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.

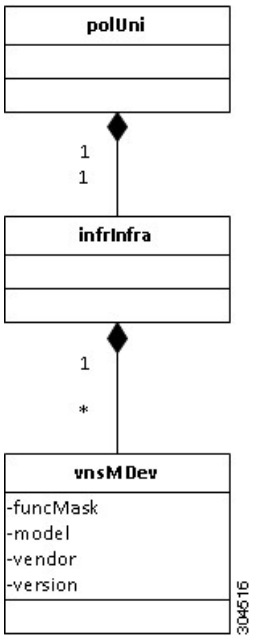
About Device Specifications

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 1: 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.

Attribute	Type	Description
`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

Attribute	Mandatory	Description
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
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`.

Attribute	Mandatory	Description
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`.
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.

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.

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 28.
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 38.

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`".

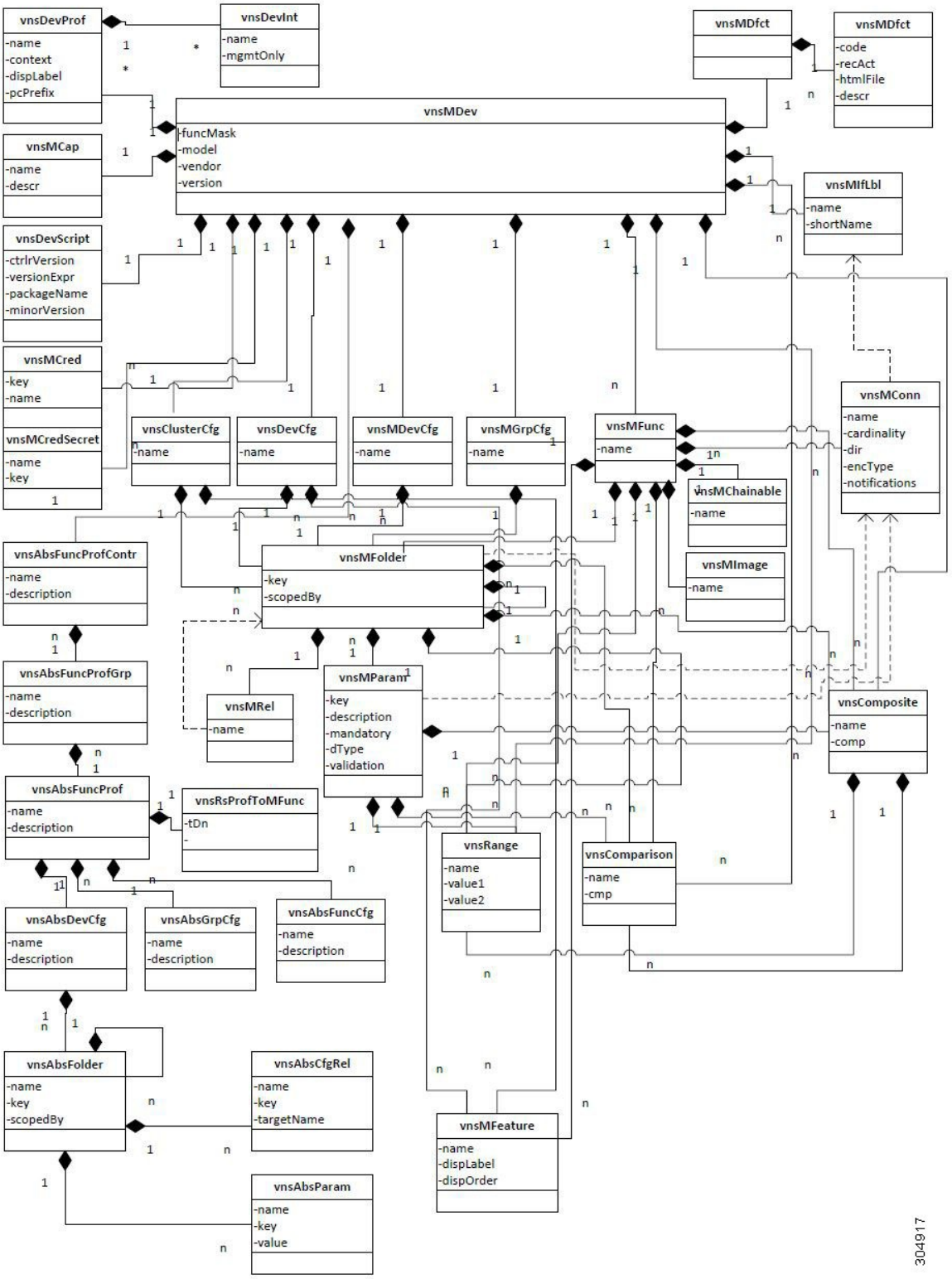
A function profile does not allow instantiating multiple instances of a folder with a cardinality value of "`n`". Only one instance can be defined within the profile.

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 2: 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>
```