CHAPTER 4

Cisco XML and Native Data Operations

Native data operations <Get>, <Set>, and <Delete> provide basic access to configuration and operational data residing on the router.

This chapter describes the content of native data operations and provides an example of each operation type.

Native Data Operation Content

The content of native data operations includes the request type and relevant object class hierarchy as described in these sections:

- Request Type Tag and Namespaces, page 4-54
- Object Hierarchy, page 4-54
- Dependencies Between Configuration Items, page 4-58
- Null Value Representations, page 4-58
- Operation Triggering, page 4-58
- Native Data Operation Examples, page 4-60

This example shows a native data operation request:

Sample XML Client Native Data Operation Request

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Operation>
    <Request Type>
      .
      .
      .
        object hierarchy goes here
      .
      .
      .
    </Request Type>
  </Operation>
</Request>
```

Sample XML Response from the Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Operation>
    <Request Type>
      .
      .
      .
        response content returned here
      .
      .
      .
    </Request Type>
  </Operation>
  <ResultSummary ErrorCount="0"/>
</Response>
```

Request Type Tag and Namespaces

The request type tag must follow the operation type tag within a native data operation request.

Table 4-1 describes the type of request that must be specified as applying to one of the namespaces.

Table 4-1 Namespace Descriptions

Namespace	Description
<Configuration>	Provides access to the router configuration data analogous to CLI configuration commands. The allowed operations on configuration data are <Get>, <Set>, and <Delete>.
<Operational>	Provides access to the router operational data and is analogous to CLI show commands. The only operation allowed on operational data is <Get>.
<Action>	Provides access to the action data, for example, the **clear** commands. The only allowed operation on action data is <Set>.
<AdminOperational>	Provides access to the router administration operational data. The only operation allowed on administration operational data is <Get>.
<AdminAction>	Provides access to the router administration action data; for example, the **clear** commands. The only allowed operation on administration action data is <Set>.

Object Hierarchy

A hierarchy of elements is included to specify the items to get, set, or delete, and so on, after the request type tag is specified. The precise hierarchy is defined by the XML component schemas.

Note You should use only the supported XML schema objects; therefore, do not attempt to write a request for other objects.

The XML schema information is mapped to the XML instance.

Main Hierarchy Structure

The main structure of the hierarchy consists of the native data model organized as a tree of nodes, where related data items appear in the same branch of the tree. At each level of the tree, a node is a container of further, more specific, sets of related data, or a leaf that holds an actual value.

For example, the first element in the configuration data model is <Configuration>, which contains all possible configuration items. The children of this element are more specific groups of configuration, such as <BGP> for Border Gateway Protocol (BGP) configuration and <ISIS> for Intermediate System-to-Intermediate System (ISIS) configuration. Beneath the <BGP> element, data is further compartmentalized with the <Global> element for global BGP configuration and <BGPEntity> element for per-entity BGP configuration. This compartmentalization continues down to the elements that hold the values, the values being the character data of the element.

This example shows the main hierarchy structure:

```
<Configuration>
  <BGP>
    .
    .
    .
    <Global>
    .
    .
    .
        <DefaultMetric>10</DefaultMetric>
    .
    .
    .
    </Global>
    <BGPEntity>
    .
    .
    .
    </BGPEntity>
    .
    .
    .
  </BGP>
  <ISIS>
        .
    .
    .
  </ISIS>
</Configuration>
```

Data can be retrieved at any level in the hierarchy. One particular data item can be examined, or all of the data items in a branch of the tree can be returned in one request.

Similarly, configuration data can be deleted at any granularity—one item can be deleted, or a whole branch of related configuration can be deleted. So, for example, all BGP configuration can be deleted in one request, or just the value of the default metric.

Hierarchy Tables

One special type of container element is a table. Tables can hold any number of keyed entries, and are used when there can be multiple instances of an entity. For example, BGP has a table of multiple neighbors, each of which has a unique IP address "key" to identify it. In this case, the table element is

<NeighborTable>, and its child element signifying a particular neighbor is <Neighbor>. To specify the key, an extension to the basic parent-child hierarchy is used, where a <Naming> element appears under the child element, containing the key to the table entry.

This example shows hierarchy tables:

```
<Configuration>
  <BGP>
    .
    .
    .
    <BGPEntity>
      <NeighborTable>
        <Neighbor>
          <Naming>
            <NeighborAddress>
              <IPV4Address>10.0.101.6</IPV4Address>
            </NeighborAddress>
          </Naming>
          <RemoteAS>
             <AS_XX>
                0
             </AS_XX>
             <AS_YY>
                6
             </AS_YY>
           </RemoteAS>
        </Neighbor>
        <Neighbor>
          <Naming>
            <NeighborAddress>
              <IPV4Address>10.0.101.7</IPV4Address>
            </NeighborAddress>
          </Naming>
          <RemoteAS>
             <AS_XX>
                0
             </AS_XX>
             <AS_YY>
                6
             </AS_YY>
          </RemoteAS>
        </Neighbor>
      </NeighborTable>
    </BGPEntity>
    .
    .
    .
  </BGP>
  <ISIS>
    .
    .
    .
  </ISIS>
</Configuration>
```

Use tables to access a specific data item for an entry (for example, getting the remote autonomous system number for neighbor 10.0.101.6), or all data for an entry, or even all data for all entries.

Tables also provide the extra feature of allowing the list of entries in the table to be returned.

Returned entries from tables can be used to show all neighbors configured; for example, without showing all their data.

Tables in the operational data model often have a further feature when retrieving their entries. The tables can be filtered on particular criteria to return just the set of entries that fulfill those criteria. For instance, the table of BGP neighbors can be filtered on address family or autonomous system number or update group, or all three. To apply a filter to a table, use another extension to the basic parent-child hierarchy, where a <Filter> element appears under the table element, containing the criteria to filter on.

This example shows table filtering:

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Get>
    <Operational>
      <BGP>
        <Active>
          <VRFTable>
            <VRF>
              <Naming>
            <VRFName>one<VRFName>
          </Naming>
          <NeighborTable>
            <Filter>
              <BGP_AFFilter>
                <AFName>IPv4Unicast</AFName>
              </BGP_AFFilter>
            </Filter>
          </NeighborTable>
            </VRF>
          </VRFTable>
        </Active>
      </BGP>
    </Operational>
  </Get>
</Request>
```

Leaf Nodes

The leaf nodes hold values and are generally simple one-value items where the element representing the leaf node uses character data to specify the value (as in `<DefaultMetric>10</DefaultMetric>` in the example in the "Main Hierarchy Structure" section on page 4-55. In some cases there may be more than one value to specify—for example, when you configure the administrative distance for an address family (the <Distance> element), three values must be given together. Specifying more than one value is achieved by adding further child elements to the leaf, each of which indicates the particular value being configured.

This example shows leaf nodes:

```
<Configuration>
  <BGP>
    .
    .
    .
      <Distance>
        <ExternalRoutes>20</ExternalRoutes>
        <InternalRoutes>250</InternalRoutes>
        <LocalRoutes>200</LocalRoutes>
      </Distance>
    .
    .
    .
  </BGP>
</Configuration>
```

Sometimes there may be even more structure to the values (with additional levels in the hierarchy beneath the <Distance> tag as a means for grouping the related parts of the data together), although they are still only "setable" or "getable" as one entity. The extreme example of this is that in some of the information returned from the operational data model, all the values pertaining to the status of a particular object may be grouped as one leaf. For example, a request to retrieve a particular BGP path status returns all the values associated with that path.

Dependencies Between Configuration Items

Dependencies between configuration items are not articulated in the XML schema nor are they enforced by the XML infrastructure; for example, if item A is this value, then item B must be one of these values, and so forth. The back-end for the Cisco IOS XR applications is responsible for preventing inconsistent configuration from being set. In addition, the management agents are responsible for carrying out the appropriate operations on dependent configuration items through the XML interface.

Null Value Representations

The standard attribute "xsi:nil" is used with a value of "true" when a null value is specified for an element in an XML request or response document.

This example shows how to specify a null value for the element <HoldTime>:

```
<Neighbor>
  <Timers>
    <KeepAliveInterval>60</KeepAliveInterval>
    <HoldTime xsi:nil="true"/>
  </Timers>
</Neighbor>
```

Any element that can be set to "nil" in an XML instance has the attribute "nillable" set to "true" in the XML schema definition for that element. For example:

```
<xsd:element name="HoldTime" type="xsd:unsignedInt" nillable="true"/>
```

Any XML instance document that uses the nil mechanism must declare the "XML Schema for Instance Documents" namespace, which contains the "xsi:nil" definition. Responses to native data operations returned from the router declares the namespace in the operation tag. For example:

```
<Get xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

Operation Triggering

When structuring an XML request, the user should remember the general rule regarding what to specify in the XML for an operation to take place: As a client XML request is parsed by the router, the specified operation takes place whenever a closing tag is encountered after a series of one or more opening tags (but only when the closing tag is not the </Naming> tag).

This example shows a request to get the confederation peer information for a particular BGP autonomous system. In this example, the <Get> operation is triggered when the `<ConfederationPeerASTable/>` tag is encountered.

Sample XML Client Request to Trigger a <Get> Operation for BGP Timer Values

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
```

```
  <Get>
    <Configuration>
      <BGP>
        <AS>
           <Naming>
             <AS>0</AS>
           </Naming>
         <FourByteAS>
           <Naming>
            <AS>3</AS>
          </Naming>
            <DefaultVRF>
              <Global>
                <ConfederationPeerASTable/>
              </Global>
            </DefaultVRF>
          </FourByteAS>
        </AS>
      </BGP>
    </Configuration>
  </Get>
</Request>
```

Sample XML Response from the Router

```
<Response MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration>
      <BGP MajorVersion="30" MinorVersion="0">
        <AS>
          <Naming>
            <AS>0</AS>
          </Naming>
        <FourByteAS>
          <Naming>
            <AS>3</AS>
          </Naming>
            <DefaultVRF>
              <Global>
                <ConfederationPeerASTable>
                  <ConfederationPeerAS>
                    <Naming>
                      <AS_XX>0</AS_XX>
                      <AS_YY>10</AS_YY>
                    </Naming>
                    <Enable>true</Enable>
                  </ConfederationPeerAS>
                </ConfederationPeerASTable>
              </Global>
            </DefaultVRF>
          </FourByteAS>
        </AS>
      </BGP>
    </Configuration>
  </Get>
  <ResultSummary ErrorCount="0"/>
</Response>
```

Native Data Operation Examples

These sections provide examples of the basic <Set>, <Get>, and <Delete> operations:

- Set Configuration Data Request: Example, page 4-60
- Get Request: Example, page 4-62
- Get Request of Nonexistent Data: Example, page 4-63
- Delete Request: Example, page 4-65
- GetDataSpaceInfo Request Example, page 4-66

Set Configuration Data Request: Example

This example shows a native data request to set several configuration values for a particular BGP neighbor. Because the <Set> operation in this example is successful, the response contains only the <Set> operation and <Configuration> request type tags.

This request is equivalent to these CLI commands:

```
router bgp 3
  address-family ipv4 unicast!
  address-family ipv4 multicast!
  neighbor 10.0.101.6
    remote-as 6
    ebgp-multihop 255
    address-family ipv4 unicast
    orf route-policy BGP_pass all
      capability orf prefix both
     !
    address-family ipv4 multicast
      orf route-policy BGP_pass all
    !
  !
!
```

Sample XML Client Request to <Set> Configuration Values for a BGP Neighbor

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Set>
    <Configuration>
    <BGP>
      <AS>
        <Naming>
          <AS>0</AS>
        </Naming>
        <FourByteAS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <BGPRunning>true</BGPRunning>
          <DefaultVRF>
            <Global>
              <GlobalAFTable>
                <GlobalAF>
                  <Naming>
                    <AFName>IPv4Unicast</AFName>
                  </Naming>
                  <Enable>true</Enable>
                </GlobalAF>
                <GlobalAF>
                  <Naming>
                    <AFName>IPv4Multicast</AFName>
                  </Naming>
                  <Enable>true</Enable>
                </GlobalAF>
              </GlobalAFTable>
            </Global>
            <BGPEntity>
              <NeighborTable>
                <Neighbor>
                  <Naming>
                    <NeighborAddress>
                      <IPV4Address>10.0.101.6</IPV4Address>
                    </NeighborAddress>
                  </Naming>
                  <RemoteAS>
                    <AS_XX>0</AS_XX>
                    <AS_YY>6</AS_YY>
                  </RemoteAS>
                  <EBGPMultihop>
                     <MaxHopCount>255</MaxHopCount>
                     <MPLSDeactivation> false </MPLSDeactivation>
                  </EBGPMultihop>
                  <NeighborAFTable>
                    <NeighborAF>
                      <Naming>
                        <AFName>IPv4Unicast</AFName>
                      </Naming>
                      <Activate>true</Activate>
                      <PrefixORFPolicy>BGP_pass_all</PrefixORFPolicy>
                      <AdvertiseORF> Both </AdvertiseORF>
                    </NeighborAF>
                    <NeighborAF>
                      <Naming>
                        <AFName>IPv4Multicast</AFName>
                      </Naming>
                      <Activate>true</Activate>
                      <PrefixORFPolicy>BGP_pass_all</PrefixORFPolicy>
                    </NeighborAF>
                  </NeighborAFTable>
                </Neighbor>
              </NeighborTable>
            </BGPEntity>
          </DefaultVRF>
        </FourByteAS>
      </AS>
    </BGP>
  </Configuration>
 </Set>
 <Commit/>
</Request>
```

Sample XML Response from the Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Set>
    <Configuration/>
  </Set>
  <Commit CommitID="1000000029"/>
  <ResultSummary ErrorCount="0"/>
</Response>
```

Get Request: Example

This example shows a native data request to get the address independent configuration values for a specified BGP neighbor (using the same values set in the previous example).

Sample XML Client Request to <Get> Configuration Values for a BGP Neighbor

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration>
      <BGP>
       <AS>
         <Naming>
           <AS>0</AS>
         </Naming>
         <FourByteAS>
           <Naming>
             <AS>3</as>
           </Naming>
           <DefaultVRF>
            <BGPEntity>
              <NeighborTable>
                <Neighbor>
                  <Naming>
                    <NeighborAddress>
                     <IPV4Address>10.0.101.6</IPV4Address>
                    </NeighborAddress>
                  </Naming>
                </Neighbor>
              </NeighborTable>
           </BGPEntity>
          </DefaultVRF>
         </FourByteAS>
         </AS>
       </BGP>
     </Configuration>
   </Get>
</Request>
```

Sample XML Response from the Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get>
   <Configuration>
    <BGP MajorVersion="35" MinorVersion="2">
     <AS>
      <Naming>
       <AS>0</AS>
      </Naming>
      <FourByteAS>
       <Naming>
        <AS>3</AS>
        </Naming>
        <DefaultVRF>
         <BGPEntity>
          <NeighborTable>
           <Neighbor>
            <Naming>
             <NeighborAddress>
              <IPV4Address>10.0.101.6</IPV4Address>
             </NeighborAddress>
          </Naming>
          <RemoteAS>
           <AS XX>0</AS XX>
           <AS YY>6</AS YY>
          </RemoteAS>
         <EBGPMultihop>
              <MaxHopCount>255</MaxHopCount>
              <MPLSDeactivation>false</MPLSDeactivation>
          <EBGPMultihop>
          <NeighborAFTable>
           <NeighborAF>
            <Naming>
             <AFName>IPv4Unicast</AFName>
            </Naming>
            <Activate>true</Activate>
         <PrefixORFPolicy>BGP_pass_all</PrefixORFPolicy>
          <AdvertiseORF>Both</AdvertiseORF>
         </NeighborAF>
         <NeighborAF>
          <Naming>
           <AFName>IPv4Multicast</AFName>
          </Naming>
          <Activate>true</Activate>
         <PrefixORFPolicy>BGP_pass_all</PrefixORFPolicy>
         </NeighborAF>
        </NeighborAFTable>
       </Neighbor>
      </NeighborTable>
     </BGPEntity>
    </DefaultVRF>
    </FourByteAS>
   </AS>
  </BGP>
 </Configuration>
</Get>
<ResultSummary ErrorCount="0"/>
</Response>
```

Get Request of Nonexistent Data: Example

This example shows a native data request to get the configuration values for a particular BGP neighbor; this is similar to the previous example. However, in this example the client application is requesting the configuration for a nonexistent neighbor. Instead of returning an error, the router returns the requested object class hierarchy, but without any data.

Note Whenever an application attempts to get nonexistent data, the router does not treat this as an error and returns the empty object hierarchy in the response.

Sample XML Client Request to <Get> Configuration Data for a Nonexistent BGP Neighbor

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration>
      <BGP MajorVersion=”24” MinorVersion=”0”>
        <AS>
         <Naming>
           <AS>0</AS>
         </Naming>
         <FourByteAS>
```

```
          <Naming>
            <AS>3</AS>
          </Naming>
          <DefaultVRF>
            <BGPEntity>
              <NeighborTable>
                <Neighbor>
                  <Naming>
                     <NeighborAddress>
                       <IPV4Address>10.0.101.99</IPV4Address>
                     </NeighborAddress>
                   </Naming>
                 </Neighbor>
               </NeighborTable>
             </BGPEntity>
           </DefaultVRF>
         </FourByteAS>
       </AS>
     </BGP>
   </Configuration>
  </Get>
</Request>
```

Sample XML Response from the Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get ItemNotFound ="true">
    <Configuration>
      <BGP MajorVersion=”35” MinorVersion=”2”>
        <AS>
         <Naming>
            <AS>0</AS>
         </Naming>
         <FourByteAS>
          <Naming>
           <AS>3<AS>
          </Naming>
          <DefaultVRF>
           <BGPEntity>
            <NeighborTable>
             <Neighbor NotFound=”true”>
              <Naming>
               <NeighborAddress>
                <IPV4Address>10.0.101.99</IPV4Address>
               </NeighborAddress>
              </Naming>
             </Neighbor>
            </NeighborTable>
           </BGPEntity>
          </DefaultVRF>
         </FourByteAS>
        </AS>
      </BGP>
    </Configuration>
  </Get>
  <ResultSummary ErrorCount="0" ItemNotFound="true"/>
</Response>
```

Delete Request: Example

This example shows a native data request to delete the address-independent configuration for a particular BGP neighbor. Note that if a request is made to delete an item that does not exist in the current configuration, an error is not returned to the client application. So in this example, the returned result is the same as in the previous example: the empty <Delete/> tag, whether or not the specified BGP neighbor exists.

This request is equivalent to these CLI commands:

```
router bgp 3
  no neighbor 10.0.101.9
exit
```

Sample XML Client Request to <Delete> the Address-Independent Configuration Data for a BGP Neighbor

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Delete>
    <Configuration>
      <BGP MajorVersion=”24” MinorVersion=”0”>
        <AS>
          <Naming>
            <AS>0</AS>
          </Naming>
          <FourByteAS>
            <Naming>
              <AS>3</AS>
            </Naming>
            <DefaultVRF>
              <BGPEntity>
                <NeighborTable>
                  <Neighbor>
                    <Naming>
                      <NeighborAddress>
                        <IPV4Address>10.0.101.6</IPV4Address>
                      </NeighborAddress>
                    </Naming>
                  </Neighbor>
                </NeighborTable>
              </BGPEntity>
            </DefaultVRF>
          </FourByteAS>
        </AS>
      </BGP>
    </Configuration>
  </Delete>
  <Commit/>
</Request>
```

Sample XML Response from the Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Delete>
    <Configuration/>
  </Delete>
  <Commit CommitID=”1000000030”/>
  <ResultSummary ErrorCount="0"/>
</Response>
```

GetDataSpaceInfo Request Example

This example shows a <GetDataSpaceInfo> operation used to retrieve the native data branch names dynamically. This is useful, for example, for writing a client application that can issue a <GetVersionInfo> operation without having to hardcode the branch names. The <GetDataSpaceInfo> operation can be invoked instead to retrieve the branch names. The returned branch names can then be included in a subsequent <GetVersionInfo> request.

Sample XML Client Request to Retrieve Native Data

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <GetDataSpaceInfo/>
</Request>
```

Sample XML Response from the Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1"
     MinorVersion="0">
  <GetDataSpaceInfo>
    <Configuration/>
    <Operational/>
    <Action/>
    <AdminOperational/>
    <AdminAction/>
  </GetDataSpaceInfo>
  <ResultSummary ErrorCount="0"/>
</Response>
```