objects have no real association in EOS to a specific tenant or VRF, the contents of all routing_policy-tagged context objects are aggregated and rendered in the TENANTS files built by the templating logic. It is possible to define multiple context objects per tenant and create context objects that are shared across multiple tenants. If a policy object like a prefix list is to be used in routing policies for multiple tenants, they SHOULD be defined in a shared context object.

The routing policy context objects support the following dictionary keys/policy objects, any unsupported keys will simply be ignored:

- route_maps
- prefix_lists
- ipv6_prefix_lists
- community_lists
- ip_extcommunity_lists
- ip_extcommunity_lists_regexp

The value of each key must be a valid eos_cli_config_gen data model as outlined here: https://avd.sh/en/latest/roles/eos_cli_config_gen/index.html#filters

Example contents of a routing_policy config context object:

```
{
   "prefix_lists": {
       "BAR-PLIST": {
           "sequence_numbers": {
               "10": {
                   "action": "permit 15.0.0.0/8"
               },
               "20": {
                   "action": "permit 16.0.0.0/8 eq 24"
               },
               "30": {
```

```
                "action": "deny 17.0.0.0/8 le 32"
            }
        }
    },
    "FOO-PLIST": {
        "sequence_numbers": {
            "10": {
                "action": "permit 10.0.0.0/8 ge 12 le 24"
            },
            "20": {
                "action": "permit 11.0.0.0/8"
            },
            "30": {
                "action": "permit 12.0.0.0/8 le 32"
            },
            "40": {
                "action": "permit 13.0.0.0/8 eq 32"
            },
            "50": {
                "action": "deny 0.0.0.0/0 eq 32"
            }
        }
    }
},
"route_maps": {
    "RM-BAR": {
        "sequence_numbers": {
            "10": {
                "description": "permit some stuff",
                "match": [
                    "ip address prefix-list BAR-PLIST",
                    "source-protocol ospf"
                ],
                "set": [
                    "as-path prepend auto repeat 3"
                ],
                "type": "permit"
            },
            "20": {
                "match": [
                    "ip address prefix-list FOO-PLIST"
                ],
```

```
                    "type": "deny"
                }
            }
        },
        "RM-FOO": {
            "sequence_numbers": {
                "10": {
                    "description": "permit some stuff",
                    "match": [
                        "ip address prefix-list FOO-PLIST",
                        "source-protocol ospf"
                    ],
                    "set": [
                        "as-path prepend auto repeat 3"
                    ],
                    "type": "permit"
                },
                "20": {
                    "match": [
                        "ip address prefix-list BAR-PLIST"
                    ],
                    "type": "deny"
                }
            }
        }
    }
}
```

Note: To apply a routing policy to a fabric-external BGP peer, use the route_map_in and route_map_out custom fields as outlined in section 16.2.