Cisco APIC Layer 4 to Layer 7 Device Package Development Guide, Release 1.1(1j)

First Published: June 12, 2015

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

CONTENTS

Preface

This preface includes the following sections:

Audience

This guide is intended primarily for data center administrators with responsibilities and expertise in one or more of the following:

- Virtual machine installation and administration
- Layer 4 to Layer 7 Services installation and administration
- Switch and network administration

Document Conventions

Command descriptions use the following conventions:

Convention	Description
bold	Bold text indicates the commands and keywords that you enter literally as shown.
Italic	Italic text indicates arguments for which the user supplies the values.
[x]	Square brackets enclose an optional element (keyword or argument).

Convention	Description
[x | y]	Square brackets enclosing keywords or arguments separated by a vertical bar indicate an optional choice.
{x | y}	Braces enclosing keywords or arguments separated by a vertical bar indicate a required choice.
[x {y | z}]	Nested set of square brackets or braces indicate optional or required choices within optional or required elements. Braces and a vertical bar within square brackets indicate a required choice within an optional element.
`variable`	Indicates a variable for which you supply values, in context where italics cannot be used.
string	A nonquoted set of characters. Do not use quotation marks around the string or the string will include the quotation marks.

Examples use the following conventions:

Convention	Description
`screen font`	Terminal sessions and information the switch displays are in screen font.
`boldface screen font`	Information you must enter is in boldface screen font.
italic screen font	Arguments for which you supply values are in italic screen font.
< >	Nonprinting characters, such as passwords, are in angle brackets.
[ ]	Default responses to system prompts are in square brackets.
!, #	An exclamation point (!) or a pound sign (#) at the beginning of a line of code indicates a comment line.

This document uses the following conventions:

Note Means *reader take note*. Notes contain helpful suggestions or references to material not covered in the manual.

Caution Means *reader be careful*. In this situation, you might do something that could result in equipment damage or loss of data.

Warning IMPORTANT SAFETY INSTRUCTIONS

This warning symbol means danger. You are in a situation that could cause bodily injury. Before you work on any equipment, be aware of the hazards involved with electrical circuitry and be familiar with standard practices for preventing accidents. Use the statement number provided at the end of each warning to locate its translation in the translated safety warnings that accompanied this device.

SAVE THESE INSTRUCTIONS

Related Documentation

The Application Centric Infrastructure documentation set includes the following documents that are available on Cisco.com at the following URL: http://www.cisco.com/c/en/us/support/cloud-systems-management/application-policy-infrastructure-controller-apic/tsd-products-support-series-home.html.

Web-Based Documentation

- *Cisco APIC Management Information Model Reference*
- *Cisco APIC Online Help Reference*
- *Cisco APIC Python SDK Reference*
- *Cisco ACI Compatibility Tool*
- *Cisco ACI MIB Support List*

Downloadable Documentation

- *Knowledge Base Articles* (KB Articles) are available at the following URL: http://www.cisco.com/c/en/us/support/cloud-systems-management/application-policy-infrastructure-controller-apic/products-configuration-examples-list.html
- *Cisco Application Centric Infrastructure Controller Release Notes*
- *Cisco Application Centric Infrastructure Fundamentals Guide*
- *Cisco APIC Getting Started Guide*
- *Cisco ACI Virtualization Guide*
- *Cisco APIC REST API User Guide*
- *Cisco APIC Command Line Interface User Guide*
- *Cisco APIC Faults, Events, and System Messages Management Guide*
- *Cisco ACI System Messages Reference Guide*
- *Cisco APIC Layer 4 to Layer 7 Services Deployment Guide*
- *Cisco APIC Layer 4 to Layer 7 Device Package Development Guide*
- *Cisco APIC Layer 4 to Layer 7 Device Package Test Guide*
- *Cisco ACI Firmware Management Guide*

- *Cisco ACI Troubleshooting Guide*
- *Cisco ACI Switch Command Reference, NX-OS Release 11.0*
- *Verified Scalability Guide for Cisco ACI*
- *Cisco ACI MIB Quick Reference*
- *Cisco Nexus CLI to Cisco APIC Mapping Guide*
- *Application Centric Infrastructure Fabric Hardware Installation Guide*
- *Cisco NX-OS Release Notes for Cisco Nexus 9000 Series ACI-Mode Switches*
- *Nexus 9000 Series ACI Mode Licensing Guide*
- *Cisco Nexus 9332PQ ACI-Mode Switch Hardware Installation Guide*
- *Cisco Nexus 9336PQ ACI-Mode Switch Hardware Installation Guide*
- *Cisco Nexus 9372PX ACI-Mode Switch Hardware Installation Guide*
- *Cisco Nexus 9372TX ACI-Mode Switch Hardware Installation Guide*
- *Cisco Nexus 9396PX ACI-Mode Switch Hardware Installation Guide*
- *Cisco Nexus 9396TX ACI-Mode Switch Hardware Installation Guide*
- *Cisco Nexus 93128TX ACI-Mode Switch Hardware Installation Guide*
- *Cisco Nexus 9504 ACI-Mode Switch Hardware Installation Guide*
- *Cisco Nexus 9508 ACI-Mode Switch Hardware Installation Guide*
- *Cisco Nexus 9516 ACI-Mode Switch Hardware Installation Guide*

Cisco Application Centric Infrastructure (ACI) Simulator Documentation

The following Cisco ACI Simulator documentation is available at http://www.cisco.com/c/en/us/support/cloud-systems-management/application-centric-infrastructure-simulator/tsd-products-support-series-home.html.

- *Cisco ACI Simulator Release Notes*
- *Cisco ACI Simulator Installation Guide*
- *Cisco ACI Simulator Getting Started Guide*

Cisco Nexus 9000 Series Switches Documentation

The Cisco Nexus 9000 Series Switches documentation is available at http://www.cisco.com/c/en/us/support/switches/nexus-9000-series-switches/tsd-products-support-series-home.html.

Cisco Application Virtual Switch Documentation

The Cisco Application Virtual Switch (AVS) documentation is available at http://www.cisco.com/c/en/us/support/switches/application-virtual-switch/tsd-products-support-series-home.html.

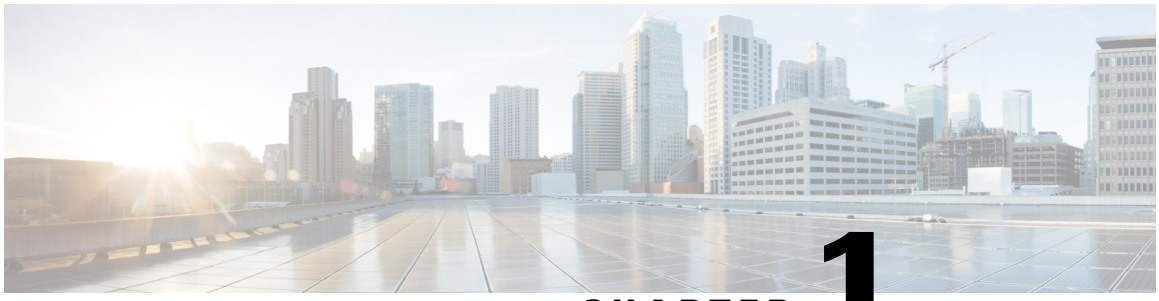
Documentation Feedback

To provide technical feedback on this document, or to report an error or omission, please send your comments to apic-docfeedback@cisco.com. We appreciate your feedback.

Obtaining Documentation and Submitting a Service Request

For information on obtaining documentation, using the Cisco Bug Search Tool (BST), submitting a service request, and gathering additional information, see *What's New in Cisco Product Documentation* at: http://www.cisco.com/c/en/us/td/docs/general/whatsnew/whatsnew.html

Subscribe to *What's New in Cisco Product Documentation*, which lists all new and revised Cisco technical documentation as an RSS feed and delivers content directly to your desktop using a reader application. The RSS feeds are a free service.



CHAPTER 1

Overview

About Service Integration with the Application Policy Infrastructure Controller

The Application Policy Infrastructure Controller (APIC) automates the insertion and provisioning of network services, such as Secure Sockets Layer (SSL) offload, server load balancing (SLB), Web Application Firewalls (WAFs), and traditional firewalls. The network services are rendered by service appliances, such as Application Delivery Controllers (ADCs) and firewalls. A service appliance can perform one or more service function.

The APIC enables you to define a service graph. Each node in the service graph represents a network function. A graph defines set of service functions that are based on user-defined policies. A service appliance (device) performs a service function within the graph. One or more service appliances can render the services that are required by a graph. One or more service functions can be performed by a single service device.

The APIC requires a device package that you can use to insert and configure network service functions on a network service appliance (device). A device package is a zip file that contains the following:

- `DeviceModel.xml`—The device package must contain a single XML file called `DeviceModel.xml` that is the device specification. The device specification is an XML file that provides a hierarchical description of the device, including the configuration of each function, and is mapped to a set of managed objects on the APIC. The device specification defines the following:
  - Device functions
  - Parameters that are required by the device to configure each function
  - Interfaces and network connectivity information for each function

- `DeviceScript.py`—The device package must contain a single Python file called `DeviceScript.py`. You should define the APIs for interfacing between the APIC and the device in this Python file. The device specification XML file associates the device script to this Python file.

  The device script manages communication between the APIC and the device. It defines the mapping between APIC events and the function calls representing device interactions, converting calls from a generic API to device-specific calls.

  When you upload a device package to the APIC, the APIC creates a hierarchy of managed objects representing the device and validates the device script interface.

- Additional files and directories that contain Python or text files. The device package can include any supporting Python libraries for interfacing and configuring the device. The supporting Python files can be split across multiple directories. The package can also include any supporting text files. A device package can contain supporting Python egg files.

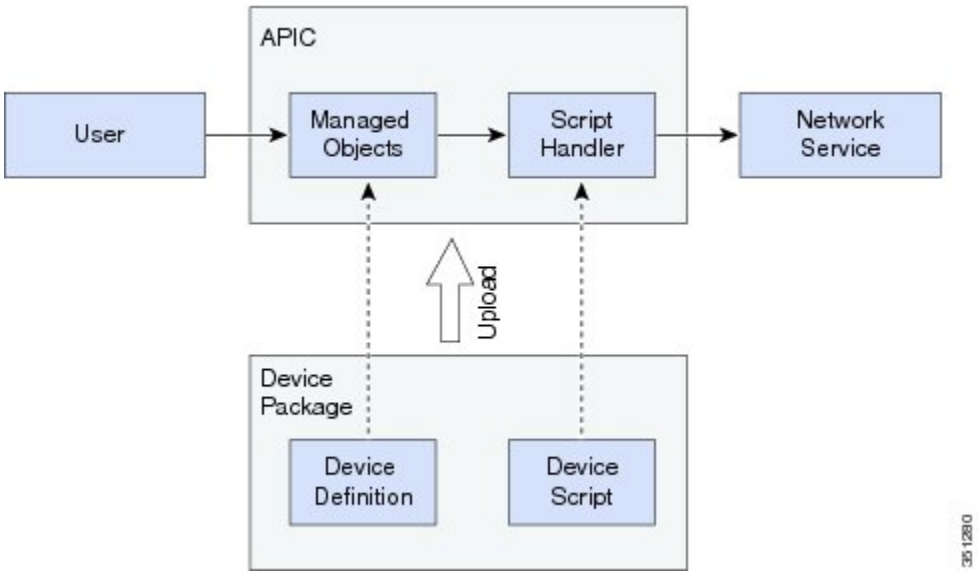
- `images` directory—The device package must contain the `images` directory, and the directory must contain a single file named *`vendor_name`*`.gif`. The image size must be 28 pixels x 28 pixels.

The following example shows a listing of a package zip file from the vendor named Insieme:

```
bash-4.1$ unzip -l insiemeDevicePackage.zip
Archive:  insiemeDevicePackage.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
   309597  03-17-2014 17:39   DeviceModel.xml
     1597  03-17-2014 17:39   DeviceScript.py
        0  01-30-2014 15:36   common/
     1919  02-06-2014 11:35   common/deviceInterface.py
        0  01-30-2014 15:36   feature/
    21919  02-06-2014 11:35   feature/functionCommon.py
     6485  10-31-2013 06:32   feature/function2.py
     7747  10-31-2013 06:32   feature/function1.py
        0  10-31-2013 06:32   feature/__init__.py
        0  01-30-2014 15:36   lib/
     1919  02-06-2014 11:35   lib/
        0  01-30-2014 15:36   util/
    21919  02-06-2014 11:35   util/logging.py
        0  10-31-2013 06:32   parser/configParser.py
        0  02-12-2014 10:07   images/
     1380  02-12-2014 10:07   images/insieme.gif
```

The following figure describes the relationship between a device package and the APIC.

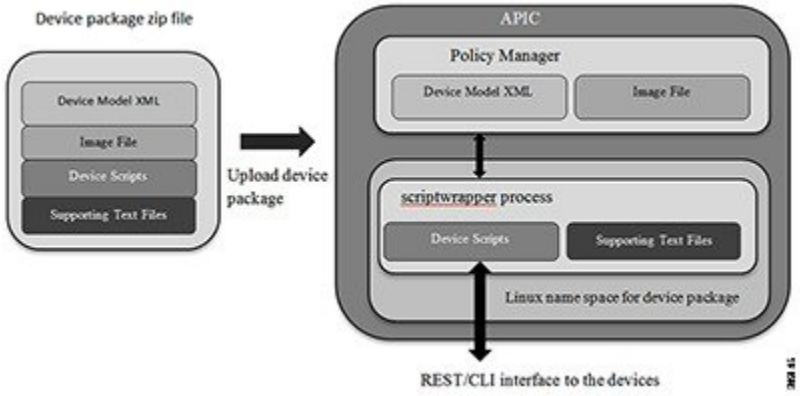
Figure 1: APIC Package Upload



About the Device Package Architecture

The following figure shows the Application Policy Infrastructure Controller (APIC) service automation and insertion architecture through the device package.

Figure 2: Device Package Architecture



When you upload a device package through the GUI or northbound APIC interface, the APIC creates a namespace for each unique device package. The content of the device package is unzipped and copied to the namespace. The file structure created for a device package namespace is as follows:

```
root@apic1:/# ls
bin  dbin  dev  etc  fwk  install images lib  lib64  logs  pipe  sbin  tmp  usr util
```

```
root@apic1:/install# ls
DeviceScript.py DeviceSpecification.xml feature common images  lib  util.py
```

The contents of the device package are copied under the `install` directory.

The APIC parses the device model. The managed objects that are defined in the XML file are added to the APIC's managed object tree that is maintained by the Policy Manager.

The Python scripts that are defined in the device package are launched within a script wrapper process in the name space. The access to the file system is restricted. Python scripts can create temporary files under `/tmp` and can access any text files that were bundled as part of the device package. However, you should not create Python scripts that create or store any persistent data in a file.

The logs are written to two files: the debug.log and the periodic.log. Any configuration API event logs are written to the debug.log and any periodic poll API logs are written to the periodic.log. The logging framework is similar to the python logging framework.

The log files are accessible by logging in to the APIC as the fabric administrator. The log files are located in `/data/devicescript/<vendorname-model-pkgversion>/logs`.

Multiple versions of a device package with different major version numbers can coexist on the APIC, because each device package version runs in its own namespace. You can select a specific version for managing a set of devices.

About the Debug Logs

The Application Policy Infrastructure Controller (APIC) maintains log files that you can use to debug a device script. The log files are saved in the following directories:

Directory	Log Files
`/data/devicescript`	`debug.log` and `periodic.log`
`/var/log/dme`	• DME logs—requires administrator privileges to view. • Core files—requires root privileges to use backtrace to check the process stack of a core file. • `svc_ifc_*.log`—requires administrator privileges to view. You need to view these log files only in the event of an issue with the APIC. For more information about exporting log files, see the *Cisco ACI Troubleshooting Guide*.

You must have administrator privileges to access these directories.

About IPv6 Support

You can use the IPv6 communications protocol instead of the IPv4 protocol. Enabling IPv6 does not impact the APIs between the Application Policy Infrastructure Controller (APIC) and Layer 4 to Layer 7 service

device package. You must model configuring IPv6 features through the APIC and enable IPv6 data path on the devices. The dictionary format for conveying the IPv6 configuration is identical to the IPv4 dictionary format.

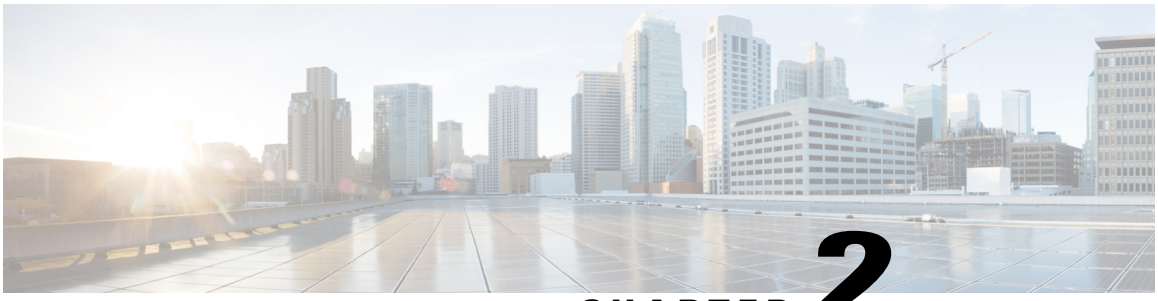
Note The management connectivity between the device and the APIC continues to be IPv4. Devices are still managed through IPv4 network. IPv6 is only enabled for user data traffic. IPv6 for management connectivity to the device is not supported.

About Route Peering

The Application Policy Infrastructure Controller (APIC) supports the following protocols between the fabric and the Layer 4 to Layer 7 service devices:

- Open Shortest Path First
- Open Shortest Path First v3 (IPv6)
- Border Gateway Protocol (IPv4 and IPv6)

The APIC provides native support for configuring Open Shortest Path First (OSPF) and Border Gateway Protocol (BGP) parameters. You do not need to model the OSPF and BGP configuration in your device model. The OSPF and BGP configuration is passed to the device script in a service API callout as part of the configuration dictionary along with other function configurations.



CHAPTER 2

Developing Device Specifications

About Device Types

The Application Policy Infrastructure Controller (APIC) classifies network service devices into two types:

- GoTo—Represents any device that is Layer 3 (L3) attached. The packet is delivered to a GoTo device because either the destination MAC or destination IP within the packet identifies the device. Typically, Application Delivery Controllers (ADCs) or L3 firewalls represent a GoTo device.
- GoThrough—Represents any transparent device. The destination MAC or destination IP address is not addressed to the device, but the packet is steered through the device due to VLAN stitching. Typically, Layer 2 (L2) firewall or Intrusion Detection System (IDS) devices represent a GoThrough device. The end stations that exchange packets are not aware of the presence of a GoThrough (transparent device) within the path.

The APIC further classifies device instances registered with an APIC into two categories:

- Concrete device—Represented by vnsCDev, which identifies an instance of a service device. A concrete device can be physical or virtual. A concrete device has its own management IP address to configure and monitor through the APIC.
- Logical device—Represented by vnsLDevVip. vnsLDevVip identifies a cluster of one or more concrete devices. A logical device is addressed and managed through a management IP address that is assigned to the cluster. The service functions offered by the service device are always rendered on a logical device. Typically, a logical device represents a cluster of devices deployed in active-active mode or active-standby high availability mode. If you deploy a device in standalone mode, the logical device contains only one

concrete device. The management IP address for logical devices and concrete devices will be the same. All service operations are always done on a logical device instance.

For information about registering a device with an APIC, see the *Cisco APIC Layer 4 to Layer 7 Services Deployment Guide*.

A service device can be single-context or multi-context. A multi-context device supports multiple routing domains, which means that the device supports overlapping IP addresses to be configured across different routing contexts.

A single-context device must be registered to a specific tenant. A single-context device cannot be shared by multiple tenants. A multi-context device can be registered under a common tenant and can be shared by multiple tenants.

About Device Specifications

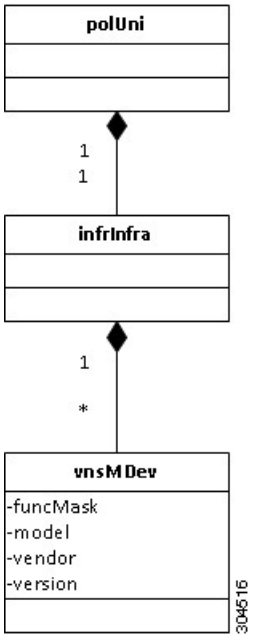
The configuration of the Application Policy Infrastructure Controller (APIC) is represented by an object model that consists of a large number of managed objects (MOs). A device type is defined by a tree of managed objects that have Meta Device (MDev) at the root. The device specification XML file extends the APIC's managed object model by defining a new MDev object.

A device specification file must define a Meta Device (`vnsMDev`) object. The `vnsMDev` object contains metadata that describes vendor-specific information, such as the vendor name, device package version, device version supported, device script binding, and device model describing that functions and parameters that are required to realize these functions on the device.

Each unique major version of a device package results in the creation of one instance of a `vnsMDev` object instance with the APIC Policy Manager. The APIC can support many instances of the `vnsMDev` object. The `vnsMDev` object is contained within an infra-policy (represented by infraInfra) under the APIC global policy.

The global policy is the universe of policies, which is represented as polUni. The following figure describes the relations of `vnsMDev` to the APIC's managed object hierarchy.

Figure 3: Relations of vnsMDev to the APIC's Managed Object Hierarchy



The device model is contained by a `vnsMDev` object. The device specification file must have the following structure:

```
<poliUni>
   <infraInfra>
      <vnsMDev>
         <!-- device Sepcification-->
      </vnsMdev>
   </infraInfra>
</poliUni>
```

`vnsMDev` must have the following attributes:

- vendor—Identifies the device package vendor.
- model—Identifies the device models that are managed by the device specification.
- version—Identifies the device package version, which is also referred to in the document as the major version. You can upload and use one or more versions of a device package on the APIC. The APIC allows you to select a device package to be used for managing a device instance that is registered with the APIC.

  The device package version is incremented when major structural changes are made to the device model and properties of existing device objects are modified or existing objects are deleted or when the device package is updated to manage later revisions of the device. You must increment a minor version for any bug fixes or minor enhancements that are made or additional that objects are augmented to the device package.
- funcMask—Indicates whether a device package can support service functions deployed in GoTo or GoThrough mode. A device package can support both the GoTo and the GoThrough mode of service

insertion. If both modes are supported, define funcMask as a comma-separated list in the following format:

`GoTo,GoThrough`

A service function on a device can be deployed as GoTo or GoThrough only when a device package supports such a configuration. Typically, funcMask for firewall device packages supports both the GoTo mode and the GoThrough mode to allow firewalls to be deployed in routed or transparent bridge mode.

The following example shows the `vnsMDev` attributes:

```
<vnsMDev vendor="Insieme"
  model="NetworkService"
  version="1.0"
  funcMask="GoTo,GoThrough">
```

The `vnsMDev` object instance is identified by the <vendor-model-version> string. The APIC creates a `vnsMDev` instance for each unique <vendor-model-version> string.

The device model is divided into following parts:

- Generic Part—Defines generic information about the device. It consists of the following objects:
  - Device Credentials
  - Interface Labels
  - Device Profiles
- Cluster and Device Configuration Part—Defines any cluster or device specific configuration. It consists of the following objects:
  - Cluster Configuration
  - Device Configuration
- Functional Part—Describes the service functions and its configuration. The configuration is divided under the following objects:
  - Global Functional Device Configuration
  - Group Configuration
  - Function Configuration

Device Script

The device script information is defined through the `vnsDevScript` object. The device Script object associates the python file defining Application Policy Infrastructure Controller (APIC) APIs. The APIC calls these python APIs to instantiate any service functions defined by the device package.

The device script object contains following attributes:

Attribute	Type	Description
`ctrlrVersion`	String	Identifies controller API version compatibility. It is a string whose value must match the APIC API version . The currently accepted values are "1.0" and "1.1". A device package that supports route peering with Cisco Application Centric Infrastructure (ACI) fabric must set the controller version to "1.1". If route peering is not supported, the device package can set the controller version to "1.0" to allow the package to work with all APIC releases. A device package with controller version set to "1.1" will not work with the "1.0" controller version. A device package with the controller version "1.0" will work with all APIC releases with controller version "1.0" and higher.
`minorversion`	String (512 characters)	Identifies the minor version of the device package. The device package developers should use this version string to track any revisions that are made to the device script or model without making structural changes to existing objects in the device model.
`versionExpr`	String (512 characters)	APIC passes this `versionExpr` string to the script during a `deviceValidate`() call. The device package developer defines any string (it can be regular expression) to indicate device versions that this device package can support.

The `minorversion` string provides a non-disruptive upgrade of a device package. If only the device scripts have changed, the device package developer must update only the minor version string. When only the `minorversion` has changed and the device package version has not been incremented, the APIC restarts the scripts associated with the package with the new set of files bundled in the device package. The Managed Object Model is refreshed with the new objects defined in the device model specified in the device package. This enables efficient upgrade of the script without triggering re-rendering of the graphs that use the device package.

Devices Credentials

The devices credentials object allows vendors to specify the type of credentials that the Application Policy Infrastructure Controller (APIC) passes to the device script for authentication while communicating with the device. Currently, only the username and password-based authentication is supported. The device specification file must define the following object:

```
<vnsMCred name=”username” key=”username”/>
<vnsMCredSecret name=”password” key=”password”/>
```

The device specification file must define only one instance of `vnsMCred` and `vnsMCredSecret`. During the device registration, you provide a value for the username and password object. For more information, see the *Cisco APIC Layer 4 to Layer 7 Services Deployment Guide* .

Interface Labels

Interfaces on the device must be labeled in an abstract way. A function associates with these interfaces to represent a logical flow of packets through the service function. For example, a firewall device could label the interfaces as trusted, untrusted, cluster, and management interfaces. Packets that are received from an untrusted interface could be directed through the firewall function and emitted out of a trusted interface. As another example, a device could label its interface as an external, internal, HA, and management interface. A load balancing function could receive packet from an external interface and load balance to a pool through an internal interface. A single physical interface (or vNIC in case of virtual service) can be assigned one or more labels. The labels are assigned to the interfaces on a device at the time of registering logical and concrete devices. You can assign multiple labels to a single interface for single arm deployment. The device models must specify labels for its interfaces. The labels are defined using the `vnsMIfLbl` object type.

The following example defines the labels:

```
<vnsMIfLbl name="external" shortName="ext"/>
<vnsMIfLbl name="internal" shortName="int"/>
<vnsMIfLbl name="management" shortName="mgmt" />
```

The `vnsMIfLbl` object must contain the name attribute and shortName attribute. The short name must be four characters or less. The device specification can define one or more types of the `vnsMIfLbl` object.

Vendor Device Profile

The vendor device profile (`vnsDevProf`) is a new object in the Layer 4 - Layer 7 management information tree. This object allows a vendor to add device model information to the Application Policy Infrastructure Controller (APIC). Vendors provide it as part of device package or provide it separately. `vnsDevProf` is contained within `vnsMDev`. A `vnsMDev` can have one or more `vnsDevProf`. `vnsDevProf` contains information pertaining to a specific device model, its interface and other properties. The APIC GUI uses `vnsDevProf` to provide users the option to select a model while registering concrete devices with the APIC. `vnsDevProf` provides an ease of use enhancement to the APIC GUI experience . `vnsDevProf` simplifies the device registration process and reduces user error when specifying physical interface name and other parameters during registering with the APIC and forming a logical cluster.

The APIC also uses `vnsDevProf` to update a device package after it has been uploaded. `vnsDevProf` can be augmented by a tenant administrator. Vendors define a new `vnsDevProf` and make it available independently of the device package in order to support new profile information such as chassis, model or IO module. Or, tenant administrators define their own device profile and use it for registering devices.

The `vnsDevProf` object has the following attributes:

Attribute	Mandatory	Description
name	Yes	Uniquely identifies the object. Each object name must have a unique value within the containing object. The name can contain only alphanumeric characters, '_' or '-'. The name cannot contain any other characters. The APIC uses the name to lookup a specific object within a containing object. The name size is limited to a maximum of 512 characters. The name attribute identifies a specific device model supported by the device package. For example: • ASA5585-S20K-X9 • ASA558-S60P60SK9
type	Yes	Specifies whether the device type is pyhsical or virtual. The values are: • PHYSICAL • VIRTUAL
context	Yes	Specifies if the device is context-aware (supports multiple contexts on the same logical cluster). For example, it has support for multiple routing domains and supports unique configuration for each user on the same logical device cluster. The values for this attribute can be: • single-context (default) • multiple-contect

Attribute	Mandatory	Description
pcPrefix	No	Provides a prefix that identifies the logical interface created by link aggregation (with or without the LACP protocol). The GUI uses the `pcPrefix` as a prefix when a user selects a link bundle (Port-channel or Etherchannel) as the device interface while registering a device with the APIC. A device package developer defines one `pcPrefix` for a given `vnsDevProf`. The following are `pcPrefix` examples: • `pcPrefix='Port-Channel'` • `pcPrefix='LA'` • `pcPrefix='Etherchannel'`

Vendor Device Interface Name

The `vnsDevInt` is a new object in the Layer 4 to Layer 7 management information tree. The `vnsDevInt` object describes an interface name on a given chassis. The Application Policy Infrastructure Controller (APIC) GUI uses the `vnsDevInt` information provided by the user during device registration. Users map a logical interface name to one of the `vnsDevInt` found on the device. The APIC GUI provides a drop down list based on `vnsDevInt` contained in the `vnsDevProf`. Users select one of the interfaces while associating a logical interface with a physical interface.

Note Users are not limited to the interfaces defined under `vnsDevProf`. Users can select the 'other' option in the APIC GUI and provide any arbitrary string as the interface name. `vnsDevInt` should have the list of all supported interface names.

The `vnsDevInt` object has the following attributes:

Attribute	Mandatory	Description
name	Yes	Uniquely identifies the object. Each object name must have a unique value within the containing `vnsDevProf` object . The name cannot contain ' '. The name size is limited to a maximum of 512 characters. For example: • eth1.1 • 1.1 • 1/1 • Gig0/1/1 • Tunnel0 • Ehternet0 • Eth0
mgmtOnly	No	Specifies whether the device type is whether the device is reserved for management access. The values are: • yes • no

About Cluster and Device Configurations

The Application Policy Infrastructure Controller (APIC) allows devices to be deployed in standalone, High-Available Active-Standby mode or as a Cluster in Active-Active mode. The cluster and device configuration section allows vendors to specify any configuration that applies to the cluster or a specific node within a cluster irrespective of the HA mode. Cluster and device specification is not mandatory.

Cluster Configurations

The device specification file can define just one cluster configuration object referred to as `vnsClusterCfg`. The cluster configuration contains the configuration for an entire cluster. The configuration that applies to a cluster is represented by one or more objects of type `vnsMParam` that can be further grouped logically under one or more `vnsMFolder` objects.

You can instantiate parameters and folders defined under the cluster configuration for a logical device registered with an Application Policy Infrastructure Controller (APIC). The configuration defined under a cluster

configuration is passed to the device script only during a `clusterModify`() or `clusterAudit`() call. The configuration defined under a cluster cannot be referenced by a service function. The cluster configuration is not passed to the scripts during a `serviceModify`(), `serviceAudit`(), `serviceHealth`(), or `serviceCounters`() API call.

`vnsClusterCfg` can contain one or more `vnsMFeature` objects. The `vnsMFeature` object allows logical grouping of cluster configurations. Folders are grouped based on the `dispFeature` attribute defined under folder. The Application Policy Infrastructure Controller (APIC) GUI uses the `vnsMFeature` object to order and group the folders for user input.

A device package developer should define any cluster level configuration within a `vnsClusterCfg` object. For example, a cluster configuration can include a Network Time Protocol (NTP) server configuration and the syslog server IP address.

The following example shows a cluster configuration:

```
<vnsClusterCfg name="ClusterConfig">
  <vnsMFolder key="SyslogConfig">
    <vnsMParam key="ipaddress"
      description="Syslog Server IP address"
      dType="str"
      validation="isIPAddress"/>
  </vnsMFolder>

  <vnsMFolder key="NTPConfig">
    <vnsMParam key="ipaddress"
      description="NTP Server IP address"
      dType="str"
      validation="isIPAddress"/>
  </vnsMFolder>
</vnsClusterCfg>
```

Routing Capability

The device model can indicate which routing protocols can be configured through the device package. The routing protocol configuration capability can be defined by using the `vnsRoutingCfg` object. This object is contained within `ClusterCfg`.

The `vnsRoutingCfg` object has the following attributes:

Attribute	Mandatory	Description
supportedProtocols	Yes	Specifies a set of routing protocols that are supported by the device package. The value is a comma-separated list of the supported protocols. The values are: • ospf • ospfv3 • bgp • bgpv6 The list must not contain any whitespace.

The absence of the vnsRoutingCfg object in the device package indicates that the device package does not support any routing protocols. The route peering configuration is pushed to a device package only when routing protocol support is explicitly indicated by the vnsRoutingCfg object.

The Application Policy Infrastructure Controller (APIC) raises a fault when you try to configure a protocol that is not supported by the device.

The following example shows a routing protocol configuration:

```
<vnsClusterCfg name="ClusterConfig">
  <vnsRoutingCfg supportedProtocols="ospf,ospfv3,bgp,bgpv6"/>
    ...
</vnsClusterCfg>
```

Device Configurations

The device specification file can contain one instance of vnsDevCfg that contains a device-specific configuration. The vnsDevCfg is contained within a vnsClusterCfg. The device-specific configuration is represented by one or more vnsMParam, which can be further grouped under one or more vnsMFolder.

The configuration that is defined under a device configuration is instantiated by the user during concrete device registration within a logical device. The device configuration is passed to the device scripts only during the deviceAudit(), deviceModify(), deviceHealth(), and deviceCounters() calls. The device configuration cannot be referenced from a service function, during the clusterModify() call, or during the clusterAudit() call.

A device configuration can contain a configuration such as the HA mode on the device, the peer IP address for cluster, or the port-channel (LACP) configuration that must be pushed to a specific device within a cluster.

vnsDevCfg can contain one or more vnsMFeature objects. The vnsMFeature object allows logical grouping of cluster configurations. Folders are grouped based on the dispFeature attribute defined under folder. The Application Policy Infrastructure Controller (APIC) GUI uses the vnsMFeature object to order and group the folders for user input.

The following example shows a device configuration:

```
<vnsClusterCfg name="ClusterConfig">
  <vnsDevCfg name="DevCfg">
    <vnsMFolder key="HighAvailabilityCfg" cardinality="n">
      <vnsMParam key="peerIP"
        description="HA Pair peer IP address"
        dType="str"
        validation="isIPAddress"/>
    </vnsMFolder>
  </vnsDevCfg>
</vnsClusterCfg>
```

About Functional Configurations

A device package and a device can support many service functions. Typically, any function that transforms and influences packet forwarding on the device can be represented as a service function. For example, SSL offload, VPN, server load balancing, and web application filtering can be modeled as functions that are supported by the device. One or more such functions can be modeled in the device specification file.

The functions are represented by a `vnsMFunc` object. The `vnsMFunc` object has a name attribute. Each function that is defined within the device package must have a unique name. The name is used to look up a function that is defined under an instance of an MDev.

The `vnsMFunc` object must contain the following object:

- vnsMConn

The parameters that are required to render a specific service function can be defined under the following categories:

- Function
- Group
- Device global

The following example shows the structure of a function configuration:

```
<poliUni>
   <infraInfra>
      <vnsMDev>
        <!-- Generic Part -->

        <!-- Device Credentials -->
        <vnsMCred name=”username” key=”username”/>
        <vnsMCredSecret name=”password” key=”password”/>

        <!-- Interface Labels -->
        <vnsMIfLbl name="external" shortName="ext"/>

        <!-- Device Profiles -->

        <!-- Cluster Configuration -->
        <vnsClusterCfg name="ClusterCfg">

          <!-- Device Configuration -->
          <vnsDevCfg name="DeviceConfig">

          </vnsDevCfg>
        </vnsClusterCfg>

        <!-- Functional Configuration -->

        <!-- Global Functional Device Configuration -->
        <vnsMDevCfg>
        </vnsMDevCfg>

        <!-- Group Configuration -->
        <vnsGrpCfg>
        </vnsGrpCfg>

        <!—Function configuration: Could be one or more such configuration -->
        <vnsMFunc>
        </vnsMFunc>
      </vnsMdev>
   </infraInfra>
</poliUni>
```

Connector Objects

A function must have at least one connector object: `vnsMConn`. The connector object is used to link one or more functions to form a service graph. If a function is a transit function, it must have at least two connectors. If a function is a stub function, such as a collector, it can have just one connector. Typically, only IDS devices

that are in passive mode and are capturing packets that are copied to the device have just one connector defined for the capture function. All other functions, such as a firewall, load balancers, and SSL offload, have two or more connectors. Currently, the Application Policy Infrastructure Controller (APIC) supports a maximum of two connectors per function, which means that you can define an input and output connector for any transit function.

The connector has the following attributes:

Attribute	Mandatory	Description
name	Yes	Specifies the name of the connector. Every connector within a function must have a unique name.
encType	Yes	Specifies the connector encapsulation type. This attribute is the encapsulation that is used for traffic on the connector and is specified as a value of `vlan` or `vxlan`. The value specifies whether the packet is sent encapsulated from the network to the device VLAN or VXLAN encapsulated. On a virtual device, the encapsulation might be removed by the virtual switch and the VLAN or VXLAN encapsulation header might not be seen by the virtual service device. Currently, the APIC supports only VLAN encapsulation.
dir	No	Specifies the connector direction. This direction can be specified as either `input` or `output`.
cardinality	No	If a function supports multiple instances of a given connector type, the device model can specify this explicitly by setting the cardinality to `n`. By default, the cardinality is `1`.

Attribute	Mandatory	Description
notification	No	Deprecated in controller version 1.1. The APIC still supports this attribute in device packages for backward compatibility. Allows endpoint or network attach/detach notifications to be generated for the function. This attribute is used to determine whether the APIC calls the device script when an endpoint or subnet association changes for an endpoint group (EPG) that is attached directly or indirectly to this connector. The notification can take the following values: • `none` • `subnet` • `endpoint` If the notification attribute is not specified, it defaults to `none`, which means that the APIC will not attach nor detach the network or endpoint APIs. This attribute is type enum. Device packages can either allow subnet or endpoint notification. The check 'epNotifications' attribute was added in controller version 1.1 to allow both notifications on a connector.

Attribute	Mandatory	Description
epNotifications	No	Allows endpoint or network attach/detach notifications to be generated for the function. This attribute is used to determine whether the APIC calls the device script when an endpoint or subnet association changes for an endpoint group (EPG) that is attached directly or indirectly to this connector. The notification can take the following values: • `none` • `subnet` • `endpoint` If the notification attribute is not specified, it defaults to `none`, which means that the APIC will not attach nor detach the network or endpoint APIs. This attribute is of type bitmask. The attribute allows a device package to allow endpoint, subnet, or both subnet and endpoint notifications. This attribute is supported only in controller version 1.1 or later.

A connector must contain just one `vnsRsInterface` object. This object associates a connector to a specific interface type that is identified by the labels that are defined by using `vnsMIfLbl`. The APIC uses this relation to pass the specific interface information while rendering the service function. For more information, see Fabric Connectivity, on page 97.

Images

The device package must contain the `images` directory, and the directory must contain a single file named *`vendor_name`*`.gif`. The image size must be 28 pixels x 28 pixels.

The following example shows a listing of a package zip file from the vendor named Insieme:

```
bash-4.1$ unzip -l insiemeDevicePackage.zip
Archive:  insiemeDevicePackage.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
   309597  03-17-2014 17:39   DeviceModel.xml
     1597  03-17-2014 17:39   DeviceScript.py
        0  01-30-2014 15:36   common/
```

```
 1919  02-06-2014 11:35   common/deviceInterface.py
    0  01-30-2014 15:36   feature/
21919  02-06-2014 11:35   feature/functionCommon.py
 6485  10-31-2013 06:32   feature/function2.py
 7747  10-31-2013 06:32   feature/function1.py
    0  10-31-2013 06:32   feature/__init__.py
    0  01-30-2014 15:36   lib/
 1919  02-06-2014 11:35   lib/
    0  01-30-2014 15:36   util/
21919  02-06-2014 11:35   util/logging.py
    0  10-31-2013 06:32   parser/configParser.py
    0  02-12-2014 10:07   images/
 1380  02-12-2014 10:07   images/insieme.gif
```

Function Configurations

The `vnsMFunc` object identifies a specific function on a device that can be managed through the device package. Each `vnsMFunc` defined in the device package must be assigned a unique name. A device package developer can also define a `dispLabel` attribute for a `vnsMFunc` object. The `dispLabel` is a 512 character string. It allows a device package developer to provide a more user friendly name for the function. When a `dispLabel` attribute is defined for a `vnsMFunc`, the APIC GUI displays the `dispLabel` string instead of the name attribute. Device package developers must provide a user friendly name for the functions exposed through the device package.

A device package developer defines parameters that are required to configure a service function under a function object. Any parameter that is defined under `vnsMFunc` is scoped under a specific function. The parameters that are defined under a function can be further grouped logically under one or more folders.

The parameter and folders defined under a function persist if the instance of the function persists. The APIC deletes the parameters and folders that are defined under a function when the function instance is deleted.

The parameter and folders under a function cannot be shared or referenced by any other function within the same graph or a different graph that is rendered on the same device. The parameter and folders defined under the function must have a unique instance on the device for each function instance. The scope of the parameter and folders the are being limited within a functions context is similar to a local variable in the C language.

The following example defines the parameters of a service function:

```
<vnsMFunc name="SLB">
    <vnsMConn name="external"
      dir="input"
      encType="vlan"
      epNotifications="endpoint">
        <vnsRsInterface tDn="uni/infra/mDev-Insieme-SampleDevice-1.0/mIfLbl-external"/>
    </vnsMConn>

    <vnsMConn name="internal"
      dir="output"
      encType="vlan"
      epNotifications="endpoint">
        <vnsRsInterface tDn="uni/infra/mDev-Insieme-SampleDevice-1.0/mIfLbl-internal"/>
    </vnsMConn>

    <vnsMFolder key="VServer"
      scopedBy="epg">
        <vnsMParam key="vservername"
          description="Name of VServer"
          mandatory="true"
          dType="str"
          validation="isAlpha"/>
        <vnsMParam key="port"
          description="Port for Virtual server"
          validation="isL4Port"/>
        <vnsMParam key="persistencetype"
          description="persistencetype"/>
```

```
        <vnsMParam key="servicename"
          description="Service bound to this vServer"/>
        <vnsMParam key="servicetype"
          description="Service bound to this vServer"
          dType="str"
          validation="isProtocol"/>
        <vnsMParam key="clttimeout"
          description="Client timeout"/>
    </vnsMFolder>
</vnsMFunc>
```

Group Configurations

Any parameter and folders that are defined under a group configuration can be shared across multiple functions in a graph. A device package developer can define parameters and folders that can be shared across multiple functions that are rendered on a single device within a single graph under a group configuration.

The parameters and folders within a group configuration are scoped under a graph instance. Any function within a graph instance can share and reference the configuration.

Objects defined under a group configuration persists as long as the graph instance persists. The Application Policy Infrastructure Controller (APIC) deletes the parameter and folder defined under a group configuration when the graph instance is deleted. Any parameter that is defined under a group configuration must have a unique instance per graph on a device; a parameter must not be shared or referenced by any other graph instance that is rendered on the same device.

The group configuration is represented by the `vnsGrpCfg` object. Only one definition of `vnsGrpCfg` can be under `vnsMDev`. All group parameters and folders that are scoped under a group must be contained within a `vnsGrpCfg` object.

Parameters and folders that are defined under a group configuration are similar to static variables in the C language. The variables persist beyond a function.

Global Function Configurations

Any parameter and folders defined under an `vnsMDev` configuration can be shared across multiple functions across multiple graphs. A device package developer can define parameters and folders that can be shared across multiple functions across multiple graphs that are rendered on a single device under `vnsMDevCfg`.

Objects defined under a `vnsMDev` configuration persist if there is at least one graph instance refers to the parameter or the folder. The Application Policy Infrastructure Controller (APIC) deletes the parameter and folder that is defined under a `vnsMDev` configuration when all functions across all graph instances are deleted from a specific device.

On a multi-context device, the global configuration must have a unique instance per context. The parameters and folders that are defined under `vnsMDev` must not be shared across multiple contexts.

The parameter and folders that are defined under a `vnsMDev` configuration are similar to global variables in the C language.

Typically, network attributes, such as an IP address configured on an interface, routes, and subnets, have a global scope. The encapsulation tags that are allocated by the APIC are globally scoped, which allows multiple parallel functions to be deployed on the same network across multiple graphs.

Relations

A service function can reference a particular parameter or a folder that is defined under a group or `vnsMDevCfg`, which allows the function to use an instance of a parameter or a folder that is defined under a group or a device scope. The relation to a folder is defined using the `vnsMRel` object. A `vnsMRel` object can exist only within a `vnsMFolder` object. A folder can have one or more relations objects defined.

The `vnsMRel` object has the following attributes:

Attribute	Mandatory	Description
key	Yes	Identifies the object. Each object key must have a unique value within the containing object. The key can contain only alphanumeric characters, '_', or '-'. A key cannot contain any other characters. The Application Policy Infrastructure Controller (APIC) uses the key to look up a specific object within a containing object. The key size is limited to a maximum of 512 characters.
Description	Yes	Holds the description of this configuration item. The description field is used by the APIC GUI to provide help to the user. A device package developer should provide an accurate description and intent of the relation. The description field size is limited to a maximum of 512 characters.
mandatory	No	Indicates whether this relation is mandatory. This property is a Boolean value (yes or no). By default, a relation is not mandatory unless explicitly specified, meaning that the user is not required to specify a relations object. The given relation is not necessary to render a function on the device.

Attribute	Mandatory	Description
cardinality	No	Specifies the number of occurrences of this relation. By default, only one instance of a relation is permitted under the contained object. If a user is allowed to instantiate more than one instance of the relation object, the device specification file should define the relation with `cardinality="n"`.
dispLabel	No	This is a 512 character string. If this attribute is specified in the model, the APIC GUI will display a string defined by `dispLabel` instead of the key. A device package developer provides a user friendly name for the folder.

The `vnsMRel` object contains a `vnsRsTarget` object that identifies the object to which a relation is referring. The target is a fully qualified key of the object that is defined in the device specification file. The `vnsMRel` object can contain only one instance of a `vnsRsTarget` object.

The following example defines a relations object:

```
<vnsMRel key="ServerConfig">
   <vnsRsTarget tDn="uni/infra/mDev-Insieme-SampleDevice-1.0/mDevCfg/mFolder-Server"/>
</vnsMRel>
```

The above example indicates that the `ServerConfig` that is defined within a function has a relation to an instance of a server folder that is defined under `vnsMDevCfg`. You can instantiate a relation by specifying the target folder instance name qualified by a full path under a device configuration. When a service function is rendered on a device, the APIC looks for a specific instance of the folder that is referred to by the relations. If the APIC finds a matching instance, it includes the folder in the configuration dictionary that is passed in the service API call. The APIC also passes an instance of relations as part of the function configuration dictionary. For an example of a configuration dictionary that is passed in the API, see Developing Device Scripts, on page 51.

Parameter Scope and API Configuration Dictionary

Any parameter and folders that are defined under vnsMDevCfg, vnsGrpCfg, or vnsMFunc are passed to the device script only during the serviceAudit(), serviceModify(), serviceHealth(), and serviceCounters() function calls. The parameters and folders that are defined in a vnsMDevCfg object are passed in a service API call only if there is a service function with a relations object that refers to that parameter and folder.

About Parameter Objects and Folders

The cluster, device, and functional configuration is defined by one or more `vnsMParam` objects. These objects can be grouped logically under one or more folders that are represented as the `vnsMFolder` object.

Parameter Objects

The configuration parameters are represented by the `vnsMParam` object type. A device package can have one or more `vnsMParam` objects. A parameter object contains the following attributes:

Attribute	Mandatory	Description
key	Yes	Specifies the key for the meta parameter. This property uniquely identifies the parameter. Each parameter key must have a unique value within the containing object. The key can contain only alphanumeric characters, "_", or "-". The key cannot contain any other characters. The Application Policy Infrastructure Controller (APIC) uses the key to look up a specific object within a containing object, which is typically the `vnsMFolder` object. The key size is limited to a maximum of 512 characters.
Description	Yes	Holds the description of this configuration item. The description field is used by the APIC GUI to provide help to the user. The device package developer should provide an accurate description and intent of the parameter. The description field size is limited to a maximum of 512 characters.
mandatory	No	Indicates whether this parameter is mandatory. This property is a Boolean value (yes or no). By default, a parameter is not mandatory unless explicitly specified.

Attribute	Mandatory	Description
dType	No	Specifies the data type for this parameter. It can take following values: • `int` • `real` • `str` If the dType is not specified, the parameter defaults to `int`.
validation	No	Specifies the validation expression to be used by the APIC for validating a value for this parameter. The validation string cannot exceed 255 characters. The dType need not be `str` if validation is specified. The validation string refers to a composite or a comparison object name. For more information, see Parameter Validation, on page 35.
cardinality	No	Specifies the number of occurrences of this parameter. By default, only one instance of a parameter is permitted under the contained object. If a user is allowed to instantiate more than one instance of the parameter object, the device specification file should define the parameter with `cardinality="n"`. For example, if you can instantiate multiple static routes on a device that has a parameter object called route, set the cardinality of the route parameter to `cardinality="n"`.

Attribute	Mandatory	Description
dispLabel	No	This is a 512 character string. If this attribute is specified in the model, the APIC GUI will display a string defined by `dispLabel` instead of the key. A device package developer provides a user friendly name for the folder. For example, the configuration for a server IP address can be labeled as `dispLabel = "Server IP Address"` while the key is `srvIpAddr`. The GUI displays "Server IP Address" as the name for the parameter instead of "srvIpAddr."

The following example defines a parameter object:

```
 <vnsMParam key="vservername"
  description="Name of VServer"
  mandatory="true"
  dType="str"
  validation="isAlpha"/>

<vnsMParam key="subnetipaddress"
  description="Subnet IPAddress of the Device"
  dType="str"
  cardinality=”n”
  validation="isIPAddress"/>

<vnsMParam dispLabel="Network Mask"
    key="netmask"
    dType="str"
    mandatory="true"/>

<vnsMParam dispLabel="Default Gateway"
    key="gateway"
    dType="str"
    mandatory="true"/>
```

Folders

The configuration parameters can be logically grouped under folders. A folder can contain one or more folders and parameters. A folder is represented by the `vnsMFolder` object and has the following attributes:

Attribute	Mandatory	Description
key	Yes	Specifies the key for the meta folder. This property uniquely identifies the folder. Each folder key must have a unique value within the containing object. The key can contain only alphanumeric characters, '_', or '-'. The key cannot contain any other characters. The Application Policy Infrastructure Controller (APIC) uses the key to look up a specific object within a containing object. The key size is limited to a maximum of 512 characters. The key for the top most folder defined under `vnsMDevCfg`, `vnsGrpCfg` or `vnsMfunc` must be unique within the device package.
Description	Yes	Holds the description of this configuration item. The description field is used by the APIC GUI to provide help to the user. The device package developer should provide an accurate description and intent of the folder. The description field size is limited to a maximum of 512 characters.

Attribute	Mandatory	Description
scopedBy	No	Specifies the scope for this configuration folder. This attribute specifies where in the Management Information Tree (MIT) to look for the value of this folder when instantiating a function. The APIC resolves the value by looking up an instance that is defined under different objects to which a graph is associated. The scopedBy attribute can contain the following values: • tenant—The folder can be instantiated only under a tenant. • ap—The folder can be instantiated only under an application profile or tenant. • bd—The folder can be instantiated only under a bd or tenant. • epg—The folder can be instantiated only under an endpoint group (EPG), bridge domain, application profile, or tenant. • none A device package developer can limit the resolution to a higher level. By default, scopedBy is defined as "none", which means that the device package does not impose any restriction on where a particular folder can be instantiated. The APIC user can define an instance of the folder under a tenant, application profile, bridge domain, or EPG. **Note** The current version supports only `scopedby` EPG.

Attribute	Mandatory	Description
cardinality	No	Specifies the number of occurrences of this folder. By default, only one instance of a folder is permitted under the contained object. If the user is allowed to instantiate more than one instance of the folder object, the device specification file should define the folder with `cardinality="n"`.
dispLabel	No	This is a 512 character string. If this attribute is specified in the model, the APIC GUI will display a string defined by `dispLabel` instead of the key. A device package developer provides a user friendly name for the folder. For example, the configuration for syslog can be grouped under a folder with `dispLabel = "Syslog Server Configuration"` while the key is `syslogSrvCfg`. The GUI displays "Syslog Server Configuration" as the name for the folder instead of "syslogSrvCfg."

Attribute	Mandatory	Description
dispFeature	No	

Attribute	Mandatory	Description
		This is a 512-character string. This attribute allows the grouping of multiple folders based on a feature. A given feature, such as network, might require multiple parameters. These parameters can be further grouped in one or more folders. A set of folders can define a feature configuration. This attribute defines which feature requires this folder. The APIC GUI matches the feature name specified fro this attribute with the `vnsFeature` to identify under which feature this folder should be displayed. This attribute takes a comma-separated list of feature names. The folder can be included for one or more features. For example, the `LBMonitor` folder can be included under the "LoadBalancing" and "ContentSwitching" features. Match the feature names specified for this attribute with the `vnsFeature` name that is defined under the function, `vnsDevCfg`, or `vnsClusterVfg`. The APIC GUI groups the folders based on the `vnsFeature` name. If the feature name specified in the `dispFeature` attribute matches a `vnsMFeature`, the GUI will show this folder under that specific feature. If the name does not match any `vnsMFeature`, the GUI will default to display this folder under the "All" feature tab. If the dispFeature attribute is not defined, the GUI will display the folder the "All" feature tab. In the example that follows this table, the GUI displays the `dispFeature="LoadBalancing,`

Attribute	Mandatory	Description
		SSLOffload", folder under both Load Balancing and SSL Offload. For more examples, see Managed Object Example for v1.1, on page 45.

The following example defines a folder object:

```
<vnsMFolder key="Server"
  scopedBy="epg">
  <vnsMParam key="servername"
    description="Server Name"
    dType="str"
    validation="isAlpha"/>
  <vnsMParam key="domain"
    description="Domain name of the server"/>
  <vnsMParam key="ipaddress"
    description="Server IP address"
    dType="str"
    validation="isIPAddress"/>
</vnsMFolder>
```

Features

The vnsMFeature object is a new object in the Layer 4 to Layer 7 management information tree. The vnsMFeature object allows logical grouping of folders based on a feature. This object along with dispFeature allows one or more folders to be grouped for configuring a specific feature. The Application Policy Infrastructure Controller (APIC) GUI uses this object to determine a set of features that can be configured on a device cluster or a function supported by the cluster. The vnsMFeature object and has the following attributes:

Attribute	Mandatory	Description
name	Yes	Uniquely identifies the object. Each object name must have a unique value within the containing object. The name can contain only alphanumeric characters, '_' or '-'. The name cannot contain any other characters. The APIC uses the name to lookup a specific object within a containing object. The name size is limited to a maximum of 512 characters.

Attribute	Mandatory	Description
dispOrder	Yes	Specifies the order in which the `vnsMFeature` object is arranged within the parent object. Each instance of the `vnsMFeature` object in the parent object should have a unique `dispOrder`. This object is a string of numeric characters. The APIC GUI uses the numeric value for ordering the feature tabs on the screen. The features are ordered in ascending order.

Parameter Validation

The Application Policy Infrastructure Controller (APIC) can do parameter validation by using the `vnsComparison` and `vnsComposite` objects. A device package developer can define and associate validation to any string type parameter by using either basic or composite comparisons.

The basic comparisons (`vnsComparison`) object can perform the following operations:

- Equal—`eq` (the default)
- Not equal—`ne`
- Less than—`lt`
- Greater than—`gt`
- Greater than or equal to—`ge`
- Less than or equal to—`le`
- Match—`match` (requires a regular expression)

The comparison object `vnsComparison` is defined under the `vnsMDev`, `vnsMFunc`, `vnsMFolder`, `vnsMParam`, or `vnsComposite` objects. The `vnsComparison` object has the following attributes:

Attribute	Mandatory	Description
name	Yes	Holds the name of the comparison assertion. **Note** The name field allows only alphanumeric characters. The maximum length for this field is 16 characters; and you cannot use special characters.

Attribute	Mandatory	Description
cmp	Yes	Defines the comparison operator: • `eq`—Equal, which is the default. • `ne`—Not equal. • `lt`—Less than. • `gt`—Greater than. • `ge`—Greater than or equal to. • `le`—Less than or equal to. • `match`—Match. The `match` comparison requires a regular expression.

In the following example, the parameter validates IP addresses using a regular expression match:

```
<vnsMParam key="vipaddress"
  description="VIP IPAddress"
  dType="str"
  validation="isIPAddress"/>

<vnsComparison name="isIPAddress"
  cmp="match"
  value="([01]?\d\d?|2[0-4]\d|25[0-5])\.([01]?\d\d?|2[0-4]\d|25[0-5])\.
    ([01]?\d\d?|2[0-4]\d|25[0-5])\.([01]?\d\d?|2[0-4]\d|25[0-5])"/>
```

The composite comparison (`vnsComposite`) object provides the following types of comparisons to be performed:

- All match (the default)—Validation passes when the parameter value matches all of the comparison objects that are defined by the composite object.
- Any match—Validation passes when a parameter value matches one of the comparison objects that is defined within the composite object.
- Exactly one match—Validation passes when a parameter value matches one of the comparison objects.

The composite object can contain one or more `vnsComparison` objects. A composite object can be defined under `vnsMDev`, `vnsMFunc`, `vnsMFolder`, or `vnsMParam`. A `vnsComposite` object has the following attributes:

Attribute	Mandatory	Description
name	Yes	Holds the name of the composite.

Attribute	Mandatory	Description
cmp	Yes	Defines the type of comparison to be performed. It takes the following values: • `and`—All comparison strings that are contained within the composite must match for the validation to return as a success. The `and` type is the default comparison. • `or`—Any comparison string that is contained within the composite can match for the validation to return as a success. • `one`—Only one comparison string contained within the composite can match for the validation to return as a success. This operator enables the package developer to define a mutual exclusion.

In the following example, the element defines a match with any of the contained values:

```
<vnsComposite name="isProtocol" comp="or">
    <vnsComparison name="ip" cmp="eq" value="IP" />
    <vnsComparison name="tcp" cmp="eq" value="TCP" />
    <vnsComparison name="udp" cmp="eq" value="UDP" />
    <vnsComparison name="http" cmp="eq" value="HTTP" />
 </vnsComposite>

 <vnsComposite name="yesNo" comp="one">
     <vnsComparison name="yes" cmp="eq" value="YES" />
     <vnsComparison name="No" cmp="eq" value="NO" />
 </vnsComposite>
```

Faults Codes

The device specification file can define fault codes with help strings that describe the nature of a fault and possible corrective action. When a device script encounters an issue with rendering a function due to a parameter or folder, the script can return a specific fault code with a path of the object that had an issue. The Application Policy Infrastructure Controller (APIC) refers to the fault code that is defined in the device specification file and picks the description and corrective action that is described while displaying the fault. Defining a fault code provides a description of the reason for the fault and the corrective action that the user can take to resolve the fault.

The fault codes are defined under a `vnsMDfcts` object. A device specification can have one instance of `vnsMDfcts` under `vnsMDev`. A `vnsMDfcts` object can contain one or more fault codes that are described by the `vnsMDfct` object.

The vnsMDfct object contains the following attributes:

Attribute	Mandatory	Description
code	Yes	Specifies a unit 16 value that identifies a unique defect.
Description	Yes	Describes the defect. The description field is used by the APIC GUI to provide help to the user. The device package developer should provide an accurate description. The field size is limited to a maximum of 512 characters.
htmlFile	No	Specifies the URL link to online help that can help a user understand and correct the issue. The field size is limited to a maximum of 512 characters.
recAct	Yes	Specifies the recommended action. This field is used by the APIC GUI to provide the recommended action for the user to take. The device package developer should provide an accurate recommended action to resolve the defect. The field size is limited to a maximum of 512 characters.

The following example defines a fault object:

```
<vnsMDfcts>
    <vnsMDfct code="100"
      recAct="Configure a Netmask for the vipaddress"
      descr="VIP requires vipaddress and NetMask"/>
    <vnsMDfct code="200"
      recAct="Configure a relation to VIP Folder"
      descr="A function should have a valid relations to a VIP folder that is
        specifying the VIP Address and Netmask"/>
</vnsMDfcts>
```

Function Profile

The APIC requires a device package developer to define a function profile within a device model. A function profile is a template for one or more functions suitable for a specific application. A function profile is the

equivalent of defining an abstract graph within a device package with meaningful defaults for a function that defines the graph. The user can leverage the built-in function profile by referencing the built-in function profile in the device package at the time of defining a service graph. Function profiles reduce the number of parameters that a user has to provide to instantiate a service function for a specific application. A device package developer must include as many function profiles as applicable.

The APIC GUI wizard for configuring service graphs requires function profiles. It expects the user to associate a function profile to a function while defining a graph template. If a device package does not define a function profiles, the user will not be able to use the APIC GUI service deployment wizard. Overall user experience suffers as a result. Define at least one function profile for each function type defined in the device package. Users can further clone and customize these function profiles.

Following is an example of defining function profile in a device package:

```
<vnsAbsFuncProfContr name = "FunctionProfiles">

          <vnsAbsFuncProfGrp name = "Function Profiles for Service graph
            for an Application 1 ">
              <vnsAbsFuncProf name = "Function 1 Name">
                  <vnsRsProfToMFunc
                    tDn="uni/infra/mDev-<vendor-model-version>/mFunc-function1"/>

                  <vnsAbsDevCfg>
                      <vnsAbsFolder key="Folder_Key"
                        name="<Folder_Key>-Default" scopedBy="epg">
                          <vnsAbsParam name="Param Instance name"
                            "key="Param Name" value="Value"/>
                          ...
                      </vnsAbsFolder>
                      ...

                  </vnsAbsDevCfg>

                  <vnsAbsFuncCfg>

                       <vnsAbsFolder key="Folder_Key"
                         name="<Folder_Key>-Default" scopedBy="epg">
                           <vnsAbsCfgRel key="relation_key"
                             name="rel name" targetName="targetValue"/>
                       </vnsAbsFolder>
                      ...
                  <vnsAbsFuncCfg>

              </vnsAbsFuncProf>

              <vnsAbsFuncProf name = "Function 2 Name">
                  <vnsRsProfToMFunc
                    tDn="uni/infra/mDev-<vendor-model-version>/mFunc-function2"/>
                  ...
              <vnsAbsFuncProf>
          </vnsAbsFuncProfGrp>

          <vnsAbsFuncProfGrp name = "Function Profiles for
            Service graph for an Application 2">
              ...
          </vnsAbsFuncProfGrp>
</vnsAbsFuncProfContr>
```

The function profile definition is contained within `vnsAbsFuncProfContr`. The profile for each unique application is identified by `vnsAbsFuncProfContr`. The `vnsAbsFuncProfGrp` name should be intuitive to relate to an application for which the template is being defined. For example, if the function profile is for a load balancing function for a web application, the `vnsAbsFuncProfGrp` should be named "Web Application Virtual Server".

A function profile identified by `vnsAbsFuncProfContr` can contain one or more functions as applicable. If the graph requires the chaining of multiple functions on the same device, the profile could define defaults for

these functions within the `vnsAbsFuncProfContr`. Each function configuration within the profile is contained within `vnsAbsFuncProf`.

Each `vnsAbsFuncProf` has one relation to a function defined by the device model. The relation identifies type of function being instantiated by the function profile. The relation to the function is defined by object `vnsRsProfToMFunc` contained within `vnsAbsFuncProf`. The `vnsRsProfToMFunc` has a tDn attribute identifying a function with a fully qualified name of the function object. The example shows a sample tDn for identifying a function within a device model.

The mechanism to configure parameters for these functions are identical to creating a service graph on the APIC. The parameter, relations, and folders in a function profile can be an instance of the parameters, relations and folders defined under `vnsMDevCfg`, `vnsGrpCfg`, and `vnsFuncCfg`.

Note The names of the folder in the function profile must be folder key appended with the following string: `-Default`

For example, if the folder key has a value of "`Network`", then the folder instance will have a value of "`Network-Default`".

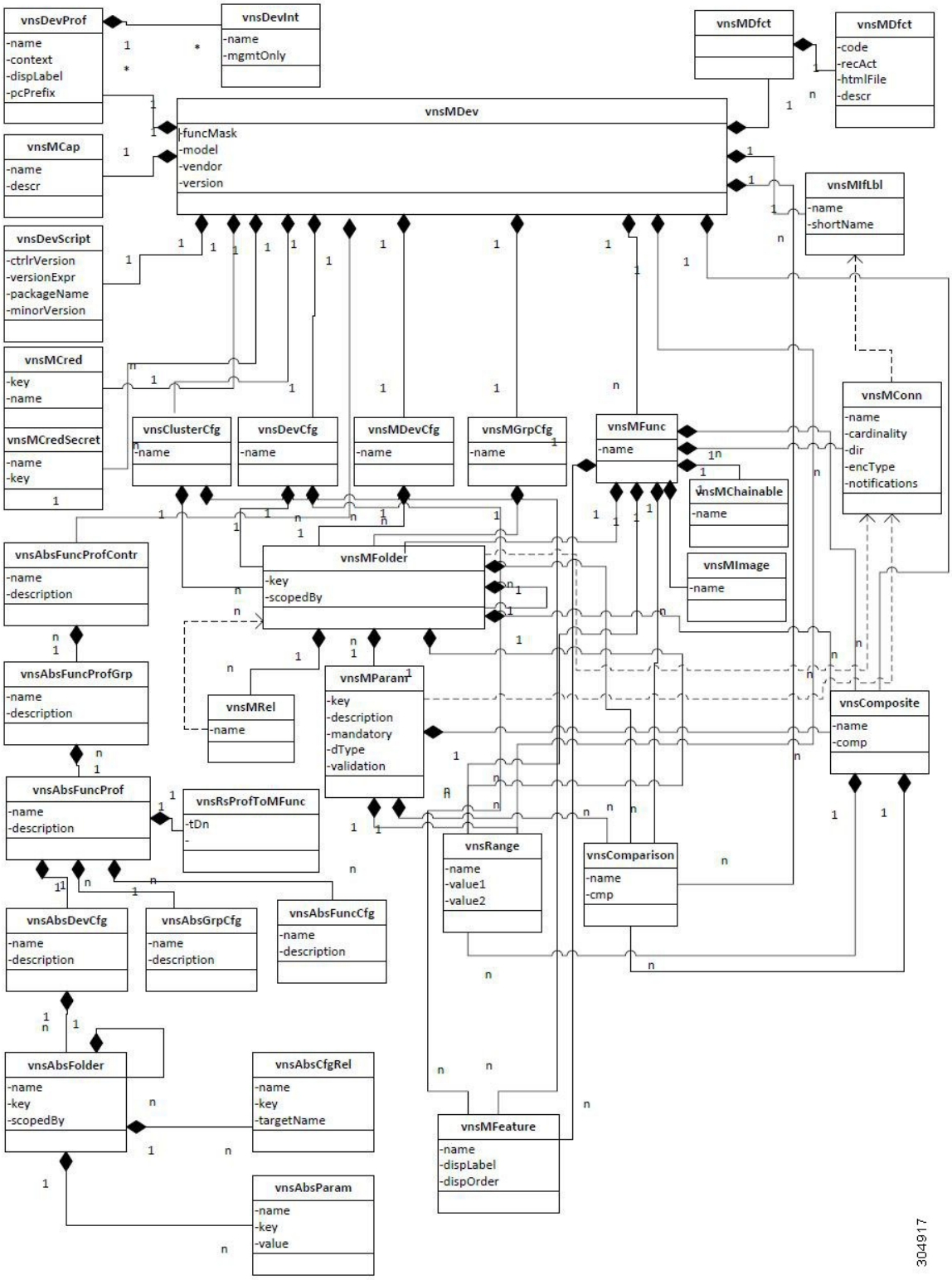
A function profile does not allow instantiating multiple instances of a folder with a cardinality value of "`n`". Only one instance can be defined within the profile.

For information about creating a service graph through the northbound API, see the *Cisco APIC Layer 4 to Layer 7 Services Deployment Guide* .

Managed Object Model

The following figure shows the object model for representing a device.

Figure 4: Managed Object Model



The following table describes the objects in the object model.

Component	Description
vnsMDev	Contains definitions of the metadata for a service device type. The metadata contains vendor-specific data, including the vendor name, device model, and device version. The service devices are categorized as GoTo and GoThrough devices. A device is a GoTo device if the packet is addressed to the device's MAC address or IP address. A device is considered as a GoThrough device if a packet transits through the device by in-path insertion and the packet is not addressed to the device's MAC address or IP address. A firewall in transparent mode is an example of a GoThrough device. A device package and device specification model could support devices in both GoTo and GoThrough mode. By default, the device specification is assumed to represent devices in GoTo mode. The device specification file can be changed to support both modes or the GoThrough mode only by using the following attribute: `funcMask: "GoTo,GoThrough"`
vnsMCred	Represents the credentials necessary to authenticate a user into the device. For example, `key` is used for key-based authentication schemes. This model details the meta-information for such key-based authentication of credentials.
vnsDevScript	Represents a device script handler. This managed object contains meta-information about the script handler's related attributes, including its name, package name, and version.
vnsClusterCfg	Contains the cluster configuration folders and parameters. The cluster configuration affects the functionality of the device cluster independent of graphs rendered on the device cluster.
vnsDevCfg	Contains device-specific configuration folders and parameters. The device configuration affects the functionality of a specific device within a cluster independent of the graphs rendered on the device cluster.
vnsMCredSecret	Contains the password for logging into a service device.
vnsMDevCfg	Represents the base level device configuration. This object serves as an anchor to differentiate between different device configurations and the shared configuration (`MGrpCfg`). The configuration under `MDevCfg` can be shared across multiple instances of a function across multiple graphs.
vnMGrpCfg	Represents the meta-group configuration. It contains the part of the configuration that can be shared across multiple functions in a graph. A configuration under a group configuration is scoped within a graph instance and cannot be referred to by another graph.
vnsMFolder	Represents meta-folder information. The model uses a generic configuration that consists of `MFolders` and `MParams`. This object allows the configuration to be specified as a hierarchy.
vnsMParam	Enables a configuration to be specified as a hierarchy. The metadata within this model consists of a key, a type (integer, string), and other attributes that are related to parameters.

Component	Description
vnsMRel	Represents a meta-relation to another object. It allows the referencing of another folder or parameter.
vnsMFunc	Contains the metadata for a single function on a device. A function contains a set of connectors and a function-specific configuration tree. This managed object contains the metadata for all such operations.
vnsMConn	Represents a connector between logical functions. The metadata includes the cardinality, direction, and encapsulation type (VXLAN or VLAN) for the given connection.
vnsMIfLbl	Represents an interface label. Interfaces can be labeled in an abstract way on devices. For example, a firewall device can implement trusted, untrusted, and management interfaces. The concrete models specify how many labels that a device supports.
vnsMChainable	Identifies the function names on a device that can immediately follow the parent function. This managed object contains the function names that can be chained together.
vnsAbsFuncProfContr	A Function profile group container. Defines a collection of function profile groups (graphs) for a specific application. Each function profile group can contain one or more functions initialized with certain default parameters for a specific application. A Function profile container can be defined within a device model by a device package developer or can be defined by the tenant to provide a catalog of graphs for a set of applications.
vnsAbsFuncProfGrp	Represents a function profile group. A collection of functions initialized with default parameters for a specific application. A function profile group can be defined within a device package by a device package developer or it can be defined by an APIC tenant as a catalog of graph for a specific applications.
vnsAbsFuncProf	Represents a function profile. It contains vnsAbsDevCfg (an instance of vnsMDevCfg), vnsAbsGrpCfg (an instance of vnsMGrpCfg) and vnsAbsFuncCfg (an instance of vnsMFunc). A function profile is linked to a specific function defined in the device model. A function profile can be defined within a device package or can be defined by an APIC tenant as a catalog of function within a vnsAbsFuncProfGrp.
vnsDevProf	Identifies a device model and associated attributes. It defines whether a device model is type virtual or physical, whether it supports multiple contexts, and so on. This object is primarily used to simplify device registration through the APIC.
vnsDevInt	Allows device package vendors to define acceptable interface names for a given device profile.

Component	Description
vnsMFeature	Represents a a list of features applicable to the function, device or cluster configuration. The APIC allows device package developers to group the folders based on features. A given folder may be part of one or more features. Based on the APIC GUI uses the vnsMFeature to display a subset of fulders while configuring a function, device, or cluster.

Managed Object Example for v1.1

The following XML file contains a sample managed object configuration, including the dispLabel, dispFeatrue, and vnsMFeature objects. You can use a similar XML file to instantiate a network device on the APIC.

```
<polUni>
    <infraInfra>
        <vnsMDev vendor="Insieme"
          model="NetworkService"
          version="1.0"
          funcMask="GoTo,GoThrough">

            <!-- Associate a device script that defines APIs required by APIC script
                 Engine -->
            <vnsDevScript name="InsiemeNetworkService"
              packageName="DeviceScript.py"
              versionExpr="1.0"
              ctrlrVersion="1.0"
              minorversion="01"/>

            <!-- Define inteface labels for logical interface -->
            <vnsMIfLbl name="external"/>
            <vnsMIfLbl name="internal"/>
            <vnsMIfLbl name="mgmt"/>

            <!-- Describe device models and interface names allowed on the model -->
            <vnsDevProf  name = "N9k" type = "PHYSICAL" context="multi-Context"
              pcPrefix="Port-channel">
                <vnsDevInt  name="eth1_0" mgmtOnly="yes"/>
                <vnsDevInt  name="eth1_1"/>
                <vnsDevInt  name="eth1_2"/>
                <vnsDevInt  name="eth1_3"/>
                <vnsDevInt  name="eth1_4"/>
                <vnsDevInt  name="eth1_5"/>
            </vnsDevProf>

            <vnsDevProf  name = "N9kv" type = "VIRTUAL" pcPrefix="Port-channel">
                <vnsDevInt  name="eth1_0" mgmtOnly="yes"/>
                <vnsDevInt  name="eth1_2"/>
                <vnsDevInt  name="eth1_3"/>
                <vnsDevInt  name="eth1_4"/>
                <vnsDevInt  name="eth1_5"/>
                <vnsDevInt  name="eth1_6"/>
            </vnsDevProf>

            <vnsMCred name="username" key="username"/>
            <vnsMCredSecret name="password" key="password"/>

            <vnsComparison name="enable" cmp="match" value="^enable$"/>
            <vnsComparison name="enableDisable" cmp="match" value="^(enable|disable)$"/>
            <vnsComparison name="trueFalse" cmp="match" value="^(true|false)$"/>
            <vnsComparison name="macAddress" cmp="match"
              value="^([0-9a-fA-F]{1,4}.){2}[0-9a-fA-F]{1,4}$"/>
            <vnsComparison name="ipv4Addr" cmp="match"
              value="^(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.
```

```
    (25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.
    (25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.
    (25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$"/>
<vnsComparison name="netmask" cmp="match"
  value="^((25[0-5]|2[0-4][0-9]|1[0-9]{2}|[0-9]{1,2})\.)
    {3}(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[0-9]{1,2})$"/>
<vnsComparison name="hexKey" cmp="match" value="^[0-9a-fA-F]{32}$"/>
<vnsComparison name="str38" cmp="match" value="^\S{1,38}$"/>
<vnsComparison name="str128" cmp="match" value="^\S{1,128}$"/>
<vnsComparison name="any46" cmp="match" value="^any[46]?$"/>
<vnsComposite name="domainName" comp="and">
    <vnsComparison name="dn_len" cmp="match" value="^.{1,63}$"/>
    <vnsComparison name="dn_str" cmp="match"
      value="^[a-zA-Z0-9-]+(\.[a-zA-Z0-9-]+)*$"/>
</vnsComposite>

<vnsComposite name="permitDeny" comp="or">
    <vnsComparison name="permit" cmp="eq" value="permit"/>
    <vnsComparison name="deny" cmp="eq" value="deny"/>
</vnsComposite>

<vnsMDfcts>
    <vnsMDfct code="10"
      descr="Configuration error"
      recAct="Fix the configuration error and retry.">
        <vnsRsDfctToCat tDn="dfctCats/dfctCat-major"/>
    </vnsMDfct>
    <vnsMDfct code="20"
      descr="Connection error"
      recAct="Check the device IP address and network connectivity.">
        <vnsRsDfctToCat tDn="dfctCats/dfctCat-major"/>
    </vnsMDfct>
    <vnsMDfct code="30"
      descr="Unexpected error"
      recAct="Report this error to Insieme.">
         <vnsRsDfctToCat tDn="dfctCats/dfctCat-critical"/>
    </vnsMDfct>
    <vnsMDfct code="40"
      descr="Unsupported device version"
      recAct="Upgrade to a device version that is supported by this Device
        Package.">
        <vnsRsDfctToCat tDn="dfctCats/dfctCat-major"/>
    </vnsMDfct>
    <vnsMDfct code="50"
      descr="device is busy with a previous configuration"
      recAct="Retry the operation after waiting for a short while.">
        <vnsRsDfctToCat tDn="dfctCats/dfctCat-warning"/>
    </vnsMDfct>
</vnsMDfcts>

<vnsClusterCfg name="ClusterConfig">
    <vnsMFeature name="License" dispOrder="0"/>

    <vnsDevCfg name="DeviceConfig">
        <vnsMFeature name="HighAvailability" dispOrder="0"/>
        <vnsMFolder dispFeature="HighAvailability"
          dispLabel="Failover Settings" key="HighAvailability">
          <vnsMParam dispLabel="Peer IP Address" key="ipaddress" dType="str"/>

            <vnsMParam dispLabel="Peer NetMask" key="netmask" dType="str"/>
            <vnsMParam dispLabel="Peer Unit ID" key="id" mandatory="true"/>
        </vnsMFolder>
    </vnsDevCfg>

    <vnsMFolder dispFeature="License" dispLabel="Licensed Features"
      key="enableFeature">
        <vnsMParam dispLabel="L4 Load Balancing" key="LBV4" dType="str"/>
        <vnsMParam dispLabel="L7 Load Balancing" key="LBV7" dType="str"/>
    </vnsMFolder>

</vnsClusterCfg>

<vnsMDevCfg name="DeviceConfig">
```

```
<vnsMFolder dispFeature="Network"
  dispLabel="Configure Network"
  key="Network"
  scopedBy="epg"
  cardinality="n">

    <vnsMFolder dispLabel="Routing" key="route" cardinality="n">
        <vnsMParam dispLabel="Subnet" key="network" dType="str"
          validation="netmask" mandatory="true"/>
        <vnsMParam dispLabel="Network Mask" key="netmask" dType="str"
          validation="netmask" mandatory="true"/>
        <vnsMParam dispLabel="Default Gateway" key="gateway" dType="str"
          validation="ipv4Addr" mandatory="true"/>
    </vnsMFolder>

    <vnsMFolder dispLabel="Device IP" key="ip" cardinality="n">
        <vnsMParam dispLabel="IP Address" key="ipaddress" dType="str"
          validation="ipv4Addr" mandatory="true"/>
        <vnsMParam dispLabel="Network Mask" key="netmask" dType="str"
          validation="netmask" mandatory="true"/>
    </vnsMFolder>

</vnsMFolder>

<vnsMFolder dispFeature="Policy"
  dispLabel="Configure Traffic Processing Policies"
  key="Policy"
  scopedBy="epg"
  cardinality="n">

    <vnsMFolder dispFeature="Policy"
      dispLabel="L7 Load Balancing"
      key="l7policy"
      cardinality="n">
        <vnsMParam dispLabel="Name" key="policyname" dType="str"
          mandatory="true"/>
        <vnsMParam dispLabel="URL" key="url" dType="str"/>
        <vnsMParam dispLabel="Rule" key="rule" dType="str"/>
    </vnsMFolder>

    <vnsMFolder dispFeature="Policy"
      dispLabel="Caching Policy"
      key="cachepolicy"
      cardinality="n">
        <vnsMParam dispLabel="Name" key="policyname" dType="str"
          mandatory="true"/>
      <vnsMParam dispLabel="Rule" key="rule" dType="str" mandatory="true"/>

        <vnsMParam dispLabel="Action" key="action" dType="str"
        validation="permitDeny" mandatory="true"/>
    </vnsMFolder>
</vnsMFolder>

<vnsMFolder dispFeature="Server" dispLabel="Configure Server Pool"
  key="serverpool" cardinality="n">
    <vnsMParam dispLabel="Pool Name" key="serverpoolname" dType="str"
      mandatory="true"/>
    <vnsMParam dispLabel="Type" key="type" dType="str" mandatory="true"/>

    <vnsMFolder dispLabel="LB Monitor" key="lbmonitor" cardinality="n">
        <vnsMRel dispLabel="Select LB Monitor" key="monitorRel" >
            <vnsRsTarget tDn="uni/infra/mDev-Insieme-NetworkService-1.0/
              mDevCfg/mFolder-lbmonitor"/>
        </vnsMRel>
        <vnsMParam dispLabel="Monitor State"
          key="monstate" dType="str"
          validation="enableDisable"/>
    </vnsMFolder>

   <vnsMFolder dispLabel="Server Pool Member" key="server" cardinality="n">

        <vnsMParam dispLabel="Server Name" key="servername" dType="str"/>
```

```
                <vnsMParam dispLabel="Port" key="port" dType="str"/>
                <vnsMParam dispLabel="IP Address" key="ip" dType="str"
                  validation="ipv4Addr" mandatory="true"/>
            </vnsMFolder>
        </vnsMFolder>

        <vnsMFolder dispFeature="LBMonitor" dispLabel="Configure LB Monitor"
          key="lbmonitor" cardinality="n">
            <vnsMParam dispLabel="Name" key="monitorname" dType="str"
              mandatory="true"/>
            <vnsMParam dispLabel="Type" key="type" dType="str" mandatory="true"/>
        </vnsMFolder>

        <vnsMFolder dispFeature="SSL" dispLabel="Configure SSL Certificate Key"
          key="sslcertkey" cardinality="n">
            <vnsMParam dispLabel="Certificate Key Name" key="certkey" dType="str"
              mandatory="true"/>
            <vnsMParam dispLabel="Certficate Name" key="cert" dType="str"
              mandatory="true"/>
           <vnsMParam dispLabel="Key Name" key="key" dType="str" mandatory="true"/>

        </vnsMFolder>

        <vnsMFolder dispFeature="SLB" dispLabel="Virtual Server Configuration"
          key="lbvserver" cardinality="n">
            <vnsMParam dispLabel="Name" key="name" dType="str" mandatory="true"/>
            <vnsMParam dispLabel="Type" key="servicetype" dType="str"
              mandatory="true"/>
            <vnsMParam dispLabel="IP Address" key="ipv4" dType="str"
              mandatory="true" validation="ipv4Addr"/>
            <vnsMParam dispLabel="Subnet" key="ipmask" dType="str"
              mandatory="true" validation="netmask"/>
            <vnsMParam dispLabel="Port" key="port" mandatory="true"/>
        </vnsMFolder>
    </vnsMDevCfg>

    <vnsMFunc name="LoadBalancing" dispLabel="Load Balancing">

      <vnsMConn name="external" dir="input" encType="vlan" epNotifications="subnet">

            <vnsRsInterface
              tDn="uni/infra/mDev-Insieme-NetworkService-1.0/mIfLbl-external"/>
        </vnsMConn>
        <vnsMConn name="internal" dir="output" encType="vlan"
          epNotifications="endpoint">
            <vnsRsInterface tDn="uni/infra/mDev-Insieme-NetworkService-1.0/
              mIfLbl-internal"/>
        </vnsMConn>

        <vnsMFeature name="SLB" dispOrder="0"/>
        <vnsMFeature name="Server" dispOrder="1"/>
        <vnsMFeature name="Monitor" dispOrder="2"/>
        <vnsMFeature name="Policy" dispOrder="3"/>
        <vnsMFeature name="Network" dispOrder="4"/>
        <vnsMFeature name="SSL" dispOrder="5"/>

       <vnsMFolder dispFeature="SLB" dispLabel="Virtual Server" key="lbvserverCfg"

          cardinality="n">
            <vnsMRel dispLabel="Select Virtual Server" key="lbvserverRel" >
              <vnsRsTarget tDn="uni/infra/mDev-Insieme-NetworkService-1.0/mDevCfg/

                  mFolder-lbvserver"/>
            </vnsMRel>
            <vnsRsConnector tDn="uni/infra/mDev-Insieme-NetworkService-1.0/
              mFunc-LoadBalancing/mConn-external"/>
        </vnsMFolder>

        <vnsMFolder dispFeature="Server" dispLabel="Server Pool"
          key="serverpoolCfg" cardinality="n">
            <vnsMRel dispLabel="Select Server Pool" key="serverpoolRel" >
              <vnsRsTarget tDn="uni/infra/mDev-Insieme-NetworkService-1.0/mDevCfg/
```

```
                mFolder-serverpool"/>
            </vnsMRel>
        </vnsMFolder>

        <vnsMFolder dispFeature="Monitor" dispLabel="Monitor" key="lbmonitorCfg"
          cardinality="n">
            <vnsMRel dispLabel="Select Monitor" key="lbmonitorRel" >
              <vnsRsTarget tDn="uni/infra/mDev-Insieme-NetworkService-1.0/mDevCfg/

                mFolder-lbmonitor"/>
            </vnsMRel>
        </vnsMFolder>

        <vnsMFolder dispFeature="Policy" dispLabel="Policies" key="policyCfg"
          cardinality="n">
            <vnsMRel dispLabel="Select Policies" key="policyRel" >
              <vnsRsTarget tDn="uni/infra/mDev-Insieme-NetworkService-1.0/mDevCfg/

                mFolder-Policy"/>
            </vnsMRel>
        </vnsMFolder>

       <vnsMFolder dispFeature="SLB" dispLabel="vip" key="vipCfg" cardinality="n">

            <vnsMRel dispLabel="Select Network" key="vipRel">
              <vnsRsTarget tDn="uni/infra/mDev-Insieme-NetworkService-1.0/mDevCfg/

                mFolder-Network/mFolder-ip"/>
            </vnsMRel>
        </vnsMFolder>

        <vnsMFolder dispFeature="Network" dispLabel="Internal Network"
          key="internalNetwork" cardinality="n">
            <vnsMRel dispLabel="Select Internal Network" key="internalNetworkRel">
              <vnsRsTarget tDn="uni/infra/mDev-Insieme-NetworkService-1.0/mDevCfg/

                mFolder-Network/mFolder-ip"/>
            </vnsMRel>
            <vnsRsConnector tDn="uni/infra/mDev-Insieme-NetworkService-1.0/
              mFunc-LoadBalancing/mConn-internal"/>
        </vnsMFolder>

        <vnsMFolder dispFeature="Network" dispLabel="Internal Route"
          key="internalRoute" cardinality="n">
            <vnsMRel dispLabel="Select Internal Route" key="internalRouteRel" >
              <vnsRsTarget tDn="uni/infra/mDev-Insieme-NetworkService-1.0/mDevCfg/

                mFolder-Network/mFolder-route"/>
            </vnsMRel>
            <vnsRsConnector tDn="uni/infra/mDev-Insieme-NetworkService-1.0/
              mFunc-LoadBalancing/mConn-internal"/>
        </vnsMFolder>

        <vnsMFolder dispFeature="Network" dispLabel="External Network"
          key="externalNetwork" cardinality="n">
            <vnsMRel dispLabel="Select External Network" key="externalNetworkRel"
>
              <vnsRsTarget tDn="uni/infra/mDev-Insieme-NetworkService-1.0/mDevCfg/

                mFolder-Network/mFolder-ip"/>
            </vnsMRel>
            <vnsRsConnector tDn="uni/infra/mDev-Insieme-NetworkService-1.0/
              mFunc-LoadBalancing/mConn-external"/>
        </vnsMFolder>

        <vnsMFolder dispFeature="Network" dispLabel="External Route"
          key="externalRoute" cardinality="n">
            <vnsMRel dispLabel="Select External Route" key="externalRouteel" >
              <vnsRsTarget tDn="uni/infra/mDev-Insieme-NetworkService-1.0/mDevCfg/

                mFolder-Network/mFolder-route"/>
            </vnsMRel>
            <vnsRsConnector tDn="uni/infra/mDev-Insieme-NetworkService-1.0/
```

```
                        mFunc-LoadBalancing/mConn-external"/>
                </vnsMFolder>

            </vnsMFunc>

            <vnsAbsFuncProfContr name="FunctionProfiles">
                <vnsAbsFuncProfGrp name = "GroupCfg">
                    <vnsAbsFuncProf name = "WebLoadBalancer">
                        <vnsRsProfToMFunc tDn=
                         "uni/infra/mDev-Insieme-NetworkService-1.0/mFunc-LoadBalancing"/>

                        <vnsAbsDevCfg>
                            <vnsAbsFolder key="lbvserver" name="lbvserver-Default">
                             <vnsAbsParam key="name" name="WebVServer" value="WebVServer"/>

                                <vnsAbsParam key="servicetype" name="servicetype"
                                  value="http"/>
                                <vnsAbsParam key="port" name="port" value="80"/>
                            </vnsAbsFolder>
                            <vnsAbsFolder key="serverpool" name="serverpool-Default">
                                <vnsAbsParam key="serverpoolname" name="serverpoolname"
                                  value="webserverpool"/>
                                <vnsAbsParam key="servicetype" name="servicetype"
                                  value="http"/>
                                <vnsAbsParam key="port" name="port" value="8080"/>
                            </vnsAbsFolder>
                        </vnsAbsDevCfg>
                        <vnsAbsFuncCfg>
                            <vnsAbsFolder key="lbvserverCfg" name="lbvserver-Default">
                                <vnsAbsCfgRel name="lbvserverRel"
                                  key="lbvserverRel" targetName="lbvserver-Default"/>
                            </vnsAbsFolder>
                           <vnsAbsFolder key="serverpoolCfg" name="serverpoolCfg-Default">

                                <vnsAbsCfgRel name="serverpoolRel"
                                  key="serverpoolRel" targetName="serverpool-Default"/>
                            </vnsAbsFolder>
                        </vnsAbsFuncCfg>
                    </vnsAbsFuncProf>
                </vnsAbsFuncProfGrp>
            </vnsAbsFuncProfContr>
        </vnsMDev>
    </infraInfra>
</polUni>
```

CHAPTER 3

Developing Device Scripts

About Device Scripts

The device script acts as an adapter between the Application Policy Infrastructure Controller (APIC) and the network service by converting calls to the APIC service API into device-specific calls.

Figure 5: Device Script Model

The device script runs in the context of a `ScriptWrapper`, which is an environment that handles calls for each device type. A `ScriptWrapper` runs within a namespace that limits CPU, file, and socket resource consumption.

Uploading a device package creates a `ScriptWrapper` that imports the module from the device script file. The module exposes the functions described in the API.

The device scripts must be stateless and idempotent (producing the same result if run more than once). No file I/O operations are permitted within a script, except for generating a temporary state in the `/tmp` directory. You should not use any file that is created within the `/tmp` directory for storing a persistent state. The APIC can be deployed in a cluster. The device script can get invoked from any one of the APIC instances. Any data stored in the `/tmp` directory is not guaranteed to be available across two API calls. A script must not store its own state in any file.

The APIC requires scripts to be developed for Python 2.7. Other than standard libraries that are available in Python 2.7, the script environment provides Python Requests library v1.2.3. If the device package requires any other libraries, the device package developer can bundle those libraries in the device script's `zip` file.

Note The script must be thread-safe, which means that for any instance, multiple threads can invoke the same function in the script in order to configure different devices. The script should execute in the invoking thread context and must not spawn any new threads as part of its execution.

Guidelines for Creating Device Scripts

You must implement all the scripting APIs to establish the adapters between the Application Policy Infrastructure Controller (APIC) and the network services. The APIs receive the Python dictionaries that correspond to the device specification hierarchy. You must convert the Python dictionaries to the internal format that is needed by the specific device. Similarly, when a device returns a value, the API must convert the return value to the format needed by the APIC. For examples, see the specification in Developing Device Specifications and the sample script at the end of this section.

Device Script APIs

The device script APIs are divided into four categories:

- Device
- Cluster
- Service
- Endpoint and Network Event

The Application Policy Infrastructure Controller (APIC) requires users to register one or more device cluster within a tenant. All service functions are applied to a cluster. A cluster can contain one or more network service devices. A device can be deployed in standalone mode without any redundancy by defining a cluster with a single device. A device can be deployed in active-standby HA mode by registering two devices configured as active-standby peers within a cluster. Similarly devices can be deployed in active-active mode by registering multiple devices configured as active peers within a cluster. The devices registered within a cluster are assumed to have active-active or active-standby pairing. The HA configuration of devices within a cluster can be pushed by way of APIC or it could be done out-of-band directly on the device prior to registering the devices with the APIC.

The configuration on the device is split into three categories:

- Service function specific configuration
- Device specific configuration
- Cluster specific configuration

The service configuration is pushed by way of the service APIs, the device configuration is pushed by way of the device APIs, and the cluster configuration is pushed by way of the cluster APIs.

Device APIs

The following APIs are called for each device registered within a cluster with APIC:

```
def deviceValidate( device, version )
def deviceModify( device, interfaces, configuration)
def deviceAudit( device, interfaces, configuration )
def deviceHealth( device, interfaces, configuration )
def deviceCounters( device, interfaces, configuration )
```

The configuration dictionary passed in these APIs contains any device-specific configuration that is done on the APIC.

Note The APIC does not pass any cluster-level or service function configuration information during device API callouts.

The device APIs are assumed to act on any device specific configuration and should not reference or affect cluster-level configuration or affect service functions.

Typically, configurations that are device-specific, such as link bundling (LACP), can be done in the deviceModify() and deviceAudit() callouts.

The configuration passed in the dictionary will be an instance of any folder and parameter that is defined under `vnsDevCfg` in the device Model.

Cluster APIs

The following APIs are called for each device cluster that is registered with the APIC:

```
def clusterModify( device, interfaces, configuration )
def clusterAudit( device, interfaces, configuration )
```

The configuration dictionary contains any cluster configuration that is done on the APIC.

Note The APIC does not pass any device configuration or service configuration information during cluster API callouts.

The cluster APIs are assumed to act on any cluster-level configuration and should not reference or affect device-specific of function specific configuration.

The cluster APIs are assumed to act on any cluster-level configuration. A cluster API should not reference nor change a device-specific or function-specific configuration.

A cluster-level configuration, such as for an NTP server or a global syslog server, should be done through the cluster API.

The configuration passed in the dictionary is an instance of any folder and parameter defined directly under `vnsClusterCfg` in the device Model.

Service APIs

The following APIs are called for any service function that is rendered on the device:

```
def serviceModify( device, configuration )
def serviceAudit( device, configuration )
def serviceHealth( device, configuration )
def serviceCounters( device, configuration )
```

The configuration dictionary contains an instance of parameters and folders that are defined under `vnsMDevCfg`, `vnsGrpCfg`, or `vnsMFunc`.

Note An instance of folders or parameters defined under `vnsMDevCfg` within a device model is passed in a service API callout if and only if an instance of a function in the service graph has a reference to these parameters. Not all parameter and folder instances defined under `vnsMDevCfg` are passed to the service function. The APIC passes only those parameters and folders that are used by a specific function instance that is being referenced in the service API callout.

Endpoint and Network Event

The APIC provides device script with information about the endpoints within an endpoint group (EPG) and the subnet associated with the endpoint group. This information is passed through explicit API calls, as shown in the following example:

```
def attachEndpoint(device, configuration, endpoint)
def detachEndpoint(device, configuration, endpoint)
def attachNetwork(device, configuration, network)
def detachNetwork(device, configuration, network)
```

With controller version 1.1 and later, the APIC also conveys the endpoint and network information within the configuration dictionary that is passed in the serviceModify() API and serviceAudit() API.

The service functions defined by a device package have two connectors. A user can define a service graph by attaching one or more service functions through the connectors. A service graph can be deployed by associating the graph to a pair of EPGs. The connectors of end node functions in the graph are attached to one of the EPGs. Once a service graph is deployed between a pair of EPGs, each function connector within the graph is associated to one of the EPGs. For each function connector within the graph, the APIC provides endpoint and network information for the EPG associated with the connector. The endpoint and network information is conveyed to the device script only after the following criteria are satisified:

1. The device model indicates that notification is supported by the device script for the function connector.

   ```
   <vnsMConn ... epNotifications="endpoint">
   ```

   or

   ```
   <vnsMConn ... epNotifications="endpoint, subnet">
   ```

Note If a device package uses the epNotifications attribute, the APIC supports notification of both the subnet and endpoint on the same connector. If the device package uses the notifications attribute, the APIC does not support notifications for both the endpoint and subnet on same connector.

2. The user enables notification on the connector while defining a service graph template.

Endpoint and network information allows scripts to modify a configuration dynamically on the service device. An example use case for using the attachEndpoint() and detachEndpoint() API calls for notification is with a load balancing function so that the function can dynamically update servers within a pool by using endpoint information. An example use case for subnet for notification is with a firewall so that the firewall can dynamically add access rules for subnets that are associated with an EPG. Such dynamic updating of the configuration can eliminate user error and provide automation through the APIC.

Script Framework

The following two modules must be imported by a device script:

- Import Insieme.Logger
- Import Insieme.Config

Logging

The `Insieme.Logger` module defines a logging utility. A device script can use this utility to log debug information. The logging utility writes the configuration API logs to a file called `debug.log`. This file is included in any technical support data that is collected from the Application Policy Infrastructure Controller (APIC). A device script developer should log as much information as possible to help debug any script issues.

Logs for periodic APIs, such as `serviceHealth`() and `serviceCounters`(), are redirected to the `periodic.log`. The `debug.log` and `periodic.log` files can be accessed as a fabric administrator on the APIC under the `/data/devicescript/`*`vendor_model_version`*`/logs` directory.

The logging function is similar to the Python logging function. The logs can be split into the following categories:

- CRIT
- ERROR
- WARN
- INFO
- DEBUG
- DEBUG2
- DEBUG3
- DEBUG4

The script can invoke the API as follows:

```
Logger.log( level,  Log String)
```

The following example invokes the API:

```
Logger.log( Logger.DEBUG, 'Connection to device failed')
```

Constants

The `Insieme.Config` module defines constants that can be used for parsing the dictionary:

```
Type = Insieme.Fwk.Enum(
    DEV=0,
    GRP=1,
    CONN=2,
    FUNC=3,
    FOLDER=4,
    PARAM=5,
    RELATION=6,
    ENCAP=7,
    ENCAPASS=8,
    ENCAPREL=9,
    VIF=10,
```

```
    CIF=11,
    LIF=12,
    OSPFDEV=13,
    OSPFFUNC=14,
    OSPFCONN=15,
    OSPFVENCAPASC=16,
    BGPDEV=17,
    BGPFUNC=18,
    BGPCONN=19,
    BGPVENCAPASC=20,
    DEVMGR=21,
    ENDPOINT=22,
    SUBNET=23,
)

State = Insieme.Fwk.Enum(
    UNCHANGED=0,
    NEW=1,
    CHANGED=2,
    DELETED=3,
)

Result = Insieme.Fwk.Enum(
    SUCCESS=0,
    TRANSIENT=1,
    PERMANENT=2,
    AUDIT=3,
)
```

Note Device package developers should use the `enums` defined in the `Insieme.Fwk` Python module. The integer values shown above might have changed in the final implementation. Using the absolute values shown in this document can lead to unexpected behavior.

Configuration Dictionary Format

The configuration dictionary that is passed in the cluster API, device API, and service API follows the same structure as defined in the device specification file. The configuration is passed as a hierarchy of dictionaries, with each level identifying a folder. The dictionary format is as follows:

```
(type,  key, name) : { 'state': ...
                       'transaction': ...
                       'connector': ...
                       'value': ...
                       'target': ...
                       'device': ...
                      }
```

The fields are as follows:

Field	Description

type	Identifies the type of the object represented by the dictionary. The field can have one of the following values:
	`DEV=0,`
	`GRP=1,`
	`CONN=2,`
	`FUNC=3,`
	`FOLDER=4,`
	`PARAM=5,`
	`RELATION=6,`
	`ENCAP=7,`
	`ENCAPASS=8,`
	`ENCAPREL=9,`
	`VIF=10,`
	`CIF=11,`
	`LIF=12,`
	`OSPFDEV=13,`
	`OSPFFUNC=14,`
	`OSPFCONN=15,`
	`OSPFVENCAPASC=16,`
	`BGPDEV=17,`
	`BGPFUNC=18,`
	`BGPCONN=19,`
	`BGPVENCAPASC=20,`
	`DEVMGR=21,`
	`ENDPOINT=22,`
	`SUBNET=23,`

Key	Specifies the key or name attribute that is defined in the device specification file for the object.
Name	Specifies the parameter or folder instance name that is provided by the user.

State	Identifies the object's state. This field can have one of the following values:
	`UNCHANGED=0,`
	`NEW=1,`
	`CHANGED=2,`
	`DELETED=3,`

Connector	Specifies the name of the connect instance that is resolved according to the relations that are defined the specification file. This field is populated for a folder or a relation dictionary only if the corresponding `vnsMFolder` object or `vnsMRel` object has `vnsRsConnector` relations defined in the device specification file.
Value	Defines the value for the object. In the case of a folder, this field can contain another dictionary. A relations object does not contain a value element, and instead has a target element. A value for a parameter object cannot exceed 512 characters.

Field	Description
Target	Defines the target folder to which a relations object is resolved. This element is populated only for a relations object.
Transaction	Contains the Application Policy Infrastructure Controller (APIC) transaction ID that resulted in a specific API callout. The transaction ID is used for correlating request/response between APIC and the device script. It is used primarily for debugging convenience. A script can ignore this value.
Device	Any configuration passed in a cluster API or service API is typically applied to the cluster and it is in effect on all devices within the cluster. However, there can be cases in which the configuration must be applied to a specific device within the cluster, such as when configuring the interface IP address for devices within the cluster. Each device can be assigned a unique IP address. As a result, the interface IP address configuration must be applied to one specific device within the cluster. You accomplish this by using device context labels on the APIC. When you configure the parameter on the APIC, you can associate a device context label that identifies a device within the cluster on which the configuration should be applied. If a parameter is tied to a specific device context within a cluster, the APIC instantiates a device key in the dictionary with device name as its value. The script can lookup the device name in the device dictionary passed in the callout. A script can apply the parameter configuration to a specific device identified by device field.

For more information about the configuration dictionary format, see Sample Script, on page 73. For an example dictionary for connector, encapsulation, and interface information, see Fabric Connectivity, on page 97

API Return Value

The APIs return a dictionary with the following format:

```
{ 'state':
  'health': []
  'fault': []
}
```

The state returns one of the following values:

```
SUCCESS=0
TRANSIENT=1
PERMANENT=2
AUDIT=3
```

For information about the health, see Health Monitoring, on page 105. For information about faults, see Faults Codes, on page 37.

You should configure the device script to set a timeout of at least 30 seconds to establish a connection with the device. If the device script fails to establish network connectivity within the time interval, it returns a `TRANSIENT (1)` state in the return dictionary. The APIC retries the transaction until the Transient state is cleared.

Transient faults indicate failures that don't require immediate user attention to resolve the issue. It could be a temporary event that prevents a script from pushing the configuration. APIC will retry pushing the configuration aggressively till the fault is cleared. If a transient fault fails to clear after multiple (5) retries, APIC marks the failure as permanent.

A device script can request for an audit call by returning the `AUDIT` state in the return dictionary. APIC will trigger a clusterAudit(), deviceAudit(), or serviceAudit() call depending on whether the cluster API, device API, or service API returned an `AUDIT` state. The script can request for an audit in the event it detects a configuration mismatch between the APIC and a device which cannot be resolved within the current API call

A device script returns a `PERMANENT` fault if the parameter values or configuration has an issue. User intervention might be required to resolve the problem.

A persistent transient fault may translate to a permanent fault. APIC will continue to periodically push the configuration till the fault is resolved. The retry is spaced at larger interval. Some faults might require a user intervention to clear, such as an invalid parameter value.

The **scriptwrapper** process that invokes the device script API expects the API to return within 120 seconds. If the script takes longer than 120 seconds, the **scriptwrapper** process terminates and restarts. Any outstanding transactions are replayed after the restart.

Service Configuration

The Application Policy Infrastructure Controller (APIC) creates an instance of a metadevice (`MDev`) for each tenant context. An `MDev` instance is referred to as a virtual device, or `vDev`. All service configuration instances for a tenant are rooted under a `vDev`. The APIC generates a unique id for identifying each `vDev`. The APIC also generates a unique ID for each graph instance, which is represented as `vGrp`. The group configuration and function configuration are rooted under this `vGrp` instance that identifies a specific graph instance.

The configuration dictionary that is passed in the serviceAudit(), **serviceModify**(), serviceHealth(), and serviceCounter() APIs always contains a `vDev` object and a `vGrp` object. A multi-context device script should use the `vDev` object to identify a tenant context uniquely, which could map to a specific routing domain or context.

Parameter Instance Name on Device

Any folders and parameters that are defined under `vnsMDevCfg` are instantiated under `vDev`. Either a multi-context device must create a configuration folder for each `vDev` ID, or the device or device script must concatenate the `vDev` ID that is passed in the configuration dictionary to generate a unique name across multiple contexts.

The following example shows a dictionary with a global folder and parameter for a function:

```
{
  (0,'',4304): {
     'state': 1,
     'transaction': 10000,
     'value': {
          (4,'Server','webserver1'): {
             'state': 1,
              'transaction': 10000,
              'value': {
              (5,'ipaddress','ipaddress'): {
                 'state': 1,
                 'transaction': 10000,
                 'value': '192.168.100.2'
               },
               (5,'servername','servername'): {
                 'state': 1,
                 'transaction': 10000,
                 'value': 'webserver1'
               }
          }
      }
}
```

A device script must make sure that the servername instance is uniquely identified across different contexts. Because the names can overlap across different contexts that are configured on the same device, a device script can append the `vDev` ID to the servername value to make the parameter and folder name unique across different contexts. The following examples are unique servername values:

```
4304_webserver1
webserver1_4304
webserver1.4304
```

Another method for creating unique servername values is by creating a folder called `4304`, and then creating the `webserver1` instance under the `4304` folder.

A single context device can ignore the `vDev` argument that is passed in the configuration dictionary. Similarly, a single context or multi-context device must append the group ID to keep the parameter and folder name that are configured for a graph instance unique from another graph instance that is rendered on the same device, as shown in the following example:

```
(0,'',4304): {
 'state': 1,
 'transaction': 10000,
 'value': {
    (1,'',4368): {
       'state': 1,
       'transaction': 10000,
       'value': {
          (3,'SLB','SLB'): {
             'state': 1,
              'transaction': 10000,
              'value': {
                  (5,'servicename','servicename'): {
                     'state': 1,
                     'transaction': 10000,
                     'value': 'webservice'
                  },
              }
          }
      }
    }
  }
}
```

The device script must ensure that the servicename instance that is created for this graph instance does not overlap with another graph instance that is rendered on the same device. The reason is because the parameter and folder that are defined under a Group or Function configuration in the device specification is unique for

a graph instance or function instance within a graph. Such parameter or folder names might not be unique across different instances of the graph or instances of a function within a graph, respectively. The device script can append the group ID or the group ID and function name that is passed in the device dictionary to make the folder parameter name unique across graphs or functions within a graph, as shown in the following example:

```
4368.SLB.webservice
```

If the device supports creating folders, the script can create a folder for the graph identified by the vgrp ID passed in the configuration dictionary and group specific parameters can be instantiated under the vgrp folder. Similarly function configuration can created under a function specific folder within the group folder, thus maintaining uniqueness of each parameter/folder across multiple graph instances.

API Callouts

Cluster Configuration

Registering the first concrete device within a cluster with Application Policy Infrastructure Controller (APIC) triggers the following sequence of API callouts:

1. `deviceValidate`— This API validates whether a device version registered with the APIC can be supported by the device package.
2. `deviceAudit`— This APIC call clears any device global configuration that is not pushed from the APIC. The device script does not clear the management IP address, login credentials, and any configuration that the device needs to be operational that is not supported through the APIC. The deviceAudit call needs to be selective in clearing the configuration. Only the configuration that can be pushed from the APIC is cleared.
   deviceAudit() should not modify any service function parameter/configuration or cluster level configuration. The service functions should not be affected on a deviceAudit() call. The purpose of the deviceAudit() call is to bring the device level configuration in sync with the APIC. The script should bring the device in sync with minimal disruption to data path. It should identify the configuration found on the device which was not pushed by APIC, such a configuration should be removed.

Note This should be done only for a configuration that can be managed through the device package.

   The script should push any missing configuration or configuration that is not in sync to the device.

3. `clusterAudit`—The APIC calls this API when the first device is added to the logical device (device cluster). This API clears any configuration from the cluster that is not pushed by the APIC. Similar to the `deviceAudit`() call, this API clears only the configuration that can be supported through the APIC.
   The clusterAudit() call should not modify any service function parameter/configuration or device specific parameters. The service functions should not be affected on a clusterAudit() call. The purpose of the clusterAudit() call is to bring the cluster level configuration in sync with the APIC. The script should bring the device in sync with minimal disruption to the data path. It should identify the configuration found on the device which was not pushed by APIC, such a configuration should be removed.

Note This should be done only for configuration that can be managed through a device package.

   A script should push any missing configuration or configuration that is not in sync to the device.

Device packages that support multiple contexts must use the clusterAudit() call for clearing any unwanted tenant configurations from the device. The APIC allocates a unique `VDev` instance for each tenant context. A device can use the APIC for isolating tenant-specific configurations. With the 1.1 release, the APIC provides a list of `VDev` IDs during the clusterAudit() call. The `VDev` ID list is passed in the device dictionary. The device script can use the `VDev` ID list information to identify contexts that should be cleared from the device.

The following example shows a `VDev` ID list that was passed in the device dictionary:

```
{'creds': {'password': '<hidden>', 'username': 'nsroot'},
 'devs': {'Generic1': {'creds': {'password': '<hidden>',
                                 'username': 'nsroot'},
                       'host': '42.42.42.100',
                       'port': 80,
                       'version': '1.0',
                       'virtual': True},
          'Generic2': {'creds': {'password': '<hidden>',
                                 'username': 'nsroot'},
                       'host': '42.42.42.101',
                       'port': 80,
                       'version': '1.0',
                       'virtual': True}},
 'host': '42.42.42.99',
 'name': 'InsiemeCluster',
 'port': 80,
 'vdevs': [{'ctxName': 'cokectx1', 'id': 9597, 'tenant': 'coke'}],
 'virtual': True}
```

Note The APIC generates a serviceAudit() for each `VDev` ID and provides a complete configuration. The configuration within the context must be synchronized during the serviceAudit() call.

4. `clusterModify`—This API is called for any device that is registered and added to a logical device (device cluster). The call results in configuring the cluster configuration.
5. `serviceAudit`—This API is called with function configuration applied by the user on APIC. The script should push any service functions passed in the configuration. If the device has any service function that is not configured on APIC but could be managed through the API, the script should remove such a configuration. The purpose of the serviceAudit() call is to make sure service specific functionality on the device is in sync with the APIC.

Note A device cluster state maintained within the APIC is changed to be operationally up when clusterAudit() returns success. All service level APIs are invoked only after cluster is operational.

The cluster can be marked operational once a single device within the cluster is operational.

Registering additional devices within a cluster triggers the following calls:

- deviceValidate()
- deviceAudit()
- clusterAudit()

Note clusterAudit() should not disrupt devices that are operational within the cluster. Besides service functions that are deployed on the cluster others should not be impacted by adding more devices (such as **deviceAudit**() or clusterAudit()) should not impact the service functions that are in operational state on the cluster.

After the device is registered, APIC periodically calls the following APIs:

- `deviceHealth`
- `deviceCounter`

Removing a device within a cluster results in calling clusterAudit(). If the last device is removed from the cluster and the cluster state changes to operationally down, APIC calls serviceModify() to remove any service specific configuration.

Service Graph Configuration

When you instantiate a graph by associating an abstract graph to a contract that is bound to an endpoint group (EPG), the APIC calls the following API:

- `serviceModify`—This API instantiates service functions on the device.

After a service has been rendered, the APIC periodically calls the following APIs:

- serviceHealth
- serviceCounter

Audit Calls

APIC will trigger a serviceAudit() when the APIC cluster changes while there is an outstanding transaction with the device. Since the cluster change can cause a different APIC to resume communication with the device, a previous configuration transaction may not have completed causing the device and the APIC cluster to go out of sync. The serviceAudit() call issued by a new master ensures the device state is kept in sync with the APIC.

APIC passes the entire service configuration that is associated with a given device in a single serviceAudit() call. A script is required to process serviceAudit() with a minimum disruption to services configured on the

device. The serviceAudit() call should not result in clearing any device specific global configuration or cluster configuration.

Similar to serviceAudit(), APIC triggers a deviceAudit() and clusterAudit() depending on whether there were any outstanding device configurations or cluster configuration transactions in progress.

The device script can trigger an audit call when it detects that the configuration on the device and the APIC has changed and the script cannot resolve the difference from within the API call.

The `clusterAudit`() call can be triggered by returning "3" (AUDIT) as the return state for the `clusterModify`() call. The APIC passes the entire cluster configuration that is defined by the folders and parameters under `vnsClusterCfg` in `clusterAudit`(). The service function configuration is not passed in `clusterAudit`(). The script must apply the configuration that is passed in the dictionary and remove any configuration that is not defined by the APIC. The script removes only unwanted configurations that are found on the device that can be managed by the APIC and is defined in a device specification file under `vnsClusterCfg`.

The `deviceAudit`() call can be triggered by returning "3" (AUDIT) as the return state for the `deviceModify`(), `deviceHealth`(), or `deviceCounter`() call. The APIC passes the entire device configuration that is defined by the folders and parameters under `vnsDevCfg` in the `deviceAudit`() call. The service function configuration is not passed in `deviceAudit`(). The script must apply the configuration that is passed in the dictionary and remove any device configuration that is not defined by the APIC. The script removes only a configuration that is defined on the APIC and can be managed through the APIC.

The `serviceAudit`() can be triggered by returning '3' as the return state for the `serviceModify`(), `serviceHealth`(), or `serviceCounter`() call. The APIC passes all service function configurations across all `vDev` (tenant) and graph instances that are rendered on the device cluster. The device removes any configuration that is not configured by the APIC and can be managed through the APIC.

Passing Parameters

The following example shows a configuration dictionary that is passed for service APIs:

```
Configuration = {
   (0, mDev-key, mDev-Instance-Name) : {
       'state': state
       'value': {
            (1, '', functionGroup-Instance) : {
               'state': state
               'value': {
                   (3, mDevFunction-Key, mDev-Function-Instance-Name): {
                       'state': state,
                       'value': {
                            (2, mDevFunction-Connector-Key, InstanceName): {
                               'state': state
                               'value': {
                                  'CDev-Instance-Name': 'EncapAssociation-Instance',
                                   ...
                               }
                            }
                            (4, mFolder-Key, mFolder-Instance): {
                                  'state': state
                                  'value': {
                                       (5, mParam-Key, mParam-Instance): {
                                            'state': state
                                             'device': cluster-node-instance
                                              'value': {
                                              }
                                       }
                                  }
                            }
                       }
```

```
                        }
                    }
                }
                (4, mFolder-Key, mFolder-Instance): {
                      'state': state
                      'value': {
                           (5, mParam-Key, mParam-Instance): {
                                'state': state
                                 'value': {
                                 }
                           }
                      }
                },
                (7, '', <encap-instance>): {'state': 1, 'tag': TagValue, 'type': TagType },
                (8, '', <encap-association-Instance>): {
                     'state': 1,
                     'encap': <encapInstance>,
                     'vif': <LogicalInterfaceInstanceID>
                },
                (10, '', <LogicalInterfaceInstance>): {
                     'state': 0,
                     'cifs': {'
                        'state': 0,
                        'value': <Interface Value>
                      }
                }
          }
    },
}         '

Device = {
   'devs': {
      cluster-node-Instance: {
            'host': cluster-node-ipaddress
            'port': cluster-node-port-number,
            'creds': {
               'username': username,
               'password': password
            }
      }
    },
    'name': cluster-name,
    'host': cluster-ipaddress
    'port': cluster-port-number,
    'creds': {
        'username': username,
        'password': password
    }
}
```

Note All parameters are strings.

The following example shows parameters that are passed in the deviceValidate() callout.

```
Device =  {
           'creds': {
                  'password': '<password>',
                   'username': 'admin'
             },
            'host': '10.0.0.2',
            'port': 443,
            'virtual': False
       }

version: '1.0'
```

The following is an example of a parameter dictionary that is passed in deviceAudit(), deviceModify(), deviceHealth(), and deviceCounter() call outs. This is parameter structure is identical to a configuration dictionary that is passed in the service or cluster API call outs.

```
devices: = {
          'host': '10.0.0.2',
          'port': 443,
          'creds': {'username': 'admin',
                    'password': '<password>'},
          'virtual': False},

interfaces =  {
      (11, '', 'eth1_0'): {'state': 0, 'label': ''},
      (11, '', 'eth1_1'): {'state': 0, 'label': ''}
},
```

The following is an example of parameters that are passed in the clusterAudit() and clusterModify() APIs. The configuration dictionary format is identical to the example that is shown in the service call out.

```
Device =  {
      'name': 'LB',
      'virtual': False,
      'devs': {
            'Device1': {
                  'host': '10.0.0.2',
                  'port': 443,
                  'creds': {
                        'username': 'admin',
                        'password': 'password'
                   },
                   'virtual': False}
               },
         'host': '10.0.0.1',
         'port': 443,
         'creds': {
               'username': 'admin',
               'password': '<password>'
         }
  },

Interfaces =  {
    (12, '', 'internal'): {
            'state': 0,
            'cifs': {
                  'Device1': 'eth1_1'
            },
            'label': ''
     },
     (12, '', 'external'): {
              'state': 0,
               'cifs': {
                     'Device1': 'eth1_0'
                },
              'label': ''
       }
 }
```

Device Identification

The device parameter is a simple dictionary that contains the device configuration and the credentials required to access the device. Most of the functions in the device script take a device parameter that identifies the device intended for modification, as shown in the following example:

```
{
  'creds': {
    'password': 'admin',
    'username': 'admin'
  },
```

```
  'host': '10.30.13.153',
  'port': 443
}
```

The Application Policy Infrastructure Controller (APIC) stores credentials in an encrypted partition.

The script must not store the credentials in temporary files and should not print the credential in debug logs.

Endpoint and Network Event Callouts

If the service function connector requires notification and you have enabled notification, the Application Policy Infrastructure Controller (APIC) calls the attachEndpoint() and detachEndpoint() APIs with endpoint information for an endpoint group (EPG) that is associated with a given connector. The APIC calls the APIs for each function connector that supports notification and for which you have administratively enabled notification. The endpoint is passed as a list.

The APIC controller version 1.1 and later supports passing endpoint information by using the serviceModify() and serviceAudit() call. The device script must use endpoint and network information that is passed in serviceModify() or serviceAudit() callout instead of the attachEndpoint() and detachEndpoint() callouts and the attachNetwork() and detachNetwork() callouts. The serviceAudit() callout provides the capability to audit and keep dynamically added endpoint and subnet information in sync with the APIC. The attachEndpoint(), detachEndpoint(), attachNetwork(), and detachNetwork() APIs alone do not support audit capability.

For controller version 1.1 and later, if the device script supports handling endpoint or network information through the service API callouts, the device script should not implement the attachEndpoint(), detachEndpoint(), attachNetwork(), and detachNetwork() APIs. If these APIs are also defined, the APIC calls the attachEndpoint(), detachEndpoint(), and serviceModify() APIs on the endpoint event. Similarly, the APIC calls the attachEndpoint(), detachEndpoint(), and serviceModify() APIs on the network event.

Controller version 1.0 does not provide endpoint and network information in the serviceAudit() and serviceModify() APIs.

The endpoint information in the serviceModify() and serviceAudit() APIs is passed as a new object type: 22. The endpoint information is contained under the connector within a function. The endpoint information for an EPG is contained under the function connector that is associated with the EPG.

The network information in the serviceModify() and serviceAudit() APIs is passed as a new object type: 23. The network information is contained under the connector within a function. The network information for an EPG is contained under the function connector that is associated with the EPG.

Following is a sample configuration dictionary for conveying endpoint information in a serviceModify() API call:

```
{'args': ({
    (0, '', 5348): {
        'ackedState': 0,
        'ctxName': 'tenant1ctx1',
        'state': 2,
        'tenant': 'tenant1',
        'transaction': 0,
        'txid': 10000,
        'value': {
            (1, '', 5661): {
                'absGraph': 'G1',
                'ackedState': 0,
                'state': 2,
                'transaction': 0,
                'value': {
                    (3, 'SubnetFunc', 'Node'): {
                        'ackedState': 0,
                        'state': 2,
                        'transaction': 0,
```

```
                        'value': {
                            (2, 'external', 'inside'): {
                                'ackedState': 0,
                                'state': 2,
                                'transaction': 0,
                                'value': {
                                    (9, '', 'InsiemeCluster_inside_2621440_32773'): {
                                        'ackedState': 0,
                                        'state': 0,
                                        'target': 'InsiemeCluster_inside_2621440_32773',
                                        'transaction': 0},
                                    (23, '', u'10.10.10.0/24'): {
                                        'ackedState': 0,
                                        'state': 0,
                                        'epg': 'app',
                                        'transaction': 0},
                                    (23, '', u'10.10.11.0/24'): {
                                        'ackedState': 0,
                                        'state': 0,
                                        'epg': 'app',
                                        'transaction': 0},
                                    (22, '', u'10.10.12.0/24'): {
                                        'ackedState': 0,
                                        'state': 0,
                                        'epg': 'app',
                                        'transaction': 0}}},
                            (2, 'internal', 'outside'): {
                                'ackedState': 0,
                                'state': 2,
                                'transaction': 0,
                                'value': {
                                    (9, '', 'InsiemeCluster_outside_2621440_16388'): {
                                        'ackedState': 0,
                                        'state': 0,
                                        'target': 'InsiemeCluster_outside_2621440_16388',
                                        'transaction': 0}}},
                            (4, 'folder', 'folder1'): {
                                'ackedState': 0,
                                'device': 'Generic1',
                                'state': 0,
                                'transaction': 0,
                                'value': {
                                    (5, 'param', 'param'): {
                                        'ackedState': 0,
                                        'state': 0,
                                        'transaction': 0,
                                        'value': 'value'}}},
                            (4, 'folder', 'folder2'): {
                                'ackedState': 0,
                                'device': 'Generic2',
                                'state': 0,
                                'transaction': 0,
                                'value': {
                                    (5, 'param', 'param'): {
                                        'ackedState': 0,
                                        'state': 0,
                                        'transaction': 0,
                                        'value': 'value'}}}}}}},
    (4, 'oneFolder', 'f1'): {
        'ackedState': 0,
        'state': 0,
        'transaction': 0,
        'value': {
            (5, 'oneParam', 'p1'): {
                'ackedState': 0,
                'state': 0,
                'transaction': 0,
                'value': 'v1'}}},
    (7, '', '2621440_16388'): {
        'ackedState': 0,
        'state': 0,
        'tag': 237,
        'transaction': 0,
```

```
                'type': 1},
            (7, '', '2621440_32773'): {
                'ackedState': 0,
                'state': 0,
                'tag': 238,
                'transaction': 0,
                'type': 1},
            (8, '', 'InsiemeCluster_inside_2621440_32773'): {
                'ackedState': 0,
                'encap': '2621440_32773',
                'state': 0,
                'transaction': 0,
                'vif': 'InsiemeCluster_inside'},
            (8, '', 'InsiemeCluster_outside_2621440_16388'): {
                'ackedState': 0,
                'encap': '2621440_16388',
                'state': 0,
                'transaction': 0,
                'vif': 'InsiemeCluster_outside'},
            (10, '', 'InsiemeCluster_inside'): {
                'OspfVEncapAscCfg': {},
                'ackedState': 0,
                'cifs': {'Generic1': 'ext',
                'Generic2': 'ext'},
                'state': 0,
                'transaction': 0},
            (10, '', 'InsiemeCluster_outside'): {
                'OspfVEncapAscCfg': {},
                'ackedState': 0,
                'cifs': {'Generic1': 'int',
                'Generic2': 'int'},
                'state': 0,
                'transaction': 0}}}},),
    'device': {
        'creds': {'password': '<hidden>', 'username': 'nsroot'},
        'devs': {
            'Generic1': {
                'creds': {
                    'password': '<hidden>',
                    'username': 'nsroot'},
                'host': '42.42.42.100',
                'port': 80,
                'version': '1.0',
                'virtual': True},
            'Generic2': {
                'creds': {
                    'password': '<hidden>',
                    'username': 'nsroot'},
                'host': '42.42.42.101',
                'port': 80,
                'version': '1.0',
                'virtual': True}},
        'host': '42.42.42.99',
        'name': 'InsiemeCluster',
        'port': 80,
        'virtual': True}}
```

Following is a sample configuration dictionary for conveying network information in a serviceModify() API call:

```
{'args': ({
    (0, '', 5348): {
        'ackedState': 0,
        'ctxName': 'tenant1ctx1',
        'state': 2,
        'tenant': 'tenant1',
        'transaction': 0,
        'txid': 10000,
        'value': {
            (1, '', 5661): {
                'absGraph': 'G1',
                'ackedState': 0,
                'state': 2,
```

```
    'transaction': 0,
    'value': {
        (3, 'SubnetFunc', 'Node'): {
            'ackedState': 0,
            'state': 2,
            'transaction': 0,
            'value': {
                (2, 'external', 'inside'): {
                    'ackedState': 0,
                    'state': 2,
                    'transaction': 0,
                    'value': {
                        (9, '', 'InsiemeCluster_inside_2621440_32773'): {
                            'ackedState': 0,
                            'state': 0,
                            'target': 'InsiemeCluster_inside_2621440_32773',
                            'transaction': 0},
                        (22, '', u'10.10.10.11'): {
                            'ackedState': 0,
                            'state': 0,
                            'transaction': 0},
                        (22, '', u'10.10.10.12'): {
                            'ackedState': 0,
                            'state': 0,
                            'transaction': 0},
                        (22, '', u'10.10.10.13'): {
                            'ackedState': 0,
                            'state': 0,
                            'transaction': 0}}},
                (2, 'internal', 'outside'): {
                    'ackedState': 0,
                    'state': 2,
                    'transaction': 0,
                    'value': {
                        (9, '', 'InsiemeCluster_outside_2621440_16388'): {
                            'ackedState': 0,
                            'state': 0,
                            'target': 'InsiemeCluster_outside_2621440_16388',
                            'transaction': 0}}},
                (4, 'folder', 'folder1'): {
                    'ackedState': 0,
                    'device': 'Generic1',
                    'state': 0,
                    'transaction': 0,
                    'value': {
                        (5, 'param', 'param'): {
                            'ackedState': 0,
                            'state': 0,
                            'transaction': 0,
                            'value': 'value'}}},
                (4, 'folder', 'folder2'): {
                    'ackedState': 0,
                    'device': 'Generic2',
                    'state': 0,
                    'transaction': 0,
                    'value': {
                        (5, 'param', 'param'): {
                            'ackedState': 0,
                            'state': 0,
                            'transaction': 0,
                            'value': 'value'}}}}}}},
(4, 'oneFolder', 'f1'): {
    'ackedState': 0,
    'state': 0,
    'transaction': 0,
    'value': {
        (5, 'oneParam', 'p1'): {
            'ackedState': 0,
            'state': 0,
            'transaction': 0,
            'value': 'v1'}}},
(7, '', '2621440_16388'): {
    'ackedState': 0,
```

```
                'state': 0,
                'tag': 237,
                'transaction': 0,
                'type': 1},
            (7, '', '2621440_32773'): {
                'ackedState': 0,
                'state': 0,
                'tag': 238,
                'transaction': 0,
                'type': 1},
            (8, '', 'InsiemeCluster_inside_2621440_32773'): {
                'ackedState': 0,
                'encap': '2621440_32773',
                'state': 0,
                'transaction': 0,
                'vif': 'InsiemeCluster_inside'},
            (8, '', 'InsiemeCluster_outside_2621440_16388'): {
                'ackedState': 0,
                'encap': '2621440_16388',
                'state': 0,
                'transaction': 0,
                'vif': 'InsiemeCluster_outside'},
            (10, '', 'InsiemeCluster_inside'): {
                'OspfVEncapAscCfg': {},
                'ackedState': 0,
                'cifs': {
                    'Generic1': 'ext',
                    'Generic2': 'ext'},
                'state': 0,
                'transaction': 0},
            (10, '', 'InsiemeCluster_outside'): {
                'OspfVEncapAscCfg': {},
                'ackedState': 0,
                'cifs': {
                    'Generic1': 'int',
                    'Generic2': 'int'},
                'state': 0,
                'transaction': 0}}}},),
    'device': {
        'creds': {'password': '<hidden>', 'username': 'nsroot'},
        'devs': {
            'Generic1': {
                'creds': {
                    'password': '<hidden>',
                    'username': 'nsroot'},
                'host': '42.42.42.100',
                'port': 80,
                'version': '1.0',
                'virtual': True},
            'Generic2': {
                'creds': {
                    'password': '<hidden>',
                    'username': 'nsroot'},
                'host': '42.42.42.101',
                'port': 80,
                'version': '1.0',
                'virtual': True}},
        'host': '42.42.42.99',
        'name': 'InsiemeCluster',
        'port': 80,
        'virtual': True}}
```

The state field that is associated with the endpoint (22) object and network (23) object has the following meanings:

State	Description
0	No change to the endpoint or network. The endpoint or network continues to be associated with the EPG.

State	Description
1	Identifies an attach event. A new endpoint or network was associated to the EPG.
2	This state typically does not occur. This state can be treated the same as a new endpoint or network association (state 1).
3	Identifies a detach event. An endpoint or network was dissociated from the EPG.

Handling Script Failures

The Application Policy Infrastructure Controller (APIC) services integration model is based on the promise theory, in which individual agents join in a system of voluntary cooperation. The APIC pushes the intended state to the script and provides an API to raise faults on parts of the configuration.

Failures can occur when parameter values change or when a device error occurs. In the case of an APIC failure, the new APIC determines whether to reissue the serviceModify call or audit the function groups. The APIs can return a fault list that provides details about the cause for the fault and the potential corrective action for resolving the fault.

The APIC does not maintain a transaction history. The state of an object within the configuration dictionary is determined based on the state of the managed object information tree. The object state is marked "NEW (1)" when a new object is inserted in the managed object tree. Any subsequent API callouts from the APIC sets that objects state to "UNCHANGED (0)" till either the object value is changed by an explicit user configuration or an implicit the APIC event that causes the object value to change.

On a change in objects value, the state of the object in the next modify() API call is set to "CHANGED(2)". If the object is deleted, the APIC indicates the deletion by setting the object state to "DELETED (3)".

If the API returns the "PERMANENT(2)" state, which indicates that the script encountered an error while processing the configuration, the APIC retries the configuration until it receives the "SUCCESS(0)" state from the script. If the user has not changed the value of any object between retries, the APIC sets the object state to "UNCHANGED(0)" during these retries. The state of the object is set to "NEW(1)", "CHANGED(2)", or "DELETED(3)" only if a new object was created, an existing object was modified, or an existing object was deleted between the retries.

The following example shows a case that can occur due to a lack of transaction history on the APIC. The device package developer should be aware of such issues and address the issues in the device script.

Assume that a device package has parameters A, B, and C, where A depends on B and B depends on C.

- When A, B, and C are created on the APIC, this causes the APIC to call a device script with the "create (1)" state for all three parameters, such that A=a(1) -> B=b (1) -> C=c (1).

  The script raises a fault for B and configures C, but does not configure A and returns the "PERMANENT" state. The end of this callout device has only "c".

  The APIC raises a fault on B because the configuration for object B was invalid.

- When the value of B is changed to b', this resolves the configuration issue. This change in configuration results in the APIC calling the device script with the modification. The parameter state during the subsequent API callout is as follows:

  A=a(0) -> B=b'(2) -> C=c(0)

  The APIC does not maintain a transaction history. The APIC does not have any knowledge that A=a was not applied on the device during the previous transaction. The APIC retries the configuration and updates the parameter state for B, as it was the only parameter that changed between retries.

  The script tries to update parameter B to b'. The script should be able to identify that parameter B does not exist on the device and is being modified before the create operation. Ideally, the modify request to the device should return an error.

  If the device is capable of identifying a modify operation before a create operation as an error, the script should handle such an error from the device using one of the following options:

  - Option 1

    Create the parameter B with value b' on the device and resolve any dependent objects that could be missing from the device. The script must walk through the configuration that is passed in the dictionary and check if the objects that depend on the modified object were created on the device. If the objects were not created, the script should create the object. In this example, the script should create A=a that depends on B=b' because object A=a is not found on the device.

  - Option 2

    Return the AUDIT(3) state in the return response. The APIC replays the entire configuration.

    Note Requesting an audit call can be an expensive operation because the APIC passes the entire configuration.

    The device script needs to identify the missing configuration and apply any missing configuration parameters. This option should be used only when a modified object in the dictionary has dependent objects in the configuration.

  If the device is not capable of returning an error when a parameter is modified before a create operation, the device script should read the object that is being modified before performing a modify operation on it. This approach of reading the configuration from the device before modifying can be expansive and impact performance. To keep the solution optimal, reading the configuration from the device should be done only if the object being modified could have other objects depending on it.

Sample Script

The following example shows a device script in Python:

```
import pprint
import sys
import Insieme.Logger as Logger

#
# Infra API
#

def deviceValidate( device,version ):
```

```
    return {
        'state': 0,'version': '1.0'
        }

def deviceModify( device,interfaces,configuration):
    return {
        'state': 0,'faults': [],'health': []
        }

def deviceAudit( device,interfaces,configuration ):
    return {
        'state': 0,'faults': [],'health': []
        }

def deviceHealth( device,interfaces,configuration ):
    return {
        'state': 0,'faults': [],'health': [([], 100)]
        }

def deviceCounters( device,interfaces,configuration ):
    return {
        'state': 0,'counters': [
            ( [(11,'','eth0')],{
                'rxpackets':100,
                'rxerrors':101,
                'rxdrops':102,
                'txpackets':200,
                'txerrors':201,
                'txdrops':202
            }
          )
        ]
    }

def clusterModify( device,interfaces,configuration ):
    return {
        'state': 0,'faults': [],'health': []
        }

def clusterAudit( device,interfaces,configuration ):
    return {
        'state': 0,'faults': [],'health': []
        }

#
# FunctionGroup API
#

def serviceModify( device,configuration ):
    return {
        'state': 0,'faults': [],'health': []
       }

def serviceAudit( device,configuration ):
    return {
        'state': 0,'faults': [],'health': []
        }

def serviceHealth( device,configuration ):
    return {
        'state': 0,'faults': [],'health': []
        }

def serviceCounters( device,configuration ):
    externalIntferface, = [
        (0, 'Firewall', 4384),
        (1, '', 4432),
        (3, 'Firewall-Func', 'FW-1'),
        (2, 'external', 'external1')
    ]
    internalInterface = [
        (0, 'Firewall', 4384)
        (1, '', 4432)
```

```
        (3, 'Firewall-Func', 'FW-1'),
        (2, 'internal','internal1')
    ]
    Firewall-1-External-Counters = (externalInterface, {
        'rxpackets': 100,
        'rxerrors': 0,
        'rxdrops': 0
        'txpackets': 100
        'txerrors': 4
        'txdrops': 2
        }
    )
    Firewall-1-Internal-Counters = (internalInterface, {
        'rxpackets': 100,
        'rxerrors': 0,
        'rxdrops': 0
        'txpackets': 100
        'txerrors': 4
        'txdrops': 2
        }
    )
    Counters = [ Firewall-1-External-Counters, Firewall-1-Internal-Counters ]
        return {
            'state': 0,
            'counters': Counters
        }

#
# EndPoint/Network API
#

def attachEndpoint( device,
                    configuration,
                    endpoints ):
    return {
        'state': 0,
        'faults': [],
        'health': [],
        }

def detachEndpoint( device,
                    configuration,
                    endpoints ):
    return {
        'state': 0,
        'faults': [],
        'health': [],
        }

def attachNetwork( device,
                   configuration,
                   networks ):
    return {
        'state': 0,
        'faults': [],
        'health': [],
        }

def detachNetwork( device,
                   configuration,
                   networks ):
    return {
        'state': 0,
        'faults': [],
        'health': [],
        }
```

The following is an example invocation:

```
Function: deviceValidate

Arguments:
```

```
(
    {
    'creds': {
        'password': 'admin',
        'username': 'admin'
    },
    'host': '10.30.13.153','port': 443
}
u'1.0'
)
```

Function: deviceAudit

Arguments:

```
(
    {
    'host': '10.30.13.153',
    'port': 443,
    'creds': {
        'username': 'admin','password': 'admin'
    }
    },
    {
        (11,'','1_1'): {
        'state': 0,
        'label': 'int'
    },
    (11,'','1_2'): {
        'state': 0,
        'label': 'ext'
    },
    (11,'','1_3'): {
        'state': 0,
        'label': 'mgmt'
    }
    },
    {
        (4,'HighAvailabilityCfg','HA'): {
            'state': 2,'value': {
                (5,'peerIP','peerip'): {
                    'state': 2,
                    'value': '10.30.13.154'
                }
            }
        }
    }
)
```

Function: deviceCounters

Arguments:

```
(
    {
    'creds': {
        'password': 'insieme',
        'username': 'admin'
    },
    'host': '10.0.0.2',
    'port': 443,
    'virtual': False
    },
    {
    (11, '', 'eth1_0'): {
        'label': '',
        'state': 0
    },
    (11, '', 'eth1_1'): {
        'label': '',
        'state': 0
    }
    },
```

```
    {
    (4, 'HighAvailability', 'HA'): {
        'ackedState': 0,
        'state': 1,
        'transaction': 0,
        'value': {
           (5, 'id', 'id'): {
               'ackedState': 0,
               'state': 1,
               'transaction': 0,
               'value': '1'
           },
           (5, 'ipaddress', 'ipaddress'): {
               'ackedState': 0,
               'state': 1,
               'transaction': 0,
               'value': '10.0.0.3'
           },
           (5, 'netmask', 'netmask'): {
               'ackedState': 0,
               'state': 1,
               'transaction': 0,
               'value': '255.255.255.0'
           }
       }
   }
}
)
```

Function: clusterAudit

Arguments:

```
(
    {
    'name': 'Cluster1',
    'virtual': False,
    'devs': {
        'SampleDevice1': {
            'host': '10.30.13.153',
            'port': 443,
            'creds': {
                'username': 'admin',
                'password': 'admin'
            }
        }
    },
    'host': '10.30.13.153',
    'port': 443,
    'creds': {
        'username': 'admin',
        'password': 'admin'
    }
    },
    {
    (12, '', 'internal'): {
        'state': 0,
        'label': 'int'
    },
    (12, '', 'external'): {
        'state': 0,
        'label': 'ext'
    }
    },
    {
    (4,'SyslogConfig','syslogconfig'): {
        'state': 2,
        'value': {
            (5,'ipaddress','syslogip'): {
                'state': 2,
                'value': '10.168.62.100'
            }
        }
```

```
        },
        (4,'NTPConfig','ntpconfig'): {
            'state': 2,
            'value': {
                (5,'ipaddress','syslogip'): {
                    'state': 2,
                    'value': '10.168.62.1'
                }
            }
        }
    }
)
```

Function: clusterModify

Arguments:

```
(
    {
    'name': 'Cluster1',
    'virtual': False,
    'devs': {
        'SampleDevice1': {
            'host': '10.30.13.153',
            'port': 443,
            'creds': {
                'username': 'admin',
                'password': 'admin'
            }
        }
    },
    'host': '10.30.13.153',
    'port': 443,
    'creds': {
        'username': 'admin',
        'password': 'admin'
    }
    },
    {
    (12, '', 'internal'): {
        'state': 0,
        'label': 'int'
    },
    (12, '', 'external'): {
        'state': 0,
        'label': 'ext'
    }
    },
    {
    (4,'SyslogConfig','syslogconfig'): {
        'state': 2,
        'value': {
            (5,'ipaddress','syslogip'): {
                'state': 2,
                'value': '10.168.62.100'
            }
        }
    },
    (4,'NTPConfig','ntpconfig'): {
        'state': 2,
        'value': {
            (5,'ipaddress','syslogip'): {
                'state': 2,
                'value': '10.168.62.1'
            }
        }
    }
  }
)
```

Function: serviceModify

Arguments:

```
(
    {
    'Device1': {
        'creds': {
            'password': 'insieme',
            'username': 'admin'
        },
        'host': '10.0.0.2',
        'port': 443,
        'virtual': False
    }
    },
    'host': '10.0.0.1',
    'name': 'LB',
    'port': 443,
    'virtual': False
    },
    {
    (0, '', 5539): {
        'ackedState': 0,
        'ctxName': 'TenantActx1',
        'state': 1,
        'tenant': 'TenantA',
        'transaction': 0,
        'txid': 10000,
        'value': {
            (1, '', 4411): {
                'absGraph': 'WebGraph',
                'ackedState': 0,
                'state': 1,
                'transaction': 0,
                'value': {
                    (3, 'LoadBalancing', 'SLB'): {
                        'ackedState': 0,
                        'state': 1,
                        'transaction': 0,
                        'value': {
                            (2, 'external', 'external'): {
                                'ackedState': 0,
                                'state': 1,
                                'transaction': 0,
                                'value': {
                                    (9, '', 'LB_external_2424832_32771'): {
                                        'ackedState': 0,
                                        'state': 1,
                                        'target': 'LB_external_2424832_32771',
                                        'transaction': 0
                                    }
                                }
                            },
                            (2, 'internal', 'internal'): {
                                'ackedState': 0,
                                'state': 1,
                                'transaction': 0,
                                'value': {
                                    (9, '', 'LB_internal_2424832_49154'): {
                                        'ackedState': 0,
                                        'state': 1,
                                        'target': 'LB_internal_2424832_49154',
                                        'transaction': 0
                                    }
                                }
                            },
                            (4, 'externalNetwork', 'externalNetwork'): {
                                'ackedState': 0,
                                'connector': 'external',
                                'state': 1,
                                'transaction': 0,
                                'value': {
                                    (6, 'externalNetworkRel', 'externalNetworkRel'): {
                                        'ackedState': 0,
```

```
                                'state': 1,
                                'target': 'network/externalIP',
                                'transaction': 0
                            }
                        }
                    },
                    (4, 'internalNetwork', 'internalNetwork'): {
                        'ackedState': 0,
                        'connector': 'internal',
                        'state': 1,
                        'transaction': 0,
                        'value': {
                            (6, 'internalNetworkRel', 'internalNetworkRel'): {
                                'ackedState': 0,
                                'state': 1,
                                'target': 'network/internalIP',
                                'transaction': 0
                            }
                        }
                    },
                    (4, 'lbvserverCfg', 'lbvserverCfg'): {
                        'ackedState': 0,
                        'connector': 'external',
                        'state': 1,
                        'transaction': 0,
                        'value': {
                            (6, 'lbvserverRel', 'lbvserverRel'): {
                                'ackedState': 0,
                                'state': 1,
                                'target': 'lbvserver',
                                'transaction': 0
                            }
                        }
                    },
                    (4, 'serverpoolCfg', 'serverpoolCfg'): {
                        'ackedState': 0,
                        'state': 1,
                        'transaction': 0,
                        'value': {
                            (6, 'serverpoolRel', 'serverpoolRel'): {
                                'ackedState': 0,
                                'state': 1,
                                'target': 'serverpool',
                                'transaction': 0
                            }
                        }
                    },
                    (4, 'vipCfg', 'vipcfg'): {
                        'ackedState': 0,
                        'state': 1,
                        'transaction': 0,
                        'value': {
                            (6, 'vipRel', 'vipRel'): {
                                'ackedState': 0,
                                'state': 1,
                                'target': 'network/vipaddress',
                                'transaction': 0
                            }
                        }
                    }
                }
            }
        },
        (4, 'Network', 'network'): {
            'ackedState': 0,
            'state': 1,
            'transaction': 0,
            'value': {
                (4, 'ip', 'externalIP'): {
                    'ackedState': 0,
                    'state': 1,
                    'transaction': 0,
                    'value': {
```

```
                        (5, 'ipaddress', 'ipaddress'): {
                            'ackedState': 0,
                            'state': 1,
                            'transaction': 0,
                            'value': '20.0.0.1'
                        },
                        (5, 'netmask', 'netmask'): {
                            'ackedState': 0,
                            'state': 1,
                            'transaction': 0,
                            'value': '255.255.255.0'
                        }
                    }
                },
                (4, 'ip', 'internalIP'): {
                    'ackedState': 0,
                    'state': 1,
                    'transaction': 0,
                    'value': {
                        (5, 'ipaddress', 'ipaddress'): {
                            'ackedState': 0,
                            'state': 1,
                            'transaction': 0,
                            'value': '30.0.0.1'
                        },
                        (5, 'netmask', 'netmask'): {
                            'ackedState': 0,
                            'state': 1,
                            'transaction': 0,
                            'value': '255.255.255.0'
                        }
                    }
                },
                (4, 'ip', 'vipaddress'): {
                    'ackedState': 0,
                    'state': 1,
                    'transaction': 0,
                    'value': {
                        (5, 'ipaddress', 'ipaddress'): {
                            'ackedState': 0,
                            'state': 1,
                            'transaction': 0,
                            'value': '100.0.0.1'
                        },
                        (5, 'netmask', 'netmask'): {
                            'ackedState': 0,
                            'state': 1,
                            'transaction': 0,
                            'value': '255.255.255.255'
                        }
                    }
                }
            }
        },
        (4, 'lbvserver', 'lbvserver'): {
            'ackedState': 0,
            'state': 1,
            'transaction': 0,
            'value': {
                (5, 'ipmask', 'ipmask'): {
                    'ackedState': 0,
                    'state': 1,
                    'transaction': 0,
                    'value': '255.255.255.255'
                },
                (5, 'ipv4', 'ipv4'): {
                    'ackedState': 0,
                    'state': 1,
                    'transaction': 0,
                    'value': '100.0.0.1'
                },
                (5, 'name', 'name'): {
                    'ackedState': 0,
```

```
                'state': 1,
                'transaction': 0,
                'value': 'vserver1'
            },
            (5, 'port', 'port'): {
                'ackedState': 0,
                'state': 1,
                'transaction': 0,
                'value': '80'
            },
            (5, 'servicetype', 'servicetype'): {
                'ackedState': 0,
                'state': 1,
                'transaction': 0,
                'value': 'http'
            }
        }
    },
    (4, 'serverpool', 'serverpool'): {
        'ackedState': 0,
        'state': 1,
        'transaction': 0,
        'value': {
            (4, 'server', 'server1'): {
                'ackedState': 0,
                'state': 1,
                'transaction': 0,
                'value': {
                    (5, 'ip', 'ip'): {
                        'ackedState': 0,
                        'state': 1,
                        'transaction': 0,
                        'value': '30.0.0.2'
                    },
                    (5, 'port', 'port'): {
                        'ackedState': 0,
                        'state': 1,
                        'transaction': 0,
                        'value': '80'
                    }
                }
            },
            (5, 'serverpoolname', 'serverpoolname'): {
                'ackedState': 0,
                'state': 1,
                'transaction': 0,
                'value': 'webpool'
            },
            (5, 'type', 'type'): {
                'ackedState': 0,
                'state': 1,
                'transaction': 0,
                'value': 'http'
            }
        }
    },
    (7, '', '2424832_32771'): {
        'ackedState': 0,
        'state': 1,
        'tag': 436,
        'transaction': 0,
        'type': 1
    },
    (7, '', '2424832_49154'): {
        'ackedState': 0,
        'state': 1,
        'tag': 370,
        'transaction': 0,
        'type': 1
    },
    (8, '', 'LB_external_2424832_32771'): {
        'ackedState': 0,
        'encap': '2424832_32771',
```

```
                        'state': 1,
                        'transaction': 0,
                        'vif': 'LB_external'
                    },
                    (8, '', 'LB_internal_2424832_49154'): {
                        'ackedState': 0,
                        'encap': '2424832_49154',
                        'state': 1,
                        'transaction': 0,
                        'vif': 'LB_internal'
                    },
                    (10, '', 'LB_external'): {
                        'ackedState': 0,
                        'cifs': {
                            'Device1': 'eth1_0'
                        },
                        'state': 1,
                        'transaction': 0
                    },
                    (10, '', 'LB_internal'): {
                        'ackedState': 0,
                        'cifs': {
                            'Device1': 'eth1_1'
                        },
                        'state': 1,
                        'transaction': 0
                    }
                }
            }
        }
)
Function: serviceCounters

Arguments:

(
{'creds': {
    'password': 'insieme',
    'username': 'admin'
    },
    'devs': {
        'Device1': {
            'creds': {
                'password': 'insieme',
                'username': 'admin'
            },
            'host': '10.0.0.2',
            'port': 443,
            'virtual': False
        }
    },
    'host': '10.0.0.1',
    'name': 'LB',
    'port': 443,
    'virtual': False
},
{
(0, '', 5539): {
    'ctxName': 'TenantActx1',
    'state': 2,
    'tenant': 'TenantA',
    'value': {
        (1, '', 4411): {
            'absGraph': 'WebGraph',
            'state': 2,
            'value': {
            (3, 'LoadBalancing', 'SLB'): {
                'state': 2, 'value': {
                    (2, 'external', 'external'): {
                        'state': 2, 'value': {
                            (9, '', 'LB_external_2424832_32771'): {
                                'state': 0, 'target': 'LB_external_2424832_32771'
                            }
```

```
                        }
                    },
                    (2, 'internal', 'internal'): {
                        'state': 2,
                        'value': {
                            (9, '', 'LB_internal_2424832_49154'): {
                                'state': 0,
                                'target': 'LB_internal_2424832_49154'
                            }
                        }
                    },
                    (4, 'externalNetwork', 'externalNetwork'): {
                        'connector': 'external',
                        'state': 0,
                        'value': {
                            (6, 'externalNetworkRel', 'externalNetworkRel'): {
                                'state': 0,
                                'target': 'network/externalIP'
                            }
                        }
                    },
                    (4, 'internalNetwork', 'internalNetwork'): {
                        'connector': 'internal',
                        'state': 0,
                        'value': {
                            (6, 'internalNetworkRel', 'internalNetworkRel'): {
                                'state': 0,
                                'target': 'network/internalIP'
                            }
                        }
                    },
                    (4, 'lbvserverCfg', 'lbvserverCfg'): {
                        'connector': 'external',
                        'state': 0,
                        'value': {
                            (6, 'lbvserverRel', 'lbvserverRel'): {
                                'state': 0,
                                'target': 'lbvserver'
                            }
                        }
                    },
                    (4, 'serverpoolCfg', 'serverpoolCfg'): {
                        'state': 0,
                        'value': {
                            (6, 'serverpoolRel', 'serverpoolRel'): {
                                'state': 0,
                                'target': 'serverpool'
                            }
                        }
                    },
                    (4, 'vipCfg', 'vipcfg'): {
                        'state': 0,
                        'value': {
                            (6, 'vipRel', 'vipRel'): {
                                'state': 0,
                                'target': 'network/vipaddress'
                            }
                        }
                    }
                }
            }
        }
    },
    (4, 'Network', 'network'): {
        'state': 0,
        'value': {
        (4, 'ip', 'externalIP'): {
            'state': 0,
            'value': {
                (5, 'ipaddress', 'ipaddress'): {
                    'state': 0,
                    'value': '20.0.0.1'
                },
```

```
                (5, 'netmask', 'netmask'): {
                    'state': 0,
                    'value': '255.255.255.0'
                }
            }
        },
        (4, 'ip', 'internalIP'): {
            'state': 0,
            'value': {
                (5, 'ipaddress', 'ipaddress'): {
                    'state': 0,
                    'value': '30.0.0.1'
                },
                (5, 'netmask', 'netmask'): {
                    'state': 0,
                    'value': '255.255.255.0'
                }
            }
        },
        (4, 'ip', 'vipaddress'): {
            'state': 0,
            'value': {
            (5, 'ipaddress', 'ipaddress'): {
                'state': 0,
                'value': '100.0.0.1'
            },
            (5, 'netmask', 'netmask'): {
                'state': 0,
                'value': '255.255.255.255'
            }
        }
    }
}
},
(4, 'lbvserver', 'lbvserver'): {
    'state': 0,
    'value': {
    (5, 'ipmask', 'ipmask'): {
        'state': 0,
        'value': '255.255.255.255'
    },
    (5, 'ipv4', 'ipv4'): {
        'state': 0,
        'value': '100.0.0.1'
    },
    (5, 'name', 'name'): {
        'state': 0,
        'value': 'vserver1'
    },
    (5, 'port', 'port'): {
        'state': 0,
        'value': '80'
    },
    (5, 'servicetype', 'servicetype'): {
        'state': 0,
        'value': 'http'
    }
}
},
(4, 'serverpool', 'serverpool'): {
    'state': 0,
    'value': {
        (4, 'server', 'server1'): {
            'state': 0,
            'value': {
                (5, 'ip', 'ip'): {
                    'state': 0,
                    'value': '30.0.0.2'
                },
                (5, 'port', 'port'): {
                    'state': 0,
                    'value': '80'
                }
```

```
                }
            },
            (5, 'serverpoolname', 'serverpoolname'): {
                'state': 0,
                'value': 'webpool'
            },
            (5, 'type', 'type'): {
                'state': 0,
                'value': 'http'
            }
        }
},
(7, '', '2424832_32771'): {
    'state': 0,
    'tag': 436,
    'type': 1
},
(7, '', '2424832_49154'): {
    'state': 0,
    'tag': 370,
    'type': 1
},
(8, '', 'LB_external_2424832_32771'): {
    'encap': '2424832_32771',
    'state': 0,
    'vif': 'LB_external'
},
(8, '', 'LB_internal_2424832_49154'): {
    'encap': '2424832_49154',
    'state': 0,
    'vif': 'LB_internal'
},
(10, '', 'LB_external'): {
    'cifs': {
        'Device1': 'eth1_0'
    },
    'state': 0
},
(10, '', 'LB_internal'): {
    'cifs': {
        'Device1': 'eth1_1'
    },
    'state': 0
}
}
}
)
```

Function: serviceHealth

Arguments:

```
(
{'creds': {
    'password': 'insieme',
    'username': 'admin'
},
'devs': {
    'Device1': {
        'creds': {
            'password': 'insieme',
            'username': 'admin'
        },
        'host': '10.0.0.2',
        'port': 443,
        'virtual': False
    }
    },
    'host': '10.0.0.1',
    'name': 'LB',
    'port': 443,
    'virtual': False
    },
    {(0, '', 5539): {
```

```
'ctxName': 'TenantActx1',
'state': 2,
'tenant': 'TenantA',
'value': {
    (1, '', 4411): {
        'absGraph': 'WebGraph',
        'state': 2,
        'value': {
            (3, 'LoadBalancing', 'SLB'): {
                'state': 2,
                'value': {
                    (2, 'external', 'external'): {
                        'state': 2,
                        'value': {
                            (9, '', 'LB_external_2424832_32771'): {
                                'state': 0,
                                'target': 'LB_external_2424832_32771'
                            }
                        }
                    },
                    (2, 'internal', 'internal'): {
                        'state': 2,
                        'value': {
                            (9, '', 'LB_internal_2424832_49154'): {
                                'state': 0,
                                'target': 'LB_internal_2424832_49154'
                            }
                        }
                    },
                    (4, 'externalNetwork', 'externalNetwork'): {
                        'connector': 'external',
                        'state': 0,
                        'value': {
                            (6, 'externalNetworkRel', 'externalNetworkRel'): {
                                'state': 0,
                                'target': 'network/externalIP'
                            }
                        }
                    },
                    (4, 'internalNetwork', 'internalNetwork'): {
                        'connector': 'internal',
                        'state': 0,
                        'value': {
                            (6, 'internalNetworkRel', 'internalNetworkRel'): {
                                'state': 0,
                                'target': 'network/internalIP'
                            }
                        }
                    },
                    (4, 'lbvserverCfg', 'lbvserverCfg'): {
                        'connector': 'external',
                        'state': 0,
                        'value': {
                            (6, 'lbvserverRel', 'lbvserverRel'): {
                                'state': 0,
                                'target': 'lbvserver'
                            }
                        }
                    },
                    (4, 'serverpoolCfg', 'serverpoolCfg'): {
                        'state': 0,
                        'value': {
                            (6, 'serverpoolRel', 'serverpoolRel'): {
                                'state': 0,
                                'target': 'serverpool'
                            }
                        }
                    },
                    (4, 'vipCfg', 'vipcfg'): {
                        'state': 0,
                        'value': {
                            (6, 'vipRel', 'vipRel'): {
                                'state': 0,
```

```
                                'target': 'network/vipaddress'
                            }
                        }
                    }
                }
            }
        }
    },
    (4, 'Network', 'network'): {
        'state': 0,
        'value': {
            (4, 'ip', 'externalIP'): {
                'state': 0,
                'value': {
                    (5, 'ipaddress', 'ipaddress'): {
                        'state': 0,
                        'value': '20.0.0.1'
                    },
                    (5, 'netmask', 'netmask'): {
                        'state': 0,
                        'value': '255.255.255.0'
                    }
                }
            },
            (4, 'ip', 'internalIP'): {
                'state': 0,
                'value': {
                    (5, 'ipaddress', 'ipaddress'): {
                        'state': 0,
                        'value': '30.0.0.1'
                    },
                    (5, 'netmask', 'netmask'): {
                        'state': 0,
                        'value': '255.255.255.0'
                    }
                }
            },
            (4, 'ip', 'vipaddress'): {
                'state': 0,
                'value': {
                    (5, 'ipaddress', 'ipaddress'): {
                        'state': 0,
                        'value': '100.0.0.1'
                    },
                    (5, 'netmask', 'netmask'): {
                        'state': 0,
                        'value': '255.255.255.255'
                    }
                }
            }
        }
    },
    (4, 'lbvserver', 'lbvserver'): {
        'state': 0,
        'value': {
            (5, 'ipmask', 'ipmask'): {
                'state': 0,
                'value': '255.255.255.255'
            },
            (5, 'ipv4', 'ipv4'): {
                'state': 0,
                'value': '100.0.0.1'
            },
            (5, 'name', 'name'): {
                'state': 0,
                'value': 'vserver1'
            },
            (5, 'port', 'port'): {
                'state': 0,
                'value': '80'
            },
            (5, 'servicetype', 'servicetype'): {
                'state': 0,
```

```
                        'value': 'http'
                    }
                }
            },
            (4, 'serverpool', 'serverpool'): {
                'state': 0,
                'value': {
                    (4, 'server', 'server1'): {
                        'state': 0,
                        'value': {
                            (5, 'ip', 'ip'): {
                                'state': 0,
                                'value': '30.0.0.2'
                            },
                            (5, 'port', 'port'): {
                                'state': 0,
                                'value': '80'
                            }
                        }
                    },
                    (5, 'serverpoolname', 'serverpoolname'): {
                        'state': 0,
                        'value': 'webpool'
                    },
                    (5, 'type', 'type'): {
                        'state': 0,
                        'value': 'http'
                    }
                }
            },
            (7, '', '2424832_32771'): {
                'state': 0,
                'tag': 436,
                'type': 1
            },
            (7, '', '2424832_49154'): {
                'state': 0,
                'tag': 370,
                'type': 1
            },
            (8, '', 'LB_external_2424832_32771'): {
                'encap': '2424832_32771',
                'state': 0,
                'vif': 'LB_external'
            },
            (8, '', 'LB_internal_2424832_49154'): {
                'encap': '2424832_49154',
                'state': 0,
                'vif': 'LB_internal'
            },
            (10, '', 'LB_external'): {
                'cifs': {
                    'Device1': 'eth1_0'
                },
                'state': 0
            },
            (10, '', 'LB_internal'): {
                'cifs': {
                    'Device1': 'eth1_1'
                },
                'state': 0
            }
        }
    }
}
)
```

Function: attachEndpoint

Arguments:

```
(
```

```
{'creds': {
    'password': 'insieme', 'username': 'admin'
},
'devs': {
    'Device1': {
        'creds': {
        'password': 'insieme', 'username': 'admin'
    },
    'host': '10.0.0.2',
    'port': 443,
    'virtual': False
}
},
'host': '10.0.0.1',
'name': 'LB',
'port': 443,
'virtual': False
},

{(0, '', 5539): {
    'ctxName': 'TenantActx1',
    'state': 2,
    'tenant': 'TenantA',
    'value': {
        (1, '', 4411): {
            'absGraph': 'WebGraph',
            'state': 2,
            'value': {
                (3, 'LoadBalancing', 'SLB'): {
                    'state': 2,
                    'value': {
                        (2, 'external', 'external'): {
                            'state': 2,
                            'value': {
                                (9, '', 'LB_external_2424832_32771'): {
                                    'state': 0,
                                    'target': 'LB_external_2424832_32771'
                                }
                            }
                        },
                        (2, 'internal', 'internal'): {
                            'state': 2,
                            'value': {
                                (9, '', 'LB_internal_2424832_49154'): {
                                    'state': 0,
                                    'target': 'LB_internal_2424832_49154'
                                }
                            }
                        },
                        (4, 'externalNetwork', 'externalNetwork'): {
                            'connector': 'external',
                            'value': {
                                (6, 'externalNetworkRel', 'externalNetworkRel'): {
                                    'state': 0,
                                    'target': 'network/externalIP'
                                }
                            }
                        },
                        (4, 'internalNetwork', 'internalNetwork'): {
                            'connector': 'internal',
                            'state': 0,
                            'value': {
                                (6, 'internalNetworkRel', 'internalNetworkRel'): {
                                    'state': 0,
                                    'target': 'network/internalIP'
                                }
                            }
                        },
                        (4, 'lbvserverCfg', 'lbvserverCfg'): {
                            'connector': 'external',
                            'state': 0,
                            'value': {
                                (6, 'lbvserverRel', 'lbvserverRel'): {
```

```
                                    'state': 0,
                                    'target': 'lbvserver'
                                }
                            }
                        },
                        (4, 'serverpoolCfg', 'serverpoolCfg'): {
                            'state': 0,
                            'value': {
                            (6, 'serverpoolRel', 'serverpoolRel'): {
                                'state': 0,
                                'target': 'serverpool'
                            }
                            }
                        },
                        (4, 'vipCfg', 'vipcfg'): {
                            'state': 0,
                            'value': {
                                (6, 'vipRel', 'vipRel'): {
                                    'state': 0,
                                    'target': 'network/vipaddress'
                                }
                            }
                        }
                    }
                }
            }
        },
        (4, 'Network', 'network'): {
            'state': 0,
            'value': {
                (4, 'ip', 'externalIP'): {
                    'state': 0,
                    'value': {
                        (5, 'ipaddress', 'ipaddress'): {
                            'state': 0,
                            'value': '20.0.0.1'
                        },
                        (5, 'netmask', 'netmask'): {
                            'state': 0,
                            'value': '255.255.255.0'
                        }
                    }
                },
                (4, 'ip', 'internalIP'): {
                    'state': 0,
                    'value': {
                        (5, 'ipaddress', 'ipaddress'): {
                            'state': 0,
                            'value': '30.0.0.1'
                        },
                        (5, 'netmask', 'netmask'): {
                            'state': 0,
                            'value': '255.255.255.0'
                        }
                    }
                },
                (4, 'ip', 'vipaddress'): {
                    'state': 0,
                    'value': {
                        (5, 'ipaddress', 'ipaddress'): {
                            'state': 0,
                            'value': '100.0.0.1'
                        },
                        (5, 'netmask', 'netmask'): {
                            'state': 0,
                            'value': '255.255.255.255'
                        }
                    }
                }
            }
        },
        (4, 'lbvserver', 'lbvserver'): {
            'state': 0,
```

```
        'value': {
            (5, 'ipmask', 'ipmask'): {
                'state': 0,
                'value': '255.255.255.255'
            },
            (5, 'ipv4', 'ipv4'): {
                'state': 0,
                'value': '100.0.0.1'
            },
            (5, 'name', 'name'): {
                'state': 0,
                'value': 'vserver1'
            },
            (5, 'port', 'port'): {
                'state': 0,
                'value': '80'
            },
            (5, 'servicetype', 'servicetype'): {
                'state': 0,
                'value': 'http'
            }
        }
    },
    (4, 'serverpool', 'serverpool'): {
        'state': 0,
        'value': {
            (4, 'server', 'server1'): {
                'state':
                'value': {
                    (5, 'ip', 'ip'): {
                        'state': 0,
                        'value': '30.0.0.2'
                    },
                    (5, 'port', 'port'): {
                        'state': 0,
                        'value': '80'
                    }
                }
            },
            (5, 'serverpoolname', 'serverpoolname'): {
                'state': 0,
                'value': 'webpool'
            },
            (5, 'type', 'type'): {
                'state': 0,
                'value': 'http'
            }
        }
    },
    (7, '', '2424832_32771'): {
        'state': 0,
        'tag': 436,
        'type': 1
    },
    (7, '', '2424832_49154'): {
        'state': 0,
        'tag': 370,
        'type': 1
    },
    (8, '', 'LB_external_2424832_32771'): {
        'encap': '2424832_32771',
        'state': 0,
        'vif': 'LB_external'
    },
    (8, '', 'LB_internal_2424832_49154'): {
        'encap': '2424832_49154',
        'state': 0,
        'vif': 'LB_internal'
    },
    (10, '', 'LB_external'): {
        'cifs': {
            'Device1': 'eth1_0'
        },
```

```
        'state': 0
    },
    (10, '', 'LB_internal'): {
        'cifs': {
            'Device1': 'eth1_1'
        },
        'state': 0
    }
}
}
},
[
    {
    'addr': '34.34.34.12',
    'conn': 'internal'
    }
]
)
```

The `endpoints` dictionary in the API callout contains the following attributes:

- `'addr'`—The IP address of the endpoint that attached to an EPG.
- `'conn'`—The connector to which the EPG is attached directly or indirectly through other function nodes.

```
Function: attachNetwork
Arguments:

(
{'creds': {
    'password': 'insieme',
    'username': 'admin'
},
'devs': {
    'Device1': {
        'creds': {
            'password': 'insieme',
            'username': 'admin'
        },
        'host': '10.0.0.2',
        'port': 443,
        'virtual': False
    }
},
'host': '10.0.0.1',
'name': 'LB',
'port': 443,
'virtual': False
},

{(0, '', 5539): {
    'ctxName': 'TenantActx1',
    'state': 2,
    'tenant': 'TenantA',
    'value': {
        (1, '', 4411): {
            'absGraph': 'WebGraph',
            'state': 2,
            'value': {
                (3, 'LoadBalancing', 'SLB'): {
                    'state': 2,
                    'value': {
                        (2, 'external', 'external'): {
                            'state': 2,
                            'value': {
                                (9, '', 'LB_external_2424832_32771'): {
                                    'state': 0,
                                    'target': 'LB_external_2424832_32771'
                                }
                            }
                        },
```

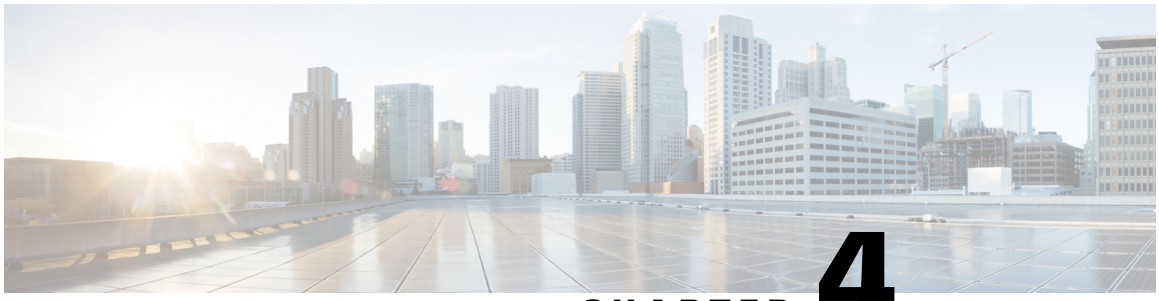
```
                    (2, 'internal', 'internal'): {
                        'state': 2,
                        'value': {
                            (9, '', 'LB_internal_2424832_49154'): {
                                'state': 0,
                                'target': 'LB_internal_2424832_49154'
                            }
                        }
                    },
                    (4, 'externalNetwork', 'externalNetwork'): {
                        'connector': 'external',
                        'state': 0,
                        'value': {
                            (6, 'externalNetworkRel', 'externalNetworkRel'): {
                                'state': 0,
                                'target': 'network/externalIP'
                            }
                        }
                    },
                    (4, 'internalNetwork', 'internalNetwork'): {
                        'connector': 'internal',
                        'state': 0,
                        'value': {
                            (6, 'internalNetworkRel', 'internalNetworkRel'): {
                                'state': 0,
                                'target': 'network/internalIP'
                            }
                        }
                    },
                    (4, 'lbvserverCfg', 'lbvserverCfg'): {
                        'connector': 'external',
                        'state': 0,
                        'value': {
                            (6, 'lbvserverRel', 'lbvserverRel'): {
                                'state': 0,
                                'target': 'lbvserver'
                            }
                        }
                    },
                    (4, 'serverpoolCfg', 'serverpoolCfg'): {
                        'state': 0,
                        'value': {
                            (6, 'serverpoolRel', 'serverpoolRel'): {
                                'state': 0,
                                'target': 'serverpool'
                            }
                        }
                    },
                    (4, 'vipCfg', 'vipcfg'): {
                        'state': 0,
                        'value': {
                            (6, 'vipRel', 'vipRel'): {
                                'state': 0,
                                'target': 'network/vipaddress'
                            }
                        }
                    }
                }
            }
        }
    },
    (4, 'Network', 'network'): {
        'state': 0,
        'value': {
            (4, 'ip', 'externalIP'): {
                'state': 0,
                'value': {
                    (5, 'ipaddress', 'ipaddress'): {
                        'state': 0,
                        'value': '20.0.0.1'
                    },
                    (5, 'netmask', 'netmask'): {
                        'state': 0,
```

```
                        'value': '255.255.255.0'
                    }
                }
            },
            (4, 'ip', 'internalIP'): {
                'state': 0,
                'value': {
                    (5, 'ipaddress', 'ipaddress'): {
                        'state': 0,
                        'value': '30.0.0.1'
                    },
                    (5, 'netmask', 'netmask'): {
                        'state': 0,
                        'value': '255.255.255.0'
                    }
                }
            },
            (4, 'ip', 'vipaddress'): {
                'state': 0,
                'value': {
                    (5, 'ipaddress', 'ipaddress'): {
                        'state': 0,
                        'value': '100.0.0.1'
                    },
                    (5, 'netmask', 'netmask'): {
                        'state': 0,
                        'value': '255.255.255.255'
                    }
                }
            }
        }
    },
    (4, 'lbvserver', 'lbvserver'): {
        'state': 0,
        'value': {
            (5, 'ipmask', 'ipmask'): {
                'state': 0,
                'value': '255.255.255.255'
            },
            (5, 'ipv4', 'ipv4'): {
                'state': 0,
                'value': '100.0.0.1'
            },
            (5, 'name', 'name'): {
                'state': 0,
                'value': 'vserver1'
            },
            (5, 'port', 'port'): {
                'state': 0,
                'value': '80'
            },
            (5, 'servicetype', 'servicetype'): {
                'state': 0,
                'value': 'http'
            }
        }
    },
    (4, 'serverpool', 'serverpool'): {
        'state': 0,
        'value': {
            (4, 'server', 'server1'): {
                'state': 0,
                'value': {
                    (5, 'ip', 'ip'): {
                        'state': 0,
                        'value': '30.0.0.2'
                    },
                    (5, 'port', 'port'): {
                        'state': 0,
                        'value': '80'
                    }
                }
            },
```

```
                (5, 'serverpoolname', 'serverpoolname'): {
                    'state': 0,
                    'value': 'webpool'
                },
                (5, 'type', 'type'): {
                    'state': 0,
                    'value': 'http'
                }
            }
        },
        (7, '', '2424832_32771'): {
            'state': 0,
            'tag': 436,
            'type': 1
        },
        (7, '', '2424832_49154'): {
            'state': 0,
            'tag': 370,
            'type': 1
        },
        (8, '', 'LB_external_2424832_32771'): {
            'encap': '2424832_32771',
            'state': 0,
            'vif': 'LB_external'
        },
        (8, '', 'LB_internal_2424832_49154'): {
            'encap': '2424832_49154',
            'state': 0,
            'vif': 'LB_internal'
        },
        (10, '', 'LB_external'): {
            'cifs': {
                'Device1': 'eth1_0'
            },
            'state': 0
        },
        (10, '', 'LB_internal'): {
            'cifs': {
                'Device1': 'eth1_1'
            },
            'state': 0
        }
    }
}
},
[
   {
    'addr': '34.34.34.0/24',
    'conn': 'internal'
   }
]
)
```

The `networks` dictionary in the API callout contains the following attributes:

- `'addr'`—Identifies the subnet configured in the bridge domain or EPG associated with the connector.
- `'conn'`—The connector to which the EPG is attached directly or indirectly through other function nodes.



CHAPTER 4

Fabric Connectivity

Registering Devices

To manage service nodes through the Application Policy Infrastructure Controller (APIC), the administrator must explicitly register the service devices. During the registration step, you must provide the following information:

- Topology information: How device interfaces are connected to the fabric leaf nodes.
- Label interfaces: Based on the device requirements. The labels are used by the APIC to bind an interface with a connector for specific functions that are provided by the service device.
- IP address and port information: Information that is needed to connect to the device.
- Username and password: Credentials used for configuring the device.

The sample firewall device specification below defines three labels for interfaces:

- `Inside`: Identifies network interfaces that are more trusted (secure).
- `Outside`: Identifies network interfaces that are less trusted.
- `Management`: Identifies the interface used for management connectivity.

The labels are defined in the device specification using the `vnsMIfLbl` tag. All interfaces on the firewall device are categorized into one of the types defined by the device specification.

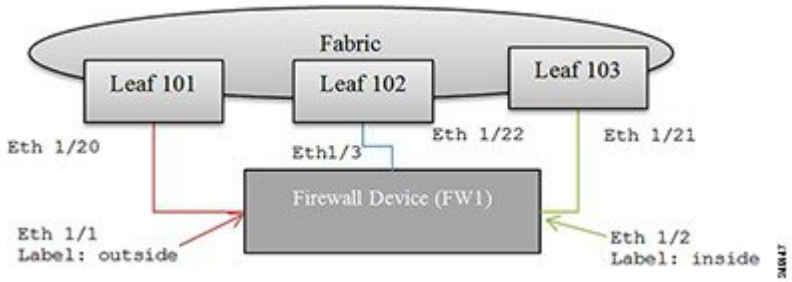
Note The APIC does not check if the interface actually exists on the device.

The following example shows a Northbound XML post for registering a device with the APIC. You can either post the request as shown or use the APIC GUI to register the device:

```
<polUni>
    <fvTenant
        dn="uni/tn-Tenant1"
        name="Tenant1">
        <vnsLDevVip name="Firewall-1">

            <vnsLIf name="external">
                <vnsRsMetaIf tDn="uni/infra/mDev-CISCO-ASA-1.0.1.16/mIfLbl-external"/>
                <vnsRsCIfAtt tDn="uni/tn-Tenant1/lDevVip-Firewall/cDev-ASA/cIf-Eth1_1"/>
            </vnsLIf>
            <vnsLIf name="internal">
                <vnsRsMetaIf tDn="uni/infra/mDev-CISCO-ASA-1.0.1.16/mIfLbl-internal"/>
                <vnsRsCIfAtt tDn="uni/tn-Tenant1/lDevVip-Firewall/cDev-ASA/cIf-Eth1_2"/>
            </vnsLIf>
            <vnsCDev name="FW1">

                <vnsCIf name="Eth1_1">
                       <vnsRsCIfPathAtt tDn="topology/pod-1/paths-101/pathep-[eth1/20]"/>
                </vnsCIf>
                <vnsCIf name="Eth1_2">
                       <vnsRsCIfPathAtt tDn="topology/pod-1/paths-102/pathep-[eth1/21]"/>
                </vnsCIf>
                <vnsCIf name="Eth1_3">
                       <vnsRsCIfPathAtt tDn="topology/pod-1/paths-103/pathep-[eth1/22]"/>
                </vnsCIf>

                <vnsCMgmt name="devMgmt"
                          host="192.168.78.62"
                          port="80"
                          />

               <vnsCCred name="username"
                          value="admin"
                          />
               <vnsCCredSecret name="password"
                          value="insieme"
                          />
            </vnsCDev>
        </vnsLDevVip>
    </fvTenant>
</polUni>
```

The following figure shows the topology of registering a device.

Figure 6: Topology of Registering a Device



The three steps the to register the device are as follows:

1 Register the interfaces:

   • Eth 1/1: Labeled as `outside`. Connected to Leaf node 101, Eth 1/20.

- Eth 1/2: Labeled as `inside`. Connected to Leaf node 102, Eth 1/21.
- Ethernet 1/3: Labeled as `management`. Connected to Leaf node 103, Eth 1/22.

2 Provide the management IP address (192.168.78.62) and port (80) to reach to the device.

3 Provide the username and password credentials to use for communicating with the device.

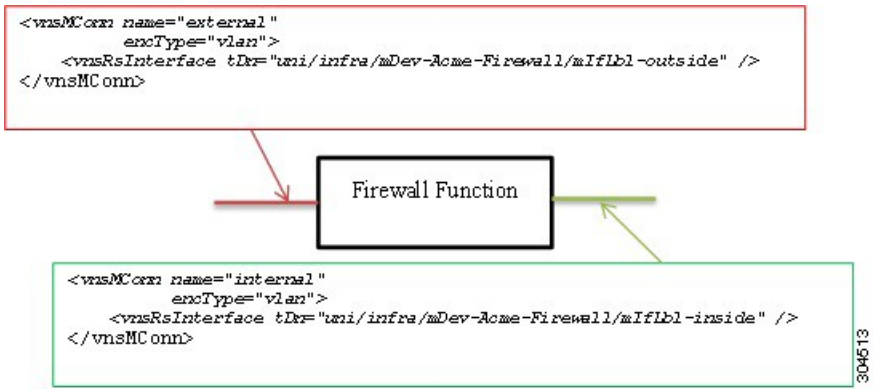
Connectors

The connectors for a `vnsMFunc` define connectivity between two or more function nodes or between a function node and the fabric within a graph. A connector has the following attributes:

- `name`: Identifies a specific connector.
- `encType`: Defines whether the packets are tagged with a VLAN header or are VXLAN encapsulated.
- `vnsRsInterface`: If the connector provides connectivity to the fabric, this interface associates the connector to an interface type on the device.

In the following figure, two connectors are associated to a firewall function. The first connector represents connectivity to an external or outside network, and the second connector represents connectivity to an internal or inside network on the firewall device. Both are tagged with a VLAN header.

Figure 7: Connectors Associated to a Firewall Function



Service Graphs

A service graph is an ordered set of functions between a set of terminals. You can manually create a service graph using the GUI or CLI, or create one programmatically using the Application Policy Infrastructure Controller (APIC) Northbound Service Integration API. A function within the graph might require one or more parameters and have one or more connectors.

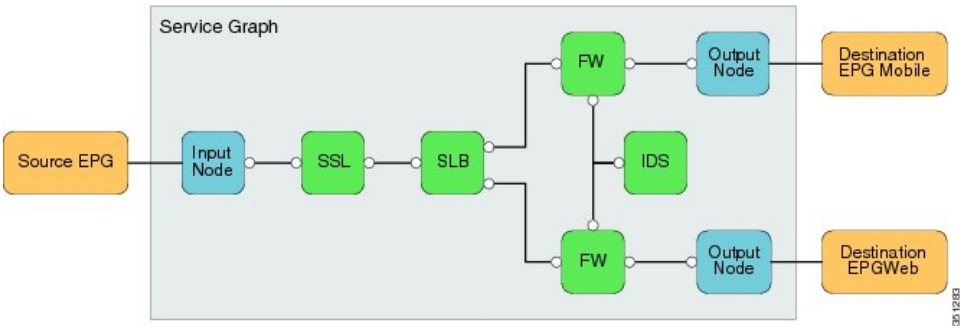
A service graph represents the network using the following elements:

- Function Nodes (green)—A function applied to traffic such as a transform (SSL termination, VPN gateway), filter (firewalls), or terminal (intrusion detection systems)

- Terminal Nodes (blue)—Input and outputs from the service graph
- Connector (white)—Input and output from a node
- Connections—How traffic is forwarded through the network

The following figure shows a service graph.

Figure 8: Service Graph



Note Although this generic service graph shows two output nodes, the fabric supports only a single input node and a single output node from a service graph at this time.

The `serviceModify` function is used to instantiate the network and function configurations.

Graph Rendering

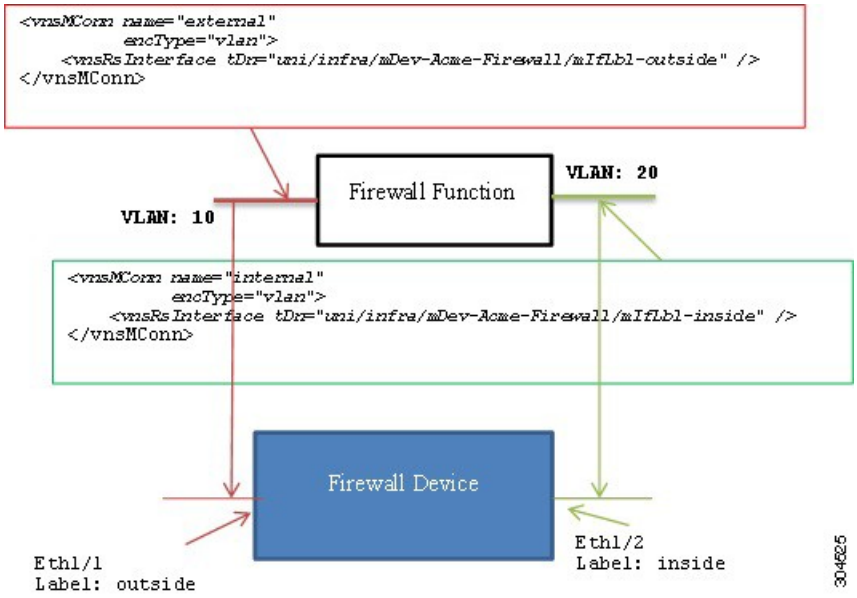
When instantiating a function on the device, the Application Policy Infrastructure Controller (APIC) does the following:

- Assigns a VLAN/VXLAN ID for the connector. The APIC checks whether the previous node has been allocated a VLAN/VXLAN ID. It either uses the previous node value or allocates a new tag for the connector. The `encType` indicates whether the VLAN/VXLAN ID is allocated.
- Uses interface relation and device interface label information to associate an interface to a connector. In the previous figure, rendering a firewall function on the firewall device will result in these bindings:
  - Connector labeled as `external`: Device Interface Eth1/1
  - Connector labeled as `internal`: Device Interface Eth1/2

You should enable or bind the VLAN or VXLAN tags that are assigned to the connector to the associated interfaces.

In the following figure, the APIC assigns VLAN 10 to the outside connector and VLAN 20 to the inside connector. The device script must configure VLAN 10 on interface Eth1/1 and bind it to the firewall function. Similarly, the device script must configure VLAN 20 on interface Eth1/2 and bind it to the firewall function.

Figure 9: A Firewall With Bound VLANs



Device Script Interface

The VXLAN/VLAN encapsulation, interface, and association between the interface and VLAN/VXLAN are passed as device configuration parameters to the device script. In the previous figure, the Application Policy Infrastructure Controller (APIC) provides the following information in the device configuration:

- Encapsulation: VLAN 10, VLAN 20
- Interface: Eth1/1, Eth1/2
- Association of encapsulation to a logical interface:
  - 'Firewall-1_outside_1553': (VLAN -10, Eth1/1)
  - 'Firewall-1_inside_7697': (VLAN-20, Eth1/2)

The encapsulation tags, interfaces, and association are destroyed only when all functions across all graphs using the tag are deleted from the device. By providing encapsulation information in the device configuration, the APIC ensures the encapsulation tag is not deleted from the device until all functions that refer to the tag are deleted. The following example shows a sample dictionary:

```
Configuration =
{
 (0, '', <LdevInstance>): {
   ...
   (7, '', <encap-instance>): {'state': 1, 'tag': TagValue, 'type': TagType},
   (7, '', <encap-instance>): {'state': 1, 'tag': TagValue, 'type': 0},
   (8, '', <encap-association-Instance>): {
```

```
          'state': 1,
          'encap': <encapInstance>,
          'vif': <LogicalInterfaceInstanceID>}
   },
   (8, '', <encap-association-Instance>): {
          'state': 1,
          'encap': <encapInstance>,
          'vif': <LogicalInterfaceInstanceID>},
   },
   (10, '', <LogicalInterfaceInstance>): {
          'state': 0,
          'cifs': {
             'cDevInstance': <Interface Value>
          }
   },
   (10, '', <LogicalInterfaceInstance>): {
          'state': 0,
          'cifs': {
             'cDevInstance': <Interface Value>
          }
   },
 }
}
```

Legend:

```
The dictionary format is as follows:

(type, key, name): {
   'state': StateValue,
   'device': CDevName,
   'connector': connectorValue,
   'value': Parameter Value,
}
```

type:

```
7 – Encap Instance [Encap Type=0 (VLAN), Encap Type=1 (VXLAN))]
                     Encap Tag = VLAN ID or VNID (VXLAN case)
8 – VEncapAss (Device Interface and Encap (VXLAN/VLAN) association)
10 – VIF (logical interface) – Identifies interface on the device.
```

CDevName: Identifies a specific device within a cluster node. This attribute is not applicable to encap, VIF, or vEncapAss.

connectorValue: Identifies the connector to which this parameter should be bound. This attribute is not applicable to encap, VIF, or vEncapAss.

value: Value of the parameter.

StateValue:

```
0 – No change
1 – Create
2 – Modify
3 - Destroy
```

The connectors within the function are related to the encapsulation association parameter specified in the device configuration. The encapsulation association parameter binds the connector to a specific VLAN/VXLAN tag and interface. In the above example, the dictionary for the function would contain the following connector information:

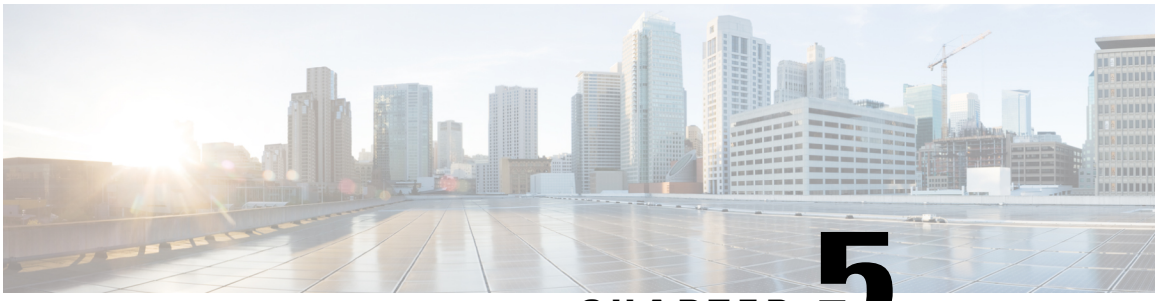
```
Configuration =
{
    (0, '', 'Firewall-1'): {
      'state': 2,
      'value': {
         (7, '', '1553'): {'state': 1, 'tag': 10, 'type': 0},
         (7, '', '7697'): {'state': 1, 'tag': 20, 'type': 0},
         (8, '', 'Firewall-1_outside_1553'): {'state': 1,
                                              'encap': '1553',
                                              'vif': 'Firewall-1_outside'},
         (8, '', 'Firewall-1_inside_7697'): {'state': 1,
```

```
                                          'encap': '1553',
                                          'vif': 'Firewall-1_inside'},

        (10, '', 'Firewall-1_outside'): {'state': 1,
                                         'cifs': {'FW2': 'Eth1/1' }
                                        },
        (10, '', 'Firewall-1_inside'):  {'state': 1,
                                         'cifs': {'FW': 'Eth1/2'}}
                                        }
        (1, '', '4552'): {
           'state': 1,
           'value': {
             (3, 'Firewall', 'F1'): {
                 'state': 1,
                 'value': {
                     (2, 'external', 'conn1'): { 'state': 1,
                         'value': {
                            ('9', '', 'outside_1553'): {
                                 'state': 1,
                                 'value': 'Firewall-1-outside_1553'
                            },
                         }
                     },
                     (2, 'internal', 'conn2'): { 'state': 1
                         'value': {
                            ('9', '', 'inside_7697'): {
                                 'state': 1,
                                 'value': 'Firewall-1-inside_7697'
                            },
                         }
                     },
                     (4, 'Firewall-Config', 'FW-Config 1'): {
                          'state': 1,
                          'value' : {
                              (5, 'Param-1', ''): { 'state': 1, 'value': value },
                          . . .
                          }
                     },
                 },
             },
           },
        },
      },
}
```

Based on the above dictionary, you should configure the device script to do the following:

- Enable VLAN 10 on interface Eth1/1:
  - Create subinterface Eth1/1.10 with encap VLAN 10
  - Add Eth1/1 to VLAN 10
- Enable VLAN 20 on interface Eth1/2:
  - Create subinterfaceEth1/2.20 with encap VLAN 20
  - Add Eth1/2 to VLAN 20

Note The connector value is a dictionary that allows each device within the cluster to use different interfaces.



CHAPTER 5

Service Insertion Support

Health Monitoring

The Application Policy Infrastructure Controller (APIC) can query the health status of devices and services from 0 (not operational) to 100 (fully functional) by using the following functions:

- **deviceHealth—**Returns the health of a service device.
- **serviceHealth—**Returns the health of a service function, endpoint, or endpoint group.

The APIC periodically calls the deviceHealth API. The device script can return the health of the device. The health can be a normalized value that is computed by the script based on querying CPU utilization, memory utilization, and other critical resources, such as the state of the power supply or HA status. The value of the health can be a score between 0 to 100. 0 indicates that the device is not operational and 100 indicates that the device is fully functional.

The following example shows a return value from the deviceHealth API:

```
def deviceHealth (device,interfaces,configuration):
     ...
    return {
       'state': 0,'faults': [],'health': [([],80)]
    }
```

Note The health value is a normalized value based on memory utilization, CPU utilization, and the number of connections. The health element can be written as a part of the device modify or device audit API.

The device script can report the health of all the functions provided by the service node using the serviceHealth API. The APIC periodically invokes the serviceHealth API with the service node configuration.

The serviceHealth API is defined as follows:

```
serviceHealth (device, configuration)
```

The serviceHealth parameters are defined as follows:

- `device`—A dictionary providing the device IP and credentials. The APIC uses this information to connect to the service node.
- `configuration`—The service node configuration. The APIC pushes the entire device configuration across all graphs during the serviceHealth poll.

The serviceHealth API can query the device and accumulate information regarding the health of a service. For example, the script may collect CPU utilization, memory utilization, or the number of connections associated with the service. The script uses the data collected from the device to compute a normalized value between 0 – 100, representing the health of the service. 0 indicates bad health, and 100 indicates that the service is in a good state.

The APIC expects the script to return the service health as a list of `(path, Service Health)` tuples.

- `path`: A list of tuples identifying a specific service function within the device:

  ```
  Path = [ (type, key, name) (type, key, name) ...]
  ```

The `state` can take one of the following values:

- `OK`—Success.
- `TRANSIENT`—Temporary failure. This typically indicates a transient issue, such as a device connectivity problem.
- `PERMANENT`—This typically indicates a persistent fault due to a misconfiguration or device failure.
- `AUDIT`—The device script can request the APIC to issue an audit callout to resolve configuration issues found on the device.
- `True`—Success. The device script must use the `OK` state value instead of `True`.
- `False`—This state is equivalent to `PERMANENT`. The device script must use the `PERMENANT` or `TRANSIENT` state values to indicate a fault instead of using the `False` state.

Cisco recommends that you use `OK`, `TRANSIENT`, `PERMANENT`, or `AUDIT` as `state` values, rather than the boolean `True` and `False` values.

Faults are returned as a list of `(object, fault)` tuples, and are updated in the system as follows:

- If the script returns a `state` value of `OK`, `True`, `False`, `AUDIT`, or `PERMANENT`—Faults are replaced with the set of faults in the return value. Any previous fault that was not reported will be implicitly cleared. For example, if you had a fault on `obj1` but on the second attempt you return a fault only on `obj2`, the `obj1` fault is cleared and the APIC now reports a fault only on `obj2`.
- If the script returns a `state` value of `TRANSIENT`—Faults are augmented. For example, if the script had reported a fault on `obj1` with the `TRANSIENT` state, the APIC will re-issue the API callout. On the subsequent callout, if the script returns a fault on `obj2`, the APIC reports a faults on both `obj1` and `obj2`.

The script should return a `TRANSIENT` state along with the fault on the device when the connection to the device breaks. Otherwise, it can report transient fault on objects if the configuration could not be applied due to a temporary device resource issue.

The script must return a PERMANENT state along with faults on objects when the configuration could not be applied because of an invalid parameter or configuration issue. The user must change the configuration to clear the fault.

The script must return a PERMANENT state with the fault on the device if the configuration parsing fails.

The script should return an OK state if the configuration could be successfully applied.

The following example illustrates a return value in the case in which the device is configured with multiple instances of an SLB function:

```
device =
       {'creds': {'password': 'admin', 'username': 'admin'},
        'devs': {'cdev1': {'creds': {'password': 'admin',
                                     'username': 'admin'},
                           'host': '172.21.158.182',
                           'port': 80},
                 'cdev2': {'creds': {'password': 'admin',
                                     'username': 'admin'},
                                     'host': '172.21.158.224',
                                     'port': 80}},
        'host': '1.1.1.3',
        'name': 'cluster1',
        'port': 80}

configuration =
     {(0, '', 4447): {'state': 1, 'transaction': 10000,
                      'value': {(1, '', 4208): {'state': 1,
                                'transaction': 10000,
                                'value': {(3, 'SLB', 'Node1'): {'state': 1,
                                                                'transaction': 10000,
                                                                'value': { ... }
                                                               }
                                         }
                                }
                      }
     }
```

For each function:

- Query the physical devices.
- Determine a score for each device based on certain criteria relevant to the device, such as connections, CPU usage, or errors.
- Determine a score for the cluster. For an Active-Active cluster, determine a score using a minimum set of nodes and normalize the scores from each device. For an Active-Standby cluster, use active nodes for the score as well as the high availability state.

Note The health element can also be returned as part of the return dictionary for the service modify or service audit API.

- Return the health score the cluster and each device using this format:

```
func1 = [(0, '', 4447), (1, '', 4208), (3, 'SLB', 'Node1')]
   return { 'state': OK,
            'health': [ (func1, 100) ],
              'devs': {
                     'cdev1': {
                         'state': True,
                         'health': [ (func1, 100) ]
                     },
                     'cdev2': {
                         'state': True,
                         'health': [ (func1, 100) ]
```

```
                    },
                }
            }
```

Faults

The APIC has a comprehensive infrastructure for alarms, notifications and logging that you can use within the device script. The device package developer can define a set of faults using the `MDfct` object. The `MDfct` objects are contained within an `MDfcts` object. The APIC allows a device package developer to define one instance of `MDfcts` that is contained within `MDev`. The `MDfcts` object can contain one or more `MDfct` object. The hierarchy of the `MDfcts` object and the `MDfct` object is as follows:

```
<polUni>
    <infraInfra>
       <vnsMDev ...>
            <vnsMDfcts>
                <vnsMDfct ...>
                    <vnsRsDfctToCat.../>
                </vnsMDfct>
            </vnsMDfcts>
      ...
       </vnsMDev>
    </infraInfra>
</polUni>
```

Each `MDfct` object describes a class of fault that the device script can return, and provides additional information about the fault to the user. The `MDfct` object has following attributes:

Attribute	Mandatory	Description
`Code`	Yes	The `code` uniquely identifies a class of defect. The device script must return a `code` value along with a fault string.
`Description`	Yes	This field describes the fault. The `description` field is used by the APIC GUI to provide help to the user. A device package developer should provide an accurate description of the fault. The `description` field size is limited to maximum of 512 characters.
`recAct`	Yes	This field specifies the recommended corrective action for the user to resolve the fault. This field is limited to maximum of 512 characters.
`htmlFile`	No	The device package developer can add a link to additional help on the fault.

APIC classifies faults into four categories or severity levels:

- warning(1)
- minor(2)
- major(3)
- critical(4)

The device package developer should associate the fault that is described by the `MDfct` object to one of the severity levels. The relation to a severity level is specified using the `vnsRsDfctToCat` object, as shown in the following example:

```
<vnsRsDfctToCat tDn="dfctCats/dfctCat-warning"/>
<vnsRsDfctToCat tDn="dfctCats/dfctCat-minor"/>
<vnsRsDfctToCat tDn="dfctCats/dfctCat-major"/>
<vnsRsDfctToCat tDn="dfctCats/dfctCat-critical"/>
```

Following is an example of defining a fault code in the device package:

```
    <vnsMDfcts>
        <vnsMDfct code="10"
                  descr="This is a description of the fault 10"
                  recAct="Recommended action for resolving fault 10"
                  htmlFile="http://somewhere/file.html">
                 <vnsRsDfctToCat tDn="dfctCats/dfctCat-minor"/>
        </vnsMDfct>
        <vnsMDfct code="20"
                  descr="This is a description of the fault 20"
                  recAct="Recommended action for resolving fault 20"
                  htmlFile="http://somewhere/file.html">
                 <vnsRsDfctToCat tDn="dfctCats/dfctCat-major"/>
       </vnsMDfct>
   </vnsMDfcts>
```

The device script can return faults in the return dictionary for any API. The APIC allows scripts to return 'faults'. The return dictionary contains a python list of tuples. Each tuple in the fault list must contain the following elements:

```
(object-path, code, fault string)
```

- `object-path`—The object path uniquely identifies an object in the configuration dictionary that caused a fault. The script can raise a fault on one or more objects that are passed in the configuration that require user intervention to correct the issue.

  To raise a fault on the device in a specific tenant context, the script can return the object path as the `vDev` that is passed in the dictionary. For example:

  ```
  (0, '', 4133)
  ```

- `code`—The script must return a fault code that is defined in the `MDfct` object in the device package.
- `Fault string`—The device script can optionally add a fault string to provide more specific information about the fault. This fault string can be null or can be alphanumeric string with up to 512 characters.

The APIC reports faults that are explicitly returned by the device script. If the script stops raising a fault on an object for any state other than the `TRANSIENT` state, the APIC implicitly clears the fault. The faults returned by the cluster API are associated to the `vnsLDevVip` object. Faults returned by the device API are associated to the `vnsCDev` object. Faults returned on all other APIs are associated to the `vnsVDev` object or to a specific configuration object that is identified by the path returned in the fault tuple.

If the script returns a state as `TRANSIENT`, the faults are augmented. That is, the APIC will not clear any previously raised faults. The faults persist untill the script returns any other state with a different fault string.

The faults that are returned by the device script can be queried through the APIC north bound API. The APIC GUI also reports faults that are returned by the device script. The APIC augments the severity to the fault code and string returned by the API.

The following example defines a fault code:

```
<vnsMDfcts>
    <vnsMDfct code="10"
              descr="Invalid VIP address "
              recAct="Please enter valid unicast VIP address"
              htmlFile="http://insieme.net/SLBCfgExample.html">
             <vnsRsDfctToCat tDn="dfctCats/dfctCat-major"/>
    </vnsMDfct>
</vnsMDfcts>
```

```
def serviceModify(device, configuration):
    path = [
        (0,'',1234), (1,'',2345), (3, 'Function', 'SLB'),
        (4, 'folder', 'network'), (5,'param','vip')
    ]
    code = 10
    message = '225.0.0.1'

    faults = [ (path,code,message) ]

    return {
      'state': Config.SUCCESS,
      'faults': faults,
      'health': []
    }
```

The APIC will display the fault and append the following text:

```
(APIC defect category defined in MDfct object in device package,
 Defect-code defined in MDfct object in device package,
 Defect-Description defined in MDfct object in device package,
 Fault string passed in the return dictionary by the device script)
```

In the `serviceModify` fault example, the APIC will report following fault string on the VIP object:

```
Major Fault: 10, "Invalid VIP address ": '225.0.0.1'
```

Note If the device script encounters a fault, such as the device credentials that were passed in device dictionary are invalid or mismatch, the connectivity to the device is lost, or there is an issue that affects the entire device rather than a specific configuration, the device script can raise a fault on `VDev` by returning an empty path list ("[]").

For example:

```
([],10,"Device Credentials invalid")
```

Counters

The Application Policy Infrastructure Controller (APIC) can query packet counters using the `deviceCounter` and `serviceCounters` functions, which returns a dictionary with transmit and receive counters for packets, errors, and drops for interfaces and connectors that are associated with a service function, respectively.

The `deviceCounters` API returns interface statistics from a specific device and is defined as follows:

```
def deviceCounters( device,interfaces,configuration ):
```

The following example is a `deviceCounters` call:

```
def deviceCounters( device, interfaces, configuration ):
        return {
              'state': 0,
              'counters': [ ([cif], counters), ...]

counters: {
   'rxpackets': <rxpackets>,
   'rxerrors': <rxerrors>,
   'rxdrops': <rxdrops>,
   'txpackets': <txpackets>
   'txerrors': <txerrors>
   'txdrops': <txdrops>
}
```

`cif` is a (type, key, value) tuple that identifies an interface.

For example:

```
  eth0Count = {
   'rxpackets': 100,
   'rxerrors': 0,
   'rxdrops': 0
   'txpackets': 10
   'txerrors': 4
   'txdrops': 2
  }


  return {
        'state': 0,
        'counters': [ ([(11, '', 'eth0')], eth0Count) ]
  }
```

The `serviceCounters` API returns statistics for connectors associated with a service function and is defined as follows:

```
serviceCounters (device, configuration)
```

The `serviceCounters` parameters are defined as follows:

- `device`: A dictionary providing the device IP and credentials. APIC uses this information to connect to the service node.
- `configuration`: The service node configuration.

The following example illustrates how APIC can query packet counters for service functions:

```
def serviceCounters(device, configuration):
   externalIntferface, = [(0, 'Firewall', 4384), (1, '', 4432), (3, 'Firewall-Func', 'FW-1'),
 (2, 'external', 'external1') ]
    internalInterface = [(0, 'Firewall', 4384) (1, '', 4432) (3, 'Firewall-Func', 'FW-1'),
(2, 'internal','internal1') ]

   Firewall-1-External-Counters = (externalInterface,
                                    {  'rxpackets': 100,
                                       'rxerrors': 0,
                                       'rxdrops': 0
                                       'txpackets': 100
                                       'txerrors': 4
                                       'txdrops': 2} )

   Firewall-1-Internal-Counters =  (internalInterface,
                                     {  'rxpackets': 100,
                                        'rxerrors': 0,
                                        'rxdrops': 0
                                        'txpackets': 100
                                        'txerrors': 4
```

```
                                          'txdrops': 2} )

   Counters = [ Firewall-1-External-Counters,
                Firewall-1-Internal-Counters ]
   return {
     'state': 0,
     'counters': Counters
   }
```

Open Shortest Path First

The Application Policy Infrastructure Controller (APIC) supports the Open Shortest Path First (OSPF) and OSPF version 3 routing protocols for route peering. The APIC provides native support for configuring OSPF parameters. A device package developer does not need to model the OSPF configuration in the device model. The OSPF configuration is passed to the device script in service API callouts as part of the configuration dictionary along with other function configurations. The OSPF configuration is split into an interface-specific configuration and an OSPF process configuration. The following object types in the dictionary are used for OSPF configuration:

```
Type = Insieme.Fwk.Enum(
    ...
    OSPFDEV=13,
    OSPFVENCAPASC=16,
    ...
)
```

Object Type	Description
OSPFDEV	Defines the OSPF process configuration.
OSPFVENCAPASC	Defines the interface-specific OSPF configuration.

Open Shortest Path First Interface Configuration

The Open Shortest Path First (OSPF) interface configuration is embedded as a field within Encap Association:

```
(8, '', 'ADCCluster1_outside_2850816_32771'): {
        'ackedState': 0,

        'encap': '2850816_32771',

        'state': 1,

        'transaction': 0,

        'vif': 'ADCCluster1_outside',

        'OspfVIfCfg': {...}
}
```

On a virtual device, the OSPF interface configuration should be applied on the interface that is identified by the `vif` field. On a physical device, the configuration should be applied on the `<interface, vlan>` tuple that is identified by the `vif` and `encap` fields.

The Application Policy Infrastructure Controller (APIC) allows users to configure the following OSPF parameters on the interface:

Parameter	Description
Area	OSPF area to which this interface is associated. If the device does not support per-interface area configuring, the script can ignore this field.
mtu	Interface MTU in bytes. The value is an integer from 64 to 9000 bytes. The default value is 1500 bytes.
authKey	OSPF authentication-key password. This parameter is a plain text password that is used when the authentication type is configured as `plain` (Type 1). This parameter is ignored when the authentication type is `MD5`.
authKeyId	Key identifier for the password. The key-ID is a number from 1 to 255. This parameter allows you to configure one or more plain text passwords. This parameter is ignored when authentication type is `MD5`.
authType	Indicates the authentication type if OSPF authentication is enabled. The following values are valid: • `None`—No Authentication (Type 0). • `Plain`—Simple plain text password authentication (Type 1). • `message-digest`—Use message digest authentication (Type 2).
md5Key	A list of dictionaries that identifies key-id/encrypted password pairs. This parameter is applicable only when the authentication is configured as message-digest (Type 2). Create the list in the following format: `[ { 'md5KeyId' : 1, 'md5Password': 'encrypted_password' }, { 'md5KeyId' : 2, 'md5Password': 'encrypted_password' } ]` • `md5KeyId`—Identifies the MD5 key-ID. A valid value is an integer from 1 to 255. • `md5Password`— A string that represents an encrypted MD5 password.
cost	Interface cost. A valid value is an integer from 1 to 65535.
deadIntvl	Interval defined in seconds after which a neighbor is declared dead. A valid value is an integer from 1 to 8192.
helloIntvl	Time in seconds between HELLO packets. A valid value is an integer from 1 to 8192.
addressFamily	A tuple identifying whether OSPF configuration is applied to IPv4 or IPv6. The tuple can take the following values: `[ 'ipv4', 'ipv6' ]`

Parameter	Description
nwT	A string that identifies the network type. The following values are valid: • `p2p`—Interface is OSPF point-to-point network. • `broadcast`—Interface is OSPF broadcast multi-access network.
ctrl	Defines the OSPF control parameters for the interface. This is a list that can contain one or more of following values: • `passive`—Suppress routing updates on this interface. • `mtu-ignore`—Ignore MTU mismatch in DBD packets. By default MTU check is enabled. MTU check should be disregarded only when this attribute is set. • `bfd`—Enable BFD on this interface.
priority	Network Priority. A valid value is an integer from 0 to 255.
rexmitIntvl	Time between retransmitting lost link state advertisements. The time is represented in seconds. A valid value is an integer from 1 to 8192.
xmitDelay	Link state transmit delay. The value is specified in seconds. A valid value is an integer from 1 to 8192.
State	OSPF state. The following values are valid: • `0`—Indicates that none of the fields defined within the dictionary has changed. • `1`—Indicates that the `OspfVIFCfg` is newly created. All fields in the dictionary should be applied on interface or <interface, encap> as applicable. • `2`—Indicates that one or more fields within the `OspfVIfCfg` has changed. The device script should push the modified parameters to the device. The APIC does not indicate which field has changed. It is left to the user to either apply all fields or query the device and identify the parameter that has change. • `3`—Indicates that `OspfVIfCfg` is deleted. The device script should remove the OSPF configuration from the interface or <interface, vlan> tuple as applicable.

The following is an example OSPF interface configuration dictionary:

```
'OspfVIfCfg': {
            (16, '', u'OspfVIfCfg'): {
                            'area': 111,
                            'authKey': u'passphrase',
                            'authKeyId': 1,
                            'authType': u'message-digest',
                            'md5Key' : [ { 'md5KeyId': 1,
                                           'md5Password': 'passphrase1',
                                         },
                                         { 'md5KeyId': 2,
                                           'md5Password': 'passphrase2'
                                         }
```

```
                                        ],
                                'cost': 1,
                                'deadIntvl': 45,
                                'helloIntvl': 15,
                                'addressFamily': [ 'ipv4', 'ipv6' ],
                                'nwT': 'p2p',
                                'ctrl': [ 'mtu-ignore' ],
                                'prio': 1,
                                'rexmitIntvl': 5,
                                'state': 1,
                                'value': {},
                                'xmitDelay': 1
                        }
}
```

Open Shortest Path First Authentication

Open Shortest Path First (OSPF) provides the following authentication options:

Authentication Type	APIC Dictionary	Example IOS Commands
No Authentication (Type 0)	• `authType = None` • `authKey`—This field is ignored. • `authKeyId`—This field is ignored.	`no ospf authentication`
Plain Text Authentication (Type 1)	• `authType = plain` • `authKey = password` • `authKeyId`—This field is ignored.	`ospf authentication` `ospf authentication-key` *`authKey`*
MD5 Authentication (Type 2)	• `authType = message-digest` • `authKey`—This field is ignored. • `authKeyId`—This field is ignored. • `md5Key: [ { 'md5KeyId': This field provides the key-id. 'md5Password': This field provides the encrypted password. } ]`	`ospf authentication message-digest` `ospf message-digest-key` *`md5KeyId`* `md5` *`md5Password`*

Open Shortest Path First Process Configuration

The Open Shortest Path First (OSPF) process configuration is defined within the `vdev` dictionary. The configuration is as follows:

```
{ (0, '', 5849): {'ackedState': 0,
                  'state': 1,
                  'transaction': 0,
                  'txid': 10000,
                  'value': {
                        ...
                        (13, '', u'OspfDevCfg2'): { ...}
                     }
                }
}
```

OSPF device configuration is shared across multiple graphs. The Application Policy Infrastructure Controller (APIC) supports the following OSPF configuration:

Parameter	Description
Area	OSPF area ID parameter. A valid value is a decimal from 0 to 4294967295.
Enable	A boolean that Specifies whether the OSPF process is administrative enabled or disabled. If enable is set to `True`, the OSPF process is administratively enabled.
areaType	A string that specifies Area type. The following values are valid: • `nssa`—Specifies a NSSA area. • `stub`—Specifies a stub area.
redistribute	A list of strings that configures OSPF to accept routing information from another routing protocol and redistributes that information through the OSPF network. The following values are valid: • `bgp`—Specifies redistribute routes learned through the border gateway protocol (BGP). • `connected`—Specifies redistribute directly to connected subnets. • `ospf`—Specifies redistribute routes learned through different OSPF processes. • `static`—Specifies redistribute static routes.
areaCtrl	A list that defines parameters for the area. The following values are valid: • `redistribute`—Enables redistribution routes for this area. • `no-redistribute`—Disables redistribution routes for this area. • `no-summary`—Disables the sending of summary Link-State Advertisement (LSA) into not-so-stubby area (NSSA). Applies only to NSSA.
rtrId	An IPv4 address that is the router ID for the OSPF process.

Parameter	Description
processID	OSPF process ID. A valid value is an integer from 1 to 65535. The APIC supports only one process per device. This field is optional; the APIC does not need to pass a processID field in the dictionary. When processID is absent from the dictionary, the script pushes a hardcoded default value to the device. The default can be set to "1".

The following is an example OSPF process configuration dictionary:

```
(13, '', u'OspfDevCfg2'): {
    'area': 1,
    'enable': True,
    'areaType': 'nssa',
    'areaCtrl': [ 'redistribute' ],
    'redistribute' : [ 'static', 'connected' ],
    'rtrId': '11.0.1.1',
    'state': 1,
    'processID' : 1,
    'value': {}
}
```

Open Shortest Path First Network Configuration

To identify the interfaces on which Open Shortest Path First (OSPF) runs and to define the area ID for those interfaces, service devices might require the following network configuration for each OSPF process:

`network` *`ip-address wildcard-mask`* `area` *`area-id`*

The *ip-address* and *wildcard-mask* is used as a match string. If the interface IP address matches the network statement *ip-address* - *wildcard-mask*, OSPF is enabled on the matching interface. The *area-id* identifies the area attachment for the interface.

To enable OSPF on an interface, the device script can auto-generate the network configuration by combining the interface IP address configuration and the OSPFVENCAPASC (type 16) configuration that is passed in the configuration dictionary.

Typically, a device model defines a network address object for configuring an IP address and subnet mask on an interface or VLAN. The configuration dictionary passed by the Application Policy Infrastructure Controller (APIC) contains the network address (IP address and subnet mask) that is defined by the device model, along with the associated connector information. The APIC provides the OSPFENCAPASC configuration that is associated with the interface or VLAN in the configuration dictionary. The device script can combine the network configuration information and OSPFVENCAPASC configuration that is associated with the corresponding connector to auto-generate the network statement for OSPF. The script can use the area ID information from the OSPFVENCAPASC object. The *ip-address* and *wildcard-mask* can be derived from the network address configuration. The script can use the area configuration that is provided within the OSPFDEV configuration to select the OSPF process. The auto-generated network statement can be added to the appropriate OSPF process configuration that is identified by the area.

Note The auto-generation of network configuration is required only if the device depends on the configuration for enabling OSPF on an interface. If the device supports directly enabling OSPF on an interface, the device can just use the OSPFVENCAPASC configuration dictionary. The device can ignore the auto-generation step that is suggested in this section.

Expected Open Shortest Path First Behavior from a Device Script

The following behavior is expected when using Open Shortest Path First (OSPF) in a device script:

- `OSPFDEV (13, '', u' ...')` state is `create`—The device script creates an OSPF process on the device if it does not exist. If the process exists, the device script updates the configuration based on the parameters that are pushed in the dictionary. The device script ignores the network prefix state and adds all network prefixes that are passed in the dictionary.
- `OSPFDEV (13, '', u' ...')` state is `modify`—Updates the OSPF parameters that are pushed in the configuration dictionary. Based on the prefix state, the device script adds to or remote network prefixes from the process. The prefix state should be used only when the `OSPFDEV` state is `modify`.
- `OSPFDEV (13, '', u' ...')` state is `delete`—Removes the OSPF process from the device.

Multiple Open Shortest Path First Areas

Each Open Shortest Path First (OSPF) `DevCfg` represents the configuration for a given area. If you configure multiple areas on the device, the Application Policy Infrastructure Controller (APIC) pushes this information as multiple `DevCfg`s with the same process ID. The device script should configure multiple areas under the same process.

Multiple Open Shortest Path First Processes

Many service devices can support multiple Open Shortest Path First (OSPF) processes, although the Application Policy Infrastructure Controller (APIC) model does not support multiple processes. The APIC configures multiple contexts within a single OSPF process.

Border Gateway Protocol

The Application Policy Infrastructure Controller (APIC) provides a built-in model for border gateway protocol (BGP) configuration. This configuration can be associated with the device or function on any layer 4 to layer 7 service device that is capable of supporting BGP. The BGP configuration is pushed to the device script as a device configuration during service API callouts. The BGP configuration is under `vdev` and is shared across multiple graphs and functions. The following object types in the dictionary are used for BGP configuration:

```
Type = Insieme.Fwk.Enum(
    ...
    BGPDEV=17,
    BGPVENCAPASC=20,
    ...
)
```

Object Type	Description
BGPDEV	Defines the BGP process configuration.
BGPVENCAPASC	Defines the interface-specific BGP configuration.

The configuration is defined within `vdev` as follows:

```
{ (0, '', 5849): {'ackedState': 0,
        'state': 1,
        'transaction': 0,
        'txid': 10000,
        'value': {
            ...
            (17, '', u'BGPDevCfg'):{ { ...}
        }
   }
}
```

Note Both OSPF and BGP configurations can co-exist. If the device supports both protocols and you have configured both of the protocols, the APIC passes the BGP and OSPF configurations as part of the configuration dictionary during service API callout.

Border Gateway Protocol Interface Configuration

The Border Gateway Protocol (BGP) interface configuration is embedded as a field within Encap Association:

```
(8, '', 'ADCCluster1_outside_2850816_32771'): {
        'ackedState': 0,
        'encap': '2850816_32771',
        'state': 1,
        'transaction': 0,
        'vif': 'ADCCluster1_outside',
        'BgpVIfCfg': {...}
}
```

On a virtual device, the BGP interface configuration should be applied on the interface that is identified by the `vif` field. On a physical device, the configuration should be applied on the `<interface, vlan>` tuple that is identified by the `vif` and `encap` fields.

The Application Policy Infrastructure Controller (APIC) allows users to configure the following BGP parameters on the interface:

Parameter	Description
mtu	Interface MTU in bytes. The value is an integer from 64 to 9000 bytes. The default value is 1500 bytes.

The following is an example BGP interface configuration dictionary:

```
'BgpVIfCfg': {
        (16, '', u'BgpVIfCfg'): {
                'mtu': 1500,
        }
}
```

Border Gateway Protocol Process Configuration

The Application Policy Infrastructure Controller (APIC) pushes the following parameters for border gateway protocol (BGP) process configuration:

Parameter	Description
localAS	Autonomous system number assigned to the service device.
rtrId	An IP address on the routher that is the BGP router identifier (ID) for the BGP process. The BGP router ID is used in the BGP algorithm for determining the best path to a destination.
state	An enum that represents the BGP configuration state. The following values are valid: • `0`—No change. • `1`—Create. • `2`—Modify. • `3`—Destroy.
Timers	A dictionary of BGP timers that must be configured for the BGP process. The following timers are supported and passed by the APIC: • `keepalive`—Frequency (in seconds) with which the device software sends keepalive messages to its peer. A valid value is an integer from 0 to 65535. • `hold`—Interval (in seconds) after not receiving a keepalive message that the software declares a peer dead. A valid value is an integer from 0 to 65535. • `stale`—Configures the `stalepath` timer (in seconds) for a BGP graceful restart. This timer sets the maximum time period that the local router will hold stale paths for a restarting peer. All stale paths are deleted after this timer expires. A valid value is an integer from 1 to 3600 seconds. The following Cisco IOS command configures the `stalepath` timer: `bgp graceful-restart stalepath-time` *`seconds`*

Parameter	Description
Context	

Parameter	Description
	Identifies a routing context. If the device does not support multiple routing contexts, the device script can ignore the context information. The APIC will not push more than one context on a single context device. The context is a dictionary that can contain the following elements: • `name`—The context name as configured on the APIC. • `ipv4`—A dictionary that contains the following elements: ◦ `networks`—A list of IPv4 networks to be added or removed from a BGP process. ◦ `redistribute`—A list of strings that defines route redistribution through a BGP network. Configures BGP to accept routing information from another routing protocol and redistribute that information through the BGP network. The `redistribute` element has following values: • `bgp`—Indicates redistribute routes that are learned through a different BGP process. • `ospf`—Indicates redistribute routes that are learned through a different OSPF process. • `connected`—Indicates redistribute directly to connected subnets. • `static`—Indicates redistribute to static routes. ◦ `neighbors`—A dictionary that identifies a neighbor. • `ipv6`—A dictionary that contains the following elements: ◦ `networks`—A list of IPv6 networks to be added or removed from a BGP process. ◦ `redistribute`—A list of strings that configures BGP to accept routing information from another routing protocol

Parameter	Description
	and redistribute that information through the BGP network. The `redistribute` element has following values: • `bgp`—Indicates redistribute routes that are learned through a different BGP process. • `ospf`—Indicates redistribute routes that are learned through a different OSPF process. • `connected`—Indicates redistribute directly to connected subnets. • `static`—Indicates redistribute to static routes. ◦ `neighbors`—A dictionary that identifies a neighbor. Each address family configuration is grouped as follows: `'context': { 'name': 'commonctx', 'ipv4': { 'networks': [{ 'ipaddress': '110.0.0.0', 'netmask': 24, 'area': 0, 'state': 1, }, { 'ipaddress': '120.0.0.0', 'netmask': 24, 'area': 0, 'state': 1, }], 'redistribute' : ['static', 'connected'], }, 'ipv6': { 'networks': [{ 'ipaddress': '2001::0', 'netmask': 64, 'area': 111 }, { 'ipaddress': '2002::0', 'netmask': 64, 'area': 111 },], 'redistribute' : ['static', 'connected'], },` The network dictionary is identical to OSPF configuration.

Parameter	Description
Neighbor	Each address family has one neighbor dictionary. The neighbor dictionary contains one or more BGP peer IP addresses and the associated configuration. Each peer is represented as a dictionary that is indexed by its IP address within the neighbor dictionary. The following attributes are configured for each neighbor: • `remoteAS`—Peers autonomous system number. • `adminStatus`—Whether peering with is enabled or disabled with a specific neighbor. • `state`—An enum that indicates whether a script should update or create a new neighbor configuration on the device. The following values are valid: ◦ `0`—No change. ◦ `1`—Create. ◦ `2`—Modify. ◦ `3`—Destroy.

The following is an example BGP configuration dictionary:

```
(17, '', u'BGPDevCfg'): {'ackedState': 0,
                         'localAS': 6501,
                         'rtrId': '11.0.1.1',
                         'state': 1,
                         'timers': {
                             'keepalive': 10,
                             'hold': 100,
                             'neighborMinHold': 100,
                          },
                         'context': {
                             'name': 'cokeCtx',
                             'ipv4': {
                                 'neighbor': {
                                       '10.0.0.2': {
                                           'remoteAS': 6501,
                                           'adminStatus': 'enabled'
                                       }
                                 },
                                 'networks': [
                                       { 'ipaddress': '110.0.0.0',
                                          'netmask': 24,
                                       },
                                       { 'ipaddress': '120.0.0.0',
                                          'netmask': 24,
                                       }
                                 ],
                                 'redistribute' : [ 'static', 'connected' ],
                             },
                         },
                         'transaction': 0,
                         'value': {}}
```

Expected Border Gateway Protocol Behavior from a Device Script

The following behavior is expected when using the border gateway protocol (BGP) in a device script:

- `BGPDEV (17, '', u' ...')` state is set to `create`—The device script creates a BGP process on the device if it does not exist. If the process exists, the device script updates the configuration based on the parameters that are pushed in the dictionary. The device script ignores the network prefix state and adds all network prefixes that are passed in the dictionary.
- `BGPDEV (17, '', u' ...')` state is set to `modify`—Updates the BGP parameters that are pushed in the configuration dictionary. Based on the prefix state, the device script adds to or remote network prefixes from the process. The prefix state should be used only when the `BGPDEV` state is `modify`. Similarly, use the per neighbor state to add or remove a neighbor's configuration from the device.
- `BGPDEV (17, '', u' ...')` state is set to `delete`—Removes the BGP process from the device.

The Application Policy Infrastructure Controller (APIC) allows multiple BGP processes to be configured on a single device. The device dictionary can have more than one `BGPDEV` configuration dictionary under a `VDev`.

The following is an example of a complete BGP dictionary:

```
configuration = {(0, '', 5849): {'ackedState': 0,
    'state': 1,
    'transaction': 0,
    'txid': 10000,
    'value': {(1, '', 4474): {'ackedState': 0,
        'state': 1,
        'transaction': 0,
        'value': {(3, 'Firewall', 'Firewall'): {'ackedState': 0,
            'state': 1,
            'transaction': 0,
            'value': {(2, 'external', 'outside'): {'ackedState': 0,
                'state': 1,
                'transaction': 0,
                'value': {(9, '', 'ADCCluster1_outside_2850816_32771'): {
                    'ackedState': 0,
                    'state': 1,
                    'target': 'ADCCluster1_outside_2850816_32771',
                    'transaction': 0}}},
                (2, 'internal', 'inside'): {'ackedState': 0,
                    'state': 1,
                    'transaction': 0,
                    'value': {(9, '', 'ADCCluster1_inside_2850816_32772'): {
                        'ackedState': 0,
                        'state': 1,
                        'target': 'ADCCluster1_inside_2850816_32772',
                        'transaction': 0}}},
                (4, 'external_network', 'external_network'): {'ackedState': 0,
                    'connector': 'outside',
                    'state': 1,
                    'transaction': 0,
                    'value': {(6, 'external_network_key', 'external_network_key'): {
                        'ackedState': 0,
                        'state': 1,
                        'target': 'network/snip2',
                        'transaction': 0}}},
                (4, 'external_route', 'external_route'): {'ackedState': 0,
                    'connector': 'outside',
                    'state': 1,
                    'transaction': 0,
                    'value': {(6, 'external_route_rel', 'external_route_rel'): {
                        'ackedState': 0,
                        'state': 1,
                        'target': 'network/route2',
                        'transaction': 0}}},
```

```
                (4, 'internal_network', 'internal_network'): {'ackedState': 0,
                    'connector': 'inside',
                    'state': 1,
                    'transaction': 0,
                    'value': {(6, 'internal_network_key', 'internal_network_key'): {
                        'ackedState': 0,
                        'state': 1,
                        'target': 'network/snip1',
                        'transaction': 0}}},
                (4, 'internal_route', 'internal_route'): {'ackedState': 0,
                    'connector': 'inside',
                    'state': 1,
                    'transaction': 0,
                    'value': {(6, 'internal_route_rel', 'internal_route_rel'): {
                        'ackedState': 0,
                        'state': 1,
                        'target': 'network/route1',
                        'transaction': 0}}},
                (4, 'mFCnglbvserver', 'lbvserver'): {'ackedState': 0,
                    'connector': 'outside',
                    'state': 1,
                    'transaction': 0,
                    'value': {}},
                (4, 'mFCngservice', 'service1'): {'ackedState': 0,
                    'connector': 'inside',
                    'state': 1,
                    'transaction': 0,
                    'value': {(6, 'service_key', 'service_key1'): {
                        'ackedState': 0,
                        'state': 1,
                        'target': 'webservice1',
                        'transaction': 0}}},
                (4, 'mFCngservice', 'service2'): {'ackedState': 0,
                    'connector': 'inside',
                    'state': 1,
                    'transaction': 0,
                    'value': {(6, 'service_key', 'service_key2'): {
                        'ackedState': 0,
                        'state': 1,
                        'target': 'webservice2',
                        'transaction': 0}}},
                (4, 'subnet-inside', 'subnet-inside'): {'ackedState': 0,
                    'connector': 'inside',
                    'state': 1,
                    'transaction': 0,
                    'value': {(5, 'subnet', 'subnet-inside'): {
                        'ackedState': 0,
                        'state': 1,
                        'transaction': 0,
                        'value': '10.10.10.10/24'}}},
                (4, 'subnet-outside', 'subnet-outside'): {'ackedState': 0,
                    'connector': 'outside',
                    'state': 1,
                    'transaction': 0,
                    'value': {(5, 'subnet', 'subnet-outside'): {
                        'ackedState': 0,
                        'state': 1,
                        'transaction': 0,
                        'value': '30.30.30.30/24'}}},
                }}}},
    (4, 'Network', 'network'): {'ackedState': 0,
        'state': 1,
        'transaction': 0,
        'value': {(4, 'nsip', 'snip1'): {'ackedState': 0,
                'state': 1,
                'transaction': 0,
                'value': {(5, 'ipaddress', 'ip1'): {'ackedState': 0,
                    'state': 1,
                    'transaction': 0,
                    'value': '10.10.10.100'},
                (5, 'netmask', 'netmask1'): {'ackedState': 0,
                    'state': 1,
                    'transaction': 0,
```

```
        'value': '255.255.255.0'}}},
(4, 'nsip', 'snip2'): {'ackedState': 0,
    'state': 1,
    'transaction': 0,
    'value': {(5, 'ipaddress', 'ip2'): {'ackedState': 0,
        'state': 1,
        'transaction': 0,
        'value': '30.30.30.101'},
    (5, 'netmask', 'netmask2'): {'ackedState': 0,
        'state': 1,
        'transaction': 0,
        'value': '255.255.255.0'}}},
(4, 'route', 'route1'): {'ackedState': 0,
    'state': 1,
    'transaction': 0,
    'value': {(5, 'gateway', 'gateway1'): {'ackedState': 0,
        'state': 1,
        'transaction': 0,
        'value': '10.10.10.10'},
    (5, 'netmask', 'netmask1'): {'ackedState': 0,
        'state': 1,
        'transaction': 0,
        'value': '255.255.255.0'},
    (5, 'network', 'network1'): {'ackedState': 0,
        'state': 1,
        'transaction': 0,
        'value': '100.100.100.0'}}},
(4, 'route', 'route2'): {'ackedState': 0,
    'state': 1,
    'transaction': 0,
    'value': {(5, 'gateway', 'gateway2'): {'ackedState': 0,
        'state': 1,
        'transaction': 0,
        'value': '30.30.30.201'},
    (5, 'netmask', 'netmask2'): {'ackedState': 0,
        'state': 1,
        'transaction': 0,
        'value': '255.255.255.0'},
    (5, 'network', 'network2'): {'ackedState': 0,
        'state': 1,
        'transaction': 0,
        'value': '40.40.40.0'}}}}},
(4, 'lbvserver', 'lbvserver1'): {'ackedState': 0,
    'state': 1,
    'transaction': 0,
    'value': {(4, 'lbvserver_service_binding', 'lbService1'): {
        'ackedState': 0,
        'state': 1,
        'transaction': 0,
        'value': {(6, 'servicename', 'webservice1'): {'ackedState': 0,
            'state': 1,
            'target': 'webservice1',
            'transaction': 0}}},
        (4, 'lbvserver_service_binding', 'lbService2'): {'ackedState': 0,
            'state': 1,
            'transaction': 0,
            'value': {(6, 'servicename', 'webservice2'): {'ackedState': 0,
                'state': 1,
                'target': 'webservice2',
                'transaction': 0}}},
    (5, 'ipv46', 'ipv46'): {'ackedState': 0,
        'state': 1,
        'transaction': 0,
        'value': '30.30.30.111'},
    (5, 'name', 'name'): {'ackedState': 0,
        'state': 1,
        'transaction': 0,
        'value': 'webVirtualServer1'},
    (5, 'port', 'port'): {'ackedState': 0,
        'state': 1,
        'transaction': 0,
        'value': '22'},
    (5, 'servicetype', 'servicetype'): {'ackedState': 0,
```

```
            'state': 1,
            'transaction': 0,
            'value': 'tcp'}}},
    (4, 'service', 'webservice1'): {'ackedState': 0,
        'state': 1,
        'transaction': 0,
        'value': {(5, 'ip', 'ip'): {'ackedState': 0,
            'state': 1,
            'transaction': 0,
            'value': '10.10.10.101'},
        (5, 'name', 'name'): {'ackedState': 0,
            'state': 1,
            'transaction': 0,
            'value': 'webservice1'},
        (5, 'port', 'port'): {'ackedState': 0,
            'state': 1,
            'transaction': 0,
            'value': '22'},
        (5, 'servicetype', 'servicetype'): {'ackedState': 0,
            'state': 1,
            'transaction': 0,
            'value': 'tcp'}}},
    (4, 'service', 'webservice2'): {'ackedState': 0,
        'state': 1,
        'transaction': 0,
        'value': {(5, 'ip', 'ip'): {'ackedState': 0,
            'state': 1,
            'transaction': 0,
            'value': '10.10.10.102'},
        (5, 'name', 'name'): {'ackedState': 0,
            'state': 1,
            'transaction': 0,
            'value': 'webservice2'},
        (5, 'port', 'port'): {'ackedState': 0,
            'state': 1,
            'transaction': 0,
            'value': '22'},
        (5, 'servicetype', 'servicetype'): {'ackedState': 0,
            'state': 1,
            'transaction': 0,
            'value': 'tcp'}}},
    (7, '', '2850816_32771'): {'ackedState': 0,
        'state': 1,
        'tag': 1802,
        'transaction': 0,
        'type': 1},
    (7, '', '2850816_32772'): {'ackedState': 0,
        'state': 1,
        'tag': 2901,
        'transaction': 0,
        'type': 1},
    (8, '', 'ADCCluster1_inside_2850816_32772'): {'ackedState': 0,
        'encap': '2850816_32772',
        'state': 1,
        'transaction': 0,
        'vif': 'ADCCluster1_inside'},
    (8, '', 'ADCCluster1_outside_2850816_32771'): {'ackedState': 0,
        'encap': '2850816_32771',
        'state': 1,
        'transaction': 0,
        'vif': 'ADCCluster1_outside',
        'OspfVIfCfg': {(16, '', u'OspfVIfCfg'): {
            'area': 111,
            'authKey': u'',
            'authKeyId': 1,
            'authType': u'none',
            'cost': 1,
            'deadIntvl': 45,
            'helloIntvl': 15,
            'addressFamily': [ 'ipv4', 'ipv6' ],
            'nwT': 'p2p',
            'prio': 1,
            'rexmitIntvl': 5,
```

```
                        'state': 0,
                        'value': {},
                        'xmitDelay': 1}}
                },
                (10, '', 'ADCCluster1_inside'): {
                    'ackedState': 0,
                    'cifs': {'ADC1': 'Gig0/1'},
                    'state': 1,
                    'transaction': 0},
                (10, '', 'ADCCluster1_outside'): {'ackedState': 0,
                    'cifs': {'ADC1': 'Gig0/2'},
                    'state': 1,
                    'transaction': 0},
                (13, '', u'OspfDevCfg2'): {'ackedState': 0,
                    'area': 10,
                    'enable': True,
                    'areaType': 'nssa',
                    'areaCtrl': [ 'redistribute' ],
                    'redistribute' : [ 'static', 'connected' ],
                    'rtrId': '11.0.1.1',
                    'state': 1,
                    'transaction': 0,
                    'processID' : 1,
                    'value': {}},
                    (17, '', u'BGPDevCfg'): {'ackedState': 0,
                        'localAS': 6501,
                        'rtrId': '11.0.1.1',
                        'state': 1,
                        'timers': {
                            'keepalive': 10,
                            'hold': 100,
                            'neighborMinHold': 100,
                        },
                        'context': {
                            'name': 'tenant1Ctx',
                            'ipv4': {
                                'neighbor': {
                                    '10.0.0.2': {
                                        'remoteAS': 6501,
                                        'adminStatus': 'enabled'
                                    }
                                },
                                'networks': [
                                    { 'ipaddress': '110.0.0.0',
                                        'netmask': 24,
                                        'state': 1
                                    },
                                    { 'ipaddress': '120.0.0.0',
                                        'netmask': 24,
                                        'state': 1
                                    }
                                ],
                                'redistribute' : [ 'static', 'connected' ],
                            },
                        },
                        'transaction': 0,
                        'value': {}
                }
            }
        }
    }
```