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': {}
                }
            }
        }
    }
```