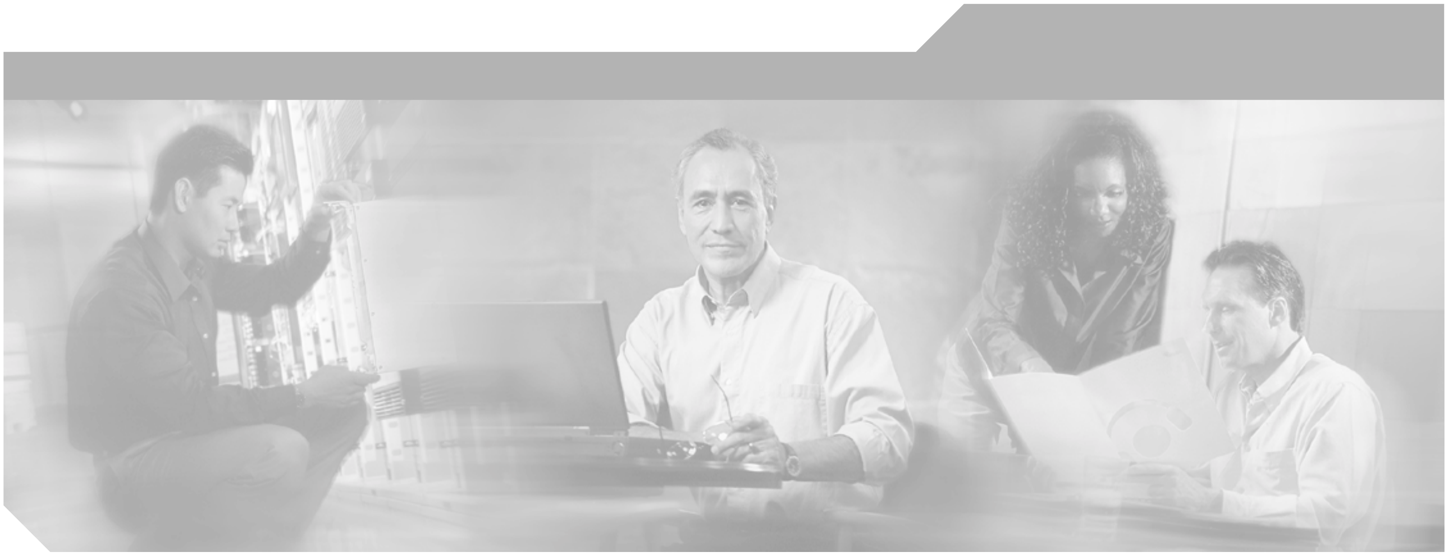


Cisco CRS-1 Series Carrier Routing System XML API Guide

Cisco IOS XR Software Release 3.0

Corporate Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 526-4100

Text Part Number: OL-4596-02

Cisco CRS-1 Series Carrier Routing System XML API Guide

CONTENTS

Preface

This document describes the extensible markup language (XML) application programming interface (API) provided by the Cisco CRS-1 Series Carrier Routing System (Cisco CRS-1 Series) to developers of external management applications. The Cisco CRS-1 Series XML interface provides a mechanism for Cisco CRS-1 Series router configuration and monitoring using XML formatted request and response streams.

The XML schemas referenced in this guide are used by the management application developer to integrate client applications with the Cisco CRS-1 Series programmable interface. The Cisco CRS-1 Series XML API is also available for use on any Cisco platform running Cisco IOS XR software.

Note The XML API code is available for use on any Cisco platform that runs Cisco IOS XR software.

The preface contains the following sections:

About This Document

This document provides a comprehensive description of the XML interface to the Cisco CRS-1 Series router. The goal of this guide is to help Cisco CRS-1 Series customers write client applications to interact with the Cisco CRS-1 Series XML infrastructure on the router, and to also use the Cisco CRS-1 Series XML API to build custom end-user interfaces for configuration and information retrieval and display.

Intended Audience

This document's audience is expected to have background knowledge and information about XML, Common Object Request Broker Architecture (CORBA), and network management.

This document will be of particular use to those who want to build management client applications.

Organization of the Document

This document consists of the following chapters:

- Chapter 1, "Cisco CRS-1 Series XML API Overview"
- Chapter 2, "Cisco CRS-1 Series XML Router Configuration and Management"
- Chapter 3, "Cisco CRS-1 Series XML Operational Requests and Fault Management"
- Chapter 4, "Cisco CRS-1 Series XML and Native Data Operations"
- Chapter 5, "Cisco CRS-1 Series XML and Native Data Access Techniques"
- Chapter 6, "Cisco CRS-1 Series XML and Encapsulated CLI Operations"
- Chapter 7, "Cisco CRS-1 Series XML and Large Data Retrieval (Iterators)"
- Chapter 8, "Cisco CRS-1 Series XML Security"
- Chapter 9, "Cisco CRS-1 Series XML Schema Versioning"
- Chapter 10, "Error Reporting in Cisco CRS-1 Series XML Responses"
- Chapter 11, "XML Transport and Event Notifications"
- Chapter 12, "Summary of Cisco CRS-1 Series XML API Configuration Tags"
- Chapter 13, "Cisco CRS-1 Series XML Schemas"
- Appendix A, "Sample BGP Configuration"

Related Documentation

The following list of related documents are useful:

- *Cisco CRS-1 Series Carrier Routing System Getting Started Guide*
- *Cisco Craft Works Interface Configuration Guide*
- *Cisco CRS-1 Series Carrier Routing System Release Notes*
- Cisco IOS XR software configuration guides and command references

Document Conventions

This publication uses the following conventions:

- The symbol ^ represents the key labeled Ctrl. For example, the key combination ^z means “hold down the Ctrl key while you press the Z key.”

Command descriptions use these conventions:

- Examples that contain system prompts denote interactive sessions, indicating the commands that you should enter at the prompt. The system prompt indicates the current level of the EXEC command interpreter. For example, the prompt router> indicates the user level, and the prompt router# indicates the privileged level. Access to the privileged level usually requires a password.
- Commands and keywords are in **boldface** font.
- Arguments for which you supply values are in *italic* font.
- Elements in square brackets ([ ]) are optional.
    - Alternative but required keywords are grouped in braces ({ }) and separated by vertical bars (|).

Examples use these conventions:

- Terminal sessions and sample console screen displays are in `screen` font.
- Information you enter is in **`boldface screen`** font.
- Nonprinting characters, such as passwords, are in angle brackets (< >).
- Default responses to system prompts are in square brackets ([ ]).
- An exclamation point (!) at the beginning of a line indicates a comment line.

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

Tip Means *the following information will help you solve a problem.* The information in tips might not contain troubleshooting or task-specific actions, but tips do contain useful information.

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

Obtaining Documentation

Cisco documentation and additional literature are available on Cisco.com. Cisco also provides several ways to obtain technical assistance and other technical resources. These sections explain how to obtain technical information from Cisco Systems.

Cisco.com

You can access the most current Cisco documentation at this URL:

http://www.cisco.com/univercd/home/home.htm

You can access the Cisco website at this URL:

http://www.cisco.com

You can access international Cisco websites at this URL:

http://www.cisco.com/public/countries_languages.shtml

Ordering Documentation

You can find instructions for ordering documentation at this URL:

http://www.cisco.com/univercd/cc/td/doc/es_inpck/pdi.htm

You can order Cisco documentation in these ways:

- Registered Cisco.com users (Cisco direct customers) can order Cisco product documentation from the Ordering tool:

  http://www.cisco.com/en/US/partner/ordering/index.shtml

- Nonregistered Cisco.com users can order documentation through a local account representative by calling Cisco Systems Corporate Headquarters (California, USA) at 408 526-7208 or, elsewhere in North America, by calling 800 553-NETS (6387).

Documentation Feedback

You can send comments about technical documentation to bug-doc@cisco.com.

You can submit comments by using the response card (if present) behind the front cover of your document or by writing to the following address:

Cisco Systems
Attn: Customer Document Ordering
170 West Tasman Drive
San Jose, CA 95134-9883

We appreciate your comments.

Obtaining Technical Assistance

For all customers, partners, resellers, and distributors who hold valid Cisco service contracts, Cisco Technical Support provides 24-hour-a-day, award-winning technical assistance. The Cisco Technical Support Website on Cisco.com features extensive online support resources. In addition, Cisco Technical Assistance Center (TAC) engineers provide telephone support. If you do not hold a valid Cisco service contract, contact your reseller.

Cisco Technical Support Website

The Cisco Technical Support Website provides online documents and tools for troubleshooting and resolving technical issues with Cisco products and technologies. The website is available 24 hours a day, 365 days a year at this URL:

http://www.cisco.com/techsupport

Access to all tools on the Cisco Technical Support Website requires a Cisco.com user ID and password. If you have a valid service contract but do not have a user ID or password, you can register at this URL:

http://tools.cisco.com/RPF/register/register.do

Submitting a Service Request

Using the online TAC Service Request Tool is the fastest way to open S3 and S4 service requests. (S3 and S4 service requests are those in which your network is minimally impaired or for which you require product information.) After you describe your situation, the TAC Service Request Tool automatically provides recommended solutions. If your issue is not resolved using the recommended resources, your service request will be assigned to a Cisco TAC engineer. The TAC Service Request Tool is located at this URL:

http://www.cisco.com/techsupport/servicerequest

For S1 or S2 service requests or if you do not have Internet access, contact the Cisco TAC by telephone. (S1 or S2 service requests are those in which your production network is down or severely degraded.) Cisco TAC engineers are assigned immediately to S1 and S2 service requests to help keep your business operations running smoothly.

To open a service request by telephone, use one of the following numbers:

Asia-Pacific: +61 2 8446 7411 (Australia: 1 800 805 227)
EMEA: +32 2 704 55 55
USA: 1 800 553 2447

For a complete list of Cisco TAC contacts, go to this URL:

http://www.cisco.com/techsupport/contacts

Definitions of Service Request Severity

To ensure that all service requests are reported in a standard format, Cisco has established severity definitions.

Severity 1 (S1)—Your network is “down,” or there is a critical impact to your business operations. You and Cisco will commit all necessary resources around the clock to resolve the situation.

Severity 2 (S2)—Operation of an existing network is severely degraded, or significant aspects of your business operation are negatively affected by inadequate performance of Cisco products. You and Cisco will commit full-time resources during normal business hours to resolve the situation.

Severity 3 (S3)—Operational performance of your network is impaired, but most business operations remain functional. You and Cisco will commit resources during normal business hours to restore service to satisfactory levels.

Severity 4 (S4)—You require information or assistance with Cisco product capabilities, installation, or configuration. There is little or no effect on your business operations.

Obtaining Additional Publications and Information

Information about Cisco products, technologies, and network solutions is available from various online and printed sources.

- Cisco Marketplace provides a variety of Cisco books, reference guides, and logo merchandise. Visit Cisco Marketplace, the company store, at this URL:

  http://www.cisco.com/go/marketplace/

- The Cisco *Product Catalog* describes the networking products offered by Cisco Systems, as well as ordering and customer support services. Access the Cisco Product Catalog at this URL:

  http://cisco.com/univercd/cc/td/doc/pcat/

- *Cisco Press* publishes a wide range of general networking, training and certification titles. Both new and experienced users will benefit from these publications. For current Cisco Press titles and other information, go to Cisco Press at this URL:

  http://www.ciscopress.com

- *Packet* magazine is the Cisco Systems technical user magazine for maximizing Internet and networking investments. Each quarter, Packet delivers coverage of the latest industry trends, technology breakthroughs, and Cisco products and solutions, as well as network deployment and troubleshooting tips, configuration examples, customer case studies, certification and training information, and links to scores of in-depth online resources. You can access Packet magazine at this URL:

  http://www.cisco.com/packet

- *iQ Magazine* is the quarterly publication from Cisco Systems designed to help growing companies learn how they can use technology to increase revenue, streamline their business, and expand services. The publication identifies the challenges facing these companies and the technologies to help solve them, using real-world case studies and business strategies to help readers make sound technology investment decisions. You can access iQ Magazine at this URL:

  http://www.cisco.com/go/iqmagazine

- *Internet Protocol Journal* is a quarterly journal published by Cisco Systems for engineering professionals involved in designing, developing, and operating public and private internets and intranets. You can access the Internet Protocol Journal at this URL:

  http://www.cisco.com/ipj

- World-class networking training is available from Cisco. You can view current offerings at this URL:

  http://www.cisco.com/en/US/learning/index.html

Cisco CRS-1 Series XML API Overview

This chapter contains the following sections:

Introduction

The *Cisco CRS-1 Series Carrier Routing System XML API Guide* explains how to use the Cisco CRS-1 Series XML API to configure Cisco CRS-1 Series routers or request information about configuration, management, or operation of Cisco CRS-1 Series routers. The goal of this guide is to help Cisco CRS-1 Series router customers write client applications to interact with the Cisco CRS-1 Series XML infrastructure on the router, and to also use the Cisco CRS-1 Series Management XML API to build custom end-user interfaces for configuration and information retrieval and display.

The extensible markup language (XML) application programming interface (API) provided by the Cisco CRS-1 Series Carrier Routing System (Cisco CRS-1 Series) is an interface used for rapid development of client applications and perl scripts to manage and monitor the Cisco CRS-1 Series router. The XML interface is specified by the Cisco CRS-1 Series XML schemas. The XML API provides a mechanism for router configuration and monitoring utilizing an exchange of XML formatted request and response streams.

Client applications can be used to configure the router, or to request status information from the router, by encoding a request in Cisco CRS-1 Series XML API tags and sending it to the router. The Cisco CRS-1 Series router processes the request and sends the response to the client by again encoding the response in Cisco CRS-1 Series XML API tags. This guide describes the XML requests that can be sent by external client applications to access router management data, and also details the responses to the client by the Cisco CRS-1 Series router.

The Cisco CRS-1 Series XML API readily supports available transport layers. The initial release of the API supports the Common Object Request Broker Architecture (CORBA). Later on, the API can support additional transport options.

Customers use a variety of vendor-specific command line interface (CLI) scripts to manage their routers because no alternative programmatic mechanism is available. In addition, a common framework has not been available to develop CLI scripts. In response to this need, Cisco CRS-1 Series XML API provides the necessary common framework for rapid development, deployment, and maintenance of Cisco CRS-1 Series router management.

Note The XML API code is available for use on any Cisco platform that runs Cisco IOS XR software.

Definition of Terms

Table 1-1 defines the words, acronyms, and actions used throughout this guide.

Table 1-1 Definition of Terms

Term	Description
AAA	Authentication, authorization, and accounting
CLI	Command-line interface
CORBA	Common Object Request Broker Architecture
CWI	Craft Works Interface
SSH	Secure Shell
XML	Extensible markup language
XML agent	A process on the Cisco CRS-1 Series router that receives XML requests by XML clients, and is responsible to carry out the actions contained in the request and to return an XML response to the client.
XML client	An external application that sends XML requests to the Cisco CRS-1 Series router and receives XML responses to those requests.
XML operation	A portion of an XML request that specifies an operation that the XML client wants the XML agent to perform.
XML operation provider	The Cisco CRS-1 Series code that carries out a particular XML operation including parsing the operation XML, performing the operation, and assembling the operation XML response.
XML request	An XML document sent to the Cisco CRS-1 Series router containing a number of requested operations to be carried out.
XML response	The response to an XML request.
XML schema	An XML document specifying the structure and possible contents of XML elements that can be contained in an XML document.

Cisco CRS-1 Series Management XML Interface

This guide provides the following information regarding the Cisco CRS-1 Series management XML interface:

- High-level structure of the Cisco CRS-1 Series XML request and response streams
- Operation tag types and usage, including their XML format and content

- How to use Cisco CRS-1 Series XML to configure the Cisco CRS-1 Series router:
  - Using the two–stage “target configuration” mechanism provided by the Cisco CRS-1 Series configuration manager
  - Using features such as locking, loading, browsing, modifying, saving, and committing the configuration
- Accessing the Cisco CRS-1 Series router’s operational data with XML
- Working with the native management data object class hierarchies:
  - To represent the native data objects in XML
  - To use various techniques for structuring XML requests to access the management data of interest, including the use of wildcards and filters
- Encapsulating CLI commands in XML
- How error information is returned to the client application
- Using iterators for large data retrieval
- Handling event notifications with XML
- How authorization of client requests is enforced
- Versioning of the Cisco CRS-1 Series XML schemas
- Generation and packaging of the Cisco CRS-1 Series XML schemas

Cisco CRS-1 Series XML API and Cisco CRS-1 Series Router System Features

Using the Cisco CRS-1 Series XML API, an external client application can send XML encoded management requests to an XML agent running on the Cisco CRS-1 Series router. The initial release of the API supports CORBA as the transport mechanism. Using CORBA, the client application sends XML encoded requests to the router through remote operation calls. The remote operation interface is implemented on the router by the CORBA specific XML agent. Later on, the Cisco CRS-1 Series XML API can support other transport mechanisms, which might include terminal based protocols such as Telnet and Secure Shell (SSH). The Cisco CRS-1 Series XML API interface described in this document applies equally to all supported transports.

Before an XML session is established, the XML transport and XML agent must be enabled on the router. For more information, see Chapter 11, “XML Transport and Event Notifications.”

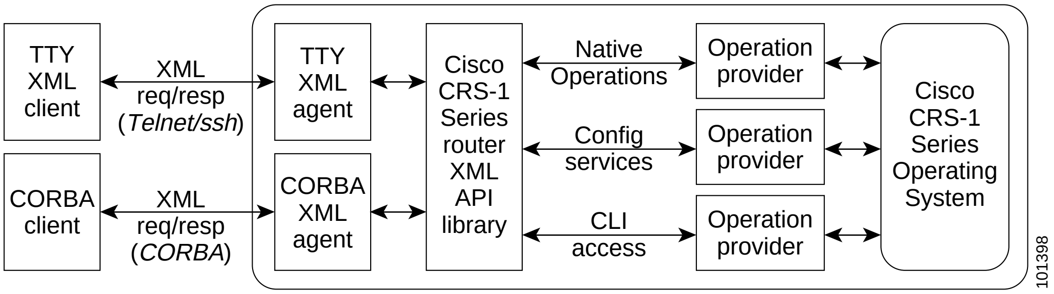
A client request sent to the Cisco CRS-1 Series router must specify the different types of operations to be carried out. The following three general types of management operations are supported through XML:

- Native data access (get, set, delete, and so on) using the native management data model.
- Configuration services for more advanced configuration management through the Configuration Manager.
- Traditional CLI access where the CLI commands and command responses are encapsulated in XML.

When a client request is received by an XML agent on the router, the request is routed to the appropriate XML operation provider in the internal Cisco CRS-1 Series XML API library for processing. After all the requested operations are processed, the XML agent receives the result and then sends the XML encoded response stream on to the client.

Figure 1-1 presents a high level view of the XML manageability-related components along with the request and response flows between the client application and the Cisco CRS-1 Series router.

Figure 1-1 XML Components and Request/Response Flows



Cisco CRS-1 Series XML API Tags

An external client application can access management data on the Cisco CRS-1 Series router through an exchange of well structured XML-tagged request and response streams. The XML taggest request and response streams are described in the following sections:

- Basic XML Request Content, page 1-16
- XML Declaration Tag, page 1-17
- Operation Type Tags, page 1-19
- XML Request Batching, page 1-21

Basic XML Request Content

This section describes the specific content and format of the XML data exchanged between the client and the Cisco CRS-1 Series router for the purpose of router configuration and monitoring.

Top-Level Structure

The top level of every request sent by a client application to the Cisco CRS-1 Series router must begin with an XML declaration tag followed by a request tag and one or more operation type tags. Similarly, every response returned by the Cisco CRS-1 Series router must begin with an XML declaration tag followed by a response tag and one or more operation type tags. Each request can contain operation tags for each supported operation type and the operation type tags can be repeated. The operation type tags contained in the response corresponds to those contained in the client request.

Sample XML Request from Client Application

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

```
         Operation-specific content goes here
        .
        .
        .
  </Operation>
</Request>
```

Sample XML Response from Router

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

Note All examples in this document are formatted with new lines and white space to aid readability. Actual XML request and response streams exchanged with the Cisco CRS-1 Series router do not include new lines and white space characters because these elements would add significantly to the size of the XML data and impact the overall performance of the Cisco CRS-1 Series XML API.

XML Declaration Tag

Each request and response exchanged between a client application and the Cisco CRS-1 Series router must begin with an XML declaration tag indicating which version of XML and (optionally) which character set are being used:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Table 1-2 defines the attributes of the XML declaration that are defined by the XML specification.

Table 1-2 Attributes for XML Declaration

Name	Description
Version	Specifies the version of XML to be used. Only Version"1.0" is supported by the Cisco CRS-1 Series router. **Note** The version attribute is required.
Encoding	Specifies the standardized character set to be used. Only "UTF-8" is supported by the Cisco CRS-1 Series router. The Cisco CRS-1 Series router includes the encoding attribute in a response only if it was specified in the corresponding request. **Note** The encoding attribute is optional.

Request and Response Tags

Following the XML declaration tag a client application must enclose each request stream within a set of <Request> start and </Request> end tags. Also, the Cisco CRS-1 Series system encloses each XML response within a set of <Response> start and </Response> end tags. A major and minor version number are carried on the <Request> and <Response> elements to indicate the overall Cisco CRS-1 Series XML API version in use by the client application and router respectively.

The Cisco CRS-1 Series XML API presents a synchronous interface to client applications. The <Request> and <Response> tags are used by the client to correlate the request and response streams. A client application can issue a request after which the Cisco CRS-1 Series router returns a response. The client can then issue another request, and so on. Therefore, the XML session between a client and the Cisco CRS-1 Series router consist of a series of alternating requests and response streams.

The client application optionally includes a ClientID attribute within the <Request> tag. The value of the ClientID attribute must be an unsigned 32-bit integer value. If the <Request> tag contains a ClientID attribute, the Cisco CRS-1 Series router includes the same ClientID value in the corresponding <Response> tag. The ClientID value is treated as opaque data and is ignored by the router.

Maximum Request Size

The maximum size of an XML request or response is determined by the restrictions of the underlying transports.

Note For more information on transport specific limitations of request and response sizes, see Chapter 11, “XML Transport and Event Notifications.”

Minimum Response Content

If a <Get> request has nothing to return for a particular operation, the Cisco CRS-1 Series router returns the original request and an appropriate empty operation type tag. The minimum response returned by the Cisco CRS-1 Series router in the case of a single operation (for example, <Get>, <Set>, <Delete>) with no result data is shown in the following example.

Sample XML Request from Client Application

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

Sample XML Minimum Response from Cisco CRS-1 Series Router

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

Operation Type Tags

Following the <Request> tag, the client application must specify the operation types to be carried out by the Cisco CRS-1 Series router. There are three general types of operations supported along with the <GetNext> operation for large responses. Each of these operation types is described in the following sections.

Native Data Operation Tags

Native data operations provide basic access to the Cisco CRS-1 Series native management data model. Table 1-3 describes the native data operation tags.

Table 1-3 Native Data Operation Tags

Native Data Tags	Description
<Get>	Get the value of one or more configuration, operational, or action data items.
<Set>	Create/modify one or more configuration or action data items.
<Delete>	Delete one or more configuration data items.
<GetVersionInfo>	Get the major and minor version numbers for one or more components.

Tip The XML schema definitions for the native data operation type tags are contained in the schema file common_types.xsd.

The native data operations are described further in Chapter 5, "Cisco CRS-1 Series XML and Native Data Access Techniques."

Configuration Services Operation Tags

Configuration services operations provide more advanced configuration management functions through the Cisco CRS-1 Series Configuration Manager. Table 1-4 describes the configuration services operation tags.

Table 1-4 Configuration Services Operation Tags

Tags	Description
<Lock>	Lock the running configuration.
<Unlock>	Unlock the running configuration.
<Load>	Load the target configuration from a binary file previously saved by way of the <Save> tag.
<Save>	Save the target configuration to a binary file.
<Commit>	Promote the target configuration to the running configuration.
<Clear>	Abort/clear the current target configuration session.
<Rollback>	Roll back the running configuration to a previous configuration state.

Table 1-4 Configuration Services Operation Tags (continued)

Tags	Description
<GetConfigurationHistory>	Get a list of commits made to the running configuration.
<GetConfigurationSessions>	Get a list of the user sessions currently configuring the box.

Note The XML schema definitions for the configuration services operation type tags are contained in the schema file cfgmgr_operations.xsd (see Chapter 13, "Cisco CRS-1 Series XML Schemas").

Note The configuration services operations are described further in Chapter 2, "Cisco CRS-1 Series XML Router Configuration and Management."

CLI Operation Tag

CLI access provides support for XML encapsulated CLI commands and responses. For CLI access, a single tag is provided. The <CLI> operation tag issues the request as a CLI command.

Note The XML schema definitions for the CLI tag are contained in the schema file common_types.xsd (see Chapter 13, "Cisco CRS-1 Series XML Schemas").

Note The CLI operations are described further in Chapter 6, "Cisco CRS-1 Series XML and Encapsulated CLI Operations."

GetNext Operation Tag

The <GetNext> tag is used to retrieve the next portion of a large response. It can be used as required to retrieve an oversize response following a request using one of the other operation types. The<GetNext> operation tag gets the next portion of a response. Iterators are supported for large requests.

Note The XML schema definition for the <GetNext> operation type tag is contained in the schema file common_types.xsd (see Chapter 13, "Cisco CRS-1 Series XML Schemas").

Note The get next operation is described further in Chapter 7, "Cisco CRS-1 Series XML and Large Data Retrieval (Iterators)."

XML Request Batching

The Cisco CRS-1 Series XML interface supports the combining of several requests or operations into a single request. When multiple operations are specified in a single request, the response contains the same operation tags and in the same order as they appeared in the request.

Batched requests are performed “best effort.” For example, if there are operations 1 through 3 in the request and operation 2 fails, operation 3 will be attempted.

Before combining multiple operations inside one <Get> tag, any operations that request multiple items of data must be sent in a separate document. For more information, see Chapter 5, “Cisco CRS-1 Series XML and Native Data Access Techniques,” “XML Request with Combined Object Class Hierarchies” section on page 5-67.

The following example shows a simple request containing six different operations:

Sample XML Client Batched Requests

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
   <Lock/>
   <Get>
     <Configuration>
        .
        .
        .
          Get operation content goes here
        .
        .
        .
     </Configuration>
   </Get>
   <Set>
     <Configuration>
        .
        .
        .
          Set operation content goes here
        .
        .
        .
     </Configuration>
   </Set>
   <Commit/>
   <Get>
     <Operational>
        .
        .
        .
          Get operation content goes here
        .
        .
        .
     </Operational>
  </Get>
 <Unlock/>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Lock/>
  <Get>
    <Configuration>
        .
        .
        .
        .
        .
        .
        .
        .
        .
        Get response content returned here
        .
        .
        .
        .
        .
        .
        .
        .
        .
    </Configuration>
  </Get>
  <Set/>
  <Commit/>
  <Get>
    <Operational>
        .
        .
        .
        .
        .
        .
        .
        .
        .
        Get response content returned here
        .
        .
        .
        .
        .
        .
        .
        .
        .
    </Operational>
  </Get>
  <Unlock/>
</Response>
```

CHAPTER 2

Cisco CRS-1 Series XML Router Configuration and Management

This chapter reviews the basic extensible markup language (XML) requests and responses used to configure and manage the Cisco CRS-1 Series Carrier Routing System (Cisco CRS-1 Series) router.

Note The XML application programming interface (API) code is available for use on any Cisco platform that runs Cisco IOS XR software.

The use of XML to configure the Cisco CRS-1 Series router is essentially an abstraction of a configuration editor where 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

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 a Cisco CRS-1 Series 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

Use the following configuration options from the client application to configure or modify the Cisco CRS-1 Series router with XML:

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).

Additional Configuration Options Using XML

Several additional, optional configuration tasks are available to the client application for configuring or modifying the Cisco CRS-1 Series router with XML:

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 Cisco CRS-1 Series 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 Cisco CRS-1 Series 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 CRS-1 Series XML and Native Data Operations.” Encoding CLI commands within XML tags is described in Chapter 6, “Cisco CRS-1 Series 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 Cisco CRS-1 Series 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>
```

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:

```
router bgp 3
  default-metric 10
  no neighbor 10.0.101.8
exit
show config
```

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>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series 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 Cisco CRS-1 Series 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>
</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 ErrorMsg 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 Cisco CRS-1 Series 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 29.

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 10, "Error Reporting in Cisco CRS-1 Series XML Responses."

The content and format of <Delete> and <Set> requests are described in additional detail in Chapter 4, "Cisco CRS-1 Series XML and Native Data Operations." Encoding CLI commands within XML tags is described in Chapter 6, "Cisco CRS-1 Series 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:

```
router bgp 3
  default-metric 10
  timers bgp 60 180
  bgp log neighbor changes
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 Cisco CRS-1 Series 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 the "Replacing the Current Running Configuration" section on page 2-44.

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 Cisco CRS-1 Series 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 ErrorMsg 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 Cisco CRS-1 Series 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 carried out on a BestEffort 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 Cisco CRS-1 Series commit database. If specified, the label must begin with an alphabetic character and cannot match any existing label in the Cisco CRS-1 Series commit database.
Comment	Use the Comment attribute as a user-specified comment to be associated with the commit in the Cisco CRS-1 Series router commit database.

If the commit operation is successful, the response contains just the <Commit/> tag along with any 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 and specifies that any failed configuration be retained in the target buffer. This request corresponds to the CLI command **commit atomic**.

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 Cisco CRS-1 Series Router

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

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

- After each successful commit operation, a commit record is created in the Cisco CRS-1 Series router commit database. The Cisco CRS-1 Series router maintains up to 20 entries in the commit database corresponding to the last 20 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 2-44.
- Applying the configuration for a trial period (“try-and-apply”) is not be 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 makes 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 ErrrorMsg 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:

```
router bgp 4
  default-metric 10
exit
commit
```

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 Cisco CRS-1 Series 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/>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router Showing Failures for Both Requested Operations

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Commit ErrorCode="0x40819c00" ErrorMsg="&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" ErrorMsg="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 2-34.

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 Cisco CRS-1 Series 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 ErrorMsg 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 Cisco CRS-1 Series Router

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

```
  <Unlock/>
</Response>
```

Additional Cisco CRS-1 Series 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 2-36
- Clearing a Target Session, page 2-38
- Rolling Back Configuration Changes to a Specified Commit Identifier, page 2-38
- Rolling Back Configuration Changes to a Specified Number of Commits, page 2-39
- Getting Rollback Changes, page 2-40
- Getting Configuration History, page 2-41
- Getting Configuration Session Information, page 2-43
- Replacing the Current Running Configuration, page 2-44

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 Cisco CRS-1 Series 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/0/0/0:Jun 18 19:16:42.561 : %CLIENTLIBCFGMGR-6-CONFIG_CHANGE : A configuration commit
by user ’root’ occurred at ’Wed Jun 18 19:16:18 2003 ’. 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 Cisco CRS-1 Series XML API, see Chapter 11, “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 1000000075**.

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=”1000000075”/>
  </Get>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration Source=”CommitChanges” ForCommitID=”1000000075”>
      .
      .
      .
        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 1000000072**.

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=”1000000072”/>
  </Get>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration Source=”CommitChanges” SinceCommitID=”1000000072”>
      .
      .
      .
        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 Cisco CRS-1 Series 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 Cisco CRS-1 Series 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 roll back in the Cisco CRS-1 Series router commit database. If specified, the label must begin with an alphabetic character and cannot match any existing label in the Cisco CRS-1 Series router commit database.
Comment	A user-specified comment to be associated with the roll back in the Cisco CRS-1 Series 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 1000000072**.

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>1000000072</CommitID>
  </Rollback>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Rollback Label=”BGPRollback1” Comment=”My BGP rollback”>
    <CommitID>1000000072</CommitID>
  </Rollback>
</Response>
```

Note The commit identifier can also be obtained by using the <GetConfigurationHistory> operation described in the section "Getting Configuration History".

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 ErrorMsg 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 roll back in the Cisco CRS-1 Series router commit database. If specified, the label must begin with an alphabetic character and cannot match any existing label in the Cisco CRS-1 Series router commit database.
Comment	A user-specified comment to be associated with the roll back in the Cisco CRS-1 Series router commit database.

The following 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 Cisco CRS-1 Series 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 1000000072**.

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=”1000000072”/>
  </Get>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration Source=”RollbackChanges” ToCommitID=”1000000072”>
      .
      .
      .
        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 Cisco CRS-1 Series 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
<CommitID>	The unique ID associated with the commit.
<Label>	The optional label associated with the commit.
<UserID>	The name of the user who created the configuration session within which the commit was performed.
<Line>	The line used to connect to the router for the configuration session.
<ClientName>	The name of the client application that performed the commit.
<Timestamp>	The date and time of the commit.
<Comment>	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 Cisco CRS-1 Series router maintains a commit database of up to 20 records corresponding to the last 20 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 history 3.**

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 Cisco CRS-1 Series 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 June 18 19:16:18 2003</Timestamp>
      <Comment>BGP config update</Comment>
    </CommitEntry>
    <CommitEntry>
      <CommitID>1000000074</CommitID>
      <Label xsi:nill=”true”>
      <UserID>unknown</UserID>
      <Line>con0_0_0</Line>
      <ClientName>CLI</ClientName>
      <Timestamp>Wed June 18 03:08:07 2003</Timestamp>
      <Comment xsi:nill=”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 Cisco CRS-1 Series 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
<SessionID>	The unique autogenerated ID for the configuration session.
<UserID>	The name of the user who created the configuration session.
<Line>	The line used to connect to the router.
<ClientName>	The user-friendly name of the client application that created the configuration session.
<Since>	The date and time of the creation of the configuration session.
<LockHeld>	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 Cisco CRS-1 Series Router

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

Sample XML Response from the Cisco CRS-1 Series 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>
      <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 Cisco CRS-1 Series 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 Cisco CRS-1 Series 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 data goes here
      .
      .
      .
    </Configuration>
  </Set>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

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

Cisco CRS-1 Series XML Operational Requests and Fault Management

A client application can send an extensible markup language (XML) request to get the Cisco CRS-1 Series Carrier Routing System (Cisco CRS-1 Series) router operational information using either a native data <Get> request along with the <Operational> tag, or the equivalent command-line interface (CLI) command. Although the CLI is more familiar to users, the advantage of using the <Get> request is that the response data is encoded in XML format instead of being just uninterpreted text enclosed within <CLI> tags.

Note The XML application programming interface (API) code is available for use on any Cisco platform that runs Cisco IOS XR software.

This chapter contains the following sections:

- Operational Get Requests, page 3-47
- Action Requests, page 3-48

Operational Get Requests

The content and format of operational <Get> requests are described in additional detail in Chapter 4, "Cisco CRS-1 Series XML and Native Data Operations."

The following example shows a <Get> request to retrieve the global Border Gateway Protocol (BGP) process information. This request returns BGP process information similar to that displayed by the CLI command **show ip bgp process detail**.

Sample XML Client Request to Get BGP Information

```
<?xml version=”1.0” encoding=”UTF-8”?>
<Request MajorVersion="1" MinorVersion="0">
  <Get>
    <Operational>
      <BGP MajorVersion=”1” MinorVersion=”0”>
        <GlobalProcessInfo/>
      </BGP>
    </Operational>
  </Get>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get>
    <Operational>
      <BGP MajorVersion=”1” MinorVersion=”0”>
        <GlobalProcessInfo>
          <InStandaloneMode>true</InStandaloneMode>
          <RouterID>10.1.0.1</RouterID>
          <ConfiguredRouterID>33.67.205.171</ConfiguredRouterID>
          <LocalAS>3</LocalAS>
          <RestartCount>1</RestartCount>
            .
            .
            .
               returned data
            .
            .
            .
          <IsGracefulRestart>true</IsGracefulRestart>
          <RestartTime>180</RestartTime>
          <StalePathTime>300</StalePathTime>
          <RIBPurgeTimeout>300</RIBPurgeTimeout>
          <UpdateDelay>55</UpdateDelay>
        </GlobalProcessInfo>
      </BGP>
    </Operational>
  </Get>
</Response>
```

Action Requests

A client application can send a <Set> request along with the <Action> tag to trigger unique actions on the router. For example, an object may be set with an action request to inform the router to clear a particular counter or reset some functionality. Most often this operation involves setting the value of a Boolean object to “true”. The content and format of <Set> requests are described in additional detail in Chapter 4, “Cisco CRS-1 Series XML and Native Data Operations.”

The following example shows an action request to clear the BGP performance statistics information. This request is equivalent to the CLI command **clear ip bgp performance-statistics**.

Sample XML Request to Clear BGP Performance Statistics Information

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Set>
    <Action>
      <BGP MajorVersion="0" MinorVersion="0">
        <ClearPerformanceStats>true</ClearPerformanceStats>
      </BGP>
    </Action>
  </Set>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

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

```
 </Set>
</Response>
```

The following is an additional example showing an action request to clear the peer drop information for all BGP neighbors. This request is equivalent to the CLI command **clear ip bgp peer-drops ***.

Sample XML Request to Clear Peer Drop Information for All BGP Neighbors

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Set>
    <Action>
      <BGP MajorVersion="0" MinorVersion="0">
        <ClearDrops>
          <All>true</All>
        </ClearDrops>
      </BGP>
    </Action>
  </Set>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

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

Cisco CRS-1 Series XML and Fault Management

When a client application successfully commits the target configuration to the Cisco CRS-1 Series router’s running configuration, the configuration manager writes a single configuration change event to system message logging (syslog). As a result, a fault management event notification is written to the Cisco CRS-1 Series Alarm Channel (that is, the Common Object Request Broker Architecture [CORBA] event notification channel for alarms) and subsequently forwarded to any registered configuration agents.

Configuration Change Notification

Table 3-1 provides event notification for configuration changes information.

Table 3-1 Event Notifications

Event Notification	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/0/0/1:Sep 18 09:43:42.747 : %CLIENTLIBCFGMGR-6-CONFIG_CHANGE : A configuration commit
by user root occurred at ’Wed Sep 18 09:43:42 2002 ’. The configuration changes are saved
on the router in file: 010208180943.0
```

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

CHAPTER 4

Cisco CRS-1 Series XML and Native Data Operations

Native data operations <Get>, <Set>, and <Delete> provide basic access to configuration and operational data residing on the Cisco CRS-1 Series Carrier Routing System (Cisco CRS-1 Series) router.

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

Note The XML application programming interface (API) code is available for use on any Cisco platform that runs Cisco IOS XR software.

Native Data Operation Content

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

The operations are described in the following sections:

- Request Type Tag and Namespaces, page 4-52
- Object Hierarchy, page 4-52
- Dependencies Between Configuration Items, page 4-55
- Null Value Representations, page 4-56
- Operation Triggering, page 4-56
- Native Data Operation Examples, page 4-57

The following 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 Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Operation>
    <Request Type>
      .
      .
      .
        response content returned here
      .
      .
      .
    </Request Type>
  </Operation>
</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 the 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 command-line interface (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 Cisco CRS-1 Series 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 Cisco CRS-1 Series 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 the 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.

The following 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 (see Chapter 5, “Cisco CRS-1 Series XML and Native Data Access Techniques,” for details on how to do perform hierarchical data retrieval).

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.

The following example shows hierarchy tables:

```
<Configuration>
  <BGP>
    .
    .
    .
    <BGPEntity>
      <NeighborTable>
        <Neighbor>
          <Naming>
            <IPAddress>
              <IPV4Address>10.0.101.6</IPV4Address>
            </IPAddress>
          </Naming>
          <RemoteAS>6</RemoteAS>
        </Neighbor>
        <Neighbor>
          <Naming>
            <IPAddress>
              <IPV4Address>10.0.101.7</IPV4Address>
            </IPAddress>
          </Naming>
          <RemoteAS>7</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. See the "XML Request Using Operation Scope (Content Attribute)" section on page 5-74 in Chapter 5, "Cisco CRS-1 Series XML and Native Data Access Techniques."

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

The following example shows table filtering:

```
<Operational>
  <BGP>
    <NeighborTable>
      <Filter>
        <BGP_AFFilter>
          <AF>IPv4Unicast</AF>
        </BGP_AFFilter>
      </Filter>
    </NeighborTable>
  </BGP>
</Operational>
```

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 4-53. 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.

The following 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 of 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 Cisco CRS-1 Series operating system applications are 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.

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

```
<Neighbor>
  <Timers>
    <KeepAlive>60</KeepAlive>
    <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 Cisco CRS-1 Series 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 following general rule regarding what to specify in the XML for an operation to take place: As a client XML request is parsed by the Cisco CRS-1 Series 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).

The following example shows a request to get the BGP timer values for a particular BGP autonomous system. In this example, the <Get> operation is triggered when the <GlobalTimers/> 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 MajorVersion=”1” MinorVersion=”0”>
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <Global>
            <GlobalTimers/>
          </Global>
        </AS>
      </BGP>
    </Configuration>
  </Get>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

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

```
      <BGP MajorVersion=”1” MinorVersion=”0”>
        <AS>
          <Naming>
            <AS>1</AS>
          </Naming>
          <Global>
            <GlobalTimers>
              <Keepalive>60</Keepalive>
              <Holdtime>180</Holdtime>
            </GlobalTimers>
          </Global>
        </AS>
      </BGP>
    </Configuration>
  </Get>
</Response>
```

Native Data Operation Examples

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

- Set Configuration Data Request: Example, page 4-57
- Get Request: Example, page 4-58
- Get Request of Nonexistent Data: Example, page 4-60
- Delete Request: Example, page 4-61

Set Configuration Data Request: Example

The following 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 the following CLI commands:

```
router bgp 3
  neighbor 10.0.101.6
    remote-as 6
    ebgp-multihop 255
    address-family ipv4 unicast
      prefix-list orf in
      capability orf prefix-list both
    exit
    address-family ipv4 multicast
      prefix-list orf in
    exit
  exit
exit
```

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 MajorVersion="1" MinorVersion="0">
         <AS>
           <Naming>
             <AS>3</AS>
           </Naming>
```

```
            <BGPEntity>
              <NeighborTable>
                <Neighbor>
                  <Naming>
                    <IPAddress>
                      <IPV4Address>10.0.101.6</IPV4Address>
                    </IPAddress>
                  </Naming>
                  <RemoteAS>6</RemoteAS>
                  <EBGPMultihopMaxHopCount>255</EBGPMultihopMaxHopCount>
                  <NeighborAFTable>
                    <NeighborAF>
                      <Naming>
                        <AF>IPv4Unicast</AF>
                      </Naming>
                      <Activate>true</Activate>
                      <PrefixListFilterIn>orf</PrefixListFilterIn>
                      <AdvertiseORF>Both</AdvertiseORF>
                    </NeighborAF>
                    <NeighborAF>
                      <Naming>
                        <AF>IPv4Multicast</AF>
                      </Naming>
                      <Activate>true</Activate>
                      <PrefixListFilterIn>orf</PrefixListFilterIn>
                    </NeighborAF>
                  </NeighborAFTable>
                </Neighbor>
              </NeighborTable>
            </BGPEntity>
          </AS>
        </BGP>
      </Configuration>
    </Set>
  </Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

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

Get Request: Example

The following 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 MajorVersion="1" MinorVersion="0">
       <AS>
         <Naming>
           <AS>3</AS>
         </Naming>
          <BGPEntity>
```

```
           <NeighborTable>
             <Neighbor>
               <Naming>
                 <IPAddress>
                    <IPV4Address>10.0.101.6</IPV4Address>
                  </IPAddress>
                </Naming>
              </Neighbor>
            </NeighborTable>
          </BGPEntity>
        </AS>
      </BGP>
    </Configuration>
  </Get>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration>
      <BGP MajorVersion="1" MinorVersion="0">
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <BGPEntity>
            <NeighborTable>
              <Neighbor>
                <Naming>
                  <IPAddress>
                    <IPV4Address>10.0.101.6</IPV4Address>
                  </IPAddress>
                </Naming>
                <RemoteAS>6</RemoteAS>
                <EBGPMultihopMaxHopCount>255</EBGPMultihopMaxHopCount>
                <NeighborAFTable>
                  <NeighborAF>
                    <Naming>
                      <AF>IPv4Unicast</AF>
                    </Naming>
                    <Activate>true</Activate>
                    <PrefixListFilterIn>orf</PrefixListFilterIn>
                    <AdvertiseORF>Both</AdvertiseORF>
                  </NeighborAF>
                  <NeighborAF>
                    <Naming>
                      <AF>IPv4Multicast</AF>
                    </Naming>
                    <Activate>true</Activate>
                    <PrefixListFilterIn>orf</PrefixListFilterIn>
                  </NeighborAF>
                </NeighborAFTable>
              </Neighbor>
            </NeighborTable>
          </BGPEntity>
        </AS>
      </BGP>
    </Configuration>
  </Get>
</Response>
```

Get Request of Nonexistent Data: Example

The following example shows a native data request to get the configuration values for a particular BGP neighbor 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 Cisco CRS-1 Series router returns the requested object class hierarchy, but without any data.

Note Whenever a client application attempts to get nonexistent data, the Cisco CRS-1 Series router usually cannot treat this as an error and returns the empty object class 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=”1” MinorVersion=”0”>
        <AS>
         <Naming>
           <AS>3</AS>
         </Naming>
          <BGPEntity>
           <NeighborTable>
             <Neighbor>
               <Naming>
                 <IPAddress>
                   <IPV4Address>10.0.101.99</IPV4Address>
                 </IPAddress>
               </Naming>
             </Neighbor>
           </NeighborTable>
         </BGPEntity>
        </AS>
      </BGP>
    </Configuration>
  </Get>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration>
      <BGP MajorVersion=”1” MinorVersion=”0”>
        <AS>
         <Naming>
            <AS>3</AS>
         </Naming>
         <BGPEntity>
            <NeighborTable>
             <Neighbor>
                <Naming>
                  <IPAddress
                    <IPV4Address>10.0.101.99</IPV4Address>
                  </IPAddress
                </Naming>
                 .
                 .
                 .
                    no data returned
                 .
                 .
                 .
              <Neighbor>
            </NeighborTable>
          </BGPEntity>
        </AS>
      </BGP>
    </Configuration>
  </Get>
</Response>
```

Delete Request: Example

The following 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 the following 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 the following 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=”1” MinorVersion=”0”>
        <AS>
         <Naming>
           <AS>3</AS>
         </Naming>
         <BGPEntity>
          <NeighborTable>
            <Neighbor>
              <Naming>
                <IPAddress>
                  <IPV4Address>10.0.101.9</IPV4Address>
                </IPAddress>
              </Naming>
            </Neighbor>
           </NeighborTable>
         </BGPEntity>
        </AS>
      </BGP>
    </Configuration>
  </Delete>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

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

CHAPTER 5

Cisco CRS-1 Series XML and Native Data Access Techniques

This chapter describes the various techniques or strategies you can use to structure native data operation requests to access the information needed within the extensible markup language (XML) schema object class hierarchy.

Note The XML application programming interface (API) code is available for use on any Cisco platform that runs Cisco IOS XR software.

Available Set of Native Data Access Techniques

The available native data access techniques are as follows:

- Request all data in the configuration hierarchy. See "XML Request for All Configuration Data" section on page 5-64.
- Request all configuration data for a component. See "XML Request for All Configuration Data per Component" section on page 5-64.
- Request all data within a container. See the "XML Request for Specific Data Items" section on page 5-66.
- Combine object class hierarchies within a request. See the "XML Request with Combined Object Class Hierarchies" section on page 5-67.
- Use wildcards in order to apply an operation to a set of entries within a table (Match attribute). See the "XML Request Using Wildcarding (Match Attribute)" section on page 5-70.
- Repeat naming information in order to apply an operation to multiple instances of an object. See the "XML Request for Specific Object Instances (Repeated Naming Information)" section on page 5-72.
- Perform a one-level <Get> in order to "list" the naming information for each entry within a table (Content attribute). See the "XML Request Using Operation Scope (Content Attribute)" section on page 5-74.
- Specify the maximum number of table entries to be returned in a response (Count attribute). See the "Limiting the Number of Table Entries Returned (Count Attribute)" section on page 5-76.
- Use custom filters to filter table entries (Filter element). See the "Custom Filtering (Filter Element)" section on page 5-77.

The actual data returned in a <Get> request depends on the value of the Source attribute as defined in the “Getting Configuration Data” section on page 2-26.

Note The term “container” is used in this document as a general reference to any grouping of related data, for example, all of the configuration data for a particular Border Gateway Protocol (BGP) neighbor. The term “table” is used more specifically to denote a type of container that holds a list of named homogeneous objects. For example, the BGP neighbor address table contains a list of neighbor addresses, each of which is identified by its IP address. All table entries in the Cisco CRS-1 Series Carrier Routing System (Cisco CRS-1 Series) XML API are identified by the unique value of their <Naming> element.

XML Request for All Configuration Data

Use the empty <Configuration/> tag to retrieve the entire configuration object class hierarchy.

The following example shows how to get the entire configuration hierarchy by specifying the empty <Configuration/> tag.

Sample XML Client Request to <Get> the Entire Configuration Object Class Hierarchy

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

Sample XML Response from the Cisco CRS-1 Series Router

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

XML Request for All Configuration Data per Component

All the configuration data for a component is retrieved by specifying the highest level tag for the component.

In the following example, all the configuration data for BGP is retrieved by specifying the empty <BGP/> tag.

Sample XML Client Request for All BGP Configuration Data

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

```
  <Get>
    <Configuration>
      <BGP MajorVersion=”1” MinorVersion=”0”/>
    </Configuration>
  </Get>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

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

XML Request for All Data Within a Container

All data within a containers is retrieved by specifying the configuration or operational object class hierarchy down to the containers of interest, including any naming information as appropriate.

The following example shows how to retrieve the configuration for the BGP neighbor with address 10.0.101.6:

Sample XML Client Request to Get All Address Family-Independent Configuration Data Within a BGP Neighbor Container

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

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration>
      <BGP MajorVersion="1" MinorVersion="0">
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <BGPEntity>
            <NeighborTable>
              <Neighbor>
                <Naming>
                  <IPAddress>
                    <IPV4Address>10.0.101.6</IPV4Address>
                  </IPAddress>
                </Naming>
                <RemoteAS>6</RemoteAS>
                <EBGPMultihopMaxHopCount>255</EBGPMultihopMaxHopCount>
                <NeighborAFTable>
                  <NeighborAF>
                    <Naming>
                      <AF>IPv4Unicast</AF>
                    </Naming>
                    <Activate>true</Activate>
                    <PrefixListFilterIn>orf</PrefixListFilterIn>
                    <AdvertiseORF>Both</AdvertiseORF>
                  </NeighborAF>
                  <NeighborAF>
                    <Naming>
                      <AF>IPv4Multicast</AF>
                    </Naming>
                    <Activate>true</Activate>
                    <PrefixListFilterIn>orf</PrefixListFilterIn>
                  </NeighborAF>
                </NeighborAFTable>
              </Neighbor>
            </NeighborTable>
          </BGPEntity>
        </AS>
      </BGP>
    </Configuration>
  </Get>
</Response>
```

XML Request for Specific Data Items

The value of a specific data item (leaf object) can be retrieved by specifying the configuration or operational object class hierarchy down to the item of interest, including any naming information as appropriate.

The following example shows how to retrieve the values of the two data items <RemoteAS> and <EBGPMultihopMaxHopCount> for the BGP neighbor with address 10.0.101.6:

Sample XML Client Request for Two Specific Data Items: RemoteAS and EBGPMultihopMaxHopCount

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

```
      <BGP MajorVersion="1" MinorVersion="0">
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <BGPEntity>
            <NeighborTable>
              <Neighbor>
                <Naming>
                  <IPAddress>
                    <IPV4Address>10.0.101.6</IPV4Address>
                  </IPAddress>
                </Naming>
                <RemoteAS/>
                <EBGPMultihopMaxHopCount/>
              </Neighbor>
            </NeighborTable>
          </BGPEntity>
        </AS>
      </BGP>
    </Configuration>
  </Get>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration>
      <BGP MajorVersion="1" MinorVersion="0">
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <BGPEntity>
            <NeighborTable>
              <Neighbor>
                <Naming>
                  <IPAddress>
                    <IPV4Address>10.0.101.6</IPV4Address>
                  </IPAddress>
                </Naming>
                <RemoteAS>6</RemoteAS>
                <EBGPMultihopMaxHopCount>255</EBGPMultihopMaxHopCount>
              </Neighbor>
            </NeighborTable>
          </BGPEntity>
        </AS>
      </BGP>
    </Configuration>
  </Get>
</Response>
```

XML Request with Combined Object Class Hierarchies

Multiple object class hierarchies can be specified in a request. For example, a portion of the hierarchy can be repeated, and multiple instances of a child object class can be included under a parent.

The object class hierarchy may also be compressed into the most “efficient” XML. In other words, it is not necessary to repeat hierarchies within a request.

Before combining multiple operations inside one <Get> tag, the following limitations should be noted for Release 3.0. Any operations that request multiple items of data must be sent in a separate XML request. They include:

- An operation to retrieve all data beneath a container. For more information, see“XML Request for All Data Within a Container” section on page 5-65.
- An operation to retrieve the list of entries in a table. For more information, see “XML Request Using Operation Scope (Content Attribute)” section on page 5-74.
- An operation which includes a wildcard. For more information, see “XML Request Using Wildcarding (Match Attribute)” section on page 5-70.

If an attempt is made to make such an operation followed by another operation within the same request, the following error will be returned:

```
XML Service Library detected the ‘fatal’ condition. The XML document which led to
this response contained a request for a potentially large amount of data, which
could return a set of iterators. The document also contained further requests for
data, but these must be sent in a separate XML document, in order to ensure that
they are serviced.
```

The error indicates that the operations must be separated out into separate XML requests.

The following two examples illustrate two different object class hierarchies that retrieve the same data: the value of the leaf object <RemoteAS> and <EBGPMultihopMaxHopCount> for the BGP neighbor with the address 10.0.101.6 and all of the configuration data for the BGP neighbor with the address 10.0.101.7:

Example 1: Verbose Form of a Request Using Duplicated Object Class Hierarchies

Sample XML Client Request for Specific Configuration Data Values

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration>
      <BGP MajorVersion="1" MinorVersion="0">
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <BGPEntity>
            <NeighborTable>
              <Neighbor>
                <Naming>
                  <IPAddress>
                    <IPV4Address>10.0.101.6</IPV4Address>
                  </IPAddress>
                </Naming>
                <!-- Gets the following two leaf objects for this neighbor -->
                <RemoteAS/>
                <EBGPMultihopMaxHopCount/>
              </Neighbor>
            </NeighborTable>
          </BGPEntity>
        </AS>
      </BGP>
    </Configuration>
  </Get>
  <Get>
    <Configuration>
      <BGP MajorVersion="1" MinorVersion="0">
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <BGPEntity>
            <NeighborTable>
              <Neighbor>
               <Naming>
                 <IPAddress>
                   <!-- Gets all configuration data for this neighbor -->
                   <IPV4Address>10.0.101.7</IPV4Address>
                 </IPAddress>
               </Naming>
              </Neighbor>
            </NeighborTable>
          </BGPEntity>
        </AS>
      </BGP>
     </Configuration>
   </Get>
 </Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration>
            .
            .
            .
        response data returned here for
        neighbor 10.0.101.6
            .
            .
            .
    </Configuration>
  </Get>
  <Get>
    <Configuration>
            .
            .
            .
        response data returned here
        neighbor 10.0.101.7
            .
            .
            .
    </Configuration>
  </Get>
</Response>
```

Example 2: Compact Form of a Request Using Compressed Object Class Hierarchies

Sample XML Client Request

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

```
      <BGP MajorVersion="1" MinorVersion="0">
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <BGPEntity>
            <NeighborTable>
              <Neighbor>
                <Naming>
                  <IPAddress>
                    <IPV4Address>10.0.101.6</IPV4Address>
                  </IPAddress>
                </Naming>
                <!-- Gets the following two leaf objects for this neighbor -->
                <RemoteAS/>
                <EBGPMultihopMaxHopCount/>
              </Neighbor>
                <Neighbor>
                  <Naming>
                    <IPAddress>
                       <!-- Gets all configuration data for this neighbor -->
                       <IPV4Address>10.0.101.7</IPV4Address>
                    </IPAddress>
                  </Naming>
                </Neighbor>
            </NeighborTable>
          </BGPEntity>
        </AS>
      </Configuration>
    </Get>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration>
            .
            .
            .
        response data returned here for both
        neighbors
            .
            .
            .
    </Configuration>
  </Get>
</Response>
```

XML Request Using Wildcarding (Match Attribute)

Wildcarding of naming information is provided by means of the Match attribute. Match="*" can be used on any Naming attribute within a <Get> or <Delete> operation to effectively specify a wildcarded value for that attribute. The operation applies to all instances of the requested objects.

"*" is the only value supported for Match, though other wildcarding or matching specifications may be supported in the future. The Match attribute is comprehensively supported for table entries in the <Configuration> namespace, but in the <Operational> space the limitation currently exists that nonwildcarded naming information cannot appear in the hierarchy below wildcarded naming information.

Note Although partial wildcarding of NodeIDs is not available in XML, each element of the NodeID has to be wildcarded, similar to the support on the CLI of */*/* as the only wildcards supported for locations.

The following example shows how to use the Match attribute to get the <RemoteAS> value for all configured BGP neighbors.

Sample XML Client Request Using the Match Attribute Wildcarding

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration>
      <BGP MajorVersion="1" MinorVersion="0">
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <BGPEntity>
            <NeighborTable>
               <Neighbor>
                <Naming>
                  <IPAddress Match=”*”/>
                </Naming>
                <RemoteAS/>
              </Neighbor>
             </NeighborTable>
           </BGPEntity>
         </AS>
       </BGP>
     </Configuration>
  </Get>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration>
      <BGP MajorVersion="1" MinorVersion="0">
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <BGPEntity>
            <NeighborTable>
              <Neighbor>
                <Naming>
                  <IPAddress>
                    <IPV4Address>10.0.101.1</IPV4Address>
                  </IPAddress>
                </Naming>
                <RemoteAS>1</RemoteAS>
              </Neighbor>
              <Neighbor>
                <Naming>
                  <IPAddress>
                    <IPV4Address>10.0.101.2</IPV4Address>
                  </IPAddress>
                </Naming>
                <RemoteAS>2</RemoteAS>
              </Neighbor>
              <Neighbor>
                <Naming>
                  <IPAddress>
                    <IPV4Address>10.0.101.3</IPV4Address>
                  </IPAddress>
                </Naming>
                <RemoteAS>3</RemoteAS>
              </Neighbor>
                .
                .
                .
                  data for more neighbors
                  returned here
                .
                .
                .
            </NeighborTable>
          </BGPEntity>
        </AS>
      </BGP>
    </Configuration>
  </Get>
</Response>
```

XML Request for Specific Object Instances (Repeated Naming Information)

Wildcarding allows the client application to effectively specify all instances of a particular object. Similarly, the client application might have a need to specify only a limited set of instances of an object. Specifying object instances can be done by simply repeating the naming information in the request.

The following example shows how to retrieve the address independent configuration for three different BGP neighbors, that is, the neighbors with addresses 10.0.101.1, 10.0.101.6, and 10.0.101.8, by repeating the naming information, once for each desired instance.

Sample XML Client Request Using Repeated Naming Information for BGP <NeighborAddress> Instances

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration>
      <BGP MajorVersion="1" MinorVersion="0">
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <BGPEntity>
            <NeighborTable>
              <Neighbor>
                <Naming>
                  <IPAddress>
                    <IPV4Address>10.0.101.1</IPV4Address>
                  </IPAddress>
                </Naming>
              </Neighbor>
            </NeighborTable>
              <NeighborTable>
                <Neighbor>
                  <Naming>
                    <IPAddress>
                      <IPV4Address>10.0.101.6</IPV4Address>
                    </IPAddress>
                  </Naming>
                </Neighbor>
              </NeighborTable>
                <NeighborTable>
                <Neighbor>
                  <Naming>
                    <IPAddress>
                      <IPV4Address>10.0.101.8</IPV4Address>
                    </IPAddress>
                  </Naming>
                </Neighbor>
               </NeighborTable>
              </BGPEntity>
            </AS>
          </BGP>
        </Configuration>
      </Get>
    </Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration>
      <BGP MajorVersion="1" MinorVersion="0">
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <BGPEntity>
            <NeighborTable>
              <Neighbor>
                <Naming>
                  <IPAddress>
                    <IPV4Address>10.0.101.1</IPV4Address>
                  </IPAddress>
                </Naming>
                .
                .
                .
                  data returned for 1st neighbor
                .
                .
                .
              </Neighbor>
              <Neighbor>
                <Naming>
                  <IPAddress>
                    <IPV4Address>10.0.101.6</IPV4Address>
                  </IPAddress>
                </Naming>
                .
                .
                .
                  data returned for 2nd neighbor
                .
                .
                .
              </Neighbor>
              <Neighbor>
                <Naming>
                  <IPAddress>
```

```
                    <IPV4Address>10.0.101.6</IPV4Address>
                  </IPAddress>
                </Naming>
                .
                .
                .
                  data returned for 3rd neighbor
                .
                .
                .
              </Neighbor>
            </NeighborTable>
          </BGPEntity>
        </AS>
      </BGP>
    </Configuration>
  </Get>
</Response>
```

XML Request Using Operation Scope (Content Attribute)

The Content attribute is used on any table element in order to specify the scope of a <Get> operation. Table 5-1 describes the content attribute values are supported.

Table 5-1   Content Attributes

Content Attribute	Description
All	Use to get all leaf items and their values. All is the default when the Content attribute is not specified on a table element.
Entries	Use to get the Naming information for each entry within a specified table object class. Entries provides a one-level get capability.

If the Content attribute is specified on a nontable element, it is ignored. Note also that the Content and Count attributes can be used together on the same table element.

The following example displays the Content attribute that is used to list all configured BGP neighbors:

Sample XML Client Request Using the All Content Attribute

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration>
      <BGP MajorVersion="1" MinorVersion="0">
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <BGPEntity>
            <NeighborTable Content=”Entries”/>
          </BGPEntity>
        </AS>
      </BGP>
    </Configuration>
  </Get>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration>
      <BGP MajorVersion="1" MinorVersion="0">
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <BGPEntity>
            <NeighborTable Content=”Entries”>
              <Neighbor>
                <Naming>
                  <IPAddress>
                    <IPV4Address>10.0.101.1</IPV4Address>
                  </IPAddress>
                </Naming>
              </Neighbor>
              <Neighbor>
                <Naming>
                  <IPAddress>
                    <IPV4Address>10.0.101.2</IPV4Address>
                  </IPAddress>
                </Naming>
              </Neighbor>
              <Neighbor>
                <Naming>
                  <IPAddress>
                    <IPV4Address>10.0.101.3</IPV4Address>
                  </IPAddress>
                </Naming>
              </Neighbor>
              <Neighbor>
                <Naming>
                  <IPAddress>
                    <IPV4Address>10.0.101.4</IPV4Address>
                  </IPAddress>
                </Naming>
              </Neighbor>
                .
                .
                .
                  more neighbors returned here
                .
                .
                .
            </NeighborTable>
          </BGPEntity>
        </AS>
      </BGP>
    </Configuration>
  </Get>
</Request>
```

Limiting the Number of Table Entries Returned (Count Attribute)

The Count attribute is used on any table element within a <Get> operation to specify the maximum number of table entries to be returned in a response. When the Count attribute is specified, the naming information within the request is used to identify the starting point within the table, that is, the first table entry of interest. If no naming information is specified, the response starts at the beginning of the table.

For a table whose entries are containers, the Count attribute can be used only if the Content attribute is also specified with a value of Entries. This restriction does not apply to a table whose children are leaf nodes.

As an alternative to the use of the Count attribute, the Cisco CRS-1 Series XML interface supports the retrieval of large XML responses in blocks through iterators. For more information on iterators, see Chapter 7, "Cisco CRS-1 Series XML and Large Data Retrieval (Iterators)."

The following example shows how to use the Count attribute to retrieve the configuration information for the first five BGP neighbors starting with the address 10.0.101.1:

Sample XML Client Request Using the Count Attribute

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration>
      <BGP MajorVersion="1" MinorVersion="0">
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <BGPEntity>
            <NeighborTable Count=”5”>
              <Neighbor>
                <Naming>
                  <IPAddress>
                    <IPV4Address>10.0.101.1</IPV4Address>
                  </IPAddress>
                </Naming>
              </Neighbor>
            </NeighborTable>
          </BGPEntity>
        </AS>
      </BGP>
    </Configuration>
  </Get>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration>
      <BGP MajorVersion="1" MinorVersion="0">
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <BGPEntity>
            <NeighborTable Count=”5”>
              <Neighbor>
                <Naming>
                  <IPAddress>
```

```
                    <IPV4Address>10.0.101.1</IPV4Address>
                  </IPAddress>
                </Naming>
                  .
                  .
                  .
                   data for 1st neighbor returned
                   here
                  .
                  .
                  .
              </Neighbor>
              <Neighbor>
                <Naming>
                  <IPAddress>
                    <IPV4Address>10.0.101.2</IPV4Address>
                  </IPAddress>
                </Naming>
                  .
                  .
                  .
                   data returned for 2nd neighbor
                   here
                  .
                  .
                  .
              </Neighbor>
                  .
                  .
                  .
                 data returned for remaining
                 neighbors here
                  .
                  .
                  .
            </NeighborTable>
          </BGPEntity>
        </AS>
      </BGP>
    </Configuration>
  </Get>
</Response>
```

Custom Filtering (Filter Element)

Some of the tables from the operational namespace support the selection of rows of interest based on predefined filtering criteria. Filters can be applied to such tables in order to reduce the number of table entries retrieved in a request.

Client applications specify filtering criteria for such tables by using the <Filter> tag and including the filter specific parameters as defined in the XML schema definition for that table. If no table entries match the specified filter criteria, the response contains the object class hierarchy down to the specified table, but does not include any table entries. The Content attribute can be used with a filter to specify the scope of a <Get> request.

In the following example, the filter <BGP_ASFilter> is used to retrieve operational information for all neighbors in autonomous system 6:

Sample XML Client Request Using Filtering

```
<?xml version=”1.0” encoding=”UTF-8”?>
<Request MajorVersion="1" MinorVersion="0">
  <Get>
    <Operational>
      <BGP MajorVersion=”1” MinorVersion=”0”>
        <NeighborTable>
          <Filter>
            <BGP_ASFilter>
              <AS>6</AS>
            </BGP_ASFilter>
          <Filter>
        </NeighborTable>
      </BGP>
    </Operational>
  </Get>
</Request>
```

Sample Filtered XML Response from the Cisco CRS-1 Series Router

```
<?xml version=”1.0” encoding=”UTF-8”?>
<Response MajorVersion="1" MinorVersion="0">
  <Get>
    <Operational>
      <BGP MajorVersion=”1” MinorVersion=”0”>
        <NeighborTable>
          <Filter>
            <BGP_ASFilter>
              <AS>6</AS>
            </BGP_ASFilter>
          </Filter>
          <Neighbor>
            .
            .
            .
              data for 1st neighbor returned here
            .
            .
            .
          </Neighbor>
          <Neighbor>
            .
            .
            .
              data for 2nd neighbor returned here
              returned here
            .
            .
            .
          </Neighbor>
            .
            .
            .
            data for remaining neighbors returned
            here
            .
            .
            .
        </NeighborTable>
      </BGP>
    </Operational>
  </Get>
</Response>
```

CHAPTER 6

Cisco CRS-1 Series XML and Encapsulated CLI Operations

The extensible markup language (XML) interface for the Cisco CRS-1 Series Carrier Routing System (Cisco CRS-1 Series) provides support for XML encapsulated command-line interface (CLI) commands and responses.

This chapter provides information on XML CLI command tags.

Note The XML application programming interface (API) code is available for use on any Cisco platform that runs Cisco IOS XR software.

XML CLI Command Tags

A client application can request a CLI command by encoding the text for the command within a pair of <CLI> start and </CLI> end tags, and <Configuration> tags. The router responds with the uninterpreted CLI text result.

Note XML encapsulated CLI commands use the same target configuration as the corresponding XML operations <Get>, <Set>, and <Delete>.

When used for CLI operations, the <Configuration> tag supports the optional Operation attribute, which can take one of the values listed in Table 6-1.

Table 6-1 Operational Attribute Values

Operational Attribute Value	Operational Attribute Value Description
Apply	Specifies that the commands should be executed or applied (default).

Table 6-1 Operational Attribute Values (continued)

Help	Gets help on the last command in the list of commands sent in the request. There should not be any empty lines after the last command (because the last command is considered to be the one on the last line)
CommandCompletion	Completes the last keyword of the last command. Apart from not allowing empty lines at the end of the list of commands sent in the request, when this option is used there should not be any white spaces after the partial keyword to be completed.

The following example uses the <CLI> operation tag:

Sample XML Client Request for CLI Command Using CLI Tags

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <CLI>
   <Configuration>
     router bgp 3
      default-metric 10
      timers bgp 80 160
     exit
    show config
   </Configuration>
  </CLI>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <CLI>
    <Configuration>
     Building configuration...
     router bgp 3
      default-metric 10
      timers bgp 80 160
     end
    </Configuration>
  </CLI>
</Response>
```

CLI Command Limitations

The initial CLI command support through XML is limited to CLI configuration and subsequent responses wrapped in <CLI> tags.

The following conditions are not supported:

- <Operational> namespace commands.
- <Action> namespace commands.
- Sending a request in <CLI> format and getting back an XML encoded response.
- Sending an XML encoded request and getting back a response in <CLI> format.
- “Long running” commands, for example, “ping” and “top.”

- Iterators in responses to CLI commands such as the **show run** and **show configuration** commands.
- Multiple simultaneous CLI requests; that is, only one XML request can be issued at a time across all client sessions on the router.

Cisco CRS-1 Series XML and Large Data Retrieval (Iterators)

The extensible markup language (XML) for the Cisco CRS-1 Series router supports the retrieval of large XML responses in blocks (that is, in chunks or sections).

Note The XML application programming interface (API) code is available for use on any Cisco platform that runs Cisco IOS XR software.

This chapter provides information on large data retrieval. See the section“Terminating an Iterator” for information on terminating an iterator.

When a client application makes a request, the resulting response data size is checked to determine if it is larger than a predetermined block size. If it is not larger, then the complete data is returned in a normal response. However, if the response data is larger than the block size, then the first set of data is returned according to the block size along with an iterator ID included as the value of the IteratorID attribute. The client must then send <GetNext> requests including the iterator ID until all the data is retrieved. The client application knows that all of the data is retrieved when it receives a response that does not contain an IteratorID attribute.

The following points should be noted by the client application when iterators are used:

- The block size is a fixed value specific to each transport mechanisms on the router, that is, the XML agent for Common Object Request Broker Architecture (CORBA) and Secure Shell (SSH) or Telnet. No mechanism is provided for the client application to specify a desired block size.
- The block size refers to the entire XML response, not just the payload portion of the response.
- Large responses are divided based on the requested block size, not on the contents. However, each response is always a complete XML document.
- Requests containing multiple operations are treated as a single entity when the block size and IteratorID are applied. As a result, the IteratorID is an attribute of the <Response> tag, never of an individual operation.
- If the client application sends a request that includes an operation resulting in the need for an iterator to return all of the response data, any further operations contained within that request are rejected. The rejected operations are resent in another request.
- The IteratorID is an unsigned 32-bit value that should be treated as opaque data by the client application. Furthermore, the client application should not assume that the IteratorID is constant between <GetNext> operations.

To reduce memory overhead and avoid memory starvation of the router, the following limitations are placed on the number of allowed iterators:

- 10—Maximum number of iterators allowed at any one time on a given client session.
- 100—Maximum number of iterators allowed at any one time for all client sessions.

Note If a <Get> request is issued that results in an iterated response, it is counted as 1 iterator, regardless of the number of <GetNext> operations required to retrieve all of the response data. For example, a <Get> request may require 10, 100, or more <GetNext> operations to retrieve all of the associated data, but during this process only 1 iterator is being used. Also, an iterator is considered to be in use until all of the response data associated with that iterator (that is, all of the response data associated with the original <Get> request) is retrieved or the iterator is terminated with the Abort attribute.

The following example shows a client request that utilizes an iterator to retrieve all global Border Gateway Protocol (BGP) configuration data for a specified autonomous system:

Sample XML Client Request to Retrieve All BGP Configuration Data

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration>
      <BGP MajorVersion=”1” MinorVersion=”0”>
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <Global/>
        </AS>
      </BGP>
     </Configuration>
  </Get>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router Containing the First Block of Retrieved Data

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0" IteratorID=”1”>
  <Get>
    <Configuration>
      <BGP MajorVersion=”1” MinorVersion=”0”>
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <Global>
            .
            .
            .
              1st block of data returned here
            .
            .
            .
          </Global>
        <AS>
      </BGP>
    </Configuration>
  <Get>
</Response>
```

Second XML Client Request Using the <GetNext> Iterator to Retrieve the Next Block of BGP Configuration Data

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

Sample XML Response from the Cisco CRS-1 Series Router Containing the Second Block of Retrieved Data

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0" IteratorID=”1”>
  <Get>
    <Configuration>
      <BGP MajorVersion=”1” MinorVersion=”0”>
       <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <Global>
            .
            .
            .
              2nd block of data returned here
            .
            .
            .
          </Global>
        </AS>
      </BGP>
    </Configuration>
  <Get>
</Response>
```

Third XML Client Request Using the <GetNext> Iterator to Retrieve the Next Block of BGP Configuration Data

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

Sample XML Response from the Cisco CRS-1 Series Router Containing Third Block of Retrieved Data

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0" IteratorID=”1”>
  <Get>
    <Configuration>
      <BGP MajorVersion=”1” MinorVersion=”0”>
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <Global>
            .
            .
            .
              3rd block of data returned here
            .
            .
            .
          </Global>
        </AS>
      </BGP>
    </Configuration>
  </Get>
</Response>
```

Final XML Client Request Using the <GetNext> Iterator to Retrieve the Last Block of BGP Configuration Data

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

Final XML Response from the Cisco CRS-1 Series Router Containing the Final Block of Retrieved Data

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration>
      <BGP MajorVersion=”1” MinorVersion=”0”>
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <Global>
            .
            .
            .
              Final block of data returned here
            .
            .
            .
          </Global>
        </AS>
      </BGP>
    </Configuration>
  </Get>
</Response>
```

Terminating an Iterator

A client application may terminate an iterator without retrieving all of the response data by including an Abort attribute with a value of “true” on the <GetNext> operation. A client application that does not complete or terminate its requests risks running out of iterators.

The following example shows a client request using the Abort attribute to terminate an iterator:

Sample XML Request

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration>
      <BGP MajorVersion=”1” MinorVersion=”0”>
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <Global/>
        </AS>
      </BGP>
    </Configuration>
  </Get>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0" IteratorID=”2”>
  <Get>
    <Configuration>
      <BGP MajorVersion=”1” MinorVersion=”0”>
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <Global>
            .
            .
            .
              1st block of data returned here
            .
            .
            .
          </Global>
        <AS>
      </BGP>
    </Configuration>
  <Get>
</Response>
```

Sample XML Request Using the Abort Attribute to Terminate an Iterator

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <GetNext IteratorID=”2” Abort=”true”/>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <GetNext IteratorID=”2” Abort=”true”/>
</Response>
```

Cisco CRS-1 Series XML Security

Specific security privileges are required for a client application requesting information from the Cisco CRS-1 Series Carrier Routing System (Cisco CRS-1 Series) router.

Note The extensible markup language (XML) application programming interface (API) code is available for use on any Cisco platform that runs Cisco IOS XR software.

This chapter contains the following sections:

Authentication

User authentication through authentication, authorization, and accounting (AAA) is handled on the router by the transport-specific XML agent and is not exposed through the XML interface.

Authorization

Every operation request by a client application is authorized. If the client is not authorized to perform an operation, the operation is not performed by the Cisco CRS-1 Series router and an error is returned.

Authorization of client requests is handled through the standard AAA "task permissions" mechanism. The XML agent caches the AAA user credentials obtained from the user authentication process, and then each client provides these to the XML infrastructure on the Cisco CRS-1 Series router. As a result, no AAA information needs to be passed in the XML request from the client application.

Each object class in the schema has a task ID associated with it. A client application's capabilities and privileges in terms of task IDs are exposed by AAA through a **show** command. A client application can use the XML interface to retrieve the capabilities prior to sending configuration requests to the router.

A client application requesting an operation through the XML interface must have the appropriate task privileges enabled/assigned for any objects accessed in the operation:

- <Get> operations require AAA “read” privileges.
- <Set> and <Delete> operations require AAA “write” privileges.

The “configuration services” operations through configuration manager can also require the appropriate predefined task privileges.

If an operation requested by a client application fails authorization, an appropriate <Error> element is returned in the response sent to the client. For “native data” operations, the <Error> element is associated with the specific element or object classes where the authorization error occurred.

Retrieving Task Permissions

A client application’s capabilities and privileges in terms of task permissions are exposed by AAA through command-line interface (CLI) **show** commands. A client application can also use the XML interface to programatically retrieve the current AAA capabilities from the router. This retrieval can be done by issuing the appropriate <Get> request to the <AAA> component.

The following example shows a request to retrieve all of the AAA configuration from the router:

Sample XLM Request to Retrieve AAA Configuration Information

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

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration>
      <AAA MajorVersion=”1” MinorVersion=”0”>
        .
        .
        .
         AAA configuration returned here
        .
        .
        .
      </AAA>
    </Configuration>
  </Get>
</Response>
```

Task Privileges

A client application requesting a native data operation through the XML interface must have the appropriate task privileges enabled/assigned for any items accessed in the operation, as follows:

- <Get>, <GetNext>, and <GetVersionInfo> operations require AAA “read” privileges.
- <Set> and <Delete> operations require AAA “write” privileges.

The “configuration services” operations through configuration manager can also require the appropriate predefined task privileges.

Task Names

Each object (that is, data item or table) exposed through the Cisco CRS-1 Series XML interface and accessible to the client application has one or more task names associated with it. The task names are published in the XML schema documents as <appinfo> annotations.

For example, the complex type definition for the top-level element in the Border Gateway Protocol (BGP) configuration schema contains the following annotation:

```
<xsd:appinfo>
  <MajorVersion>1</MajorVersion>
  <MinorVersion>0</MinorVersion>
  <TaskIdInfo TaskGrouping="Single">
    <TaskName>bgp</TaskName>
  </TaskIdInfo>
</xsd:appinfo>
```

Here is another example from a different component schema. This annotation includes a list of task names.

```
<xsd:appinfo>
  <MajorVersion>1</MajorVersion>
  <MinorVersion>0</MinorVersion>
  <TaskIdInfo TaskGrouping="And">
    <TaskName>ouni</TaskName>
    <TaskName>mpls-te</TaskName>
  </TaskIdInfo>
</xsd:appinfo>
```

The task names indicate what permissions are required to access the data below the object. In this example, the task names “ouni” and “mpls-te” have been specified for the object. These task names apply to this object and are inherited by all of the object’s descendents in the schema, unless a descendant has a task names of its own, in which case the descendant (and all of its descendants) assumes the more specific task name (that is, overriding the task name of the ancestor). Essentially, the rule for a particular object is that it assumes the task name of the closest ancestor for which there is a task name specified in the schema.

The TaskGrouping attribute is used to specify the logical relationship between the task names when multiple task names are specified for an object. For example, for a client application to issue a <Get> request for the object containing the annotation shown in the example, the corresponding AAA user credentials must have “read” permissions set for both the “ouni” *and* “mpls-te” tasks. The possible values for the TaskGrouping attribute are And, Or, and Single. Single is used when there is only a single task name specified for the object.

Authorization Failure

If an operation requested by a client application fails authorization, an appropriate <Error> element is returned in the response sent to the client. For “native data” operations, the <Error> element is associated with the specific element or object where the authorization error occurred.

If a client application issues a <Get> request to retrieve all data below a container object, and if any subsections of that data require permissions that the user does not have, then an error is not returned. Instead, the subsection of data is not included in the <Get> response.

Cisco CRS-1 Series XML Schema Versioning

Before the Cisco CRS-1 Series Carrier Routing System (Cisco CRS-1 Series) can carry out a client application request, it must verify version compatibility between the client request and router component versions.

A major and minor version number are carried on the <Request> and <Response> elements to indicate the overall Cisco CRS-1 Series extensible markup language (XML) application programming interface (API) version in use by the client application and router. In addition, each component XML schema exposed through the Cisco CRS-1 Series XML API has a major and minor version number associated with it.

Note The XML API code is available for use on any Cisco platform that runs Cisco IOS XR software.

This chapter describes the format of the version information exchanged between the client application and the Cisco CRS-1 Series router, and how the Cisco CRS-1 Series router uses this information at run time to check version compatibility.

The following sections are included:

- Major and Minor Version Numbers, page 9-95
- Run-Time Use of Version Information, page 9-96
- Retrieving Version Information, page 9-99

Major and Minor Version Numbers

The top-level or root object (that is, element) in each component XML schema carries the major and minor version numbers for that schema. A minor version change is defined as an addition to the XML schema. All other changes, including deletions and semantic changes, are considered major version changes.

The version numbers are documented in the header comment contained in the XML schema file. They are also available as <xsd:appinfo> annotations included as part of the complex type definition for the top-level schema element. This enables you to programmatically extract the version numbers from the XML schema file to include in XML request instances sent to the router. The version numbers are carried in the XML instances using the MajorVersion and MinorVersion attributes.

The following example shows the relevant portion of the complex type definition for an element that carries version information:

```
<xsd:complexType name="ipv4_bgp_cfg_BGP_type">
  <xsd:annotation>
    <xsd:documentation>Global BGP config</xsd:documentation>
    <xsd:appinfo>
      <MajorVersion>1</MajorVersion>
      <MinorVersion>0</MinorVersion>
      <TaskIdInfo TaskGrouping="Single">
        <TaskName>bgp</TaskName>
      </TaskIdInfo>
    </xsd:appinfo>
  </xsd:annotation>
            .
            .
            .
  <xsd:attributeGroup ref="VersionAttributeGroup"/>
            .
            .
            ..
</xsd:complexType>
```

The attribute group VersionAttributeGroup is defined as follows:

```
<xsd:attributeGroup name="VersionAttributeGroup">
  <xsd:annotation>
    <xsd:documentation>
      Common version information attributes
    </xsd:documentation>
  </xsd:annotation>
  <xsd:attribute name="MajorVersion" type="xsd:unsignedInt" use="required"/>
  <xsd:attribute name="MinorVersion" type="xsd:unsignedInt" use="required"/>
</xsd:attributeGroup>
```

Run-Time Use of Version Information

Each XML request must contain the client’s major and minor version numbers at the appropriate locations in the XML. These version numbers are compared to the version numbers running on the router. The request is then accepted or rejected based on the following rules:

- If there is a major version discrepancy, then the request fails.
- If there is a minor version lag, that is, the client minor version is behind that of the router, then the request is attempted.
- If there is a minor version creep, that is, the client minor version is ahead of that of the router, then the request fails.
- If the version information has not been included in the request, then the request fails.

Each XML response can also contain the version numbers at the appropriate locations in the XML.

Note If the client minor version is behind that of the router, then the response may contain elements that are not recognized by the client application. The client application must be able to handle these additional elements.

Placement of Version Information

The following example shows the placement of the MajorVersion and MinorVersion attributes within a client request to retrieve the global BGP configuration data for a specified autonomous system:

Sample Client Request Showing Placement of Version Information

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration>
      <BGP MajorVersion=”1” MinorVersion=”0”>
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <Global/>
        </AS>
      </BGP>
    </Configuration>
  </Get>
</Request>
```

Sample Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration>
      <BGP MajorVersion=”1” MinorVersion=”0”>
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <Global>
            .
            .
            .
              data returned here
            .
            .
            .
          </Global>
        <AS>
      </BGP>
    </Configuration>
  <Get>
</Response>
```

Version Lag

The following example shows a request and response with a version mismatch. In this case, the client minor version is behind that of the router, so the request is attempted.

Note The version number, which is returned in the response, is the version number running on the router.

Sample XML Client Request with a Version Mismatch

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Get>
    <Configuration>
      <BGP MajorVersion=”1” MinorVersion=”0”>
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <Global/>
        </AS>
      </BGP>
    </Configuration>
  </Get>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0" IteratorID=”1”>
  <Get>
    <Configuration>
      <BGP MajorVersion=”1” MinorVersion=”1”>
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <Global>
            .
            .
            .
              data returned here
            .
            .
            .
          </Global>
        <AS>
      </BGP>
    </Configuration>
  <Get>
</Response>
```

Version Creep

The following example shows a request and response with a version mismatch. In this case, the client minor version is ahead of that of the router minor version, resulting in an error response.

Sample XML Request with a Minor Version Mismatch Ahead of the Cisco CRS-1 Series Router

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

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0" IteratorID=”12345678”>
  <Get xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ErrorCode=”0x43679000"
       ErrorMsg="&apos;XML Service Library&apos; detected the &apos;warning&apos;
       condition &apos;An error was encountered in the XML beneath this operation
       tag&apos;" >
    <Configuration>
      <BGP MajorVersion=”1” MinorVersion=”0” ErrorCode="0x4368ac00"
           ErrorMsg="&apos; XMLMDA&apos; detected the &apos;warning&apos; condition
           &apos; The XML version specified in the XML request is not compatible
           with the version running on the router&apos;"/>
    </Configuration>
  <Get>
</Response>
```

Retrieving Version Information

The version of the XML schemas running on the Cisco CRS-1 Series router can be retrieved using the <GetVersionInfo> tag followed by the appropriate tags identifying the names of the desired components.

In the following example, the <GetVersionInfo> tag is used to retrieve the major and minor version numbers for the BGP component configuration schema:

Sample XML Request to Retrieve Major and Minor Version Numbers

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

Sample XML Response from the Cisco CRS-1 Series Router

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

The following example shows how to retrieve the version information for all configuration schemas available on the router:

Sample XML Request to Retrieve Version Information for All Configuration Schemas

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

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <GetVersionInfo>
    <Configuration>
            .
            .
            .
      <MPLS_LSD MajorVersion="1" MinorVersion="0"/>
      <MPLS_TE MajorVersion="1" MinorVersion="0"/>
      <OUNI MajorVersion="1" MinorVersion="0"/>
      <OLM MajorVersion="1" MinorVersion="0"/>
      <BGP MajorVersion=”1” MinorVersion=”0”/>
      <CDP MajorVersion=”1” MinorVersion=”1”/>
      <RSVP MajorVersion=”1” MinorVersion=”0”/>
            .
            .
            .
      <InterfaceConfiguration>
        <CDP MajorVersion="1" MinorVersion="0"/>
        <SONET MajorVersion="1" MinorVersion="2"/>
        <PPP MajorVersion="1" MinorVersion="0">
          <IPCP MajorVersion="1" MinorVersion="0"/>
        </PPP>
            .
            .
            .
      </InterfaceConfiguration>
            .
            .
            .
    </Configuration>
  </GetVersionInfo>
</Response>
```

CHAPTER 10

Error Reporting in Cisco CRS-1 Series XML Responses

The extensible markup language (XML) responses returned by the Cisco CRS-1 Series Carrier Routing System (Cisco CRS-1 Series) router contains error information as appropriate, including the operation, object, and cause of the error when possible. The error codes and messages returned from the Cisco CRS-1 Series router may originate in the XML agent or in one of the other Cisco CRS-1 Series infrastructure layers; for example, the XML Service Library, XML Parser Library, or Configuration Manager.

Note The XML application programming interface (API) code is available for use on any Cisco platform that runs Cisco IOS XR software.

Types of Reported Errors

The types of potential errors in Cisco CRS-1 Series XML Responses are categorized in Table 10-1.

Table 10-1 Reported Error Types

Error Type	Description
Transport errors	Transport-specific errors are detected within the XML agent (and include failed authentication attempts).
XML parse errors	XML format or syntax errors are detected by the XML Parser Library (and include errors resulting from malformed XML, mismatched XML tags, and so on).
XML schema errors	XML schema errors are detected by the XML operation provider within the Cisco CRS-1 Series infrastructure (and include errors resulting from invalid operation types, invalid object hierarchies, values out of range, and so on).
Operation processing errors	Operation processing errors are errors encountered during the processing of an operation, typically as a result of committing the target configuration (and include errors returned from Configuration Manager and the Cisco CRS-1 Series infrastructure such as failed authorization attempts, and “invalid configuration errors” returned from the back-end Cisco CRS-1 Series operating system applications).

These error categories are described in the following sections:

- Error Attributes, page 10-102
- Transport Errors, page 10-102
- XML Parse Errors, page 10-102
- XML Schema Errors, page 10-103
- Operation Processing Errors, page 10-105
- Error Codes and Messages, page 10-105

Error Attributes

If one or more errors occur during the processing of a requested operation, the corresponding XML response includes error information for each element or object class in error. The error information is included in the form of ErrorCode and ErrorMsg attributes providing a relevant error code and error message respectively.

If one or more errors occur during the processing of an operation, error information is included for each error at the appropriate point in the response. In addition, error attributes are added at the operation element level. As a result, the client application does not have to search through the entire response to determine if an error has occurred. However, the client can still search through the response to identify each of the specific error conditions.

Transport Errors

Transport-specific errors, including failed authentication attempts, are handled by the appropriate XML agent.

XML Parse Errors

This general category of errors includes those resulting from malformed XML and mismatched XML tags.

The Cisco CRS-1 Series router checks each XML request, but does not validate the request against an XML schema. If the XML contains invalid syntax and thus fails the well-formedness check, the error indication is returned in the form of error attributes placed at the appropriate point in the response. In such cases, the response may not contain the same XML as was received in the request, but just the portions to the point where the syntax error was encountered.

In the following example, the client application sends a request to the Cisco CRS-1 Series router that contains mismatched tags, that is, the opening <BGPEntity> tag is not paired with a closing </BGPEntity> tag. This example illustrates the format and placement of the error attributes.

Note The actual error codes and messages might be different than what is shown in this example. Also, the actual error attributes does not contain new line characters.

Sample XML Client Request Containing Mismatched Tags

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

```
  <Get>
    <Configuration>
      <BGP MajorVersion=”1” MinorVersion=”0”>
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <BGPEntity>
        </AS>
      </BGP>
    </Configuration>
  </Set>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0">
  <Get xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ErrorCode="0x43679000"
      ErrorMsg="&apos;XML Service Library&apos; detected the &apos;warning&apos;
      condition &apos;An error was encountered in the XML beneath this operation tag
      &apos;">
    <Configuration ErrorCode="0xa240da00" ErrorMsg="&apos;XML Infrastructure &apos;
        detected the &apos;fatal&apos; condition &apos;Opening and ending tag does not
        match&apos;"/>
  </Get>
</Response>
```

XML Schema Errors

XML schema errors are detected by the XML operation providers. This general category of errors includes those resulting from invalid operation types, invalid object hierarchies, and invalid naming or value elements. However, some schema errors may go undetected because, as previously noted, the Cisco CRS-1 Series router does not validate the request against an XML schema.

In the following example, the client application has requested a <Set> operation specifying an object <ExternalRoutes> that does not exist at this location in the Border Gateway Protocol (BGP) component hierarchy. This example illustrates the format and placement of the error attributes.

Note The actual error codes and messages may be different than those shown in the example.

Sample XML Client Request Specifying an Invalid Object Hierarchy

```
<?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>
            <ExternalRoutes>10</ExternalRoutes>
          </Global>
        </AS>
      </BGP>
    </Configuration>
  </Set>
```

```
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Set xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ErrorCode="0x4368a400"
       ErrorMsg="&apos;XML Service Library&apos; detected the &apos;warning&apos;
       condition &apos;An error was encountered in the XML beneath this operation
       tag&apos;">
    <Configuration>
      <BGP MajorVersion=”1” MinorVersion=”0”
        <AS>
          <Naming>
            <AS>3</AS>
          </Naming>
          <Global ErrorCode="0x4368a400" ErrorMsg="&apos;XMLMDA&apos; detected the
              &apos;warning&apos; condition &apos;
              The XML request does not conform to the schema. A child element of
              the element on which this error appears is invalid. No such child
              element name exists at this location in the schema. Please check
              the request against the schema.&apos;"/>
        </AS>
      </BGP>
    </Configuration>
  </Set>
</Request>
```

The following example also illustrates a schema error. In this case, the client application has requested a <Set> operation specifying a value for the <GracefulRestartTime> object that is not within the range of valid values for this item.

Sample XML Request Specifying an Invalid Object Value Range

```
<?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>
            <GracefulRestartTime>6000</GracefulRestartTime>
          </Global>
        </AS>
      </BGP>
    </Configuration>
  </Set>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <Set xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ErrorCode="0x4368a800"
       ErrorMsg="&apos;XML Service Library&apos; detected the &apos;warning&apos;
       condition &apos;An error was encountered in the XML beneath this operation
       tag&apos;">
    <Configuration>
      <BGP MajorVersion=”1” MinorVersion=”0”>
        <AS>
```

```
          <Naming>
            <AS>3</AS>
          </Naming>
          <Global>
            <GracefulRestartTime ErrorCode="0x4368a800" ErrorMsg=&apos;
                XMLMDA&apos; detected the &apos;warning&apos; condition &apos;
                The XML request does not conform to the schema. The character data
                contained in the element on which this error appears (or one of its
                child elements) does not conform to the XML schema for its datatype.
                Please check the request against the schema.&apos;"/>
          </Global>
        </AS>
      </BGP>
    </Configuration>
  </Set>
</Request>
```

Operation Processing Errors

Operation processing errors include errors encountered during the processing of an operation, typically as a result of committing the target configuration after previous <Set> or <Delete> operations. While processing an operation, errors are returned from Configuration Manager and the Cisco CRS-1 Series infrastructure, failed authorization attempts, and “invalid configuration errors” returned from the back-end Cisco CRS-1 Series operating system applications.

The following example illustrates an operation processing error resulting from a <GetNext> request specifying an unrecognized iterator ID:

Sample XML Client Request and Processing Error

```
<?xml version="1.0" encoding="UTF-8"?>
<Request MajorVersion="1" MinorVersion="0">
  <GetNext IteratorID="1" Abort="true"/>
</Request>
```

Sample XML Response from the Cisco CRS-1 Series Router

```
<?xml version="1.0" encoding="UTF-8"?>
<Response MajorVersion="1" MinorVersion="0" ErrorCode="0xa367a800" ErrorMsg="&apos;
     XML Service Library&apos; detected the &apos;fatal&apos; condition &apos;The XML
     Infrastructure has been provided with an iterator ID which is not recognized. The
     iterator is either invalid or has timed out.&apos;"/>
```

For more information on errors resulting from a commit of the target configuration, see the “Commit Errors” section on page 2-33.

Error Codes and Messages

The error codes and messages returned from the Cisco CRS-1 Series router may originate in any one of several components.

The error codes (cerrnos) returned from these layers are 32-bit integer values. In general, for a given error condition, the error message returned in the XML is the same as the error message displayed on the command line interface (CLI).

CHAPTER 11

XML Transport and Event Notifications

This chapter discusses the Common Object Request Broker Architecture (CORBA) as the extensible markup language (XML) transport mechanism for the Cisco CRS-1 Series Carrier Routing System (Cisco CRS-1 Series), and its Internet Inter-ORB Protocol (IIOP), the protocol used for accessing objects across the Internet. CORBA-based event notifications are also discussed in this chapter.

The following Object Request Brokers are supported by Cisco IOS XR for the Cisco CRS-1 Series system as clients:

- Java ORB
- IONA Orbix (running on Solaris 2.8)

Note CORBA/IIOP is the transport mechanism provided with the initial release of the Cisco CRS-1 Series XML API. Additional transport mechanisms including Telnet and Secure Shell (SSH) are provided in future releases of the Cisco CRS-1 Series XML application programming interface (API).

This chapter contains the following sections:

Cisco CRS-1 Series XML Transport and CORBA IDL

CORBA/IIOP is the Cisco CRS-1 Series XML transport mechanism. External client applications use CORBA Interface Definition Language (IDL) to exchange XML encoded request and response streams with the CORBA XML agent running on the router.

Communication between the client application and the server (that is, through the CORBA XML agent) is through IIOP. The client application must first bind to the CORBA XML agent object by obtaining the appropriate Interoperable object reference (IOR) from the CORBA Naming Service. The client then obtains a login authentication using the “login()” method. After the client has obtained a login authentication, the client can use the “invoke()” method to send XML requests to the CORBA XML agent.

When the CORBA XML agent receives the XML request, it uses the XML infrastructure on the router to parse and process the request. The agent then obtains the XML response from the XML infrastructure and passes this back to the client as a response parameter on the “invoke()” method return.

IDL Interface

The following IDL example defines the interfaces used to make Cisco CRS-1 Series XML API requests. The IDL interface was designed to be simple for easy migration to other transports. Request details, including operation name, request parameters, and versioning information, are not exposed through the IDL, but are instead encoded in the request itself. To make a Cisco CRS-1 Series XML API request, the client application calls the “invoke()” method sending the “stringified” XML request as the request parameter. The stringified XML response along with error information is returned in the response parameter to the client application.

```
module Manageability
{
    interface XMLAgent
    {
/*
 * edt: * * XMLAgent::login
 *
 * Provides the definition for the login() method for the XMLAgent idl
 *
 * Return:CORBA::Boolean
 *
 *  TRUE  - The method succeeded
 *  FALSE - The method failed
 *
 * Argument: username
 *
 * IN - valid login username on the router.
 *
 * Argument: password
 *
 * IN - valid unencrypted password for the given username on the router.
 *
 * Argument: session_context
 *
 * OUT - internally generated context for a given valid login session.
 * Clients will use this in the invoke call.
 *
 * Argument: response
 *
 * OUT - contains error code and error message string in case login failed.
 * Clients will use this to know the reason of failure.
 *
 */
        boolean login (in string username,
                       in string password,
                       out long session_context,
                       out string response);

/*
 * edt: * * XMLAgent::invoke
 *
 * Provides the definition for the invoke() method for the XMLAgent idl
 *
 * Return: None
 *
 * Argument: session_context
 *
 * IN - internally generated context for a given valid login session.
 * Clients pass this as it is obtained in login method.
 *
 * Argument: request
 *
 * IN - XML request string. Passed to XML Infra for processing
 *
 * Argument: response
 *
 * OUT - contain response obtained from XML Infra or error code
 * and error message string in case of response failure from XML Infra.
 *
 */
        void invoke (in long session_context,
                     in string request,
                     out string response);

/*
 * edt: * * XMLAgent::logout
 *
 * Provides the definition for the logout() method for the XMLAgent idl
 *
 * Return: None
 *
 * Argument: session_context
 *
 * IN - internally generated context for a given valid login session.
 * Clients will pass as it is obtained in login method.
 *
 * Argument: response
 *
 * OUT - contain error code and error message string in case of failure.
 *
 */
        void logout (in long session_context,
                     out string response);

     };
};
```

Authentication

Client applications must authenticate with the CORBA XML agent before sending any requests. The authentication is done through the “login()” method of the CORBA XML agent IDL. The username and password supplied are used to contact and call authentication, authorization, and accounting (AAA) server APIs for authentication on the router. Only after successful authentication is the client application able to send XML encode requests to the router using the “invoke()” method.

Table 11-1 describes the “login()” implementation in the CORBA XML agent that uses AAA options as part of the Authentication API.

Table 11-1 Authentication, Authorization, and Accounting Options

Name	Description
default	Determines the authentication method.
ASCII authentication	Determines the username and password. As a result, the username and password supplied in the “login()” method call should be unencrypted ASCII text. Any security for transport should use SSL[1].

[1] SSL = secure socket layer

CORBA NameServer

The NameServer is used by client applications to obtain the interface object request (IOR) of the CORBA XML agent object. In order to make the IOR available to clients, the CORBA XML agent exports it to the NameServer upon startup.

The CORBA Naming Service stores a name with each object reference and provides a hierarchical namespace to allow server implementers to logically organize the IORs of the objects that they export. Naming Contexts are CORBA objects within the Naming Service. There are several operations and methods defined in the naming context IDL. The most commonly used of these are "bind()" and "resolve()". Servers export IORs in a particular naming context by using the "bind()" operation, specifying the name of the object being exported along with the corresponding IOR. Clients then use the "resolve()" operation to obtain the IOR corresponding to a particular name.

Client Access to the Root Naming Context

The standard mechanism for finding the Naming Service and obtaining the root Naming Context is the “resolve_initial_references()” API. This API makes use of proprietary mechanisms to contact the Naming Service and extract the root naming context. Generally, this involves having the environment previously configured with the host on which the name server is running, the port ID, and so on.

The client application can connect to the Naming Service running on the Cisco CRS-1 Series router using a Domain Name Server (DNS) configured router name or through the Naming Service IOR as follows:

```
client -ORBInitRef NameService=iioploc://<router_name>:10001/NameService

client -ORBInitRef NameService=file://<IOR file>
```

If CORBA is used with SSL, then the IOR method of contacting to the Naming Service must be used along with a service configurator file:

```
client -ORBInitRef NameService=file://<IOR file> -ORBSvcConf <config file>
```

Here the service configurator file would contain the SSL variable definitions, for example:

```
static SSLIOP_Factory "-SSLAuthenticate NONE -SSLPrivateKey
                  PEM:server_key.pem -SSLCertificate PEM:server_cert.pem"
static Resource_Factory "-ORBProtocolFactory SSLIOP_Factory"
```

The Naming Service IOR file can be obtained from the router through HTTP at:

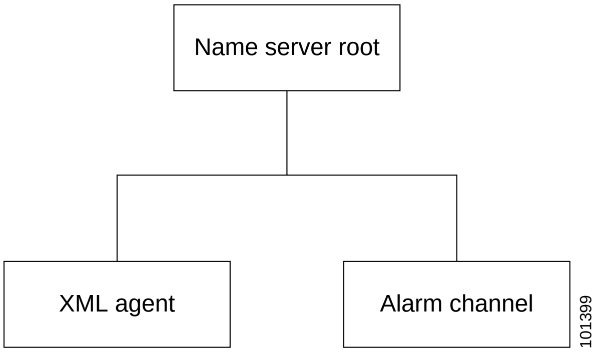
https://<router-name>/cwi/ns_ssl.ior

Note Connection to the Naming Service through an IP address is not supported.

LR Name Server Tree

The name hierarchy for the Cisco CRS-1 Series router is shown in Figure 11-1. The root naming context identifies the particular logical router (LR) on which the Name Server is running. Within the root context is the list of object references that components within the LR have exported.

Figure 11-1 Name Server Root



Event Notifications and Alarms

The CORBA Notification Service is used to deliver events asynchronously to registered clients. Each LR exports one structured notification channel for alarms called “AlarmChannel.” This channel is exported to the Name Server as shown in the Name Server tree for the LR. Client applications can obtain the AlarmChannel IOR and use its standard IDL interface to register interest in events and alarms based on filters.

CORBA Notification Structure

Client applications can receive asynchronous events through the CORBA Notification Service. The alarm notifications are in the form of CORBA structured events where the fixed header consists of the following definitions:

- Domain = LR DNS Name
- Type = alarm
- Instance = Event ID

The complete definition of the structured event is as follows:

Fixed header:

```
<LR_dnsname>.alarm.<event_id>
```

Filterable data:

```
filter:SourceID :
filter:Category :
filter:Severity :

Variable header:
variable:SourceID :
variable:Category :
variable:Severity :
```

Payload:

```
<?xml version= "1.0" encoding= "UTF-8"?>
<Event Version= "1.0" Module= "AlarmAgent">
  <Alarm>
    <SourceID>String</SourceID>
    <EventID>Number</EventID>
    <Timestamp>Number</Timestamp>
    <Category>String</Category>
    <Group>String</Group>
    <Code>String</Code>
    <Severity>String</Severity>
    <State>String</State>
    <CorrelationID>Number</CorrelationID>
    <AdditionalText>String</AdditionalText>
  </Alarm>
</Event>
```

Severity and State are enumerations are as follows:

Severity:

```
Unknown,
Emergency,
Alert,
Critical ,
Error ,
Warning,
Notice,
Informational,
Debug
```

State:

```
NotAvailable,
Active,
Clear
```

CORBA XML Agent Initiative

The CORBA XML agent is started on the router through the **xml agent corba** command-line interface (CLI) command. The **ssl** option is used to enable SSL for any subsequent client-to-agent CORBA connections.

The CORBA XML agent is started as follows:

```
router# configure terminal
router(config)# xml agent corba ssl
router(config)# commit
router(config)# exit
```

CORBA XML Limitations

The following limitations apply to the XML transport over CORBA:

- The maximum size of the response data returned to the client application is 32 KB. Responses containing more than 32 KB is returned through iterators as described in Chapter 7, "Cisco CRS-1 Series XML and Large Data Retrieval (Iterators)."
- The maximum number of simultaneous client logins that is accepted by the CORBA XML agent is five. Each client may in turn have up to four open connections.
- The inactivity timeout for a client connection is 15 minutes. After this timeout, the client connection is terminated by the CORBA XML agent.
- Client Object Request Brokers may have buffer limits that need to be adjusted for large XML responses.

XML Agent Errors

All the errors are returned as XML responses with embedded error codes and messages as defined in Chapter 10, "Error Reporting in Cisco CRS-1 Series XML Responses."

Table 11-2 describes the error codes specific to the XML agent for CORBA.

Table 11-2 XML Agent Error Codes

Error Code	Description
0x00000000	Success
0x7FFF0001	Login session failed to authenticate
0x7FFF0002	Login session start failed
0x7FFF0003	Login session activation failed
0x7FFF0004	Login session context invalid
0x7FFF0005	XML response dump from XML infrastructure has failed
0x7FFF0006	XML parsing in XML infrastructure has failed
0x7FFF0007	Login session handle or context is invalid

CHAPTER 12

Summary of Cisco CRS-1 Series XML API Configuration Tags

Table 12-1 provides the command-line interface (CLI) to extensible markup language (XML) application programming interface (API) tag mapping for the Cisco CRS-1 Series Carrier Routing System (Cisco CRS-1 Series) router target configuration.

Table 12-1 CLI Command or Operation to XML Tag Mapping

CLI Command or Operation	XML Tag
`To end, abort, or exit`[1] `(from top config mode)`	`<Unlock>`[2]
`clear`	`<Clear>`
`show config`	`<Get> with <Configuration Source=“ChangedConfig”>`
`show config running`	`<Get> with <Configuration Source=“CurrentConfig”>`
`show config merge`	`<Get> with <Configuration Source=“MergedConfig”>`
`show config failed`	`<Load> with <FailedConfig> followed by <Get> with <Configuration Source=”ChangedConfig”>`
`configure exclusive`[3]	`<Lock>`[4]
`To change the selected config`	`<Set> with <Configuration>`
`To delete the selected config`	`<Delete> with <Configuration>`
`commit`	`<Commit Mode=“BestEffort”>`
`commit atomic`	`<Commit Mode=“Atomic”>`
`show config failed`	`<Load> with <FailedConfig>`
`show commit changes` ***commitid***	`<Get> with <Configuration Source=“CommitChanges” ForCommitID=“`***commitid***`”>`
`show commit changes since` ***commitid***	`<Get> with <Configuration Source=“CommitChanges” SinceCommitID=“`***commitid***`”>`
`rollback configuration to` ***commitid***	`<Rollback> with <CommitID>`
`rollback configuration last` ***number***	`<Rollback> with <Previous>`
`show rollback changes to` ***commitid***	`<Get> with <Configuration Source=“RollbackChanges” ToCommitID=“`***commitid***`”>`
`show rollback changes last` ***number***	`<Get> with <Configuration Source=“RollbackChanges” PreviousCommits=“`***number***`”>`
`show rollback points`	`<GetConfigurationHistory RollbackOnly=“true”>`
`show configuration sessions`	`<GetConfigurationSessions>`

[1] These CLI operations end the configuration session and unlock the running configuration session if it was locked.

2. This XML tag releases the lock on a running configuration but does not end the configuration session.
3. This CLI command starts a new configuration session and locks the running configuration.
4. This XML tag locks the running configuration from a configuration session that is already in progress.

CHAPTER 13

Cisco CRS-1 Series XML Schemas

This chapter contains information about common XML schemas. The structure and allowable content of the extensible markup language (XML) request and response instances supported by the Cisco IOS XR XML application programming interface (API) are documented by means of XML schemas (.xsd files).

The Cisco CRS-1 Series XML schemas are documented using the standard World Wide Web Consortium (W3C) XML schema language, which provides a much more powerful and flexible mechanism for describing schemas than can be achieved using Document Type Definitions (DTDs). The set of Cisco CRS-1 Series XML schemas consist of a small set of common high-level schemas and a larger number of component-specific schemas as described in this chapter.

Note For more information on the W3C XML Schema standard, refer to http://www.w3.org/XML/Schema.

The following sections are described:

- XML Schema Retrieval, page 13-117
- Common XML Schemas, page 13-117

XML Schema Retrieval

The XML schemas that belong to the features in a particular package are obtained as a .tar file from cisco.com at http://www.cisco.com/cgi-bin/Software/IOXPlanner/ioxplanner.cgi. Once untarred, all the XML schema files appear as a flat directory of .xsd files, and can be opened with any XML schema viewing application such as XMLSpy.

Common XML Schemas

Among the .xsd files that belong to a BASE package are the common Cisco IOS XR XML schemas that include definitions of the high-level XML request and response instances and a number of common datatype definitions. These common XML schemas are:

- common_types.xsd
- error_types.xsd
- namespace_types.xsd

The XML schema file containing all configuration operations is also among the BASE package common schemas, and is named cfgmgr_operations.xsd.

Component XML Schemas

In addition to the common XML schemas, component XML schemas (such as native data) are provided and contain the data model for each feature. There is typically one component XML schema for each major type of data supported by the component—configuration, operational, action, administration operational, and administration action data—plus any complex data type definitions in the operational space.

Note Sometimes common schema files exist for a component that contain resources used by the component's other schema files (for example, the data types to be used by both configuration data and operational data).

You should use only the XML objects that are defined in the XML schema files.

Note You should not use any unpublished objects that may be shown in the XML returned from the router.

Schema File Organization

For the Cisco CRS-1 Series system, there is no hard link from the high-level XML request schemas (namespace_types.xsd) and the component schemas. Instead, links appear in the component schemas in the form of include elements that specify the file in which the parent element exists. The name of the component .xsd file also indicates where in the hierarchy the file's contents reside. If the file ends with _cfg.xsd, it appears as a child of "Configuration"; if it ends with _if_cfg.xsd, it appears as a child of "InterfaceConfiguration", and so on. In addition, the comment header in each .xsd file names the parent object of each top level object in the schema.

Schema File Upgrades

If a new version of a schema file becomes available (or has to be uploaded to the router as part of an upgrade), the new version of the file can replace the old version of the file in a straight swap. All other files are unaffected. Therefore, if a component is replaced, only the .xsd files pertaining to that component is replaced.

Sample BGP Configuration

The following is an excerpt of the relevant portions of the command-line interface (CLI) configuration used as a basis for the Border Gateway Protocol (BGP) examples contained within this document:

```
router bgp 3
 timers bgp 60 180
 bgp router-id 10.1.0.1
 bgp update-delay 55
 bgp cluster-id 10.1.0.2
 bgp graceful-restart purge-time 300
 default-information originate
 bgp graceful-restart restart-time 180
 bgp log neighbor changes disable
 default-metric 10
 bgp graceful-restart stalepath-time 300
 bgp graceful-restart
 bgp as-path-loopcheck
 socket send-buffer-size 131072
 bgp bestpath med always
 bgp bestpath compare-routerid
 bgp bestpath med missing-as-worst
 socket receive-buffer-size 131072
 address-family ipv4 unicast
  distance bgp 140 145 150
  bgp dampening 1 1400 1800 2
  bgp scan-time 30
  network 10.100.1.0/24
  network 10.100.2.0/24
  aggregate-address 10.100.0.0/16 summary-only
  redistribute connected route-map MATCH_ONE_CONNECT
  redistribute static route-map MATCH_ONE_STATIC
exit
 address-family ipv4 multicast
  distance bgp 120 125 130
  maximum-paths 6
  bgp dampening 2 2400 2800 3
  bgp scan-time 40
  network 10.10.1.0/24
  network 10.10.2.0/24
  aggregate-address 80.100.0.0/16 summary-only
  redistribute connected
exit
.
.
.
neighbor 10.0.101.1
  remote-as 1
  ebgp-multihop 255
  address-family ipv4 unicast
   send-community-ebgp
   route-map EBGP_IN_1 in
  exit
  address-family ipv4 multicast
   send-community-ebgp
   route-map EBGP_IN_1 in
  exit
 exit
 neighbor 10.0.101.2
  remote-as 2
  ebgp-multihop 255
  address-family ipv4 unicast
   send-community-ebgp
   capability orf prefix-list receive
   route-map EBGP_IN_1 in
  exit
  address-family ipv4 multicast
   send-community-ebgp
   route-map EBGP_IN_1 in
  exit
 exit
 neighbor 10.0.101.3
  remote-as 3
  address-family ipv4 unicast
   route-map IBGP_IN_1 in
  exit
  address-family ipv4 multicast
   route-map IBGP_IN_1 in
  exit
 exit
 neighbor 10.0.101.4
  remote-as 4
  ebgp-multihop 255
  address-family ipv4 unicast
   route-map EBGP_IN_2 in
  exit
  address-family ipv4 multicast
   route-map EBGP_IN_2 in
  exit
 exit
neighbor 10.0.101.5
  remote-as 5
  ebgp-multihop 255
  address-family ipv4 unicast
   route-map EBGP_IN_3 in
  exit
  address-family ipv4 multicast
   route-map EBGP_IN_3 in
  exit
 exit
 neighbor 10.0.101.6
  remote-as 6
  ebgp-multihop 255
  address-family ipv4 unicast
   prefix-list orf in
   capability orf prefix-list both
  exit
  address-family ipv4 multicast
   prefix-list orf in
  exit
 exit
 neighbor 10.0.101.7
  remote-as 7
  ebgp-multihop 255
  address-family ipv4 unicast
   prefix-list orf in
   capability orf prefix-list send
  exit
  address-family ipv4 multicast
   prefix-list orf in
  exit
 exit
 neighbor 10.0.101.8
  remote-as 8
  ebgp-multihop 255
  address-family ipv4 multicast
  exit
 exit
.
.
.
exit
```

GLOSSARY

A

AAA authentication, authorization, and accounting. A network security service that provides the primary framework to set up access control on a Cisco CRS-1 Series router or access server. AAA is an architectural framework and modular means of configuring three independent, but closely related security functions in a consistent manner.

B

BGP Border Gateway Protocol. A routing protocol used between autonomous systems. It is the routing protocol that makes the internet work. BGP is a distance vector routing protocol that carries connectivity information and an additional set of BGP attributes. These attributes allow for a rich set of policies for deciding what the best route to reach a given destination is.

Border Gateway Protocol See BGP.

C

CLI command-line interface

CORBA Common Object Request Broker Architecture. Specification that provides the standard interface definition between OMG-compliant objects. CORBA allows applications to communicate with one another no matter where they are located or who has designed them.

CWI Craft Works Interface. Remote web-based monitoring, configuration, fault management, and troubleshooting system for the Cisco CRS-1 Series router.

E

eBGP external Border Gateway Protocol. BGP sessions are established between routers in different autonomous systems. eBGPs communicate among different network domains.

EGP Exterior Gateway Protocol. Internet protocol for exchanging routing information between different autonomous systems. EGP is an obsolete protocol that was replaced by BGP. See also *BGP*.

extensible markup language See XML.

G

Gbps Gigabits per second. The amount of data that can be sent in a fixed amount of time. 1 gigabit = 2^{30} bits, 1,073,741,824 bits.

H

HTTP Hypertext Transfer Protocol. Used by web browsers and web servers to transfer files, such as text and graphic files. The Hypertext Transfer Protocol (HTTP) is the set of rules for exchanging files (text, graphic images, sound, video, and other multimedia files) on the World Wide Web. Relative to the TCP/IP suite of protocols (which are the basis for information exchange on the Internet), HTTP is an application protocol.

Hypertext Transfer Protocol See HTTP.

I

IDL Interface Definition Language. External client applications use CORBA IDL to exchange XML encoded request and response streams with the CORBA XML agent running on the router

IIOP Internet Inter-ORB Protocol (IIOP). The protocol used for accessing objects across the Internet.

IOR Interoperable Object Reference. A bundle of data that is used by a CORBA application to determine how to connect to a server and send requests to an object.

IOS XR The Cisco operating system used on Cisco CRS-1 Series routers.

L

line card See modular services card. Line cards are now referred to as MCSs in the Cisco CRS-1 Series router.

LR logical router. A Cisco CRS-1 Series routing system can be partitioned into several logical routers, each of which is managed independently. The terms Cisco CRS-1 Series router and LR are used interchangeably in this document.

M

modular services card Module where the ingress and egress packet processing and queueing functions are carried out in the Cisco CRS-1 Series router architecture. Up to 16 MSCs are installed in a line card chassis; each MSC must have an associated PLIM (of which there are several types to provide a variety of physical interfaces). The MSC and PLIM mate together on the line card chassis midplane.

MSCs are also referred to as line cards.

MPLS-TE Multiprotocol Label Switching traffic engineering

MSC See modular services card.

N

node A card installed and running in a Cisco CRS-1 Series routing system. In the Cisco 12000 series Internet router, nodes are identified by slot number (for example, node 1).

R

router Network layer device that uses one or more routing metrics to determine the optimal path along which network traffic should be forwarded. Routers forward packets from one network to another based on network layer information.

running configuration The router configuration in effect. Although, the user can save multiple versions of the router configuration for future reference, only one copy of the running configuration is in the router at any given time. An explicit commit operation must be performed to make changes to or update the running configuration on the Cisco CRS-1 Series router.

S

software configuration A list of packages activated for a particular node. A software configuration consists of a boot package and additional feature packages.

SSH Secure Shell

SSL Secure Socket Layer

startup configuration The router configuration designated to be applied on next router startup.

switchover A switch between the active and standby cards; the old active card may be dead prior to switchover (death of the active card is one of the causes for the switchover). Also known as failover.

system reload Reload of an Cisco CRS-1 Series router node.

system restart Soft reset of an Cisco CRS-1 Series router node. This involves restarting all the processes running on that node.

T

TAC Technical Assistance Center

target configuration The current Cisco IOS XR running configuration plus the autonomous changes made to that configuration by a user. The target configuration is promoted to the running configuration by means of the **commit** command.

Tbps Terabits per second = 1,000,000,000,000 (1 trillion) bits per second. The amount of data that can be sent in a fixed amount of time.

Telnet Standard terminal emulation protocol in the TCP/IP protocol stack. Telnet is used for remote terminal connection, enabling users to log in to remote systems and use resources as if they were connected to a local system. Telnet is defined in RFC 854.

Terabyte A unit of computer memory or data storage capacity equal to 1,024 gigabytes (2^{40} bytes). Approximately 1 trillion bytes.

X

XML extensible markup language. A standard maintained by the World Wide Web Consortium (W3C) that defines a syntax that lets you create markup languages to specify information structures. Information structures define the type of information, for example, subscriber name or address, not how the information looks (bold, italic, and so on). External processes can manipulate these information structures and publish them in a variety of formats. XML allows you to define your own customized markup language.

XML agent A process on the Cisco CRS-1 Series router that is sent XML requests by XML clients and is responsible for carrying out the actions contained in the request and returning an XML response back to the client. The XML Agent for CORBA is an example of an XML agent provided on the Cisco CRS-1 Series router.

XML client An external application that sends an XML request to the Cisco CRS-1 Series router and receives XML responses to those requests.

XML operation A portion of an XML request that specifies an operation that the XML client would like the XML agent to perform.

XML operation provider The Cisco CRS-1 Series router code that carries out a particular XML operation including parsing the operation XML, performing the operation, and assembling the operation XML response.

XML request An XML document sent to the Cisco CRS-1 Series router containing a number of requested operations to be carried out.

XML response The response to an XML request.

XML schema An XML document specifying the structure and possible contents of XML elements that can be contained in an XML document.

INDEX