



Cisco XML Router Configuration and Management

This chapter reviews the basic extensible markup language (XML) requests and responses used to configure and manage the router.

The use of XML to configure the router is essentially an abstraction of a configuration editor in which client applications can, load, browse, and modify configuration data without affecting the current running (that is, active) configuration on the router. This configuration is called the "target configuration" and is not the running configuration on the router. The router's running configuration can never be modified directly. All changes to the running configuration must go through the target configuration.



Note

Each client application session has its own target configuration, which is not visible to other client sessions.

This chapter contains the following sections:

- [Target Configuration Overview, page 2-29](#)
- [Configuration Operations, page 2-30](#)
- [Additional Router Configuration and Management Options Using XML, page 2-42](#)

Target Configuration Overview

The target configuration is effectively the current running configuration overlaid with the client-entered configuration. In other words, the target configuration is the client-intended configuration if the client were to commit changes. In terms of implementation, the target configuration is an operating system buffer that contains just the changes (set and delete) that are performed within the configuration session.

A "client session" is synonymous with a single Common Object Request Broker Architecture (CORBA), Telnet, or Secure Shell (SSH) connection and authentication, authorization, and accounting (AAA) login. The target configuration is created implicitly at the beginning of a client application session and must be promoted (that is, committed) to the running configuration explicitly by the client application in order to replace or become the running configuration. If the client session breaks, the current target configuration is aborted and any outstanding locks are released.

**Note**

Only the syntax of the target configuration is checked and verified to be compatible with the installed software image on the router. The semantics of the target configuration is checked only when the target configuration is promoted to the running configuration.

Configuration Operations

**Note**

Only the tasks in the “[Committing the Target Configuration](#)” section are required to change the configuration on the router (that is, modifying and committing the target configuration).

Use the following configuration options from the client application to configure or modify the router with XML:

- [Locking the Running Configuration](#), page 2-31
- [Browsing the Target or Running Configuration](#), page 2-31
 - [Getting Configuration Data](#), page 2-32
- [Browsing the Changed Configuration](#), page 2-33
- [Loading the Target Configuration](#), page 2-35
- [Setting the Target Configuration Explicitly](#), page 2-35
- [Saving the Target Configuration](#), page 2-37
- [Committing the Target Configuration](#), page 2-37
 - [Loading a Failed Configuration](#), page 2-40
- [Unlocking the Running Configuration](#), page 2-41

Additional Configuration Options Using XML

Several optional configuration tasks are available to the client application to configure or modify the router with XML:

- [Getting Commit Changes](#), page 2-42
- [Clearing a Target Session](#), page 2-44
- [Rolling Back Configuration Changes to a Specified Commit Identifier](#), page 2-44
- [Rolling Back Configuration Changes to a Specified Number of Commits](#), page 2-45
- [Getting Rollback Changes](#), page 2-46
- [Getting Configuration History](#), page 2-47
- [Getting Configuration Session Information](#), page 2-49
- [Replacing the Current Running Configuration](#), page 2-50

Locking the Running Configuration

The client application uses the <Lock> operation to obtain an exclusive lock on the running configuration in order to prevent modification by other users or applications.

If the lock operation is successful, the response contains only the <Lock/> tag. If the lock operation fails, the response also contains ErrorCode and ErrorMsg attributes that indicates the cause of the lock failure.

The following example shows a request to lock the running configuration. This request corresponds to the command-line interface (CLI) command **configure exclusive**.

Sample XML Request from the Client Application

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

Sample XML Response from the Router

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

The following issues are used to lock the running configuration:

- The scope of the lock is the entire configuration “namespace.”
- Only one client application can hold the lock on the running configuration at a time. If a client application attempts to lock the configuration while another application holds the lock, an error is returned.
- If a client application has locked the running configuration, all other client applications can read only the running configuration, but cannot modify it (that is, they cannot commit changes to it).
- No mechanism is provided to allow a client application to break the lock of another user.
- If a client session is terminated, any outstanding locks are automatically released.
- The XML API does not support timeouts for locks.
- The <GetConfigurationSessions> operation is used to identify the user session holding the lock.

Browsing the Target or Running Configuration

The client application browses the target or current running configuration using the <Get> operation along with the <Configuration> request type tag. The client application optionally uses CLI commands encoded within XML tags to browse the configuration.

The <Configuration> tag supports the optional Source attribute, which is used to specify the source of the configuration information returned from a <Get> operation.

Getting Configuration Data

Table 2-1 describes the Source options.

Table 2-1 Source Options

Name	Description
ChangedConfig	Read only from the changes made to the target configuration for the current session. This option effectively gets the configuration changes made from the current session since the last configuration commit. This option corresponds to the CLI command show configuration .
CurrentConfig	Read from the current active running configuration. This option corresponds to the CLI command show configuration running .
MergedConfig	Read from the target configuration for this session. This option should provide a view of the resultant running configuration if the current target configuration is committed without errors. For example, in the case of the “best effort” commit, some portions of the commit could fail, while others succeed. MergedConfig is the default when the Source attribute is not specified on the <Get> operation. This option corresponds to the CLI command show configuration merge .

If the get operation fails, the response contains one or more ErrorCode and ErrorMsg attributes indicating the cause of the failure.

The content and format of <Get> requests are described in additional detail in [Chapter 4, “Cisco XML and Native Data Operations.”](#) Encoding CLI commands within XML tags is described in [Chapter 6, “Cisco XML and Encapsulated CLI Operations.”](#)

The following example shows a <Get> request to browse the current Border Gateway Protocol (BGP) configuration:

Sample XML Client Request to Browse the Current BGP Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration Source="CurrentConfig">
      <BGP MajorVersion="1" MinorVersion="0"/>
    </Configuration>
  </Get>
</Request>
```

Sample XML Response from the Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration Source="CurrentConfig">
      <BGP MajorVersion="1" MinorVersion="0">
        ..
        .
        .
        response data goes here
        .
        .
        .
      </BGP>
    </Configuration>
  </Get>
</Response>
```

```

    </Configuration>
  </Get>
</Response>

```

Browsing the Changed Configuration

When a client application issues a <Get> request with a Source type of ChangedConfig, the response contains the OperationType attribute to indicate whether the returned changes to the target configuration were a result of <Set> or <Delete> operations.

Use <Get> to browse uncommitted target configuration changes.

The following example shows <Set> and <Delete> operations that modify the BGP configuration followed by a <Get> request to browse the uncommitted BGP configuration changes. These requests correspond to the following CLI commands:

```

RP/0/RP0/CPU0:router# configure
RP/0/RP0/CPU0:router(config)# router bgp 3
RP/0/RP0/CPU0:router(config-bgp)# default-metric 10
RP/0/RP0/CPU0:router(config-bgp)# no neighbor 10.0.101.8
RP/0/RP0/CPU0:router(config-bgp)# exit
RP/0/RP0/CPU0:router# show configuration

```

Sample XML to Modify the BGP Configuration

```

<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Set>
    <Configuration>
      <BGP MajorVersion="1" MinorVersion="0">
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <Global>
            <DefaultMetric>10</DefaultMetric>
          </Global>
        </AS>
      </BGP>
    </Configuration>
  </Set>
  <Delete>
    <Configuration>
      <BGP>
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <BGPEntity>
            <NeighborTable>
              <Neighbor>
                <Naming>
                  <IPAddress>
                    <IPv4Address>10.0.101.8</IPv4Address>
                  </IPAddress>
                </Naming>
              </Neighbor>
            </NeighborTable>
          </BGPEntity>
        </AS>
      </BGP>
    </Configuration>
  </Delete>

```

```

    </AS>
  </BGP>
</Configuration>
</Delete>
</Request>

```

Sample XML Response from the Router

```

<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Set>
    <Configuration/>
  </Set>
  <Delete>
    <Configuration/>
  </Delete>
</Response>

```

Sample XML Client Request to Browse Uncommitted Target Configuration Changes

```

<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration Source="ChangedConfig">
      <BGP MajorVersion="1" MinorVersion="0"/>
    </Configuration>
  </Get>
</Request>

```

Sample Secondary XML Response from the Router

```

<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration OperationType="Set">
      <BGP MajorVersion="1" MinorVersion="0">
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <Global>
            <DefaultMetric>10</DefaultMetric>
          </Global>
        </AS>
      </BGP>
    </Configuration>
    <Configuration OperationType="Delete">
      <BGP MajorVersion="1" MinorVersion="0">
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <BGPEntity>
            <NeighborTable>
              <Neighbor>
                <Naming>
                  <IPAddress>
                    <IPV4Address>10.0.101.8</IPV4Address>
                  </IPAddress>
                </Naming>
              </Neighbor>
            </NeighborTable>
          </BGPEntity>
        </AS>
      </BGP>
    </Configuration>
  </Get>

```

```

    </BGP>
  </Configuration>
</Get>
</Response>

```

Loading the Target Configuration

The client application uses the <Load> operation along with the <File> tag to populate the target configuration with the contents of a binary configuration file previously saved on the router using the <Save> operation.

Use the <File> tag to name the file from which the configuration is to be loaded. When you use the <File> tag to name the file from which the configuration is to be loaded, specify the complete path of the file to be loaded.

If the load operation is successful, the response contains both the <Load> and <File> tags. If the load operation fails, the response can also contain the ErrorCode and ErrorMessage attributes that indicates the cause of the load failure.

The following example shows a request to load the target configuration from the contents of the file my_bgp.cfg:

Sample XML Client Request to Load the Target Configuration from a Named File

```

<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Load>
    <File>disk0:/my_bgp.cfg</File>
  </Load>
</Request>

```

Sample XML Response from the Router

```

<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Load>
    <File>disk0:/my_bgp.cfg</File>
  </Load>
</Response>

```

See also the [“Setting the Target Configuration Explicitly”](#) section on page 35.

Setting the Target Configuration Explicitly

The client application modifies the target configuration as needed using the <Delete> and <Set> operations.



Note

There are not separate “Create” and “Modify” operations, because a <Set> operation for an item can result in the creation of the item if it does not already exist in the configuration, and a modification of the item if it does already exist.

The client application can optionally use CLI commands encoded within XML tags to modify the target configuration.

If the operation to modify the target configuration is successful, the response contains only the <Delete/> or <Set/> tag. If the operation fails, the response includes the element or object hierarchy passed in the request along with one or more `ErrorCode` and `ErrorMsg` attributes indicating the cause of the failure.

A syntax check is performed whenever the client application writes to the target configuration. A successful write to the target configuration, however, does not guarantee that the configuration change can succeed when a subsequent commit of the target configuration is attempted. For example, errors resulting from failed verifications may be returned from the commit. For information about the error returned from the XML API, see [Chapter 11, “Error Reporting in Cisco XML Responses.”](#)

The content and format of <Delete> and <Set> requests are described in additional detail in [Chapter 4, “Cisco XML and Native Data Operations.”](#)

Encoding CLI commands within XML tags is described in [Chapter 6, “Cisco XML and Encapsulated CLI Operations.”](#)

The following example shows how to use a <Set> request to set the default metric and routing timers and disable neighbor change logging for a particular BGP autonomous system. This request corresponds to the following CLI commands:

```
RP/0/RP0/CPU0:router# configure
RP/0/RP0/CPU0:router (config)# router bgp 3
RP/0/RP0/CPU0:router (config-bgp)# default-metric 10
RP/0/RP0/CPU0:router (config-bgp)# timers bgp 60 180
RP/0/RP0/CPU0:router (config-bgp)# bgp log neighbor changes
RP/0/RP0/CPU0:router (config-bgp)# exit
```

Sample XML Client Request to Set Timers and Disable Neighbor Change Logging for a BGP Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Set>
    <Configuration>
      <BGP MajorVersion="1" MinorVersion="0">
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <Global>
            <DefaultMetric>10</DefaultMetric>
            <GlobalTimers>
              <Keepalive>60</Keepalive>
              <Holdtime>180</Holdtime>
            </GlobalTimers>
            <DisableNbrLogging>true</DisableNbrLogging>
          </Global>
        </AS>
      </BGP>
    </Configuration>
  </Set>
</Request>
```

Sample XML Response from the Router

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

To replace a portion of the configuration, the client application should use a <Delete> operation to remove the unwanted configuration followed by a <Set> operation to add the new configuration. An explicit “replace” option is not supported.

For more information on replacing the configuration, see [“Replacing the Current Running Configuration” section on page 2-50](#).

Saving the Target Configuration

The client application uses the <Save> operation along with the <File> tag to save the contents of the target configuration to a binary file on the router.

Use the <File> tag to name the file to which the configuration is to be saved. You must specify the complete path of the file to be saved when you use the <File> tag. If the file already exists on the router, then an error is returned unless the optional Boolean attribute Overwrite is included on the <File> tag with a value of “true”.

**Note**

No mechanism is provided by the XML interface for “browsing” through the file directory structure.

If the save operation is successful, the response contains both the <Save> and <File> tags. If the save operation fails, the response can also contain the ErrorCode and ErrorMessage attributes that indicate the cause of the save failure.

The following example shows a request to save the contents of the target configuration to the file named my_bgp.cfg on the router:

Sample XML Client Request to Save the Target Configuration to a File

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Save>
    <File Overwrite="true">disk0:/my_bgp.cfg</File>
  </Save>
</Request>
```

Sample XML Response from the Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Save>
    <File Overwrite="true">disk0:/my_bgp.cfg</File>
  </Save>
</Response>
```

Committing the Target Configuration

In order for the configuration in the target area to become part of the running configuration, the target configuration must be explicitly committed by the client application using the <Commit> operation.

Commit Operation

Table 2-2 describes the four optional attributes that are specified with the <Commit> operation.

Table 2-2 Commit Operation Attributes

Name	Description
Mode	Use the Mode attribute in the request to specify whether the target configuration should be committed on an Atomic or BestEffort basis. In the case of a commit with the Atomic option, the entire configuration in the target area is committed only if application of all the configuration in the target area to the running configuration succeeds. If any errors occur, the commit operation is rolled back and the errors are returned to the client application. In the case of commit with the BestEffort option, the configuration is committed even if some configuration items fail during the commit operation. In this case, the errors are also returned to the client application. By default, the commit operation is performed on an Atomic basis.
KeepFailedConfig	Use this Boolean value to specify whether any configuration that fails during the commit operation should remain in the target configuration buffer. The default value for KeepFailedConfig is false. That is, by default the target configuration buffer is cleared after each commit. If a commit operation is performed with a KeepFailedConfig value of false, the user can then use the <Load> operation to load the failed configuration back into the target configuration buffer. The use of the KeepFailedConfig attribute makes sense only for the BestEffort commit mode. In the case of an Atomic commit, if something fails, the entire target configuration is kept intact (because nothing was committed).
Label	Use the Label attribute instead of the commit identifier wherever a commit identifier is expected, such as in the <Rollback> operation. The Label attribute is a unique user-specified label that is associated with the commit in the commit database. If specified, the label must begin with an alphabetic character and cannot match any existing label in the commit database.
Comment	Use the Comment attribute as a user-specified comment to be associated with the commit in the router commit database.

If the commit operation is successful, the response contains only the <Commit/> tag along with a unique CommitID and any other attributes specified in the request. If the commit operation fails, the failed configuration is returned in the response.

The following example shows a request to commit the target configuration using the Atomic option. The request corresponds to the CLI command **commit label BGPUpdate1 comment BGP config update**.

Sample XML Client Request to Commit the Target Configuration Using the Atomic Option

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Commit Mode="Atomic" Label="BGPUpdate1" Comment="BGP config update"/>
</Request>
```

Sample XML Response from the Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Commit Mode="Atomic" Label="BGPUpdate1"
    Comment="BGP config update"
    CommitID="1000000075"/>
</Response>
```

The following points should be noted with regard to committing the target configuration:

- After each successful commit operation, a commit record is created in the router commit database. The router maintains up to 100 entries in the commit database corresponding to the last 100 commits. Each commit is assigned a unique identifier, for example, "1000000075," which is saved with the commit information in the database. The commit identifier is used in subsequent operations such as <Get> commit changes or <Rollback> to a previous commit identifier (along with the <CommitID> tag).
- The configuration changes in the target configuration are merged with the running configuration when committed. If a client application is to perform a replace of the configuration, the client must first remove the unwanted configuration using a <Delete> operation and then add the new configuration using a <Set> operation. An explicit replace option is not supported. For more information on replacing the configuration, see ["Replacing the Current Running Configuration" section on page 2-50](#).
- Applying the configuration for a trial period ("try-and-apply") is not supported for this release.
- If the client application never commits, the target configuration is automatically destroyed when the client session is terminated. No other timeouts are supported.

Commit Errors

If any configuration entered into the target configuration fails to make its way to the running configuration as the result of a <Commit> operation (for example, the configuration contains a semantic error and is therefore rejected by a back-end application's verifier function), then all of the failed configuration is returned in the <Commit> response along with the appropriate ErrorCode and ErrorMessage attributes indicating the cause of each failure.

The OperationType attribute is used to indicate whether the failure was a result of a requested <Set> or <Delete> operation. In the case of a <Set> operation failure, the value to be set is included in the commit response.

The following example shows <Set> and <Delete> operations to modify the BGP configuration followed by a <Commit> request resulting in failures for both requested operations. This request corresponds to the following CLI commands:

```
RP/0/RP0/CPU0:router# configure
RP/0/RP0/CPU0:router(config)# router bgp 4
RP/0/RP0/CPU0:router(config-bgp)# default-metric 10
RP/0/RP0/CPU0:router(config-bgp)# exit
RP/0/RP0/CPU0:router(config)# commit best-effort
```

Sample XML Client Request to Modify the Target Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Set>
    <Configuration>
      <BGP MajorVersion="1" MinorVersion="0">
        <AS>
          <Naming>
```

```

        <AS>4</AS>
    </Naming>
    <Global>
        <DefaultMetric>10</DefaultMetric>
    </Global>
</AS>
</BGP>
</Configuration>
</Set>
</Request>

```

Sample XML Response from the Router

```

<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
    <Set>
        <Configuration/>
    </Set>
</Response>

```

Sample Request to Commit the Target Configuration

```

<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
    <Commit Mode="BestEffort"/>
</Request>

```

Sample XML Response from the Router Showing Failures for Both Requested Operations

```

<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
    <Commit Mode="BestEffort" ErrorCode="0x40819c00"
        ErrorMessage="&apos;sysdb&apos; detected the &apos;warning&apos; condition &apos;One
        or more sub-operations failed during a best effort complex operation&apos; ">
    <Configuration OperationType="Set">
        <BGP MajorVersion="1" MinorVersion="0">
            <AS>
                <Naming>
                    <AS>4</AS>
                </Naming>
                <Global>
                    <DefaultMetric ErrorCode="0x409f8c00" ErrorMessage="AS number is wrong -
                    BGP is already running with AS number 3">10</DefaultMetric>
                </Global>
            </AS>
        </BGP>
    </Configuration>
</Commit>
</Response>

```

For more information, see [“Loading a Failed Configuration”](#) section on page 2-40.

Loading a Failed Configuration

The client application uses the `<Load>` operation along with the `<FailedConfig>` tag to populate the target configuration with the failed configuration from the most recent `<Commit>` operation. Loading the failed configuration in this way is equivalent to specifying a “true” value for the `KeepFailedConfig` attribute in the `<Commit>` operation.

If the load is successful, the response contains both the `<Load>` and `<FailedConfig>` tags. If the load fails, the response can also contain the `ErrorCode` and `ErrorMsg` attributes that indicate the cause of the load failure.

The following example shows a request to load and display the failed configuration from the last <Commit> operation. This request corresponds to the CLI command **show configuration failed**.

Sample XML Client Request to Load the Failed Configuration from the Last <Commit> Operation

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Load>
    <FailedConfig/>
  </Load>
  <Get>
    <Configuration Source="ChangedConfig"/>
  </Get>
</Request>
```

Sample XML Response from the Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Load>
    <FailedConfig/>
  </Load>
  <Get>
    <Configuration OperationType="Set">
      <BGP MajorVersion="1" MinorVersion="0">
        <AS>
          <Naming>
            <AS>4</AS>
          </Naming>
          <Global>
            <DefaultMetric>10</DefaultMetric>
          </Global>
        </AS>
      </BGP>
    </Configuration>
  </Get>
</Response>
```

Unlocking the Running Configuration

The client application must use the <Unlock> operation to release the exclusive lock on the running configuration for the current session prior to terminating the session.

If the unlock operation is successful, the response contains only the <Unock/> tag. If the unlock operation fails, the response can also contain the ErrorCode and ErrorMessage attributes that indicate the cause of the unlock failure.

The following example shows a request to unlock the running configuration. This request corresponds to the CLI command **exit** when it is used after the configuration mode is entered through the CLI command **configure exclusive**.

Sample XML Client Request to Unlock the Running Configuration

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

Sample XML Response from the Router

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

Additional Router Configuration and Management Options Using XML

The following sections describe the optional configuration and router management tasks available to the client application:

- [Getting Commit Changes, page 2-42](#)
- [Clearing a Target Session, page 2-44](#)
- [Rolling Back Configuration Changes to a Specified Commit Identifier, page 2-44](#)
- [Rolling Back Configuration Changes to a Specified Number of Commits, page 2-45](#)
- [Getting Rollback Changes, page 2-46](#)
- [Getting Configuration History, page 2-47](#)
- [Getting Configuration Session Information, page 2-49](#)
- [Replacing the Current Running Configuration, page 2-50](#)

Getting Commit Changes

When a client application successfully commits the target configuration to the running configuration, the configuration manager writes a single configuration change event to the system message logging (syslog). As a result, an event notification is written to the Alarm Channel (that is, the CORBA event notification channel for alarms) and subsequently forwarded to any registered configuration agents.

[Table 2-3](#) describes the event notification.

Table 2-3 Event Notification

Name	Description
userid	The name of the user who performed the commit operation.
timestamp	The date and time of the commit.
commit	The unique ID associated with the commit.

The following example shows a configuration change notification:

```
RP/0/RP0/CPU0:Jun 18 19:16:42.561 : %CLIENTLIBCFGMGR-6-CONFIG_CHANGE : A configuration
commit by user 'root' occurred at 'Wed Jun 18 19:16:18 2004 '. The configuration changes
are saved on the router by commit ID: '1000000075'.
```

Upon receiving the configuration change notification, a client application can then use the <Get> operation to load and browse the changed configuration.

For more information on CORBA-based event notifications and alarms supported by the XML API, see [Chapter 12, "XML Transport and Event Notifications."](#)

The client application can read a set of commit changes using the <Get> operation along with the <Configuration> request type tag when it includes the Source attribute option CommitChanges. One of the additional attributes, either ForCommitID or SinceCommitID, must also be used to specify the commit identifier or commit label for which the commit changes should be retrieved.

The following example shows the use of the ForCommitID attribute to show the commit changes for a specific commit. This request corresponds to the CLI command **show commit changes 100000075**.

Sample XML Request to Show Specified Commit Changes Using the ForCommitID Attribute

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration Source="CommitChanges" ForCommitID="100000075" />
  </Get>
</Request>
```

Sample XML Response from the Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration Source="CommitChanges" ForCommitID="100000075">
      .
      .
      .
      changed config returned here
      .
      .
      .
    </Configuration>
  </Get>
</Response>
```

The following example shows the use of the SinceCommitID attribute to show the commit changes made since a specific commit. This request corresponds to the CLI command **show commit changes since 100000072**.

Sample XML Request to Show Specified Commit Changes Using the SinceCommitID Attribute

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration Source="CommitChanges" SinceCommitID="100000072" />
  </Get>
</Request>
```

Sample XML Response from the Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration Source="CommitChanges" SinceCommitID="100000072">
      .
      .
      .
      changed config returned here
      .
      .
      .
    </Configuration>
  </Get>
</Response>
```

Clearing a Target Session

Prior to committing the target configuration to the active running configuration, the client application can use the <Clear> operation to clear the target configuration session. This operation has the effect of clearing the contents of the target configuration, thus removing any changes made to the target configuration since the last commit. The clear operation does not end the target configuration session, but results in the discarding of any uncommitted changes from the target configuration.

If the clear operation is successful, the response contains just the <Clear/> tag. If the clear operation fails, the response can also contain the ErrorCode and ErrorMsg attributes that indicate the cause of the clear failure.

The following example shows a request to clear the current target configuration session. This request corresponds to the CLI command **clear**.

Sample XML Request to Clear the Current Target Configuration Session

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

Sample XML Response from a Router

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

Rolling Back Configuration Changes to a Specified Commit Identifier

The client application uses the <Rollback> operation with the <CommitID> tag to roll back the configuration changes made since (and including) the commit by specifying a commit identifier or commit label.

If the roll back operation is successful, the response contains both the <Rollback> and <CommitID> tags. If the roll back operation fails, the response can also contain the ErrorCode and ErrorMsg attributes that indicate the cause of the roll back failure.

Table 2-4 describes the optional attributes that are specified with the <Rollback> operation by the client application when rolling back to a commit identifier.

Table 2-4 Optional Attributes for Rollback Operation (Commit Identifier)

Name	Description
Label	A unique user-specified label to be associated with the rollback in the router commit database. If specified, the label must begin with an alphabetic character and cannot match any existing label in the router commit database.
Comment	A user-specified comment to be associated with the rollback in the router commit database.

The following example shows a request to roll back the configuration changes to a specified commit identifier. This request corresponds to the CLI command **rollback configuration to 100000072**.

Sample XML Request to Roll Back the Configuration Changes to a Specified Commit Identifier

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Rollback Label="BGPRollback1" Comment="My BGP rollback">
    <CommitID>100000072</CommitID>
  </Rollback>
</Request>
```

Sample XML Response from the Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Rollback Label="BGPRollback1" Comment="My BGP rollback">
    <CommitID>100000072</CommitID>
  </Rollback>
</Response>
```

**Note**

The commit identifier can also be obtained by using the <GetConfigurationHistory> operation described in the section [“Getting Configuration History” section on page 2-47](#).

Rolling Back Configuration Changes to a Specified Number of Commits

The client application uses the <Rollback> operation with the <Previous> tag to roll back the configuration changes made during the most recent [x] commits, where [x] is a number ranging from 0 to the number of saved commits in the commit database. If the <Previous> value is specified as “0”, nothing is rolled back. The target configuration must be unlocked at the time the <Rollback> operation is requested.

If the roll back operation is successful, the response contains both the <Rollback> and <Previous> tags. If the roll back operation fails, the response can also contain the ErrorCode and ErrorMessage attributes that indicate the cause of the rollback failure.

[Table 2-5](#) describes the optional attributes that are specified with the <Rollback> operation by the client application when rolling back a specified number of commits.

Table 2-5 *Optional Attributes for Rollback Operation (Number of Commits)*

Name	Description
Label	A unique user-specified label to be associated with the rollback in the router commit database. If specified, the label must begin with an alphabetic character and cannot match any existing label in the router commit database.
Comment	A user-specified comment to be associated with the rollback in the router commit database.

The example shows a request to roll back the configuration changes made during the previous three commits. This request corresponds to the CLI command **rollback configuration last 3**.

Sample XML Request to Roll Back Configuration Changes to a Specified Number of Commits

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Rollback>
    <Previous>3</Previous>
  </Rollback>
</Request>
```

Sample XML Response from the Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Rollback>
    <Previous>3</Previous>
  </Rollback>
</Response>
```

Getting Rollback Changes

The client application can read a set of rollback changes using the <Get> operation along with the <Configuration> request type tag when it includes both the Source attribute option RollbackChanges and one of the additional attributes ToCommitID or PreviousCommits.

The set of roll back changes are the changes that are applied when the <Rollback> operation is performed using the same parameters. It is recommended that the client application read or verify the set of roll back changes before performing the roll back.

The following example shows the use of the ToCommitID attribute to get the rollback changes for rolling back to a specific commit. This request corresponds to the CLI command **show rollback-changes to 100000072**.

Sample XML Client Request to Get Rollback Changes Using the ToCommitID Attribute

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration Source="RollbackChanges" ToCommitID="100000072"/>
  </Get>
</Request>
```

Sample XML Response from the Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration Source="RollbackChanges" ToCommitID="100000072">
      .
      .
      .
      rollback changes returned here
      .
      .
    </Configuration>
  </Get>
</Response>
```

The following example shows the use of the PreviousCommits attribute to get the roll back changes for rolling back a specified number of commits. This request corresponds to the CLI command **show rollback-changes last 4**.

Sample XML Client Request to Get Roll Back Changes Using the PreviousCommits Attribute

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration Source="RollbackChanges" PreviousCommits="4" />
  </Get>
</Request>
```

Sample XML Response from the Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration Source="RollbackChanges" PreviousCommits="4">
      .
      .
      .
      rollback changes returned here
      .
      .
      .
    </Configuration>
  </Get>
</Response>
```

Getting Configuration History

The client application uses the `<GetConfigurationHistory>` operation to get information regarding the most recent commits to the running configuration.

Table 2-6 describes the information that is returned for each commit.

Table 2-6 Returned Commit Information

Returned Commit Information	Commit Information Description
<code><CommitID></code>	The unique ID associated with the commit.
<code><Label></code>	The optional label associated with the commit.
<code><UserID></code>	The name of the user who created the configuration session within which the commit was performed.
<code><Line></code>	The line used to connect to the router for the configuration session.
<code><ClientName></code>	The name of the client application that performed the commit.
<code><Timestamp></code>	The date and time of the commit.
<code><Comment></code>	The comment associated with the commit.

Table 2-7 describes the optional attributes available with the <GetConfigurationHistory> operation.

Table 2-7 Optional Attributes to Get Configuration History

Name	Description
Maximum	Use the Maximum attribute to specify the maximum number of entries to return from the commit history file. If the Maximum attribute is not included in the request or if the Maximum value is greater than the actual number of entries in the commit history file, all entries are returned. The commit entries are returned with the most recent commit first in the list.
RollbackOnly	Use the RollbackOnly Boolean attribute to specify whether the response should contain only those commits that can be rolled back. In addition to the commit history file, the router maintains a commit database of up to 100 records corresponding to the last 100 commits that can be rolled back. The default value for RollbackOnly is “false”. The <GetConfigurationHistory> operation used with a RollbackOnly value of “false” corresponds to the CLI command show configuration history . When the RollbackOnly attribute is specified as “true”, the operation corresponds to the CLI command show rollback-points .

The following example shows a request to list the information associated with the previous three commits. This request corresponds to the CLI command **show configuration commit history 3 detail**.

Sample XML Request to List Configuration History Information for the Previous Three Commits

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <GetConfigurationHistory Maximum="3"/>
</Request>
```

Sample XML Response from the Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <GetConfigurationHistory Maximum="3">
    <CommitEntry>
      <CommitID>1000000075</CommitID>
      <Label>BGPUpdate1</Label>
      <UserID>cisco</UserID>
      <Line>line0</Line>
      <ClientName>XMLDemo</ClientName>
      <Timestamp>Wed Jun 18 19:16:18 2003</Timestamp>
      <Comment>BGP config update</Comment>
    </CommitEntry>
    <CommitEntry>
      <CommitID>1000000074</CommitID>
      <Label xsi:nil="true">
      <UserID>unknown</UserID>
      <Line>con0_0_0</Line>
      <ClientName>CLI</ClientName>
      <Timestamp>Wed Jun 18 03:08:07 2003</Timestamp>
      <Comment xsi:nil="true">
    </CommitEntry>
    <CommitEntry>
      <CommitID>1000000073</CommitID>
      <Label>MyCDPUpdate</Label>
      <UserID>nchomsky</UserID>
      <Line>line1</Line>
      <ClientName>XMLDemo</ClientName>
```

```

    <Timestamp>Tue June 17 11:07:55 2003</Timestamp>
    <Comment>My CDP config update</Comment>
  </ComitEntry>
</GetConfigurationHistory>
</Response>

```

Getting Configuration Session Information

The client application uses the `<GetConfigurationSessions>` operation to get the list of all users configuring the router. In the case where the configuration is locked, the list identifies the user holding the lock.

Table 2-8 describes the information that is returned for each configuration session.

Table 2-8 Returned Session Information

Returned Session Information	Session Information Description
<code><SessionID></code>	The unique autogenerated ID for the configuration session.
<code><UserID></code>	The name of the user who created the configuration session.
<code><Line></code>	The line used to connect to the router.
<code><ClientName></code>	The user-friendly name of the client application that created the configuration session.
<code><Since></code>	The date and time of the creation of the configuration session.
<code><LockHeld></code>	A Boolean operation indicating whether the session has an exclusive lock on the running configuration.

The following example shows a request to get the list of users configuring the router. This request corresponds to the CLI command **show configuration sessions**.

Sample XML Request to Get List of Users Configuring the Router

```

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

```

Sample XML Response from the Router

```

<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <GetConfigurationSessions>
    <Session>
      <SessionID>00000070-001610ae-00000000</SessionID>
      <UserID>cisco</UserID>
      <Line>line0</Line>
      <ClientName>XMLDemo</ClientName>
      <Since>Fri Jun 27 15:10:39 2003</Since>
      <LockHeld>>true</LockHeld>
    </Session>
    <Session>
      <SessionID>00000070-001650a4-00000000</SessionID>
      <UserID>unknown</UserID>
      <Line>con0_0_0</Line>
      <ClientName>XML-Agent</ClientName>
      <Since>Fri Jun 27 14:55:18 2003</Since>
    </Session>
  </GetConfigurationSessions>
</Response>

```

```

    <LockHeld>>false</LockHeld>
  </Session>
</GetConfigurationSessions>
</Response>

```

Replacing the Current Running Configuration

A client application replaces the current running configuration on the router with an off-the-box configuration file by performing the following operations in sequence:

1. Lock the configuration.
2. Delete the entire configuration using a <Delete> operation along with the <Configuration/> tag.
3. Load the desired off-the-box configuration into the target configuration using one or more <Set> operations (assuming that the entire desired configuration is available in XML format, perhaps from a previous <Get> of the entire configuration). As an alternative, use an appropriate **copy** command enclosed within <CLI> tags.
4. Commit the target configuration.

The following example illustrates these steps:

Sample XML Request to Lock the Current Running Configuration

```

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

```

Sample XML Response from the Router

```

<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Lock/>
</Response>

```

Sample XML Request to Delete the Current Running Configuration

```

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

```

Sample XML Response from the Router

```

<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Delete>
    <Configuration/>
  </Delete>
</Response>

```

Sample XML Request to Set the Current Running Configuration

```

<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Set>
    <Configuration>
      .
      .
    </Configuration>
  </Set>
</Request>

```


